

lucid Reference Manual

0.1_rc2

Generated by Doxygen 1.5.2

Tue Jun 19 20:27:20 2007

Contents

1	lucid Documentation	1
1.1	Why another library?	1
1.2	Current Status	1
1.3	Documentation	1
1.4	Bugs, Patches, Wishes	2
2	lucid Module Index	3
2.1	lucid Modules	3
3	lucid Data Structure Index	5
3.1	lucid Data Structures	5
4	lucid File Index	7
4.1	lucid File List	7
5	lucid Page Index	11
5.1	lucid Related Pages	11
6	lucid Module Documentation	13
6.1	Internet address conversion	13
6.2	Bitmap conversion	18
6.3	Character classification and manipulation	21
6.4	Secure chroot wrappers	26
6.5	Command execution wrappers	30
6.6	Flag list conversion	37
6.7	Simple doubly linked lists	46
6.8	Log system multiplexer	51
6.9	Memory area manipulation	63
6.10	Miscellaneous helpers	71
6.11	Create or open files	80

6.12	Formatted output conversion	83
6.13	Formatted input conversion	96
6.14	String classification and conversion	105
6.15	Dynamic string allocator	127
6.16	String tokenizer	136
6.17	TCP socket wrappers	144
6.18	Whirlpool hash function	146
7	lucid Data Structure Documentation	157
7.1	__printf_t Struct Reference	157
7.2	__scanf_t Struct Reference	159
7.3	_mem_pool_t Struct Reference	160
7.4	flist32_t Struct Reference	161
7.5	flist64_t Struct Reference	162
7.6	list_head Struct Reference	163
7.7	log_options_t Struct Reference	164
7.8	stralloc_t Struct Reference	166
7.9	strtok_t Struct Reference	167
7.10	whirlpool_t Struct Reference	168
8	lucid File Documentation	171
8.1	addr.h File Reference	171
8.2	addr/addr_from_str.c File Reference	172
8.3	addr/addr_hton.c File Reference	173
8.4	addr/addr_htos.c File Reference	174
8.5	addr/addr_ntoh.c File Reference	175
8.6	addr/addr_stoh.c File Reference	176
8.7	addr/addr_to_str.c File Reference	177
8.8	bitmap.h File Reference	178
8.9	bitmap/i2v32.c File Reference	179
8.10	bitmap/i2v64.c File Reference	180
8.11	bitmap/v2i32.c File Reference	181
8.12	bitmap/v2i64.c File Reference	182
8.13	char.h File Reference	183
8.14	chroot.h File Reference	185
8.15	chroot/chroot_fd.c File Reference	186
8.16	chroot/chroot_mkdirp.c File Reference	187

8.17	chroot/chroot_secure_chdir.c File Reference	188
8.18	doxygen/examples.h File Reference	189
8.19	doxygen/license.h File Reference	190
8.20	doxygen/main.h File Reference	191
8.21	exec.h File Reference	192
8.22	exec/exec_fork.c File Reference	193
8.23	exec/exec_fork_background.c File Reference	194
8.24	exec/exec_fork_pipe.c File Reference	195
8.25	exec/exec_replace.c File Reference	196
8.26	flist.h File Reference	197
8.27	flist/flist32_from_str.c File Reference	199
8.28	flist/flist32_getkey.c File Reference	200
8.29	flist/flist32_getval.c File Reference	201
8.30	flist/flist32_to_str.c File Reference	202
8.31	flist/flist64_from_str.c File Reference	203
8.32	flist/flist64_getkey.c File Reference	204
8.33	flist/flist64_getval.c File Reference	205
8.34	flist/flist64_to_str.c File Reference	206
8.35	list.h File Reference	207
8.36	log.h File Reference	209
8.37	log/log_close.c File Reference	212
8.38	log/log_init.c File Reference	213
8.39	log/log_internal.c File Reference	214
8.40	mem.h File Reference	216
8.41	mem/mem_alloc.c File Reference	217
8.42	mem/mem_ccpy.c File Reference	218
8.43	mem/mem_chr.c File Reference	219
8.44	mem/mem_cmp.c File Reference	220
8.45	mem/mem_cpy.c File Reference	221
8.46	mem/mem_dup.c File Reference	222
8.47	mem/mem_free.c File Reference	223
8.48	mem/mem_freeall.c File Reference	224
8.49	mem/mem_idx.c File Reference	225
8.50	mem/mem_internal.h File Reference	226
8.51	mem/mem_realloc.c File Reference	227
8.52	mem/mem_set.c File Reference	228

8.53	misc.h File Reference	229
8.54	misc/copy_file.c File Reference	230
8.55	misc/isdir.c File Reference	231
8.56	misc/isfile.c File Reference	232
8.57	misc/islink.c File Reference	233
8.58	misc/ismount.c File Reference	234
8.59	misc/ispath.c File Reference	235
8.60	misc/mkdirnamep.c File Reference	236
8.61	misc/mkdirp.c File Reference	237
8.62	misc/readsymlink.c File Reference	238
8.63	misc/runlink.c File Reference	239
8.64	open.h File Reference	240
8.65	open/open_append.c File Reference	241
8.66	open/open_excl.c File Reference	242
8.67	open/open_read.c File Reference	243
8.68	open/open_rw.c File Reference	244
8.69	open/open_trunc.c File Reference	245
8.70	open/open_write.c File Reference	246
8.71	printf.h File Reference	247
8.72	printf/asprintf.c File Reference	248
8.73	printf/dprintf.c File Reference	249
8.74	printf/printf.c File Reference	250
8.75	printf/snprintf.c File Reference	251
8.76	printf/vasprintf.c File Reference	252
8.77	printf/vdprintf.c File Reference	253
8.78	printf/vprintf.c File Reference	254
8.79	printf/vsnprintf.c File Reference	255
8.80	scanf.h File Reference	258
8.81	scanf/sscanf.c File Reference	259
8.82	scanf/vsscanf.c File Reference	260
8.83	str.h File Reference	263
8.84	str/str_check.c File Reference	266
8.85	str/str_chr.c File Reference	267
8.86	str/str_cmp.c File Reference	268
8.87	str/str_cmpn.c File Reference	269
8.88	str/str_cpy.c File Reference	270

8.89	str/str_cpyn.c File Reference	271
8.90	str/str_dup.c File Reference	272
8.91	str/str_equal.c File Reference	273
8.92	str/str_len.c File Reference	274
8.93	str/str_path_basename.c File Reference	275
8.94	str/str_path_concat.c File Reference	276
8.95	str/str_path_dirname.c File Reference	277
8.96	str/str_path_isabs.c File Reference	278
8.97	str/str_path_isdot.c File Reference	279
8.98	str/str_rchr.c File Reference	280
8.99	str/str_read.c File Reference	281
8.100	str/str_readfile.c File Reference	282
8.101	str/str_readline.c File Reference	283
8.102	str/str_str.c File Reference	284
8.103	str/str_tolower.c File Reference	285
8.104	str/str_toumax.c File Reference	286
8.105	str/str_toupper.c File Reference	287
8.106	stralloc.h File Reference	288
8.107	stralloc/stralloc_cat.c File Reference	290
8.108	stralloc/stralloc_catb.c File Reference	291
8.109	stralloc/stralloc_catf.c File Reference	292
8.110	stralloc/stralloc_catm.c File Reference	293
8.111	stralloc/stralloc_cats.c File Reference	294
8.112	stralloc/stralloc_cmp.c File Reference	295
8.113	stralloc/stralloc_copy.c File Reference	296
8.114	stralloc/stralloc_copyb.c File Reference	297
8.115	stralloc/stralloc_copys.c File Reference	298
8.116	stralloc/stralloc_finalize.c File Reference	299
8.117	stralloc/stralloc_free.c File Reference	300
8.118	stralloc/stralloc_init.c File Reference	301
8.119	stralloc/stralloc_ready.c File Reference	302
8.120	stralloc/stralloc_readyplus.c File Reference	303
8.121	stralloc/stralloc_zero.c File Reference	304
8.122	strtok.h File Reference	305
8.123	strtok/strtok_append.c File Reference	307
8.124	strtok/strtok_count.c File Reference	308

8.125	strtok/strtok_delete.c File Reference	309
8.126	strtok/strtok_free.c File Reference	310
8.127	strtok/strtok_init_argv.c File Reference	311
8.128	strtok/strtok_init_str.c File Reference	312
8.129	strtok/strtok_next.c File Reference	313
8.130	strtok/strtok_prev.c File Reference	314
8.131	strtok/strtok_toargv.c File Reference	315
8.132	strtok/strtok_tostr.c File Reference	316
8.133	tcp.h File Reference	317
8.134	tcp/tcp_connect.c File Reference	318
8.135	tcp/tcp_listen.c File Reference	319
8.136	whirlpool.h File Reference	320
8.137	whirlpool/whirlpool_add.c File Reference	322
8.138	whirlpool/whirlpool_digest.c File Reference	323
8.139	whirlpool/whirlpool_finalize.c File Reference	324
8.140	whirlpool/whirlpool_init.c File Reference	325
8.141	whirlpool/whirlpool_tables.h File Reference	326
8.142	whirlpool/whirlpool_transform.c File Reference	327
9	lucid Page Documentation	329
9.1	Examples	329
9.2	License	330

Chapter 1

lucid Documentation

The lucid library combines a lot of useful functions i wrote for my projects. There are a lot of custom functions for strings, doubly-linked lists, bitmaps and flag lists, input/output, cryptographic digests, and tcp connections.

Some of these functions, especially string and list functions, are completely self-contained and do not rely on libc. This makes integration in foreign projects as easy as possible.

On the other hand, functions for input/output or chroot are just wrappers around libc library functions, but may still be usefull to others as well.

The size of functions range from a few hundred bytes to about 30K.

1.1 Why another library?

This library was written for my own projects. As the number of foreign libraries i included grew, i have collected all functions i needed - and meanwhile most of them reimplemented - and made an own library.

1.2 Current Status

Actually 4 projects (libvserver, vcd, vstatd, vwrappers) are using lucid now, so an own shared library was necessary. The library contains a test suite for the most important functions. Functions not explicetely tested are either tested implicitly by other functions and a test may be written in the future, or the function is so simple that you can just hope it works ;)

1.3 Documentation

All function are documented with an inline source browser for easy learning and reference. To get an overview of function families please start at the Modules page. Experienced users may look up information in the Globals, Data Fields or File List.

Also take a look at the **Examples** (p. 329) and read the **License** (p. 330).

1.4 Bugs, Patches, Wishes

If you have found a bug in lucid that was not discovered by the test suite, or if you have any suggestion, wishes or patches for the future of lucid, please contact me at [hollow\[at\]gentoo.org](mailto:hollow[at]gentoo.org)

Chapter 2

lucid Module Index

2.1 lucid Modules

Here is a list of all modules:

Internet address conversion	13
Bitmap conversion	18
Character classification and manipulation	21
Secure chroot wrappers	26
Command execution wrappers	30
Flag list conversion	37
Simple doubly linked lists	46
Log system multiplexer	51
Memory area manipulation	63
Miscellaneous helpers	71
Create or open files	80
Formatted output conversion	83
Formatted input conversion	96
String classification and conversion	105
Dynamic string allocator	127
String tokenizer	136
TCP socket wrappers	144
Whirlpool hash function	146

Chapter 3

lucid Data Structure Index

3.1 lucid Data Structures

Here are the data structures with brief descriptions:

__printf_t	157
__scanf_t	159
__mem_pool_t	160
flist32_t (32 bit list object)	161
flist64_t (64 bit list object)	162
list_head (List head)	163
log_options_t (Multiplexer configuration data)	164
stralloc_t (Dynamic string allocator tracking data)	166
strtok_t	167
whirlpool_t (Dynamic whirlpool state data)	168

Chapter 4

lucid File Index

4.1 lucid File List

Here is a list of all files with brief descriptions:

addr.h	171
bitmap.h	178
char.h	183
chroot.h	185
exec.h	192
flist.h	197
list.h	207
log.h	209
mem.h	216
misc.h	229
open.h	240
printf.h	247
scanf.h	258
str.h	263
stralloc.h	288
strtok.h	305
tcp.h	317
whirlpool.h	320
addr/addr_from_str.c	172
addr/addr_hton.c	173
addr/addr_htos.c	174
addr/addr_ntoh.c	175
addr/addr_stoh.c	176
addr/addr_to_str.c	177
bitmap/i2v32.c	179
bitmap/i2v64.c	180
bitmap/v2i32.c	181
bitmap/v2i64.c	182
chroot/chroot_fd.c	186
chroot/chroot_mkdirp.c	187
chroot/chroot_secure_chdir.c	188
doxygen/examples.h	189
doxygen/license.h	190

doxygen/main.h	191
exec/exec_fork.c	193
exec/exec_fork_background.c	194
exec/exec_fork_pipe.c	195
exec/exec_replace.c	196
flist/flist32_from_str.c	199
flist/flist32_getkey.c	200
flist/flist32_getval.c	201
flist/flist32_to_str.c	202
flist/flist64_from_str.c	203
flist/flist64_getkey.c	204
flist/flist64_getval.c	205
flist/flist64_to_str.c	206
log/log_close.c	212
log/log_init.c	213
log/log_internal.c	214
mem/mem_alloc.c	217
mem/mem_ccpy.c	218
mem/mem_chr.c	219
mem/mem_cmp.c	220
mem/mem_cpy.c	221
mem/mem_dup.c	222
mem/mem_free.c	223
mem/mem_freeall.c	224
mem/mem_idx.c	225
mem/mem_internal.h	226
mem/mem_realloc.c	227
mem/mem_set.c	228
misc/copy_file.c	230
misc/isdir.c	231
misc/isfile.c	232
misc/islink.c	233
misc/ismount.c	234
misc/ispath.c	235
misc/mkdirnamep.c	236
misc/mkdirp.c	237
misc/readsymlink.c	238
misc/runlink.c	239
open/open_append.c	241
open/open_excl.c	242
open/open_read.c	243
open/open_rw.c	244
open/open_trunc.c	245
open/open_write.c	246
printf/asprintf.c	248
printf/dprintf.c	249
printf/printf.c	250
printf/snprintf.c	251
printf/vasprintf.c	252
printf/vdprintf.c	253
printf/vprintf.c	254
printf/vsnprintf.c	255
scanf/sscanf.c	259
scanf/vsscanf.c	260

str/str_check.c	266
str/str_chr.c	267
str/str_cmp.c	268
str/str_cmpn.c	269
str/str_cpy.c	270
str/str_cpyn.c	271
str/str_dup.c	272
str/str_equal.c	273
str/str_len.c	274
str/str_path_basename.c	275
str/str_path_concat.c	276
str/str_path_dirname.c	277
str/str_path_isabs.c	278
str/str_path_isdot.c	279
str/str_rchr.c	280
str/str_read.c	281
str/str_readfile.c	282
str/str_readline.c	283
str/str_str.c	284
str/str_tolower.c	285
str/str_toumax.c	286
str/str_toupper.c	287
stralloc/stralloc_cat.c	290
stralloc/stralloc_catb.c	291
stralloc/stralloc_catf.c	292
stralloc/stralloc_catm.c	293
stralloc/stralloc_cats.c	294
stralloc/stralloc_cmp.c	295
stralloc/stralloc_copy.c	296
stralloc/stralloc_copyb.c	297
stralloc/stralloc_copys.c	298
stralloc/stralloc_finalize.c	299
stralloc/stralloc_free.c	300
stralloc/stralloc_init.c	301
stralloc/stralloc_ready.c	302
stralloc/stralloc_readyplus.c	303
stralloc/stralloc_zero.c	304
strtok/strtok_append.c	307
strtok/strtok_count.c	308
strtok/strtok_delete.c	309
strtok/strtok_free.c	310
strtok/strtok_init_argv.c	311
strtok/strtok_init_str.c	312
strtok/strtok_next.c	313
strtok/strtok_prev.c	314
strtok/strtok_toargv.c	315
strtok/strtok_tostr.c	316
tcp/tcp_connect.c	318
tcp/tcp_listen.c	319
whirlpool/whirlpool_add.c	322
whirlpool/whirlpool_digest.c	323
whirlpool/whirlpool_finalize.c	324
whirlpool/whirlpool_init.c	325
whirlpool/whirlpool_tables.h	326

whirlpool/ whirlpool_transform.c	327
---	-----

Chapter 5

lucid Page Index

5.1 lucid Related Pages

Here is a list of all related documentation pages:

Examples	329
License	330

Chapter 6

lucid Module Documentation

6.1 Internet address conversion

6.1.1 Detailed Description

The **addr_htos()** (p. 14), **addr_hton()** (p. 14), **addr_stoh()** (p. 15) and **addr_ntoh()** (p. 14) functions convert from host- to network-byteorder, and vice versa, respectively.

The **addr_from_str()** (p. 15) function converts the Internet host address in standard numbers-and-dots notation pointed to by the string *str* into binary data and stores result in the *ip*/*mask* pair of pointers. **addr_from_str()** (p. 15) returns 0 if no argument was converted, 1 if *ip* was converted, 2 for *mask* and 3 for both.

The **addr_to_str()** (p. 16) function converts the Internet host address given in the *ip*/*mask* pair of pointers to a string in standard numbers-and-dots notation. The returned string is obtained by *malloc* and should be *free(3)*'d by the caller.

Functions

- **uint16_t addr_htos** (uint16_t *addr*)
convert address from host to network byte order
- **uint32_t addr_hton** (uint32_t *addr*)
convert address from host to network byte order
- **uint32_t addr_ntoh** (uint32_t *addr*)
convert address from network to host byte order
- **uint16_t addr_stoh** (uint16_t *addr*)
convert address from network to host byte order
- **int addr_from_str** (const char **str*, uint32_t **ip*, uint32_t **mask*)
convert string to IP address and netmask
- **char * addr_to_str** (uint32_t *ip*, uint32_t *mask*)
convert IP address and netmask to string

6.1.2 Function Documentation

6.1.2.1 `uint16_t addr_htos (uint16_t addr)`

convert address from host to network byte order

Parameters:

← *addr* address in host byte order

Returns:

address in network byte order

Definition at line 26 of file `addr_htos.c`.

Referenced by `addr_stoh()`, and `tcp_listen()`.

```

27 {
28     if (islitend())
29         return ((addr >> 8) & 0xFF) | (addr << 8);
30     else
31         return addr;
32 }
```

6.1.2.2 `uint32_t addr_hton (uint32_t addr)`

convert address from host to network byte order

Parameters:

← *addr* address in host byte order

Returns:

address in network byte order

Definition at line 26 of file `addr_hton.c`.

Referenced by `addr_from_str()`, and `addr_ntoh()`.

```

27 {
28     if (islitend())
29         return (addr >> 24) |
30             ((addr & 0xff0000) >> 8) |
31             ((addr & 0xff00 ) << 8) |
32             (addr << 24);
33     else
34         return addr;
35 }
```

6.1.2.3 `uint32_t addr_ntoh (uint32_t addr)`

convert address from network to host byte order

Parameters:

← *addr* address in network byte order

Returns:

address in host byte order

Definition at line 19 of file `addr_ntoh.c`.

References `addr_hton()`.

```
20 {  
21     return addr_hton(addr);  
22 }
```

6.1.2.4 uint16_t addr_stoh (uint16_t addr)

convert address from network to host byte order

Parameters:

← *addr* address in network byte order

Returns:

address in host byte order

Definition at line 19 of file `addr_stoh.c`.

References `addr_htos()`.

```
20 {  
21     return addr_htos(addr);  
22 }
```

6.1.2.5 int addr_from_str (const char * str, uint32_t * ip, uint32_t * mask)

convert string to IP address and netmask

Parameters:

← *str* string in CIDR or netmask notation

→ *ip* pointer to store IP address in network byte order

→ *mask* pointer to store netmask in network byte order

Returns:

0 if no argument was converted, 1 if ip was converted, 2 for mask and 3 for both.

Definition at line 21 of file `addr_from_str.c`.

References `_lucid_sscanf()`, `addr_hton()`, `str_chr()`, `str_isdigit`, `str_isempty`, and `str_len()`.

Referenced by `tcp_connect()`, and `tcp_listen()`.

```

22 {
23     int rc = 0;
24     int cidr;
25
26     union {
27         uint8_t b[4];
28         uint32_t l;
29     } u;
30
31     const char *p = str_chr(str, '/', str_len(str));
32
33     /* ip address */
34     if (!p || p - str > 0) {
35         if (_lucid_sscanf(str, "%hhu.%hhu.%hhu.%hhu",
36                             &u.b[0], &u.b[1], &u.b[2], &u.b[3]) == 4) {
37             if (ip)
38                 *ip = u.l;
39
40             rc = 1;
41         }
42     }
43
44     if (!p)
45         return rc;
46
47     p++;
48
49     /* netmask in CIDR notation */
50     if (!str_isempty(p) && str_isdigit(p)) {
51         if (_lucid_sscanf(p, "%d", &cidr) == 1 && (cidr > 0 && cidr <= 32)) {
52             if (mask)
53                 *mask = addr_hton(0xffffffff & ~((1 << (32 - cidr)) - 1));
54
55             rc += 2;
56         }
57     }
58
59     /* netmask in ip notation */
60     if (!str_isempty(p)) {
61         if (_lucid_sscanf(p, "%hhu.%hhu.%hhu.%hhu",
62                             &u.b[0], &u.b[1], &u.b[2], &u.b[3]) == 4) {
63             if (mask)
64                 *mask = u.l;
65
66             rc += 2;
67         }
68     }
69
70     return rc;
71 }

```

6.1.2.6 char* addr_to_str (uint32_t ip, uint32_t mask)

convert IP address and netmask to string

Parameters:

- ← *ip* IP address to convert in network byte order
- ← *mask* netmask to convert in network byte order

Returns:

string in netmask notation (obtained with malloc(3))

Note:

The caller should free obtained memory using `free(3)`

See also:

`malloc(3)`
`free(3)`

Definition at line 20 of file `addr_to_str.c`.

References `_lucid_asprintf()`.

```
21 {
22     char *buf;
23
24     char *ipp  = (char *) &ip;
25     char *maskp = (char *) &mask;
26
27     if (mask)
28         _lucid_asprintf(&buf, "%hhu.%hhu.%hhu.%hhu/%hhu.%hhu.%hhu.%hhu",
29                         ipp[0], ipp[1], ipp[2], ipp[3],
30                         maskp[0], maskp[1], maskp[2], maskp[3]);
31
32     else
33         _lucid_asprintf(&buf, "%hhu.%hhu.%hhu.%hhu",
34                         ipp[0], ipp[1], ipp[2], ipp[3]);
35
36     return buf;
37 }
```

6.2 Bitmap conversion

6.2.1 Detailed Description

The i2v and v2i family of functions convert between a bitmap and a bit index.

A bitmap is simply an integer with certain bits being 1 (enabled) and 0 (disabled).

These functions only return usable results if exactly one bit is enabled.

- Bit index to bitmap
The resulting bitmask is a simple arithmetic left shift of 1 index times.
- Bitmap to bit index
The resulting bit index is a simple arithmetic right shift until the map is empty.

These functions are mainly used by the flist family of functions.

Functions

- `uint32_t i2v32 (int index)`
convert bit index to 32 bit value
- `uint64_t i2v64 (int index)`
convert bit index to 64 bit value
- `int v2i32 (uint32_t val)`
convert 32 bit value to bit index
- `int v2i64 (uint64_t val)`
convert 64 bit value to bit index

6.2.2 Function Documentation

6.2.2.1 `uint32_t i2v32 (int index)`

convert bit index to 32 bit value

Parameters:

← *index* bit index (0-31)

Returns:

32 bit value

Definition at line 19 of file i2v32.c.

```

20 {
21     if (index < 0 || index > 31)
22         return 0;
23
24     return (1UL << index);
25 }
```

6.2.2.2 uint64_t i2v64 (int index)

convert bit index to 64 bit value

Parameters:

← *index* bit index (0-63)

Returns:

64 bit value

Definition at line 19 of file i2v64.c.

```
20 {  
21     if (index < 0 || index > 63)  
22         return 0;  
23  
24     return (1ULL << index);  
25 }
```

6.2.2.3 int v2i32 (uint32_t val)

convert 32 bit value to bit index

Parameters:

← *val* 32 bit value

Returns:

bit index (0-31)

Definition at line 19 of file v2i32.c.

```
20 {  
21     int index = 0;  
22  
23     if (val == 0)  
24         return -1;  
25  
26     while ((val = val >> 1))  
27         index++;  
28  
29     return index;  
30 }
```

6.2.2.4 int v2i64 (uint64_t val)

convert 64 bit value to bit index

Parameters:

← *val* 64 bit value

Returns:

bit index (0-63)

Definition at line 19 of file v2i64.c.

```
20 {  
21     int index = 0;  
22  
23     if (val == 0)  
24         return -1;  
25  
26     while ((val = val >> 1))  
27         index++;  
28  
29     return index;  
30 }
```

6.3 Character classification and manipulation

6.3.1 Detailed Description

The `char` family of macros check whether `ch`, which must have the value of an unsigned `char`, falls into a certain character class.

`char_isalnum()` (p. 24) checks for an alphanumeric character; it is equivalent to **`(char_isalpha(ch) (p. 24) || char_isdigit(ch) (p. 23))`**.

`char_isalpha()` (p. 24) checks for an alphabetic character; it is equivalent to **`(char_isupper(c) (p. 23) || char_islower(c) (p. 23))`**.

`char_isascii()` (p. 22) checks whether `ch` is a 7-bit unsigned `char` value that fits into the ASCII character set.

`char_isblank()` (p. 22) checks for a blank character; that is, a space or a tab.

`char_iscntrl()` (p. 23) checks for a control character.

`char_isdigit()` (p. 23) checks for a digit (0 through 9).

`char_isgraph()` (p. 23) checks for any printable character except space.

`char_islower()` (p. 23) checks for a lower-case character.

`char_isprint()` (p. 23) checks for any printable character including space.

`char_ispunct()` (p. 24) checks for any printable character which is not a space or an alphanumeric character.

`char_isspace()` (p. 23) checks for white-space characters. These are: space, form-feed (”), newline (”), carriage return (”), horizontal tab (”), and vertical tab (”).

`char_isupper()` (p. 23) checks for an uppercase letter.

`char_isxdigit()` (p. 24) checks for a hexadecimal digits, i.e. one of 0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F.

`char_tolower()` (p. 24) converts a character to lowercase if applicable.

`char_toupper()` (p. 24) converts a character to uppercase if applicable.

Defines

- **`#define char_isascii(ch) ((unsigned int)(ch) < 128u)`**
check for an ASCII character
- **`#define char_isblank(ch) (ch == ' ' || ch == '\t')`**
check for a blank character (space, horizontal tab)
- **`#define char_iscntrl(ch) ((unsigned int)(ch) < 32u || ch == 127)`**
check for an ASCII control character
- **`#define char_isdigit(ch) ((unsigned int)(ch - '0') < 10u)`**
check for a digit character (0-9)
- **`#define char_isgraph(ch) ((unsigned int)(ch - '!') < 94u)`**

check for graphable characters (excluding space)

- **#define char_islower(ch)** ((unsigned int)(ch - 'a') < 26u)
check for a lower-case character
- **#define char_isprint(ch)** ((unsigned int)(ch - ' ') < 95u)
check for a printable character (including space)
- **#define char_isspace(ch)** ((unsigned int)(ch - '\t') < 5u || ch == ' ')
check for a whitespace character (\t, \n, \v, \f, \r)
- **#define char_isupper(ch)** ((unsigned int)(ch - 'A') < 26u)
check for an upper-case character
- **#define char_isxdigit(ch)**
check for a hexadecimal character
- **#define char_isalpha(ch)** (char_islower(ch) || char_isupper(ch))
check for an upper- or lower-case character
- **#define char_isalnum(ch)** (char_isalpha(ch) || char_isdigit(ch))
check for an upper-, lower-case or digit character
- **#define char_ispunct(ch)**
check for a punctuation character
- **#define char_tolower(ch)** do { if (char_isupper(ch)) ch += 32; } while(0)
convert character to lower-case
- **#define char_toupper(ch)** do { if (char_islower(ch)) ch -= 32; } while(0)
convert character to upper-case

6.3.2 Define Documentation

6.3.2.1 #define char_isascii(ch) ((unsigned int)(ch) < 128u)

check for an ASCII character

Definition at line 67 of file char.h.

Referenced by str_check().

6.3.2.2 #define char_isblank(ch) (ch == ' ' || ch == '\t')

check for a blank character (space, horizontal tab)

Definition at line 70 of file char.h.

Referenced by str_check().

6.3.2.3 `#define char_iscntrl(ch) (((unsigned int)(ch) < 32u || ch == 127)`

check for an ASCII control character

Definition at line 73 of file char.h.

Referenced by str_check().

6.3.2.4 `#define char_isdigit(ch) (((unsigned int)(ch - '0') < 10u)`

check for a digit character (0-9)

Definition at line 76 of file char.h.

Referenced by str_check().

6.3.2.5 `#define char_isgraph(ch) (((unsigned int)(ch - '!') < 94u)`

check for graphable characters (excluding space)

Definition at line 79 of file char.h.

Referenced by str_check().

6.3.2.6 `#define char_islower(ch) (((unsigned int)(ch - 'a') < 26u)`

check for a lower-case character

Definition at line 82 of file char.h.

Referenced by str_check().

6.3.2.7 `#define char_isprint(ch) (((unsigned int)(ch - ' ') < 95u)`

check for a printable character (including space)

Definition at line 85 of file char.h.

Referenced by str_check().

6.3.2.8 `#define char_isspace(ch) (((unsigned int)(ch - '\t') < 5u || ch == '')`

check for a whitespace character (`\t`, `\n`, `\v`, `\f`, `\r`)

Definition at line 88 of file char.h.

Referenced by `_lucid_vsscanf()`, `str_check()`, and `str_toumax()`.

6.3.2.9 `#define char_isupper(ch) (((unsigned int)(ch - 'A') < 26u)`

check for an upper-case character

Definition at line 91 of file char.h.

Referenced by str_check().

6.3.2.10 `#define char_isxdigit(ch)`

Value:

```
(char_isdigit(ch) || \
    (unsigned int)(ch - 'a') < 6u || \
    (unsigned int)(ch - 'A') < 6u)
```

check for a hexadecimal character

Definition at line 94 of file char.h.

Referenced by str_check().

6.3.2.11 `#define char_isalpha(ch) (char_islower(ch) || char_isupper(ch))`

check for an upper- or lower-case character

Definition at line 100 of file char.h.

Referenced by str_check().

6.3.2.12 `#define char_isalnum(ch) (char_isalpha(ch) || char_isdigit(ch))`

check for an upper-, lower-case or digit character

Definition at line 103 of file char.h.

Referenced by str_check().

6.3.2.13 `#define char_ispunct(ch)`

Value:

```
(char_isprint(ch) && \
    !char_isalnum(ch) && \
    !char_isspace(ch))
```

check for a punctuation character

Definition at line 106 of file char.h.

Referenced by str_check().

6.3.2.14 `#define char_tolower(ch) do { if (char_isupper(ch)) ch += 32; } while(0)`

convert character to lower-case

Definition at line 112 of file char.h.

Referenced by str_tolower().

6.3.2.15 `#define char_toupper(ch) do { if (char_islower(ch)) ch -= 32; } while(0)`

convert character to upper-case

Definition at line 115 of file char.h.

Referenced by `str_toupper()`.

6.4 Secure chroot wrappers

6.4.1 Detailed Description

The chroot system call changes the root directory of the current process. This directory will be used for pathnames beginning with /. The root directory is inherited by all children of the current process.

The chroot family of functions provide wrappers for other library functions to happen in a chroot while the caller still remains in the old root after the functions have returned.

One can break out of the chroot in many ways due to the nature of the chroot system call:

- This call changes an ingredient in the pathname resolution process and does nothing else.
- This call does not change the current working directory.
- This call does not close open file descriptors.

The main usage of these functions is to get a file descriptor, safe against symlink attacks, referring to a directory inside a new root.

Functions

- int **chroot_fd** (int fd)
chroot(2) to the directory pointed to by a filedescriptor
- int **chroot_mkdirp** (const char *root, const char *dir, mode_t mode)
recursive mkdir(2) inside a secure chroot
- int **chroot_secure_chdir** (const char *root, const char *dir)
symlink-attack safe chdir(2) in chroot(2)

6.4.2 Function Documentation

6.4.2.1 int chroot_fd (int fd)

chroot(2) to the directory pointed to by a filedescriptor

Parameters:

← *fd* file descriptor referring to a directory (fchdir(2))

Returns:

0 on success, -1 on error with errno set

See also:

Secure chroot wrappers (p. 26)
fchdir(2)

Definition at line 21 of file `chroot_fd.c`.

Referenced by `chroot_mkdirp()`, and `chroot_secure_chdir()`.

```

22 {
23     if (fchdir(fd) == -1)
24         return -1;
25
26     return chroot(".");
27 }

```

6.4.2.2 `int chroot_mkdirp (const char * root, const char * dir, mode_t mode)`

recursive `mkdir(2)` inside a secure chroot

Parameters:

- ← *root* new root path
- ← *dir* dir to be created in root
- ← *mode* file permissions

Returns:

0 on success, -1 on error with `errno` set

See also:

`chroot_secure_chdir` (p. 28)
`mkdir(2)`

Definition at line 27 of file `chroot_mkdirp.c`.

References `chroot_fd()`, `mkdirp()`, and `open_read()`.

```

28 {
29     int orig_root, new_root;
30     int errno_orig;
31
32     if ((orig_root = open_read("/")) == -1)
33         return -1;
34
35     if (chdir(root) == -1)
36         return -1;
37
38     if ((new_root = open_read(".")) == -1)
39         return -1;
40
41     /* check cwdfd */
42     if (chroot_fd(new_root) == -1)
43         return -1;
44
45     /* now create the dir in the chroot */
46     if (mkdirp(dir, mode) == -1)
47         goto err;
48
49     /* break out of the chroot */
50     chroot_fd(orig_root);
51
52     return 0;
53 }

```

```

54 err:
55     errno_orig = errno;
56     chroot_fd(orig_root);
57     errno = errno_orig;
58     return -1;
59 }

```

6.4.2.3 int chroot_secure_chdir (const char * root, const char * dir)

symlink-attack safe chdir(2) in chroot(2)

Parameters:

- ← *root* new root path
- ← *dir* dir to chdir(2) in root

Returns:

0 on success, -1 on error with errno set

See also:

Secure chroot wrappers (p. 26)
chdir(2)

Definition at line 26 of file chroot_secure_chdir.c.

References chroot_fd(), and open_read().

```

27 {
28     int orig_root, new_root;
29
30     if ((orig_root = open_read("/")) == -1)
31         return -1;
32
33     if (chdir(root) == -1)
34         return -1;
35
36     if ((new_root = open_read(".")) == -1)
37         return -1;
38
39     int dirfd;
40     int errno_orig;
41
42     /* check cwdfd */
43     if (chroot_fd(new_root) == -1)
44         return -1;
45
46     /* now go to dir in the chroot */
47     if (chdir(dir) == -1)
48         goto err;
49
50     /* save a file descriptor of the target dir */
51     dirfd = open_read(".");
52
53     if (dirfd == -1)
54         goto err;
55
56     /* break out of the chroot */
57     chroot_fd(orig_root);
58 }

```

```
59      /* now go to the saved target dir (but outside the chroot) */
60      if (fchdir(dirfd) == -1)
61          goto err2;
62
63      close(dirfd);
64      return 0;
65
66 err2:
67      errno_orig = errno;
68      close(dirfd);
69      errno = errno_orig;
70 err:
71      errno_orig = errno;
72      chroot_fd(orig_root);
73      errno = errno_orig;
74      return -1;
75 }
```

6.5 Command execution wrappers

6.5.1 Detailed Description

The exec family of functions provide convenient wrappers around fork(2), execve(2), waitpid(2) and pipe(2).

These functions combine one or more of the above system calls in one function, thus allowing fast and simple process creation in applications.

Defines

- `#define EXEC_MAX_ARGV 64`
maximum number of arguments that will be converted for execvp(2)

Functions

- `int exec_fork (const char *fmt,...)`
fork, execvp and wait
- `int exec_fork_background (const char *fmt,...)`
fork, execvp and ignore child
- `int exec_fork_pipe (char **out, const char *fmt,...)`
pipe, fork, execvp and wait
- `int exec_replace (const char *fmt,...)`
plain execvp

6.5.2 Define Documentation

6.5.2.1 `#define EXEC_MAX_ARGV 64`

maximum number of arguments that will be converted for execvp(2)

Definition at line 35 of file exec.h.

6.5.3 Function Documentation

6.5.3.1 `int exec_fork (const char * fmt, ...)`

fork, execvp and wait

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

status obtained by wait(2) or -1 with errno set

See also:

Formatted output conversion (p. 83)

execvp(2)

Definition at line 27 of file exec_fork.c.

References `_lucid_vasprintf()`, `mem_alloc()`, `mem_free()`, `strtok_count()`, `strtok_free()`, `strtok_init_str()`, and `strtok_toargv()`.

```

28 {
29     va_list ap;
30     va_start(ap, fmt);
31
32     char *cmd;
33
34     if (_lucid_vasprintf(&cmd, fmt, ap) == -1) {
35         va_end(ap);
36         return -1;
37     }
38
39     va_end(ap);
40
41     strtok_t _st, *st = &_st;
42
43     if (!strtok_init_str(st, cmd, " ", 0)) {
44         mem_free(cmd);
45         return -1;
46     }
47
48     mem_free(cmd);
49
50     int argc    = strtok_count(st);
51     char **argv = mem_alloc((argc + 1) * sizeof(char *));
52
53     if (!argv) {
54         strtok_free(st);
55         return -1;
56     }
57
58     if (strtok_toargv(st, argv) < 1) {
59         mem_free(argv);
60         strtok_free(st);
61         return -1;
62     }
63
64     pid_t pid;
65     int status;
66
67     switch ((pid = fork())) {
68     case -1:
69         return -1;
70
71     case 0:
72         usleep(200);
73         execvp(argv[0], argv);
74
75         /* never get here */
76         exit(1);
77
78     default:
79         mem_free(argv);

```

```

80         strtok_free(st);
81
82         if (waitpid(pid, &status, 0) == -1)
83             return -1;
84     }
85
86     return status;
87 }

```

6.5.3.2 int exec_fork_background (const char * *fmt*, ...)

fork, execvp and ignore child

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

0 on success or -1 with errno set

See also:

Formatted output conversion (p. 83)
 execvp(2)

Note:

this function closes file descriptors 0-100 before execvp

Definition at line 26 of file exec_fork_background.c.

References `_lucid_vasprintf()`, `mem_alloc()`, `mem_free()`, `strtok_count()`, `strtok_free()`, `strtok_init_str()`, and `strtok_toargv()`.

```

27 {
28     va_list ap;
29     va_start(ap, fmt);
30
31     char *cmd;
32
33     if (_lucid_vasprintf(&cmd, fmt, ap) == -1) {
34         va_end(ap);
35         return -1;
36     }
37
38     va_end(ap);
39
40     strtok_t _st, *st = &_st;
41
42     if (!strtok_init_str(st, cmd, " ", 0)) {
43         mem_free(cmd);
44         return -1;
45     }
46
47     mem_free(cmd);
48
49     int argc    = strtok_count(st);
50     char **argv = mem_alloc((argc + 1) * sizeof(char *));

```



```
51
52     if (!argv) {
53         strtok_free(st);
54         return -1;
55     }
56
57     if (strtok_toargv(st, argv) < 1) {
58         mem_free(argv);
59         strtok_free(st);
60         return -1;
61     }
62
63     pid_t pid;
64     int i;
65
66     switch ((pid = fork())) {
67     case -1:
68         return -1;
69
70     case 0:
71         usleep(200);
72
73         for (i = 0; i < 100; i++)
74             close(i);
75
76         execvp(argv[0], argv);
77
78     default:
79         mem_free(argv);
80         strtok_free(st);
81         signal(SIGCHLD, SIG_IGN);
82     }
83
84     return 0;
85 }
```

6.5.3.3 int exec_fork_pipe (char ** out, const char * fmt, ...)

pipe, fork, execvp and wait

Parameters:

- **out** empty pointer to store combined stdout/stderr
- ← **fmt** format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

status obtained by wait(2) or -1 with errno set

Note:

The caller should free obtained memory for out using free(3)

See also:

Formatted output conversion (p. 83)

malloc(3)

free(3)

execvp(2)

Definition at line 28 of file `exec_fork_pipe.c`.

References `_lucid_vasprintf()`, `mem_alloc()`, `mem_free()`, `str_readfile()`, `strtok_count()`, `strtok_free()`, `strtok_init_str()`, and `strtok_toargv()`.

```
29 {
30     va_list ap;
31     va_start(ap, fmt);
32
33     char *cmd;
34
35     if (_lucid_vasprintf(&cmd, fmt, ap) == -1) {
36         va_end(ap);
37         return -1;
38     }
39
40     va_end(ap);
41
42     strtok_t _st, *st = &_st;
43
44     if (!strtok_init_str(st, cmd, " ", 0)) {
45         mem_free(cmd);
46         return -1;
47     }
48
49     mem_free(cmd);
50
51     int argc = strtok_count(st);
52     char **argv = mem_alloc((argc + 1) * sizeof(char *));
53
54     if (!argv) {
55         strtok_free(st);
56         return -1;
57     }
58
59     if (strtok_toargv(st, argv) < 1) {
60         mem_free(argv);
61         strtok_free(st);
62         return -1;
63     }
64
65     int outfds[2];
66
67     if (pipe(outfds) == -1) {
68         mem_free(argv);
69         strtok_free(st);
70         return -1;
71     }
72
73     pid_t pid;
74     int status;
75
76     switch ((pid = fork())) {
77     case -1:
78         mem_free(argv);
79         strtok_free(st);
80         close(outfds[0]);
81         close(outfds[1]);
82         return -1;
83
84     case 0:
85         usleep(200);
86
87         close(outfds[0]);
88
89         dup2(outfds[1], STDOUT_FILENO);
90         dup2(outfds[1], STDERR_FILENO);
```

```

91
92         execvp(argv[0], argv);
93
94         mem_free(argv);
95         strtok_free(st);
96
97         /* never get here */
98         exit(1);
99
100     default:
101         mem_free(argv);
102         strtok_free(st);
103
104         close(outfds[1]);
105
106         if (out && str_readfile(outfds[0], out) == -1)
107             return -1;
108
109         close(outfds[0]);
110
111         if (waitpid(pid, &status, 0) == -1)
112             return -1;
113     }
114
115     return status;
116 }
```

6.5.3.4 int exec_replace (const char * *fmt*, ...)

plain execvp

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

only returns on error with errno set

See also:

Formatted output conversion (p. 83)

execvp(2)

Definition at line 25 of file exec_replace.c.

References `_lucid_vasprintf()`, `mem_alloc()`, `mem_free()`, `strtok_count()`, `strtok_free()`, `strtok_init_str()`, and `strtok_toargv()`.

```

26 {
27     va_list ap;
28     va_start(ap, fmt);
29
30     char *cmd;
31
32     if (_lucid_vasprintf(&cmd, fmt, ap) == -1) {
33         va_end(ap);
34         return -1;
35     }
36 }
```

```
37     va_end(ap);
38
39     strtok_t _st, *st = &_st;
40
41     if (!strtok_init_str(st, cmd, " ", 0)) {
42         mem_free(cmd);
43         return -1;
44     }
45
46     mem_free(cmd);
47
48     int argc    = strtok_count(st);
49     char **argv = mem_alloc((argc + 1) * sizeof(char *));
50
51     if (!argv) {
52         strtok_free(st);
53         return -1;
54     }
55
56     if (strtok_toargv(st, argv) < 1) {
57         mem_free(argv);
58         strtok_free(st);
59         return -1;
60     }
61
62     execvp(argv[0], argv);
63
64     /* never get here */
65     mem_free(argv);
66     strtok_free(st);
67     return -1;
68 }
```

6.6 Flag list conversion

6.6.1 Detailed Description

The flag list family of functions manages a list of possible values of a bitmap using strings as key into the list.

A bitmap is simply an integer with certain bits being 1 (enabled) and 0 (disabled).

The `FLIST32_START` and `FLIST64_START` macros provides a shortcut for list declaration and initialization; followed by one or more of `FLIST32_NODE`, `FLIST32_NODE1`, `FLIST64_NODE` and `FLIST64_NODE1` to insert nodes into the list. The `FLIST32_NODE1` and `FLIST64_NODE1` macros are the same as `FLIST32_NODE` and `FLIST64_NODE`, respectively, except that they convert the bit index to a bitmap value before storing it in the list. The list should then be terminated using `FLIST32_END` or `FLIST64_END`, respectively.

The `flist32_getval()` (p. 39), `flist64_getval()` (p. 42), `flist32_getkey()` (p. 40) and `flist64_getkey()` (p. 43) functions provide lookup routines by key and value, respectively.

The `flist32_from_str()` (p. 41) and `flist64_from_str()` (p. 43) functions convert a string consisting of zero or more flag list keys separated by a delimiter and optionally prefixed with a clear modifier to a bitmap/bitmask pair according to a given list.

The `flist32_to_str()` (p. 42) and `flist64_to_str()` (p. 44) functions convert a bitmap according to a given list to a string consisting of zero or more flag list keys separated by a delimiter.

Data Structures

- struct `flist32_t`
32 bit list object
- struct `flist64_t`
64 bit list object

Defines

- `#define FLIST32_START(LIST) const flist32_t LIST[] = {`
32 bit list initialization
- `#define FLIST32_NODE(PREFIX, NAME) { #NAME, PREFIX ## _ ## NAME`
`},`
32 bit list node
- `#define FLIST32_NODE1(PREFIX, NAME) { #NAME, (1 << PREFIX ## _ ##`
`NAME) },`
32 bit list node from index
- `#define FLIST32_END { 0, 0 } };`
32 bit list termination
- `#define FLIST64_START(LIST) const flist64_t LIST[] = {`
64 bit list initialization

- `#define FLIST64__NODE(PREFIX, NAME) { #NAME, PREFIX ## _ ## NAME }`,
64 bit list node
- `#define FLIST64__NODE1(PREFIX, NAME) { #NAME, (1 << PREFIX ## _ ## NAME) }`,
64 bit list node from index
- `#define FLIST64__END { 0, 0 };`
64 bit list termination

Functions

- `uint32_t flist32_getval (const flist32_t list[], const char *key)`
get 32 bit value by key
- `const char * flist32_getkey (const flist32_t list[], uint32_t val)`
get key from 32 bit value
- `int flist32_from_str (const char *str, const flist32_t list[], uint32_t *flags, uint32_t *mask, char clmod, char *delim)`
parse flag list string
- `char * flist32_to_str (const flist32_t list[], uint32_t val, char *delim)`
convert bit mask to flag list string
- `uint64_t flist64_getval (const flist64_t list[], const char *key)`
get 64 bit value by key
- `const char * flist64_getkey (const flist64_t list[], uint64_t val)`
get key from 64 bit value
- `int flist64_from_str (const char *str, const flist64_t list[], uint64_t *flags, uint64_t *mask, char clmod, char *delim)`
parse flag list string
- `char * flist64_to_str (const flist64_t list[], uint64_t val, char *delim)`
convert bit mask to flag list string

6.6.2 Define Documentation

6.6.2.1 `#define FLIST32__START(LIST) const flist32_t LIST[] = {`

32 bit list initialization

Definition at line 61 of file flist.h.

6.6.2.2 `#define FLIST32__NODE(PREFIX, NAME) { #NAME, PREFIX ## _
NAME },`

32 bit list node

Definition at line 64 of file flist.h.

6.6.2.3 `#define FLIST32__NODE1(PREFIX, NAME) { #NAME, (1 << PREFIX
_ ## NAME) },`

32 bit list node from index

Definition at line 67 of file flist.h.

6.6.2.4 `#define FLIST32__END { 0, 0 } ;`

32 bit list termination

Definition at line 70 of file flist.h.

6.6.2.5 `#define FLIST64__START(LIST) const flist64__t LIST[] = {`

64 bit list initialization

Definition at line 135 of file flist.h.

6.6.2.6 `#define FLIST64__NODE(PREFIX, NAME) { #NAME, PREFIX ## _
NAME },`

64 bit list node

Definition at line 138 of file flist.h.

6.6.2.7 `#define FLIST64__NODE1(PREFIX, NAME) { #NAME, (1 << PREFIX
_ ## NAME) },`

64 bit list node from index

Definition at line 141 of file flist.h.

6.6.2.8 `#define FLIST64__END { 0, 0 } ;`

64 bit list termination

Definition at line 144 of file flist.h.

6.6.3 Function Documentation

6.6.3.1 `uint32__t flist32__getval (const flist32__t list[], const char * key)`

get 32 bit value by key

Parameters:

- ← *list* list to use for conversion
- ← *key* key to look for

Returns:

32 bit value ≥ 1 if key found, 0 otherwise

Definition at line 20 of file flist32_getval.c.

References flist32_t::key, and str_equal().

Referenced by flist32_from_str().

```

21 {
22     int i;
23
24     for (i = 0; list[i].key; i++)
25         if (str_equal(list[i].key, key))
26             return list[i].val;
27
28     return 0;
29 }
```

6.6.3.2 const char* flist32_getkey (const flist32_t list[], uint32_t val)

get key from 32 bit value

Parameters:

- ← *list* list to use for conversion
- ← *val* 32 bit key to look for

Returns:

key if value was found, NULL otherwise

Note:

this functions does not reset the flags or mask argument to an empty bitmap, thus allowing incremental changes to the map.

Definition at line 19 of file flist32_getkey.c.

References flist32_t::key.

```

20 {
21     int i;
22
23     for (i = 0; list[i].key; i++)
24         if (list[i].val == val)
25             return list[i].key;
26
27     return 0;
28 }
```


6.6.3.3 `int flist32_from_str (const char * str, const flist32_t list[], uint32_t * flags, uint32_t * mask, char clmod, char * delim)`

parse flag list string

Parameters:

- ← *str* string to convert
- ← *list* list to use for conversion
- *flags* pointer to a bit mask
- *mask* pointer to a set mask
- ← *clmod* clear flag modifier
- ← *delim* flag delimiter

Returns:

0 on success, -1 on error with `errno` set

Definition at line 20 of file `flist32_from_str.c`.

References `flist32_getval()`, `strtok_for_each`, `strtok_free()`, and `strtok_init_str()`.

```

23 {
24     char *token;
25     int clear = 0;
26     uint32_t cur_flag;
27
28     strtok_t _st, *st = &_st, *p;
29
30     if (!strtok_init_str(st, str, delim, 0))
31         return -1;
32
33     strtok_for_each(st, p) {
34         token = p->token;
35
36         if (*token == clmod)
37             clear = 1;
38
39         cur_flag = flist32_getval(list, token+clear);
40
41         if (!cur_flag) {
42             strtok_free(st);
43             return -1;
44         }
45
46         if (clear) {
47             *flags &= ~cur_flag;
48             *mask |= cur_flag;
49         } else {
50             *flags |= cur_flag;
51             *mask |= cur_flag;
52         }
53     }
54
55     strtok_free(st);
56
57     return 0;
58 }
```

6.6.3.4 char* flist32_to_str (const flist32_t list[], uint32_t val, char * delim)

convert bit mask to flag list string

Parameters:

- ← *list* list to use for conversion
- ← *val* bit mask
- ← *delim* flag delimiter

Returns:

flags list string

Note:

this function ignores set bits if they do not appear in the list
if no flag was found or the bitmap was empty, an empty string is returned, not NULL

Definition at line 21 of file flist32_to_str.c.

References flist32_t::key, str_len(), stralloc_catf(), stralloc_finalize(), stralloc_free(), and stralloc_init().

```

22 {
23     int i;
24     char *buf;
25     stralloc_t _sa, *sa = &_amp;sa;
26
27     stralloc_init(sa);
28
29     for (i = 0; list[i].key; i++)
30         if (val & list[i].val)
31             stralloc_catf(sa, "%s%s", list[i].key, delim);
32
33     if (sa->len > 0)
34         sa->len -= str_len(delim);
35
36     buf = stralloc_finalize(sa);
37
38     stralloc_free(sa);
39     return buf;
40 }
```

6.6.3.5 uint64_t flist64_getval (const flist64_t list[], const char * key)

get 64 bit value by key

Parameters:

- ← *list* list to use for conversion
- ← *key* key to look for

Returns:

64 bit value ≥ 1 if key was found, 0 otherwise

Definition at line 20 of file flist64_getval.c.

References flist64_t::key, and str_equal().

Referenced by flist64_from_str().

```

21 {
22     int i;
23
24     for (i = 0; list[i].key; i++)
25         if (str_equal(list[i].key, key))
26             return list[i].val;
27
28     return 0;
29 }

```

6.6.3.6 const char* flist64_getkey (const flist64_t list[], uint64_t val)

get key from 64 bit value

Parameters:

- ← *list* list to use for conversion
- ← *val* 64 bit key to look for

Returns:

key if value was found, NULL otherwise

Definition at line 19 of file flist64_getkey.c.

References flist64_t::key.

```

20 {
21     int i;
22
23     for (i = 0; list[i].key; i++)
24         if (list[i].val == val)
25             return list[i].key;
26
27     return 0;
28 }

```

6.6.3.7 int flist64_from_str (const char * str, const flist64_t list[], uint64_t * flags, uint64_t * mask, char clmod, char * delim)

parse flag list string

Parameters:

- ← *str* string to convert
- ← *list* list to use for conversion
- *flags* pointer to a bit mask
- *mask* pointer to a set mask
- ← *clmod* clear flag modifier

← *delim* flag delimiter

Returns:

0 on success, -1 on error with errno set

Note:

this functions does not reset the flags or mask argument to an empty bitmap, thus allowing incremental changes to the map.

Definition at line 20 of file flist64_from_str.c.

References flist64_getval(), strtok_for_each, strtok_free(), and strtok_init_str().

```

23 {
24     char *token;
25     int clear = 0;
26     uint64_t cur_flag;
27
28     strtok_t _st, *st = &_st, *p;
29
30     if (!strtok_init_str(st, str, delim, 0))
31         return -1;
32
33     strtok_for_each(st, p) {
34         token = p->token;
35
36         if (*token == clmod)
37             clear = 1;
38
39         cur_flag = flist64_getval(list, token+clear);
40
41         if (!cur_flag) {
42             strtok_free(st);
43             return -1;
44         }
45
46         if (clear) {
47             *flags &= ~cur_flag;
48             *mask |= cur_flag;
49         } else {
50             *flags |= cur_flag;
51             *mask |= cur_flag;
52         }
53     }
54
55     strtok_free(st);
56
57     return 0;
58 }
```

6.6.3.8 char* flist64_to_str (const flist64_t list[], uint64_t val, char * delim)

convert bit mask to flag list string

Parameters:

- ← *list* list to use for conversion
- ← *val* bit mask
- ← *delim* flag delimiter

Returns:

flags list string

Note:

this function ignores set bits if they do not appear in the list
if no flag was found or the bitmap was empty, an empty string is returned, not NULL

Definition at line 21 of file flist64_to_str.c.

References flist64_t::key, str_len(), stralloc_catf(), stralloc_finalize(), stralloc_free(), and stralloc_init().

```
22 {
23     int i;
24     char *buf;
25     stralloc_t _sa, *sa = &_amp;_sa;
26
27     stralloc_init(sa);
28
29     for (i = 0; list[i].key; i++)
30         if (val & list[i].val)
31             stralloc_catf(sa, "%s%s", list[i].key, delim);
32
33     if (sa->len > 0)
34         sa->len -= str_len(delim);
35
36     buf = stralloc_finalize(sa);
37
38     stralloc_free(sa);
39     return buf;
40 }
```

6.7 Simple doubly linked lists

6.7.1 Detailed Description

The simplest kind of linked list is a singly-linked list, which has one link per node. This link points to the next node in the list, or to a null value or empty list if it is the final node; e.g. 12 -> 99 -> 37 -> NULL.

A more sophisticated kind of linked list is a doubly-linked list. Each node has two links: one points to the previous node, or points to a null value or empty list if it is the first node; and one points to the next, or points to a null value or empty list if it is the final node; e.g. NULL <- 26 <-> 56 <-> 46 -> NULL.

The list family of functions and macros provide routines to create a list, add, move or remove elements and iterate over the list.

Data Structures

- struct **list_head**
list head

Defines

- #define **container_of**(ptr, type, member) ((type *)((char *)(ptr) - offsetof(type, member)))
get container of list head
- #define **LIST_NODE_ALLOC**(NAME) NAME = mem_alloc(sizeof(*NAME))
- #define **list_entry**(ptr, type, member) container_of(ptr, type, member)
get the struct for this entry
- #define **list_for_each**(pos, head) for (pos = (head) → next; pos != (head); pos = pos → next)
iterate over a list
- #define **list_for_each_prev**(pos, head) for (pos = (head) → prev; pos != (head); pos = pos → prev)
iterate over a list backwards
- #define **list_for_each_safe**(pos, n, head)
iterate over a list safe against removal of list entry
- #define **list_for_each_entry**(pos, head, member)
iterate over list of given type
- #define **list_for_each_entry_reverse**(pos, head, member)
iterate backwards over list of given type.
- #define **list_for_each_entry_safe**(pos, n, head, member)
iterate over list of given type safe against removal of list entry

- `#define list_for_each_entry_safe_reverse(pos, n, head, member)`
iterate backwards over list of given type safe against removal of list entry

Typedefs

- `typedef list_head list_t`
list head

6.7.2 Define Documentation

- 6.7.2.1** `#define container_of(ptr, type, member) ((type *)((char *)(ptr) - offsetof(type, member)))`

get container of list head

Definition at line 48 of file list.h.

- 6.7.2.2** `#define LIST_NODE_ALLOC(NAME) NAME = mem_alloc(sizeof(*NAME))`

Definition at line 66 of file list.h.

- 6.7.2.3** `#define list_entry(ptr, type, member) container_of(ptr, type, member)`

get the struct for this entry

Parameters:

ptr the &list_t pointer
type the type of the struct this is embedded in
member the name of the list_struct within the struct

Definition at line 250 of file list.h.

Referenced by strtok_delete(), strtok_free(), strtok_next(), and strtok_prev().

- 6.7.2.4** `#define list_for_each(pos, head) for (pos = (head) → next; pos != (head); pos = pos → next)`

iterate over a list

Parameters:

pos the &list_t to use as a loop counter
head the head for your list

Definition at line 259 of file list.h.

Referenced by strtok_count().

6.7.2.5 `#define list_for_each_prev(pos, head) for (pos = (head) → prev; pos != (head); pos = pos → prev)`

iterate over a list backwards

Parameters:

pos the &list_t to use as a loop counter

head the head for your list

Definition at line 268 of file list.h.

6.7.2.6 `#define list_for_each_safe(pos, n, head)`

Value:

```
for (pos = (head)->next, n = pos->next; pos != (head); \
     pos = n, n = pos->next)
```

iterate over a list safe against removal of list entry

Parameters:

pos the &list_t to use as a loop counter

n another &list_t to use as temporary storage

head the head for your list

Definition at line 278 of file list.h.

Referenced by strtok_delete(), and strtok_free().

6.7.2.7 `#define list_for_each_entry(pos, head, member)`

Value:

```
for (pos = list_entry((head)->next, typeof(*pos), member); \
     &pos->member != (head); \
     pos = list_entry(pos->member.next, typeof(*pos), member))
```

iterate over list of given type

Parameters:

pos the type * to use as a loop counter

head,: the head for your list

member the name of the list_struct within the struct

Definition at line 289 of file list.h.

6.7.2.8 #define list_for_each_entry_reverse(pos, head, member)**Value:**

```
for (pos = list_entry((head)->prev, typeof(*pos), member); \
    &pos->member != (head); \
    pos = list_entry(pos->member.prev, typeof(*pos), member))
```

iterate backwards over list of given type.

Parameters:

pos the type * to use as a loop counter
head the head for your list
member the name of the list_struct within the struct

Definition at line 301 of file list.h.

6.7.2.9 #define list_for_each_entry_safe(pos, n, head, member)**Value:**

```
for (pos = list_entry((head)->next, typeof(*pos), member), \
    n = list_entry(pos->member.next, typeof(*pos), member); \
    &pos->member != (head); \
    pos = n, n = list_entry(n->member.next, typeof(*n), member))
```

iterate over list of given type safe against removal of list entry

Parameters:

pos the type * to use as a loop counter
n another type * to use as temporary storage
head the head for your list
member the name of the list_struct within the struct

Definition at line 314 of file list.h.

6.7.2.10 #define list_for_each_entry_safe_reverse(pos, n, head, member)**Value:**

```
for (pos = list_entry((head)->prev, typeof(*pos), member), \
    n = list_entry(pos->member.prev, typeof(*pos), member); \
    &pos->member != (head); \
    pos = n, n = list_entry(n->member.prev, typeof(*n), member))
```

iterate backwards over list of given type safe against removal of list entry

Parameters:

pos the type * to use as a loop counter
n another type * to use as temporary storage
head the head for your list
member the name of the list_struct within the struct

Definition at line 328 of file list.h.

6.7.3 Typedef Documentation

6.7.3.1 typedef struct list__head list__t

list head

6.8 Log system multiplexer

6.8.1 Detailed Description

The log system multiplexer allows the caller to send log messages to multiple destinations; currently: syslog(3), file, stderr.

An application can only open one connection to the multiplexer during runtime. Another call to **log_init()** (p. 55) will replace the previous connection.

log_init() (p. 55) opens a connection to the multiplexer for a program. The options argument is a pointer to a **log_options_t** (p. 164) structure used for the multiplexer configuration.

See also:

- log_options_t** (p. 164)
- syslog(3)

Data Structures

- struct **log_options_t**
multiplexer configuration data

Defines

- #define **LOGD_SYSLOG** 0x01
- #define **LOGD_FILE** 0x02
- #define **LOGD_STDERR** 0x04
- #define **LOGP_ALERT** 0
- #define **LOGP_ERROR** 1
- #define **LOGP_WARN** 2
- #define **LOGP_NOTE** 3
- #define **LOGP_INFO** 4
- #define **LOGP_DEBUG** 5
- #define **LOGP_TRACE** 6
- #define **LOGO_PID** 0x01
- #define **LOGO_TIME** 0x02
- #define **LOGO_PRIO** 0x04
- #define **LOGO_IDENT** 0x08
- #define **LOG_TRACEME** log_traceme(__FILE__, __FUNCTION__, __LINE__ -
__);
simple trace helper

Functions

- void **log_init** (**log_options_t** *options)
initialize log message multiplexer
- int **log_alert** (const char *fmt,...)

send ALERT level message to the multiplexer

- `int log_error (const char *fmt,...)`
send ERR level message to the multiplexer
- `int log_warn (const char *fmt,...)`
send WARNING level message to the multiplexer
- `int log_notice (const char *fmt,...)`
send NOTICE level message to the multiplexer
- `int log_info (const char *fmt,...)`
send INFO level message to the multiplexer
- `int log_debug (const char *fmt,...)`
send DEBUG level message to the multiplexer
- `int log_trace (const char *fmt,...)`
send TRACE level message to the multiplexer
- `int log_traceme (const char *file, const char *func, int line)`
send TRACE level message to the multiplexer
- `void log_alert_and_die (const char *fmt,...)`
send ALERT level message to the multiplexer and exit(2)
- `void log_error_and_die (const char *fmt,...)`
send ERR level message to the multiplexer and exit(2)
- `int log_palert (const char *fmt,...)`
send ALERT level message to the multiplexer and append strerror(errno)
- `int log_perror (const char *fmt,...)`
send ERR level message to the multiplexer and append strerror(errno)
- `int log_pwarn (const char *fmt,...)`
send WARNING level message to the multiplexer and append strerror(errno)
- `int log_pnotice (const char *fmt,...)`
send NOTICE level message to the multiplexer and append strerror(errno)
- `int log_pinfo (const char *fmt,...)`
send INFO level message to the multiplexer and append strerror(errno)
- `int log_pdebug (const char *fmt,...)`
send DEBUG level message to the multiplexer and append strerror(errno)
- `int log_ptrace (const char *fmt,...)`
send TRACE level message to the multiplexer and append strerror(errno)

- void **log_alert_and_die** (const char *fmt,...)
send ALERT level message to the multiplexer, append strerror(errno) and exit(2)
- void **log_perror_and_die** (const char *fmt,...)
send ERR level message to the multiplexer, append strerror(errno) and exit(2)
- void **log_close** (void)
close connection to logging system

6.8.2 Define Documentation

6.8.2.1 #define LOGD_SYSLOG 0x01

Log to syslog

Definition at line 42 of file log.h.

Referenced by log_close(), and log_init().

6.8.2.2 #define LOGD_FILE 0x02

Log to a file

Definition at line 43 of file log.h.

Referenced by log_close(), and log_init().

6.8.2.3 #define LOGD_STDERR 0x04

Log to STDERR

Definition at line 44 of file log.h.

Referenced by log_init().

6.8.2.4 #define LOGP_ALERT 0

action must be taken immediately

Definition at line 47 of file log.h.

6.8.2.5 #define LOGP_ERROR 1

error conditions

Definition at line 48 of file log.h.

6.8.2.6 #define LOGP_WARN 2

warning conditions

Definition at line 49 of file log.h.

6.8.2.7 #define LOGP_NOTE 3

normal but significant condition

Definition at line 50 of file log.h.

6.8.2.8 #define LOGP_INFO 4

informational

Definition at line 51 of file log.h.

Referenced by log_init().

6.8.2.9 #define LOGP_DEBUG 5

debug-level messages

Definition at line 52 of file log.h.

6.8.2.10 #define LOGP_TRACE 6

trace messages

Definition at line 53 of file log.h.

6.8.2.11 #define LOGO_PID 0x01

log the pid with each message

Definition at line 56 of file log.h.

Referenced by log_init().

6.8.2.12 #define LOGO_TIME 0x02

log the time with each message

Definition at line 57 of file log.h.

6.8.2.13 #define LOGO_PRIO 0x04

log the priority with each message

Definition at line 58 of file log.h.

6.8.2.14 #define LOGO_IDENT 0x08

log the ident string with each message

Definition at line 59 of file log.h.

6.8.2.15 `#define LOG_TRACEME log_traceme(__FILE__,
__FUNCTION__, __LINE__);`

simple trace helper

Definition at line 62 of file log.h.

6.8.3 Function Documentation

6.8.3.1 void log_init (log_options_t * options)

initialize log message multiplexer

Parameters:

← *options* multiplexer configuration

See also:

`log_options_t` (p. 164)

Definition at line 50 of file log_init.c.

References `_log_options`, `log_options_t::log_dest`, `log_options_t::log_facility`, `log_options_t::log_fd`, `log_options_t::log_ident`, `log_options_t::log_mask`, `log_options_t::log_opts`, `LOGD_FILE`, `LOGD_STDERR`, `LOGD_SYSLOG`, `LOGO_PID`, `LOGP_INFO`, `mem_alloc()`, `mem_cpy()`, and `str_isempty`.

```

51 {
52     struct stat sb;
53
54     /* check file destination */
55     if (options->log_dest & LOGD_FILE)
56         if (options->log_fd < 0 || fstat(options->log_fd, &sb) == -1)
57             options->log_dest &= ~LOGD_FILE;
58
59     /* check if STDERR is available */
60     if (options->log_dest & LOGD_STDERR)
61         if (fstat(STDERR_FILENO, &sb) == -1)
62             options->log_dest &= ~LOGD_STDERR;
63
64     /* log up to LOGP_INFO if not specified */
65     if (options->log_mask == 0)
66         options->log_mask = ((1 << ((LOGP_INFO) + 1)) - 1);
67
68     /* sanitize ident string */
69     if (str_isempty(options->log_ident))
70         options->log_ident = "(none)";
71
72     if (options->log_dest & LOGD_SYSLOG) {
73         openlog(options->log_ident,
74                options->log_opts & LOGO_PID ? LOG_PID : 0,
75                options->log_facility);
76         setlogmask(mask_to_syslog(options->log_mask));
77     }
78
79     _log_options = (log_options_t *) mem_alloc(sizeof(log_options_t));
80
81     mem_cpy(_log_options, options, sizeof(log_options_t));
82 }

```

6.8.3.2 int log_alert (const char * *fmt*, ...)

send ALERT level message to the multiplexer

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

EXIT_FAILURE

See also:

Formatted output conversion (p. 83)

6.8.3.3 int log_error (const char * *fmt*, ...)

send ERR level message to the multiplexer

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

EXIT_FAILURE

See also:

Formatted output conversion (p. 83)

6.8.3.4 int log_warn (const char * *fmt*, ...)

send WARNING level message to the multiplexer

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 83)

6.8.3.5 int log_notice (const char * *fmt*, ...)

send NOTICE level message to the multiplexer

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 83)

6.8.3.6 int log_info (const char * *fmt*, ...)

send INFO level message to the multiplexer

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 83)

6.8.3.7 int log_debug (const char * *fmt*, ...)

send DEBUG level message to the multiplexer

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 83)

6.8.3.8 int log_trace (const char * *fmt*, ...)

send TRACE level message to the multiplexer

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 83)

Referenced by log_traceme().

6.8.3.9 int log_traceme (const char * *file*, const char * *func*, int *line*)

send TRACE level message to the multiplexer

Parameters:

- ← *file* File name
- ← *func* Function name
- ← *line* Line number

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 83)

Definition at line 117 of file log_internal.c.

References log_trace(), mem_free(), and str_path_basename().

```
118 {  
119     char *base = str_path_basename(file);  
120  
121     int rc = log_trace("%s() in %s:%d", func, base, line);  
122  
123     mem_free(base);  
124  
125     return rc;  
126 }
```

6.8.3.10 void log_alert_and_die (const char * *fmt*, ...)

send ALERT level message to the multiplexer and exit(2)

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

See also:

Formatted output conversion (p. 83)
exit(2)

6.8.3.11 void log_error_and_die (const char * *fmt*, ...)

send ERR level message to the multiplexer and exit(2)

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

See also:

Formatted output conversion (p. 83)
exit(2)

6.8.3.12 int log_alert (const char * *fmt*, ...)

send ALERT level message to the multiplexer and append strerror(errno)

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

EXIT_FAILURE

See also:

Formatted output conversion (p. 83)

6.8.3.13 int log_perror (const char * *fmt*, ...)

send ERR level message to the multiplexer and append strerror(errno)

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

EXIT_FAILURE

See also:

Formatted output conversion (p. 83)

6.8.3.14 int log_pwarn (const char * *fmt*, ...)

send WARNING level message to the multiplexer and append strerror(errno)

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 83)

6.8.3.15 int log_pnotice (const char * *fmt*, ...)

send NOTICE level message to the multiplexer and append strerror(errno)

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 83)

6.8.3.16 int log_pinfo (const char * *fmt*, ...)

send INFO level message to the multiplexer and append strerror(errno)

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 83)

6.8.3.17 int log_pdebug (const char * *fmt*, ...)

send DEBUG level message to the multiplexer and append strerror(errno)

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 83)

6.8.3.18 int log_ptrace (const char * *fmt*, ...)

send TRACE level message to the multiplexer and append strerror(errno)

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

Returns:

EXIT_SUCCESS

See also:

Formatted output conversion (p. 83)

6.8.3.19 void log_palert_and_die (const char * *fmt*, ...)

send ALERT level message to the multiplexer, append strerror(errno) and exit(2)

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

See also:

Formatted output conversion (p. 83)
exit(2)

6.8.3.20 void log__perror__and__die (const char * *fmt*, ...)

send ERR level message to the multiplexer, append strerror(errno) and exit(2)

Parameters:

- ← *fmt* format string passed to printf
- ← ... variable number of arguments according to fmt

See also:

Formatted output conversion (p. 83)
exit(2)

6.8.3.21 void log__close (void)

close connection to logging system

Definition at line 25 of file log__close.c.

References `_log_options`, `log_options_t::log_dest`, `log_options_t::log_fd`, `LOGD_FILE`, `LOGD_SYSLOG`, and `mem_free()`.

```
26 {  
27     if (!_log_options)  
28         return;  
29  
30     if (_log_options->log_dest & LOGD_SYSLOG)  
31         closelog();  
32  
33     if (_log_options->log_dest & LOGD_FILE)  
34         close(_log_options->log_fd);  
35  
36     mem_free(_log_options);  
37  
38     _log_options = 0;  
39 }
```

6.9 Memory area manipulation

Functions

- void * **mem_alloc** (int n)
allocate memory
- void * **mem_ccpy** (void *s1, const void *s2, int c, int n)
copy memory block until character is found
- void * **mem_chr** (const void *s, int c, int n)
find character in memory block
- int **mem_cmp** (const void *s1, const void *s2, int n)
compare two memory regions
- void * **mem_cpy** (void *s1, const void *s2, int n)
copy memory block
- void * **mem_dup** (const void *s, int n)
duplicate a memory block
- void **mem_free** (void *s)
free memory
- void **mem_freeall** (void)
free all memory
- int **mem_idx** (const void *s, int c, int n)
find character in memory block
- void * **mem_realloc** (void *s, int n)
reallocate memory
- void * **mem_set** (void *s, int c, int n)
fill memory block with character

6.9.1 Function Documentation

6.9.1.1 void* mem_alloc (int *n*)

allocate memory

Parameters:

← *n* allocate *n* bytes

Returns:

A pointer to newly allocated memory, NULL otherwise.

Definition at line 24 of file `mem_alloc.c`.

References `_mem_pool`, `_mem_pool_t::list`, and `mem_set()`.

Referenced by `_lucid_vasprintf()`, `exec_fork()`, `exec_fork_background()`, `exec_fork_pipe()`, `exec_replace()`, `log_init()`, `mem_dup()`, `mem_realloc()`, `readsymlink()`, `str_read()`, `str_readfile()`, `str_readline()`, `stralloc_finalize()`, `strtok_append()`, `strtok_init_argv()`, and `strtok_init_str()`.

```

25 {
26     if (!_mem_pool) {
27         if ((_mem_pool = malloc(sizeof(_mem_pool_t))) == NULL)
28             return NULL;
29
30         INIT_LIST_HEAD(&(_mem_pool->list));
31     }
32
33     _mem_pool_t *new;
34
35     if ((new = malloc(sizeof(_mem_pool_t))) == NULL)
36         return NULL;
37
38     new->len = n;
39
40     if ((new->mem = malloc(new->len)) == NULL) {
41         free(new);
42         return NULL;
43     }
44
45     mem_set(new->mem, 0, new->len);
46
47     list_add_tail(&(new->list), &(_mem_pool->list));
48
49     return new->mem;
50 }
```

6.9.1.2 void* mem_ccpy (void * s1, const void * s2, int c, int n)

copy memory block until character is found

Parameters:

- *s1* pointer to destination block
- ← *s2* pointer to source block
- ← *n* copy first n bytes of s2

Returns:

A pointer to s1.

Definition at line 19 of file `mem_ccpy.c`.

```

20 {
21     unsigned char    *a = s1;
22     const unsigned char *b = s2;
23
24     while (n--) {
25         *a++ = *b;
26
27         if (*b == c)
```



```
28             return (void *) a;
29
30         b++;
31     }
32
33     return 0;
34 }
```

6.9.1.3 void* mem_chr (const void * s, int c, int n)

find character in memory block

Parameters:

- ← *s* pointer to memory block
- ← *c* character to look for
- ← *n* scan first n bytes

Returns:

A pointer to the first matching character found, NULL otherwise.

Definition at line 19 of file mem_chr.c.

```
20 {
21     const unsigned char *p = s;
22
23     for (; n--; p++)
24         if (*p == c)
25             return (void *) p;
26
27     return 0;
28 }
```

6.9.1.4 int mem_cmp (const void * s1, const void * s2, int n)

compare two memory regions

Parameters:

- ← *s1* pointer to first memory region
- ← *s2* pointer to second memory region
- ← *n* compare n bytes

Returns:

An integer less than, equal to, or greater than zero according to whether s1 is lexicographically less than, equal to, or greater than s2.

Definition at line 19 of file mem_cmp.c.

Referenced by str_path_concat(), and str_str().

```

20 {
21     int d, i;
22     const unsigned char *a = s1;
23     const unsigned char *b = s2;
24
25     for (i = 0; i < n; i++)
26         if ((d = a[i] - b[i]) != 0)
27             return d;
28
29     return 0;
30 }

```

6.9.1.5 void* mem_cpy (void * s1, const void * s2, int n)

copy memory block

Parameters:

- **s1** pointer to destination block
- ← **s2** pointer to source block
- ← **n** copy first n bytes of s2

Returns:

A pointer to s1.

Definition at line 19 of file mem_cpy.c.

Referenced by log_init(), mem_dup(), str_cpy(), str_cpyn(), stralloc_catb(), stralloc_copyb(), stralloc_finalize(), and whirlpool_finalize().

```

20 {
21     unsigned char      *a = s1;
22     const unsigned char *b = s2;
23
24     while (n--)
25         *a++ = *b++;
26
27     return s1;
28 }

```

6.9.1.6 void* mem_dup (const void * s, int n)

duplicate a memory block

Parameters:

- ← **s** pointer to source memory area
- ← **n** duplicate first n bytes

Returns:

A pointer to the duplicated memory block, or NULL if insufficient memory was available.

Definition at line 19 of file mem_dup.c.

References mem_alloc(), and mem_cpy().

Referenced by str_dup().

```
20 {
21     void *d = mem_alloc(n);
22
23     if (d)
24         return mem_cpy(d, s, n);
25
26     return 0;
27 }
```

6.9.1.7 void mem_free (void * s)

free memory

Parameters:

← *s* memory area to free

Definition at line 23 of file mem_free.c.

References `_mem_pool`, `_mem_pool_t::list`, `_mem_pool_t::mem`, and `mem_for_each`.

Referenced by `_lucid_vdprintf()`, `exec_fork()`, `exec_fork_background()`, `exec_fork_pipe()`, `exec_replace()`, `ismount()`, `log_close()`, `log_traceme()`, `mem_realloc()`, `mkdirnamep()`, `readsym-link()`, `runlink()`, `str_path_basename()`, `str_path_dirname()`, `str_read()`, `str_readfile()`, `str_readline()`, `stralloc_catf()`, `stralloc_free()`, `strtok_append()`, `strtok_delete()`, `strtok_free()`, and `strtok_init_str()`.

```
24 {
25     int errno_orig = errno;
26     _mem_pool_t *p;
27
28     mem_for_each(_mem_pool, p)
29         if (p->mem == s)
30             break;
31
32     if (p->mem != s)
33         return;
34
35     list_del(&(p->list));
36
37     free(p->mem);
38     free(p);
39
40     errno = errno_orig;
41 }
```

6.9.1.8 void mem_freeall (void)

free all memory

Definition at line 22 of file mem_freeall.c.

References `_mem_pool`, `_mem_pool_t::list`, `_mem_pool_t::mem`, and `mem_for_each_safe`.

```
23 {
24     _mem_pool_t *p, *tmp;
25
26     if (!_mem_pool)
27         return;
```

```

28
29     mem_for_each_safe(_mem_pool, p, tmp) {
30         list_del(&(p->list));
31         free(p->mem);
32         free(p);
33     }
34 }

```

6.9.1.9 int mem_idx (const void * s, int c, int n)

find character in memory block

Parameters:

- *s* pointer to memory block
- ← *c* character to look for
- ← *n* scan first n bytes

Returns:

An integer offset to the character in the memory block starting at s.

Note:

This function will scan any number of bytes until a NULL character is found if n is less than or equal to zero.

Definition at line 19 of file mem_idx.c.

```

20 {
21     int i;
22     const unsigned char *p = s;
23
24     for (i = 0; n--, *p++; i++)
25         if (c == *p)
26             return i;
27
28     return -1;
29 }

```

6.9.1.10 void* mem_realloc (void * s, int n)

reallocate memory

Parameters:

- ← *s* memory area to reallocate
- ← *n* allocate n bytes

Returns:

A pointer to newly allocated memory, NULL otherwise.

Definition at line 23 of file `mem_realloc.c`.

References `_mem_pool`, `_mem_pool_t::len`, `_mem_pool_t::mem`, `mem_alloc()`, `mem_for_each`, `mem_free()`, and `mem_set()`.

Referenced by `readsymlink()`, `str_readfile()`, `str_readline()`, and `stralloc_ready()`.

```

24 {
25     if (!s) {
26         if (n > 0)
27             return mem_alloc(n);
28         else
29             return NULL;
30     }
31
32     else if (n < 1) {
33         mem_free(s);
34         return NULL;
35     }
36
37     _mem_pool_t *p;
38
39     mem_for_each(_mem_pool, p)
40         if (p->mem == s)
41             break;
42
43     if (p->mem != s) {
44         errno = EINVAL;
45         return NULL;
46     }
47
48     char *m = realloc(p->mem, n);
49
50     if (!m)
51         return NULL;
52
53     p->mem = m;
54
55     if (n > p->len)
56         mem_set(m + p->len, 0, n - p->len);
57
58     p->len = n;
59
60     return p->mem;
61 }

```

6.9.1.11 void* mem_set (void * s, int c, int n)

fill memory block with character

Parameters:

- *s* pointer to memory block
- ← *c* character value to be set
- ← *n* fill first *n* bytes

Returns:

A pointer to *s*.

Definition at line 19 of file `mem_set.c`.

Referenced by `_lucid_vsnprintf()`, `mem_alloc()`, `mem_realloc()`, `strtok_init_str()`, `tcp_connect()`, `tcp_listen()`, `whirlpool_finalize()`, and `whirlpool_init()`.

```
20 {  
21     unsigned char *p = s;  
22  
23     while (n--)  
24         *p++ = c;  
25  
26     return s;  
27 }
```

6.10 Miscellaneous helpers

6.10.1 Detailed Description

The misc family of functions provide wrappers not fitting in any other module and not being worth an own category for each of them.

The **isdir()** (p. 72), **isfile()** (p. 72) and **islink()** (p. 73) functions wrap the `stat(2)` system call and checks if the path in the string pointed to by `path` is a directory, regular file or link, respectively.

The **mkdirp()** (p. 74) function creates any missing parent directories of the path in the string pointed to by `path`, before creating the directory itself. The **mkdirnamep()** (p. 74) function additionally calls `dirname(3)` on the path string before calling **mkdirp()** (p. 74).

The `path_concat()` function concatenates the strings pointed to by `dirname` and `basename` and checks the latter using **str_path_isdot()** (p. 120).

The **runlink()** (p. 76) function removes all files and directories in the path pointed to by the string `path`.

Functions

- **int ispath** (const char *path)
check if given path exists
- **int isdir** (const char *path)
check if given path is a directory
- **int isfile** (const char *path)
check if given path is a regular file
- **int islink** (const char *path)
check if given path is a symbolic link
- **int ismount** (const char *path)
check if given path is a top-level mount point
- **int mkdirnamep** (const char *path, mode_t mode)
recursive mkdir(2) with dirname(3)
- **int mkdirp** (const char *path, mode_t mode)
recursive mkdir(2)
- **int runlink** (const char *path)
recursive unlink(2) and rmdir(2)
- **char * readsymlink** (const char *path)
read contents of symlink
- **int copy_file** (int srcfd, int dstfd)
copy a file

6.10.2 Function Documentation

6.10.2.1 `int ispath (const char * path)`

check if given path exists

Parameters:

← *path* path to check

Returns:

1 on success, 0 otherwise

See also:

stat(2)

Definition at line 21 of file ispath.c.

```
22 {  
23     struct stat stats;  
24     return stat(path, &stats) == 0;  
25 }
```

6.10.2.2 `int isdir (const char * path)`

check if given path is a directory

Parameters:

← *path* path to check

Returns:

1 on success, 0 otherwise

See also:

stat(2)

Definition at line 21 of file isdir.c.

```
22 {  
23     struct stat stats;  
24     return stat(path, &stats) == 0 && S_ISDIR(stats.st_mode);  
25 }
```

6.10.2.3 `int isfile (const char * path)`

check if given path is a regular file

Parameters:

← *path* path to check

Returns:

1 on success, 0 otherwise

See also:

stat(2)

Definition at line 21 of file isfile.c.

```
22 {  
23     struct stat stats;  
24     return stat(path, &stats) == 0 && S_ISREG(stats.st_mode);  
25 }
```

6.10.2.4 int islink (const char * *path*)

check if given path is a symbolic link

Parameters:

← *path* path to check

Returns:

1 on success, 0 otherwise

See also:

stat(2)

Definition at line 21 of file islink.c.

```
22 {  
23     struct stat stats;  
24     return stat(path, &stats) == 0 && S_ISLNK(stats.st_mode);  
25 }
```

6.10.2.5 int ismount (const char * *path*)

check if given path is a top-level mount point

Parameters:

← *path* path to check

Returns:

1 on success, 0 otherwise

See also:

stat(2)

Definition at line 23 of file ismount.c.

References mem_free(), str_equal(), str_isempty, and str_path_dirname().

```

24 {
25     struct stat sb_path, sb_parent;
26     int rc = 1;
27
28     if (str_isempty(path))
29         return 0;
30
31     if (str_equal(path, "/"))
32         return 1;
33
34     char *parent = str_path_dirname(path);
35
36     if (lstat(path, &sb_path) == -1 ||
37         !S_ISDIR(sb_path.st_mode) ||
38         lstat(parent, &sb_parent) == -1 ||
39         sb_path.st_dev == sb_parent.st_dev)
40         rc = 0;
41
42     mem_free(parent);
43     return rc;
44 }

```

6.10.2.6 int mkdirnamep (const char * *path*, mode_t *mode*)

recursive mkdir(2) with dirname(3)

Parameters:

- ← *path* path to create
- ← *mode* file permissions

Returns:

0 on success, -1 on error with errno set

See also:

mkdir(2)
dirname(3)

Definition at line 23 of file mkdirnamep.c.

References mem_free(), mkdirp(), str_isempty, and str_path_dirname().

```

24 {
25     if (str_isempty(path))
26         return errno = EINVAL, -1;
27
28     char *dname = str_path_dirname(path);
29     int rc      = mkdirp(dname, mode);
30
31     mem_free(dname);
32
33     return rc;
34 }

```

6.10.2.7 int mkdirp (const char * *path*, mode_t *mode*)

recursive mkdir(2)

Parameters:

- ← *path* path to create
- ← *mode* file permissions

Returns:

0 on success, -1 on error with errno set

See also:

mkdir(2)

Definition at line 26 of file mkdirp.c.

References open_read(), str_isempty, str_path_isdot(), strtok_for_each, strtok_free(), and strtok_init_str().

Referenced by chroot_mkdirp(), and mkdirnamep().

```

27 {
28     int ok = 1;
29     struct stat sb;
30
31     if (str_isempty(path) || str_path_isdot(path))
32         return errno = EINVAL, -1;
33
34     strtok_t _st, *st = &_st, *p;
35
36     int curdir = open_read(".");
37
38     if (curdir == -1)
39         return -1;
40
41     if (!strtok_init_str(st, path, "/", 0))
42         return -1;
43
44     strtok_for_each(st, p) {
45         if (mkdir(p->token, 0755) == -1) {
46             if (errno != EEXIST || stat(p->token, &sb) == -1) {
47                 ok = 0;
48                 break;
49             }
50
51             if (!S_ISDIR(sb.st_mode)) {
52                 errno = ENOTDIR;
53                 ok = 0;
54                 break;
55             }
56         }
57
58         if (chdir(p->token) == -1) {
59             ok = 0;
60             break;
61         }
62     }
63
64     if (ok && chmod(".", mode) == -1)
65         ok = 0;
66
67     fchdir(curdir);
68     close(curdir);
69
70     strtok_free(st);
71     return ok ? 0 : -1;
72 }
```

6.10.2.8 int unlink (const char * *path*)

recursive unlink(2) and rmdir(2)

Parameters:

← *path* path to remove

Returns:

0 on success, -1 on error with errno set

See also:

unlink(2)
rmdir(2)

Definition at line 26 of file unlink.c.

References `_lucid_asprintf()`, `mem_free()`, and `unlink()`.

Referenced by `unlink()`.

```

27 {
28     struct stat sb;
29
30     DIR *dp;
31     struct dirent *d;
32
33     int status = 0;
34     char *p, *new_path;
35
36     if (lstat(path, &sb) == -1) {
37         if (errno == ENOENT)
38             return 0;
39         else
40             return -1;
41     }
42
43     if (S_ISDIR(sb.st_mode)) {
44         if (!(dp = opendir(path)))
45             return -1;
46
47         while ((d = readdir(dp))) {
48             p = d->d_name;
49
50             if (p && p[0] == '.' && (!p[1] || (p[1] == '.' && !p[2])))
51                 continue;
52
53             _lucid_asprintf(&new_path, "%s/%s", path, d->d_name);
54
55             if (unlink(new_path) == -1)
56                 status = -1;
57
58             mem_free(new_path);
59         }
60
61         if (closedir(dp) == -1)
62             return -1;
63
64         if (rmdir(path) == -1)
65             return -1;
66
67         return status;

```

```

68     }
69
70     if (unlink(path) == -1)
71         return -1;
72
73     return 0;
74 }

```

6.10.2.9 char* readsymlink (const char * *path*)

read contents of symlink

Parameters:

← *path* symlink to read

Returns:

on success a pointer to a string containing the destination of the link, NULL on error with errno set

See also:

unlink(2)
rmdir(2)

Definition at line 24 of file readsymlink.c.

References CHUNKSIZE, mem_alloc(), mem_free(), and mem_realloc().

```

25 {
26     int chunks = 1, len = 0;
27     char *buf = mem_alloc(chunks * CHUNKSIZE + 1);
28
29     while (1) {
30         len = readlink(path, buf, chunks * CHUNKSIZE);
31
32         if (len == -1) {
33             mem_free(buf);
34             return NULL;
35         }
36
37         if (len >= chunks * CHUNKSIZE) {
38             chunks++;
39             buf = mem_realloc(buf, chunks * CHUNKSIZE + 1);
40         }
41
42         else
43             break;
44     }
45
46     buf[len] = '\0';
47
48     return buf;
49 }

```

6.10.2.10 int copy_file (int *srcfd*, int *dstfd*)

copy a file

Parameters:

- ← *srcfd* filedescriptor to read from
- ← *dstfd* filedescriptor to write to

Returns:

- 0 on success, -1 on error with errno set

Definition at line 66 of file copy_file.c.

References CHUNKSIZE.

```

67 {
68     int errno_orig;
69     int rc = -1, bufsize = 0;
70     void *srcbuf = MAP_FAILED, *dstbuf = MAP_FAILED;
71
72     /* install SIGBUS handler for mmap */
73     void (*oldhandler)(int) = signal(SIGBUS, __copy_file_sigbus_handler);
74
75     /* get file length */
76     struct stat sb;
77
78     if (fstat(srcfd, &sb) == -1)
79         goto out;
80
81     /* create sparse file */
82     if (ftruncate(dstfd, sb.st_size) == -1)
83         goto out;
84
85     if (sb.st_size < 1) {
86         rc = 0;
87         goto out;
88     }
89
90     /* save environment for non-local jump */
91     if (sigsetjmp(__copy_file_sigjmp_env, 1) != 0)
92         goto out;
93
94     int offset = 0;
95
96     while (offset < sb.st_size) {
97         bufsize = sb.st_size - offset;
98         bufsize = bufsize > CHUNKSIZE ? CHUNKSIZE : bufsize;
99
100         /* map source file */
101         srcbuf = mmap(0, bufsize, PROT_READ, MAP_SHARED, srcfd, offset);
102
103         if (srcbuf == MAP_FAILED)
104             goto out;
105
106         /* map destination file */
107         dstbuf = mmap(0, bufsize, PROT_WRITE, MAP_SHARED, dstfd, offset);
108
109         if (dstbuf == MAP_FAILED)
110             goto out;
111
112         offset += bufsize;
113
114         /* advise to sequential order (more aggressive read ahead) */
115         madvise(srcbuf, bufsize, MADV_SEQUENTIAL);
116         madvise(dstbuf, bufsize, MADV_SEQUENTIAL);
117
118         /* copy memory area with sparse support */
119         __copy_file_sparse_memcpy(dstbuf, srcbuf, bufsize);

```

```
120
121             munmap(srcbuf, bufsize);
122             srcbuf = MAP_FAILED;
123
124             munmap(dstbuf, bufsize);
125             dstbuf = MAP_FAILED;
126         }
127
128         rc = 0;
129
130 out:
131     if (srcbuf && srcbuf != MAP_FAILED)
132         munmap(srcbuf, bufsize);
133
134     if (dstbuf && dstbuf != MAP_FAILED)
135         munmap(dstbuf, bufsize);
136
137     errno_orig = errno;
138     signal(SIGBUS, oldhandler);
139     errno = errno_orig;
140     return rc;
141 }
```

6.11 Create or open files

6.11.1 Detailed Description

The open family of functions provide wrappers around `open(2)` with different flags.

Functions

- `int open__append (const char *filename)`
open file in append mode
- `int open__excl (const char *filename)`
open file exclusively
- `int open__read (const char *filename)`
open file for reading
- `int open__rw (const char *filename)`
open file for reading and writing
- `int open__trunc (const char *filename)`
open and truncate file for reading and writing
- `int open__write (const char *filename)`
open file for writing

6.11.2 Function Documentation

6.11.2.1 `int open__append (const char * filename)`

open file in append mode

Parameters:

filename file to open

Returns:

filedescriptor on success, -1 otherwise with `errno` set

See also:

Create or open files (p. 80)

Definition at line 21 of file `open__append.c`.

```
22 {  
23     return open(filename, O_WRONLY|O_NONBLOCK|O_APPEND|O_CREAT, 0666);  
24 }
```


6.11.2.2 int open_excl (const char * *filename*)

open file exclusively

Parameters:

filename file to open

Returns:

filedescriptor on success, -1 otherwise with errno set

See also:

Create or open files (p. 80)

Definition at line 21 of file open_excl.c.

```
22 {  
23     return open(filename, O_WRONLY|O_NONBLOCK|O_CREAT|O_EXCL, 0666);  
24 }
```

6.11.2.3 int open_read (const char * *filename*)

open file for reading

Parameters:

filename file to open

Returns:

filedescriptor on success, -1 otherwise with errno set

See also:

Create or open files (p. 80)

Definition at line 21 of file open_read.c.

Referenced by chroot_mkdirp(), chroot_secure_chdir(), and mkdirp().

```
22 {  
23     return open(filename, O_RDONLY|O_NONBLOCK);  
24 }
```

6.11.2.4 int open_rw (const char * *filename*)

open file for reading and writing

Parameters:

filename file to open

Returns:

filedescriptor on success, -1 otherwise with errno set

See also:

Create or open files (p. 80)

Definition at line 21 of file open_rw.c.

```
22 {  
23     return open(filename, O_RDWR|O_NONBLOCK|O_CREAT, 0666);  
24 }
```

6.11.2.5 int open__trunc (const char * *filename*)

open and truncate file for reading and writing

Parameters:

filename file to open

Returns:

filedescriptor on success, -1 otherwise with errno set

See also:

Create or open files (p. 80)

Definition at line 21 of file open_trunc.c.

```
22 {  
23     return open(filename, O_WRONLY|O_NONBLOCK|O_CREAT|O_TRUNC, 0666);  
24 }
```

6.11.2.6 int open__write (const char * *filename*)

open file for writing

Parameters:

filename file to open

Returns:

filedescriptor on success, -1 otherwise with errno set

See also:

Create or open files (p. 80)

Definition at line 21 of file open_write.c.

```
22 {  
23     return open(filename, O_WRONLY|O_NONBLOCK|O_CREAT, 0666);  
24 }
```

6.12 Formatted output conversion

6.12.1 Detailed Description

The functions in the `printf()` family produce output according to a format as described below.

The functions `printf()` and `vprintf()` write output to `stdout`, the standard output stream; `dprintf()` and `vdprintf()` write output to the file descriptor `fd`; `asprintf()` and `vasprintf()` allocate a string long enough to hold the output; `snprintf()` and `vsnprintf()` write to the character string `str`.

The functions `vprintf()`, `vdprintf()`, `vasprintf()` and `vsnprintf()` are equivalent to the functions `printf()`, `dprintf()`, `asprintf()` and `snprintf()`, respectively, except that they are called with a `va_`-list instead of a variable number of arguments. These functions do not call the `va_end` macro. Consequently, the value of `ap` is undefined after the call. The application should call `va_end(ap)` itself afterwards.

6.12.2 Format of the format string

The format string is composed of zero or more directives: ordinary characters (not `%`), which are copied unchanged to the output stream; and conversion specifications, each of which results in fetching zero or more subsequent arguments. Each conversion specification is introduced by the character `%`, and ends with a conversion specifier. In between there may be (in this order) zero or more flags, an optional minimum field width, an optional precision and an optional length modifier.

6.12.2.1 The flag characters

The character `%` is followed by zero or more of the following flags:

- `#`
The value should be converted to an “alternate form”. For `o` conversions, the first character of the output string is made zero (by prefixing a 0 if it was not zero already). For `x` conversions, the result has the string ‘0x’ prepended to it. For other conversions, the result is undefined.
- `0`
The value should be zero padded. For `d`, `i`, `o`, `u`, `x`, and `f` conversions, the converted value is padded on the left with zeros rather than blanks. If the `0` and `-` flags both appear, the `0` flag is ignored. If a precision is given with a numeric conversion (`d`, `i`, `o`, `u`, `x`, and `X`), the `0` flag is ignored. For other conversions, the behavior is undefined.
- `-`
The converted value is to be left adjusted on the field boundary. (The default is right justification.) Except for `n` conversions, the converted value is padded on the right with blanks, rather than on the left with blanks or zeros. A `-` overrides a `0` if both are given.
- `' '`
(a space) A blank should be left before a positive number (or empty string) produced by a signed conversion.
- `+`
A sign (`+` or `-`) should always be placed before a number produced by a signed conversion. By default a sign is used only for negative numbers. A `+` overrides a space if both are used.

6.12.2.2 The field width

An optional decimal digit string (with non-zero first digit) specifying a minimum field width. If the converted value has fewer characters than the field width, it will be padded with spaces on the left (or right, if the left-adjustment flag has been given). Instead of a decimal digit string one may write ‘*’ to specify that the field width is given in the next argument, which must be of type `int`. A negative field width is taken as a ‘-’ flag followed by a positive field width. In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is expanded to contain the conversion result.

6.12.2.3 The length modifier

Here, ‘integer conversion’ stands for `d`, `i`, `o`, `u`, or `x` conversion.

- `hh`
A following integer conversion corresponds to a signed char or unsigned char argument, or a following `n` conversion corresponds to a pointer to a signed char argument.
- `h`
A following integer conversion corresponds to a short int or unsigned short int argument, or a following `n` conversion corresponds to a pointer to a short int argument.
- `l`
(`ell`) A following integer conversion corresponds to a long int or unsigned long int argument.
- `ll`
(`ell-ell`). A following integer conversion corresponds to a long long int or unsigned long long int argument.

Note however that internally, `hh`, `h`, and `l` are handled as long int, `ll` as long long int, respectively.

6.12.2.4 The conversion specifier

A character that specifies the type of conversion to be applied. The conversion specifiers and their meanings are:

- `d,i`
The `int` argument is converted to signed decimal notation. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros. The default precision is 1. When 0 is printed with an explicit precision 0, the output is empty.
- `o,u,x,X`
The unsigned `int` argument is converted to unsigned octal (`o`), unsigned decimal (`u`), or unsigned hexadecimal (`x`) notation. The letters `abcdef` are used for `x` conversions. The precision, if any, gives the minimum number of digits that must appear; if the converted value requires fewer digits, it is padded on the left with zeros. The default precision is 1. When 0 is printed with an explicit precision 0, the output is empty.
- `c`
The `int` argument is converted to an unsigned char, and the resulting character is written.

- s

The `const char *` argument is expected to be a pointer to an array of character type (pointer to a string). Characters from the array are written up to (but not including) a terminating null byte (`'\0'`); if a precision is specified, no more than the number specified are written. If a precision is given, no null byte need be present; if the precision is not specified, or is greater than the size of the array, the array must contain a terminating null byte.

- p

The `void *` pointer argument is printed in hexadecimal.

- n

The number of characters written so far is stored into the integer indicated by the `int *` pointer argument. No argument is converted.

- %

A `'` is written. No argument is converted. The complete conversion specification is `'%'`.

6.12.3 Note on conformance

This `printf` implementation is not fully C99 or SUS compliant, though most common features are implemented in a completely self-contained way, to make integration within other applications as easy as possible.

See also:

`fmt`

Functions

- `int __lucid__vsnprintf (char *str, int size, const char *fmt, va_list ap)`
write conversion to string using va_list
- `int __lucid__snprintf (char *str, int size, const char *fmt,...)`
write conversion to string using variable number of arguments
- `int __lucid__vasprintf (char **ptr, const char *fmt, va_list ap)`
write conversion to allocated string using va_list
- `int __lucid__asprintf (char **ptr, const char *fmt,...)`
write conversion to allocated string using variable number of arguments
- `int __lucid__vdprintf (int fd, const char *fmt, va_list ap)`
write conversion to file descriptor using va_list
- `int __lucid__dprintf (int fd, const char *fmt,...)`
write conversion to file descriptor using variable number of arguments
- `int __lucid__vprintf (const char *fmt, va_list ap)`
write conversion to stdout using va_list
- `int __lucid__printf (const char *fmt,...)`
write conversion to stdout using variable number of arguments

6.12.4 Function Documentation

6.12.4.1 `int __lucid_vsnprintf (char * str, int size, const char * fmt, va_list ap)`

write conversion to string using `va_list`

Parameters:

- *str* buffer to store conversion
- ← *size* size of *str*
- ← *fmt* format string
- ← *ap* variable number of arguments

Returns:

number of bytes (that would have been) written

Note:

Every conversion happens in this functions. All other printf functions are just convenient wrappers.

Definition at line 179 of file `vsnprintf.c`.

References `EMIT`, `__printf_t::f`, `__printf_t::l`, `mem_set()`, `__printf_t::p`, `PFL_ALT`, `PFL_-BLANK`, `PFL_LEFT`, `PFL_SIGN`, `PFL_SIGNED`, `PFL_UPPER`, `PFL_ZERO`, `PFR_CHAR`, `PFR_INT`, `PFR_LLONG`, `PFR_LONG`, `PFR_MAX`, `PFR_MIN`, `PFR_SHORT`, `PFS_CONV`, `PFS_FLAGS`, `PFS_MOD`, `PFS_NORMAL`, `PFS_PREC`, `PFS_WIDTH`, `__printf_t::s`, `str_chr()`, `str_len()`, and `__printf_t::w`.

Referenced by `__lucid_snprintf()`, and `__lucid_vasprintf()`.

```

180 {
181     /* generic counter */
182     int i;
183
184     /* generic pointer */
185     const char *p;
186
187     /* keep track of string length */
188     int idx = 0;
189
190     /* save pointer to start of current conversion */
191     const char *ccp = fmt;
192
193     /* current character in format */
194     char c;
195
196     /* current conversion data */
197     __printf_t f;
198
199     /* arguments */
200     union {
201         /* signed argument */
202         signed long long int d;
203
204         /* unsigned argument */
205         unsigned long long int u;
206
207         /* float argument */

```

```

208         double f;
209
210         /* character argument */
211         int c;
212
213         /* string argument */
214         const char *s;
215
216         /* pointer argument */
217         void *p;
218
219         /* number argument */
220         int *n;
221     } arg;
222
223     /* base used for integer conversions */
224     int base;
225
226     /* number of consumed bytes in conversions */
227     int len;
228
229     /* don't consume original ap */
230     va_list ap;
231     va_copy(ap, _ap);
232
233     /* initialize conversion data */
234     f.f = 0;
235     f.l = PFR_INT;
236     f.p = -1;
237     f.s = PFS_NORMAL;
238     f.w = 0;
239
240     if (size > 0)
241         mem_set(str, 0, size);
242
243     while ((c = *fmt++)) {
244         switch (f.s) {
245             case PFS_NORMAL:
246                 if (c == '%') {
247                     f.f = 0;
248                     f.l = PFR_INT;
249                     f.p = -1;
250                     f.s = PFS_FLAGS;
251                     f.w = 0;
252                     ccp = &c;
253                 }
254
255                 else
256                     EMIT(c)
257
258                 break;
259
260             case PFS_FLAGS:
261                 switch (c) {
262                     case '#':
263                         f.f |= PFL_ALT;
264                         break;
265
266                     case '0':
267                         if (!(f.f & PFL_LEFT))
268                             f.f |= PFL_ZERO;
269                         break;
270
271                     case '-':
272                         f.f &= ~PFL_ZERO; /* left overrides zero */
273                         f.f |= PFL_LEFT;
274                         break;

```

```

275
276         case ' ':
277             f.f |= PFL_BLANK;
278             break;
279
280         case '+':
281             f.f &= ~PFL_BLANK; /* sign overrides blank */
282             f.f |= PFL_SIGN;
283             break;
284
285         default:
286             f.s = PFS_WIDTH;
287             fmt--;
288             break;
289     }
290
291     break;
292
293 case PFS_WIDTH:
294     if (c == '-') {
295         f.f &= PFL_ZERO; /* left overrides zero */
296         f.f |= PFL_LEFT;
297     }
298
299     else if (c >= '0' && c <= '9')
300         f.w = f.w * 10 + (c - '0');
301
302     else if (c == '*') {
303         f.w = va_arg(ap, int);
304
305         if (f.w < 0) {
306             f.w = -f.w;
307             f.f &= PFL_ZERO; /* left overrides zero */
308             f.f |= PFL_LEFT;
309         }
310     }
311
312     else if (c == '.') {
313         f.p = 0;
314         f.s = PFS_PREC;
315     }
316
317     else {
318         f.s = PFS_MOD;
319         fmt--;
320     }
321
322     break;
323
324 case PFS_PREC:
325     if (c >= '0' && c <= '9')
326         f.p = f.p * 10 + (c - '0');
327
328     else if (c == '*') {
329         f.p = va_arg(ap, int);
330
331         if (f.p < 0)
332             f.p = 0;
333     }
334
335     else {
336         f.s = PFS_MOD;
337         fmt--;
338     }
339
340     break;
341

```



```
342         case PFS_MOD:
343             switch (c) {
344                 case 'h':
345                     f.l--;
346                     break;
347
348                 case 'l':
349                     f.l++;
350                     break;
351
352                 default:
353                     f.s = PFS_CONV;
354                     fmt--;
355                     break;
356             }
357
358         break;
359
360     case PFS_CONV:
361         f.s = PFS_NORMAL;
362
363         if (f.l > PFR_MAX)
364             f.l = PFR_MAX;
365
366         if (f.l < PFR_MIN)
367             f.l = PFR_MIN;
368
369         switch (c) {
370             case 'P':
371                 f.f |= PFL_UPPER;
372
373             case 'p':
374                 base = 16;
375                 f.p = (8 * sizeof(void *) + 3)/4;
376                 f.f |= PFL_ALT;
377
378                 arg.u = (unsigned long long int) (unsigned long int) va_arg(ap, void *);
379
380                 goto is_integer;
381
382             case 'd':
383             case 'i': /* signed conversion */
384                 base = 10;
385                 f.f |= PFL_SIGNED;
386
387                 switch (f.l) {
388                     case PFR_CHAR:
389                         arg.d = (signed char) va_arg(ap, signed int);
390                         break;
391
392                     case PFR_SHORT:
393                         arg.d = (signed short int) va_arg(ap, signed int);
394                         break;
395
396                     case PFR_INT:
397                         arg.d = (signed int) va_arg(ap, signed int);
398                         break;
399
400                     case PFR_LONG:
401                         arg.d = (signed long int) va_arg(ap, signed long int);
402                         break;
403
404                     case PFR_LLONG:
405                         arg.d = (signed long long int) va_arg(ap, signed long long int);
406                         break;
407
408                     default:
```

```

409             arg.d = (signed long long int) va_arg(ap, signed int);
410             break;
411         }
412
413         arg.u = (unsigned long long int) arg.d;
414
415         goto is_integer;
416
417     case 'o':
418         base = 8;
419         goto is_unsigned;
420
421     case 'u':
422         base = 10;
423         goto is_unsigned;
424
425     case 'X':
426         f.f |= PFL_UPPER;
427
428     case 'x':
429         base = 16;
430         goto is_unsigned;
431
432     is_unsigned:
433         switch (f.l) {
434             case PFR_CHAR:
435                 arg.u = (unsigned char) va_arg(ap, unsigned int);
436                 break;
437
438             case PFR_SHORT:
439                 arg.u = (unsigned short int) va_arg(ap, unsigned int);
440                 break;
441
442             case PFR_INT:
443                 arg.u = (unsigned int) va_arg(ap, unsigned int);
444                 break;
445
446             case PFR_LONG:
447                 arg.u = (unsigned long int) va_arg(ap, unsigned long int);
448                 break;
449
450             case PFR_LLONG:
451                 arg.u = (unsigned long long int) va_arg(ap, unsigned long long int);
452                 break;
453
454             default:
455                 arg.u = (unsigned long long int) va_arg(ap, unsigned int);
456                 break;
457         }
458
459     is_integer:
460         len = __printf_int(str, size, arg.u, base, f);
461
462         str += len;
463         idx += len;
464         break;
465
466     case 'c': /* character conversion */
467         arg.c = (char) va_arg(ap, int);
468         EMIT(arg.c)
469         break;
470
471     case 's': /* string conversion */
472         arg.s = va_arg(ap, const char *);
473         arg.s = arg.s ? arg.s : "(null)";
474         len = str_len(arg.s);
475

```

```

476         is_string:
477             if (f.p != -1 && len > f.p)
478                 len = f.p;
479
480             if ((f.f & (PFL_LEFT|PFL_ZERO)) == 0) {
481                 while (f.w > len) {
482                     EMIT(' ')
483                     f.w--;
484                 }
485             }
486
487             if ((f.f & PFL_ZERO) > 0) {
488                 while (f.w > len) {
489                     EMIT('0')
490                     f.w--;
491                 }
492             }
493
494             for (i = len; i; i--)
495                 EMIT(*arg.s++)
496
497             if ((f.f & PFL_LEFT) > 0) {
498                 while (f.w > len) {
499                     EMIT(' ')
500                     f.w--;
501                 }
502             }
503
504             break;
505
506         case 'n':
507             arg.n = va_arg(ap, int *);
508             *arg.n = idx;
509
510             break;
511
512         case '%':
513             EMIT(c)
514             break;
515
516         default:
517             /* no padding for unknown conversion */
518             f.w = 0;
519             f.p = -1;
520
521             arg.s = ccp;
522             len = str_len(arg.s);
523             fmt = ccp + len;
524
525             p = str_chr(arg.s + 1, '%', len - 1);
526
527             if (p != 0) {
528                 len = p - arg.s - 1;
529                 fmt = p - 1;
530             }
531
532             goto is_string;
533     }
534
535     break;
536 }
537 }
538
539 va_end(ap);
540
541 return idx;
542 }

```

6.12.4.2 int __lucid_snprintf (char * *str*, int *size*, const char * *fmt*, ...)

write conversion to string using variable number of arguments

Parameters:

- *str* buffer to store conversion
- ← *size* size of str
- ← *fmt* format string
- ← ... variable number of arguments

Returns:

number of bytes (that would have been) written

Definition at line 19 of file snprintf.c.

References `__lucid_vsnprintf()`.

```

20 {
21     va_list ap;
22     va_start(ap, fmt);
23
24     return __lucid_vsnprintf(str, size, fmt, ap);
25 }
```

6.12.4.3 int __lucid_vasprintf (char ** *ptr*, const char * *fmt*, va_list *ap*)

write conversion to allocated string using va_list

Parameters:

- *ptr* pointer to string to store conversion
- ← *fmt* format string
- ← *ap* variable number of arguments

Returns:

number of bytes (that would have been) written

See also:

`malloc(3)`
`free(3)`

Definition at line 20 of file vasprintf.c.

References `__lucid_vsnprintf()`, and `mem_alloc()`.

Referenced by `__lucid_asprintf()`, `__lucid_vdprintf()`, `exec_fork()`, `exec_fork_background()`, `exec_fork_pipe()`, `exec_replace()`, and `stralloc_catf()`.

```

21 {
22     va_list ap2;
23     int len;
```

```

24     char *buf;
25
26     /* don't consume the original ap, we'll need it again */
27     va_copy(ap2, ap);
28
29     /* get required size */
30     len = _lucid_vsnprintf(0, 0, fmt, ap2);
31
32     va_end(ap2);
33
34     /* if size is 0, no buffer is allocated
35     ** just set *ptr to NULL and return size */
36     if (len > 0) {
37         if (!(buf = mem_alloc(len + 1)))
38             return -1;
39
40         _lucid_vsnprintf(buf, len + 1, fmt, ap);
41
42         *ptr = buf;
43     }
44
45     return len;
46 }

```

6.12.4.4 int _lucid_asprintf (char ** *ptr*, const char * *fmt*, ...)

write conversion to allocated string using variable number of arguments

Parameters:

- *ptr* pointer to string to store conversion
- ← *fmt* format string
- ← ... variable number of arguments

Returns:

number of bytes (that would have been) written

See also:

malloc(3)
free(3)

Definition at line 19 of file asprintf.c.

References _lucid_vasprintf().

Referenced by addr_to_str(), unlink(), and str_path_concat().

```

20 {
21     va_list ap;
22     va_start(ap, fmt);
23
24     return _lucid_vasprintf(ptr, fmt, ap);
25 }

```

6.12.4.5 int _lucid_vdprintf (int *fd*, const char * *fmt*, va_list *ap*)

write conversion to file descriptor using va_list

Parameters:

- ← *fd* open file descriptor
- ← *fmt* format string
- ← *ap* variable number of arguments

Returns:

number of bytes (that would have been) written

Definition at line 22 of file vdprintf.c.

References _lucid_vasprintf(), and mem_free().

Referenced by _lucid_dprintf(), and _lucid_vprintf().

```

23 {
24     char *buf;
25     int buflen, len;
26
27     buflen = _lucid_vasprintf(&buf, fmt, ap);
28     len = write(fd, buf, buflen);
29     mem_free(buf);
30
31     return len;
32 }
```

6.12.4.6 int _lucid_dprintf (int *fd*, const char * *fmt*, ...)

write conversion to file descriptor using variable number of arguments

Parameters:

- ← *fd* open file descriptor
- ← *fmt* format string
- ← ... variable number of arguments

Returns:

number of bytes (that would have been) written

Definition at line 19 of file dprintf.c.

References _lucid_vdprintf().

```

20 {
21     va_list ap;
22     va_start(ap, fmt);
23
24     return _lucid_vdprintf(fd, fmt, ap);
25 }
```

6.12.4.7 int _lucid_vprintf (const char * *fmt*, va_list *ap*)

write conversion to stdout using va_list

Parameters:

- ← *fmt* format string
- ← *ap* variable number of arguments

Returns:

number of bytes (that would have been) written

Definition at line 19 of file vprintf.c.

References _lucid_vdprintf().

Referenced by _lucid_printf().

```
20 {  
21     return _lucid_vdprintf(1, fmt, ap);  
22 }
```

6.12.4.8 int _lucid_printf (const char * *fmt*, ...)

write conversion to stdout using variable number of arguments

Parameters:

- ← *fmt* format string
- ← ... variable number of arguments

Returns:

number of bytes (that would have been) written

Definition at line 19 of file printf.c.

References _lucid_vprintf().

```
20 {  
21     va_list ap;  
22     va_start(ap, fmt);  
23  
24     return _lucid_vprintf(fmt, ap);  
25 }
```

6.13 Formatted input conversion

6.13.1 Detailed Description

The `scanf()` family of functions scans input according to format as described below. This format may contain conversion specifications; the results from such conversions, if any, are stored in the locations pointed to by the pointer arguments that follow format. Each pointer argument must be of a type that is appropriate for the value returned by the corresponding conversion specification.

If the number of conversion specifications in format exceeds the number of pointer arguments, the results are undefined. If the number of pointer arguments exceeds the number of conversion specifications, then the excess pointer arguments are evaluated, but are otherwise ignored.

The format string consists of a sequence of directives which describe how to process the sequence of input characters. If processing of a directive fails, no further input is read, and `scanf()` returns. A "failure" can be either of the following: input failure, meaning that input characters were unavailable, or matching failure, meaning that the input was inappropriate (see below).

A directive is one of the following:

- A sequence of white-space characters (space, tab, newline, etc; see `isspace(3)`). This directive matches any amount of white space, including none, in the input.
- An ordinary character (i.e., one other than white space or `\"`). This character must exactly match the next character of input.
- A conversion specification, which commences with a `\"` (percent) character. A sequence of characters from the input is converted according to this specification, and the result is placed in the corresponding pointer argument. If the next item of input does not match the the conversion specification, the conversion fails — this is a matching failure.

6.13.2 Format of the format string

Each conversion specification in format begins with either the character `\"` followed by:

- An optional `'*'` assignment-suppression character: `scanf()` reads input as directed by the conversion specification, but discards the input. No corresponding pointer argument is required, and this specification is not included in the count of successful assignments returned by `scanf()`.
- An optional decimal integer which specifies the maximum field width. Reading of characters stops either when this maximum is reached or when a non-matching character is found, whichever happens first. Most conversions discard initial whitespace characters (the exceptions are noted below), and these discarded characters don't count towards the maximum field width. String input conversions store a null terminator (`'\0'`) to mark the end of the input; the maximum field width does not include this terminator.
- An optional type modifier character. For example, the `l` type modifier is used with integer conversions such as `d` to specify that the corresponding pointer argument refers to a long int rather than a pointer to an int.
- A conversion specifier that specifies the type of input conversion to be performed.

6.13.2.1 Conversions

The following type modifier characters can appear in a conversion specification:

- hh
A following integer conversion corresponds to a signed char or unsigned char argument, or a following n conversion corresponds to a pointer to a signed char argument.
- h
A following integer conversion corresponds to a short int or unsigned short int argument, or a following n conversion corresponds to a pointer to a short int argument.
- l
(ell) A following integer conversion corresponds to a long int or unsigned long int argument.
- ll
(ell-ell). A following integer conversion corresponds to a long long int or unsigned long long int argument.

The following conversion specifiers are available:

- %
Matches a literal ". That is, %% in the format string matches a single input " character. No conversion is done, and assignment does not occur.
- d
Matches an optionally signed decimal integer; the next pointer must be a pointer to int.
- i
Matches an optionally signed integer; the next pointer must be a pointer to int. The integer is read in base 16 if it begins with 0x or 0X, in base 8 if it begins with 0, and in base 10 otherwise. Only characters that correspond to the base are used.
- o
Matches an unsigned octal integer; the next pointer must be a pointer to unsigned int.
- u
Matches an unsigned decimal integer; the next pointer must be a pointer to unsigned int.
- x,X
Matches an unsigned hexadecimal integer; the next pointer must be a pointer to unsigned int.
- s
Matches a sequence of non-white-space characters; the next pointer must be a pointer to character array that is long enough to hold the input sequence and the terminating null character ('\0'), which is added automatically. The input string stops at white space or at the maximum field width, whichever occurs first.
- c
Matches a sequence of characters whose length is specified by the maximum field width (default 1); the next pointer must be a pointer to char, and there must be enough room for all the characters (no terminating null byte is added). The usual skip of leading white space is suppressed. To skip white space first, use an explicit space in the format.

- **p**
Matches a pointer value (as printed by `p` in `printf(3)`; the next pointer must be a pointer to a pointer to void.
- **n**
Nothing is expected; instead, the number of characters consumed thus far from the input is stored through the next pointer, which must be a pointer to `int`. This is not a conversion, although it can be suppressed with the `*` assignment-suppression character.

Functions

- `int __lucid_vsscanf (const char *str, const char *fmt, va_list ap)`
read conversion from string using va_list
- `int __lucid_sscanf (const char *str, const char *fmt,...)`
read conversion from string using variable number of arguments

6.13.3 Function Documentation

6.13.3.1 `int __lucid_vsscanf (const char *str, const char *fmt, va_list ap)`

read conversion from string using `va_list`

Parameters:

- ← **str** source string
- ← **fmt** format string
- **ap** variable number of arguments

Returns:

Number of converted arguments

Note:

Every conversion happens in this functions. All other `scanf` functions are just convenient wrappers.

Definition at line 59 of file `vsscanf.c`.

References `char_isspace`, `__scanf_t::f`, `__scanf_t::l`, `__scanf_t::s`, `SFL_NOOP`, `SFL_WIDTH`, `SFR_CHAR`, `SFR_INT`, `SFR_LLONG`, `SFR_LONG`, `SFR_MAX`, `SFR_MIN`, `SFR_SHORT`, `SFS_CONV`, `SFS_EOF`, `SFS_ERR`, `SFS_FLAGS`, `SFS_MOD`, `SFS_NORMAL`, `SFS_WIDTH`, `str_len()`, `str_toumax()`, and `__scanf_t::w`.

Referenced by `__lucid_sscanf()`.

```

60 {
61     /* keep track of converted arguments */
62     int converted = 0;
63
64     /* current character in format */
65     char c;
```

```

66
67     /* current conversion data */
68     __scanf_t f;
69
70     /* arguments */
71     union {
72         /* unsigned argument */
73         unsigned long long int u;
74
75         /* string argument */
76         char *s;
77     } arg;
78
79     /* base used for integer conversions */
80     int base;
81
82     /* number of bytes converted in str_toumax */
83     int len;
84
85     /* pointer for string conversion */
86     char *sp;
87
88     /* don't consume original ap */
89     va_list ap;
90     va_copy(ap, _ap);
91
92     /* initialize conversion data */
93     f.f = 0;
94     f.l = SFR_INT;
95     f.s = SFS_NORMAL;
96     f.w = str_len(str);
97
98     while ((c = *fmt++)) {
99         switch (f.s) {
100             case SFS_NORMAL:
101                 if (c == '%') {
102                     f.f = 0;
103                     f.l = SFR_INT;
104                     f.s = SFS_FLAGS;
105                     f.w = str_len(str);
106                 }
107
108                 else if (char_isspace(c))
109                     while (char_isspace(*str))
110                         str++;
111
112                 else if (*str == c)
113                     str++;
114
115                 else
116                     f.s = SFS_ERR;
117
118                 break;
119
120             case SFS_FLAGS:
121                 switch (c) {
122                     case '*':
123                         f.f |= SFL_NOOP;
124                         break;
125
126                     case '0':
127                     case '1':
128                     case '2':
129                     case '3':
130                     case '4':
131                     case '5':
132                     case '6':

```

```
133         case '7':
134         case '8':
135         case '9':
136             f.w = (c - '0');
137             f.f |= SFL_WIDTH;
138             f.s = SFS_WIDTH;
139             break;
140
141         default:
142             f.s = SFS_MOD;
143             fmt--;
144             break;
145     }
146
147     break;
148
149     case SFS_WIDTH:
150         if (c >= '0' && c <= '9')
151             f.w = f.w * 10 + (c - '0');
152
153         else {
154             f.s = SFS_MOD;
155             fmt--;
156         }
157
158         break;
159
160     case SFS_MOD:
161         switch (c) {
162         case 'h':
163             f.l--;
164             break;
165
166         case 'l':
167             f.l++;
168             break;
169
170         default:
171             f.s = SFS_CONV;
172             fmt--;
173             break;
174         }
175
176         break;
177
178     case SFS_CONV:
179         f.s = SFS_NORMAL;
180
181         if (f.l > SFR_MAX)
182             f.l = SFR_MAX;
183
184         if (f.l < SFR_MIN)
185             f.l = SFR_MIN;
186
187         switch (c) {
188         case 'd':
189             base = 10;
190             goto scan_int;
191
192         case 'i': /* signed conversion */
193             base = 0;
194             goto scan_int;
195
196         case 'o':
197             base = 8;
198             goto scan_int;
199
```

```

200         case 'u':
201             base = 10;
202             goto scan_int;
203
204         case 'X':
205         case 'x':
206             base = 16;
207             goto scan_int;
208
209     scan_int:
210         while (char_isspace(*str))
211             str++;
212
213         if (!*str) {
214             f.s = SFS_EOF;
215             break;
216         }
217
218         len = str_toumax(str, &arg.u, base, f.w);
219
220         if (len <= 0) {
221             f.s = SFS_ERR;
222             break;
223         }
224
225         str += len;
226         converted++;
227
228         if (!(f.f & SFL_NOOP)) {
229             switch (f.l) {
230             case SFR_CHAR:
231                 *va_arg(ap, unsigned char *) = arg.u;
232                 break;
233
234             case SFR_SHORT:
235                 *va_arg(ap, unsigned short int *) = arg.u;
236                 break;
237
238             case SFR_INT:
239                 *va_arg(ap, unsigned int *) = arg.u;
240                 break;
241
242             case SFR_LONG:
243                 *va_arg(ap, unsigned long int *) = arg.u;
244                 break;
245
246             case SFR_LLONG:
247                 *va_arg(ap, unsigned long long int *) = arg.u;
248                 break;
249
250             default:
251                 *va_arg(ap, unsigned long long int *) = arg.u;
252                 break;
253             }
254         }
255
256         break;
257
258     case 'c': /* character conversion */
259         /* default width = 1 */
260         f.w = (f.f & SFL_WIDTH) ? f.w : 1;
261
262         if ((f.f & SFL_NOOP)) {
263             while (f.w-- > 0) {
264                 if (!*str) {
265                     f.s = SFS_EOF;
266                     break;

```

```

267         }
268
269         str++;
270     }
271 }
272
273 else {
274     arg.s = va_arg(ap, char *);
275
276     while (f.w-- > 0) {
277         if (!*str) {
278             f.s = SFS_EOF;
279             break;
280         }
281
282         *arg.s++ = *str++;
283     }
284 }
285
286 if (f.s != SFS_EOF && !(f.f & SFL_NOOP))
287     converted++;
288
289 break;
290
291 case 's': /* string conversion */
292     if (!(f.f & SFL_NOOP)) {
293         while (f.w-- && !char_isspace(*str)) {
294             if (!*str) {
295                 f.s = SFS_EOF;
296                 break;
297             }
298
299             str++;
300         }
301     }
302
303     else {
304         sp = arg.s = va_arg(ap, char *);
305
306         while (f.w-- && !char_isspace(*str)) {
307             if (!*str) {
308                 f.s = SFS_EOF;
309                 break;
310             }
311
312             *sp++ = *str++;
313         }
314
315         if (f.s != SFS_EOF)
316             *sp = '\0';
317     }
318
319     if (f.s != SFS_EOF && !(f.f & SFL_NOOP))
320         converted++;
321
322     break;
323
324 case 'p':
325 case 'p': /* pointer conversion */
326     while (char_isspace(*str))
327         str++;
328
329     if (!*str) {
330         f.s = SFS_EOF;
331         break;
332     }
333

```

```

334         len = str_toumax(str, &arg.u, 0, f.w);
335
336         if (len <= 0) {
337             f.s = SFS_ERR;
338             break;
339         }
340
341         if (!(f.f & SFL_NOOP))
342             *va_arg(ap, void **) = (void *) (unsigned long int) arg.u;
343
344         str += len;
345         converted++;
346
347         break;
348
349     case 'n':
350         *va_arg(ap, int *) = converted;
351
352         break;
353
354     case '%':
355         if (*str == '%')
356             str++;
357         else
358             f.s = SFS_ERR;
359
360         break;
361
362     default:
363         f.s = SFS_ERR;
364         break;
365     }
366
367     break;
368
369     case SFS_EOF:
370         converted = converted ? converted : -1;
371
372     case SFS_ERR:
373         va_end(ap);
374         return converted;
375     }
376 }
377
378 if (f.s == SFS_EOF)
379     converted = converted ? converted : -1;
380
381 va_end(ap);
382 return converted;
383 }

```

6.13.3.2 int _lucid_sscanf (const char * *str*, const char * *fmt*, ...)

read conversion from string using variable number of arguments

Parameters:

- ← *str* source string
- ← *fmt* format string
- ... variable number of arguments

Returns:

Number of converted arguments

Definition at line 19 of file sscanf.c.

References `_lucid_vsscanf()`.

Referenced by `addr_from_str()`.

```
20 {  
21     va_list ap;  
22     va_start(ap, fmt);  
23  
24     return _lucid_vsscanf(str, fmt, ap);  
25 }
```


6.14 String classification and conversion

6.14.1 Detailed Description

The `str_check` family of functions extend the classification of single characters to strings. The `str_check()` (p. 111) function checks the string pointed to by `str` for a set of allowed character classes. As soon as a character is found that is not allowed checking stops and 0 is returned.

The `str_cmp()` (p. 112) function compares the string pointed to by `str1` to the string pointed to by `str2`. It returns an integer less than, equal to, or greater than zero if `str1` is found, respectively, to be less than, to match, or be greater than `str2`.

The `strcpy()` function copies the string pointed to by `src` (including the terminating `'\0'` character) to the array pointed to by `dst`. The strings may not overlap, and the destination string `dst` must be large enough to receive the copy. The `strncpy()` function is similar, except that not more than `n` bytes of `src` are copied. Thus, if there is no null byte among the first `n` bytes of `src`, the result will not be null-terminated.

The `str_dup()` (p. 114) function returns a pointer to a new string which is a duplicate of the string `str`. The `str_dupn()` function is similar, but only copies at most `n` characters. If `s` is longer than `n`, only `n` characters are copied, and a terminating null byte is added.

The `str_index()` returns a pointer to the first occurrence of the character `c` in the string pointed to by `str`.

The `str_len()` (p. 116) function calculates the length of the string `str`, not including the terminating `'\0'` character.

The `str_path_concat()` (p. 119) function concatenates the directory name pointed to by `dirname` and file name pointed to by `basename` and checks that the latter does not contain any dot entries.

The `str_path_isabs()` (p. 119) and `str_path_isdot()` (p. 120) functions check if the file path pointed to by `str` is absolute or contains dots, respectively.

The `str_toupper()` (p. 121) and `str_tolower()` (p. 121) functions map lower-case to upper case and vice-versa, respectively.

The `str_zero()` function sets the first `n` bytes of the byte area starting at `s` to zero (bytes containing `'\0'`).

The `str_toumax()` (p. 122) function converts the string pointed to by `str` to an unsigned long long int `val` using `base` as conversion base.

Defines

- `#define CC_ALNUM (1 << 1)`
class for alpha-numerical characters
- `#define CC_ALPHA (1 << 2)`
class for upper- or lower-case characters
- `#define CC_ASCII (1 << 3)`
class for ASCII characters
- `#define CC_BLANK (1 << 4)`
class for blank characters

- `#define CC_CNTRL (1 << 5)`
class for ASCII control characters
- `#define CC_DIGIT (1 << 6)`
class for digit characters
- `#define CC_GRAPH (1 << 7)`
class for graphable characters
- `#define CC_LOWER (1 << 8)`
class for lower-case characters
- `#define CC_PRINT (1 << 9)`
class for printable characters
- `#define CC_PUNCT (1 << 10)`
class for punctuation characters
- `#define CC_SPACE (1 << 11)`
class for white space characters
- `#define CC_UPPER (1 << 12)`
class for upper-case characters
- `#define CC_XDIGIT (1 << 13)`
class for hexadecimal characters
- `#define str_isempty(str) (!str || str_check(str, CC_BLANK))`
check if string is empty
- `#define str_isalnum(str) str_check(str, CC_ALNUM)`
check string for alpha-numerical characters
- `#define str_isalpha(str) str_check(str, CC_ALPHA)`
check string for upper- or lower-case characters
- `#define str_iscii(str) str_check(str, CC_ASCII)`
check string for ASCII characters
- `#define str_isdigit(str) str_check(str, CC_DIGIT)`
check string for digit characters
- `#define str_isgraph(str) str_check(str, CC_GRAPH)`
check string for graphable characters
- `#define str_islower(str) str_check(str, CC_LOWER)`
check string for lower-case characters
- `#define str_isprint(str) str_check(str, CC_PRINT)`

check string for printable characters

- `#define str_isupper(str) str_check(str, CC_UPPER)`
check string for upper-case characters
- `#define str_isxdigit(str) str_check(str, CC_XDIGIT)`
check string for hexadecimal characters
- `#define CHUNKSIZE 4096`

Functions

- `int str_check (const char *str, int allowed)`
check string against classes of allowed characters
- `int str_cmp (const char *str1, const char *str2)`
compare two strings
- `int str_cmpn (const char *str1, const char *str2, int n)`
compare two strings
- `int str_equal (const char *str1, const char *str2)`
compare two strings
- `char * str_cpy (char *dst, const char *src)`
copy a string
- `char * str_cpyn (char *dst, const char *src, int n)`
copy a string
- `char * str_dup (const char *str)`
duplicate a string
- `char * str_chr (const char *str, int c, int n)`
scan string for character
- `char * str_rchr (const char *str, int c, int n)`
scan string for character beginning at the end
- `char * str_str (const char *str, const char *needle)`
locate a substring
- `int str_len (const char *str)`
calculate the length of a string
- `char * str_path_dirname (const char *path)`
parse directory component
- `char * str_path_basename (const char *path)`
parse basename component

- char * **str_path_concat** (const char *dirname, const char *basename)
concatenate dirname and basename
- int **str_path_isabs** (const char *str)
check if path is absolute and contains no dot entries or ungraphable characters
- int **str_path_isdot** (const char *str)
check if given path contains . or .. entries
- char * **str_tolower** (char *str)
convert string to lower-case
- char * **str_toupper** (char *str)
convert string to upper-case
- int **str_toumax** (const char *str, unsigned long long int *val, int base, int n)
convert string to integer
- int **str_readline** (int fd, char **str)
read a line of input
- int **str_readfile** (int fd, char **str)
read until end of file
- int **str_read** (int fd, char **str, int len)
read exact number of bytes

6.14.2 Define Documentation

6.14.2.1 `#define CC_ALNUM (1 << 1)`

class for alpha-numerical characters

Definition at line 71 of file str.h.

Referenced by str_check().

6.14.2.2 `#define CC_ALPHA (1 << 2)`

class for upper- or lower-case characters

Definition at line 74 of file str.h.

Referenced by str_check().

6.14.2.3 `#define CC_ASCII (1 << 3)`

class for ASCII characters

Definition at line 77 of file str.h.

Referenced by str_check().

6.14.2.4 `#define CC_BLANK (1 << 4)`

class for blank characters

Definition at line 80 of file str.h.

Referenced by str_check().

6.14.2.5 `#define CC_CNTRL (1 << 5)`

class for ASCII control characters

Definition at line 83 of file str.h.

Referenced by str_check().

6.14.2.6 `#define CC_DIGIT (1 << 6)`

class for digit characters

Definition at line 86 of file str.h.

Referenced by str_check().

6.14.2.7 `#define CC_GRAPH (1 << 7)`

class for graphable characters

Definition at line 89 of file str.h.

Referenced by str_check().

6.14.2.8 `#define CC_LOWER (1 << 8)`

class for lower-case characters

Definition at line 92 of file str.h.

Referenced by str_check().

6.14.2.9 `#define CC_PRINT (1 << 9)`

class for printable characters

Definition at line 95 of file str.h.

Referenced by str_check().

6.14.2.10 `#define CC_PUNCT (1 << 10)`

class for punctuation characters

Definition at line 98 of file str.h.

Referenced by str_check().

6.14.2.11 `#define CC_SPACE (1 << 11)`

class for white space characters

Definition at line 101 of file str.h.

Referenced by `str_check()`.

6.14.2.12 `#define CC_UPPER (1 << 12)`

class for upper-case characters

Definition at line 104 of file str.h.

Referenced by `str_check()`.

6.14.2.13 `#define CC_XDIGIT (1 << 13)`

class for hexadecimal characters

Definition at line 107 of file str.h.

Referenced by `str_check()`.

6.14.2.14 `#define str_isempty(str) (!str || str_check(str, CC_BLANK))`

check if string is empty

Definition at line 120 of file str.h.

Referenced by `addr_from_str()`, `ismount()`, `log_init()`, `mkdirnamep()`, `mkdirp()`, `str_path_basename()`, `str_path_concat()`, `str_path_dirname()`, `str_path_isabs()`, `str_path_isdot()`, `strtok_init_argv()`, and `strtok_init_str()`.

6.14.2.15 `#define str_isalnum(str) str_check(str, CC_ALNUM)`

check string for alpha-numerical characters

Definition at line 123 of file str.h.

6.14.2.16 `#define str_isalpha(str) str_check(str, CC_ALPHA)`

check string for upper- or lower-case characters

Definition at line 126 of file str.h.

6.14.2.17 `#define str_isspace(str) str_check(str, CC_ASCII)`

check string for ASCII characters

Definition at line 129 of file str.h.

6.14.2.18 `#define str_isdigit(str) str_check(str, CC_DIGIT)`

check string for digit characters

Definition at line 132 of file str.h.

Referenced by `addr_from_str()`.

6.14.2.19 `#define str_isgraph(str) str_check(str, CC_GRAPH)`

check string for graphable characters

Definition at line 135 of file str.h.

Referenced by `str_path_isabs()`.

6.14.2.20 `#define str_islower(str) str_check(str, CC_LOWER)`

check string for lower-case characters

Definition at line 138 of file str.h.

6.14.2.21 `#define str_isprint(str) str_check(str, CC_PRINT)`

check string for printable characters

Definition at line 141 of file str.h.

6.14.2.22 `#define str_isupper(str) str_check(str, CC_UPPER)`

check string for upper-case characters

Definition at line 144 of file str.h.

6.14.2.23 `#define str_isxdigit(str) str_check(str, CC_XDIGIT)`

check string for hexadecimal characters

Definition at line 147 of file str.h.

6.14.2.24 `#define CHUNKSIZE 4096`

bytes read at a time

Definition at line 346 of file str.h.

6.14.3 Function Documentation

6.14.3.1 `int str_check (const char * str, int allowed)`

check string against classes of allowed characters

Parameters:

← *str* string to check

← *allowed* allowed classes of characters (multiple classes by ORing)

Returns:

1 if all characters are valid, 0 otherwise

Definition at line 20 of file `str_check.c`.

References `CC_ALNUM`, `CC_ALPHA`, `CC_ASCII`, `CC_BLANK`, `CC_CNTRL`, `CC_DIGIT`, `CC_GRAPH`, `CC_LOWER`, `CC_PRINT`, `CC_PUNCT`, `CC_SPACE`, `CC_UPPER`, `CC_XDIGIT`, `char_isalnum`, `char_isalpha`, `char_isascii`, `char_isblank`, `char_iscntrl`, `char_isdigit`, `char_isgraph`, `char_islower`, `char_isprint`, `char_ispunct`, `char_isspace`, `char_isupper`, `char_isxdigit`, and `str_len()`.

```

21 {
22     int i, n;
23
24     if (!str)
25         return 1;
26
27     n = str_len(str);
28
29     for (i = 0; i < n; i++) {
30         if (allowed & CC_ALNUM  && char_isalnum (str[i])) continue;
31         if (allowed & CC_ALPHA  && char_isalpha (str[i])) continue;
32         if (allowed & CC_ASCII  && char_isascii (str[i])) continue;
33         if (allowed & CC_BLANK  && char_isblank (str[i])) continue;
34         if (allowed & CC_CNTRL  && char_iscntrl (str[i])) continue;
35         if (allowed & CC_DIGIT  && char_isdigit (str[i])) continue;
36         if (allowed & CC_GRAPH  && char_isgraph (str[i])) continue;
37         if (allowed & CC_LOWER  && char_islower (str[i])) continue;
38         if (allowed & CC_PRINT  && char_isprint (str[i])) continue;
39         if (allowed & CC_PUNCT  && char_ispunct (str[i])) continue;
40         if (allowed & CC_SPACE  && char_isspace (str[i])) continue;
41         if (allowed & CC_UPPER  && char_isupper (str[i])) continue;
42         if (allowed & CC_XDIGIT && char_isxdigit(str[i])) continue;
43
44         return 0;
45     }
46
47     return 1;
48 }
```

6.14.3.2 int str_cmp (const char * *str1*, const char * *str2*)

compare two strings

Parameters:

- ← *str1* first string
- ← *str2* second string

Returns:

An integer greater than, equal to, or less than 0, if the string pointed to by *str1* is greater than, equal to, or less than the string pointed to by *str2*, respectively.

Definition at line 19 of file `str_cmp.c`.

Referenced by `str_equal()`.


```
20 {  
21     while (*str1 && *str2 && *str1 == *str2)  
22         str1++, str2++;  
23  
24     return *str1 - *str2;  
25 }
```

6.14.3.3 int str_cmpn (const char * *str1*, const char * *str2*, int *n*)

compare two strings

Parameters:

- ← *str1* first string
- ← *str2* second string
- ← *n* compare first *n* bytes

Returns:

An integer greater than, equal to, or less than 0, if the string pointed to by *str1* is greater than, equal to, or less than the string pointed to by *str2*, respectively.

Definition at line 19 of file str_cmpn.c.

```
20 {  
21     while (--n && *str1 && *str2 && *str1 == *str2)  
22         str1++, str2++;  
23  
24     return *str1 - *str2;  
25 }
```

6.14.3.4 int str_equal (const char * *str1*, const char * *str2*)

compare two strings

Parameters:

- ← *str1* first string
- ← *str2* second string

Returns:

1 if both strings are equal, 0 otherwise.

Definition at line 19 of file str_equal.c.

References str_cmp().

Referenced by flist32_getval(), flist64_getval(), ismount(), str_path_dirname(), str_path_isabs(), str_path_isdot(), and strtok_delete().

```
20 {  
21     return str_cmp(str1, str2) == 0;  
22 }
```

6.14.3.5 char* str_cpy (char * *dst*, const char * *src*)

copy a string

Parameters:

- *dst* destination string
- ← *src* source string

Returns:

A pointer to *dst*.

Definition at line 20 of file str_cpy.c.

References mem_cpy(), and str_len().

```
21 {  
22     return mem_cpy(dst, src, str_len(src) + 1);  
23 }
```

6.14.3.6 char* str_cpyn (char * *dst*, const char * *src*, int *n*)

copy a string

Parameters:

- *dst* destination string
- ← *src* source string
- ← *n* copy at most n bytes

Returns:

A pointer to *dst*.

Definition at line 20 of file str_cpyn.c.

References mem_cpy(), and str_len().

```
21 {  
22     int len = str_len(src) + 1;  
23     return mem_cpy(dst, src, n > len ? len : n);  
24 }
```

6.14.3.7 char* str_dup (const char * *str*)

duplicate a string

Parameters:

- ← *str* source string

Returns:

A pointer to the duplicated string, or NULL if insufficient memory was available.

Definition at line 20 of file str_dup.c.

References mem_dup(), and str_len().

Referenced by str_path_basename(), str_path_dirname(), strtok_append(), strtok_init_argv(), and strtok_init_str().

```
21 {  
22     return mem_dup(str, str_len(str) + 1);  
23 }
```

6.14.3.8 char* str_chr (const char * str, int c, int n)

scan string for character

Parameters:

- ← **str** string to scan
- ← **c** character to look for
- ← **n** scan first n bytes

Returns:

A pointer to the matched character or NULL if the character is not found.

Definition at line 19 of file str_chr.c.

Referenced by _lucid_vsnprintf(), and addr_from_str().

```
20 {  
21     for (; n; str++, n--)  
22         if (*str == c)  
23             return (char *) str;  
24  
25     return 0;  
26 }
```

6.14.3.9 char* str_rchr (const char * str, int c, int n)

scan string for character beginning at the end

Parameters:

- ← **str** string to scan
- ← **c** character to look for
- ← **n** scan first n bytes

Returns:

A pointer to the matched character or NULL if the character is not found.

Definition at line 19 of file str_rchr.c.

Referenced by str_path_basename(), and str_path_dirname().

```

20 {
21     for (str += n - 1; n; str--, n--)
22         if (*str == c)
23             return (char *) str;
24
25     return 0;
26 }

```

6.14.3.10 char* str_str (const char * *str*, const char * *needle*)

locate a substring

Parameters:

- ← *str* string to scan
- ← *needle* string to look for

Returns:

A pointer to the matched substring or NULL if the substring is not found.

Definition at line 20 of file str_str.c.

References mem_cmp(), and str_len().

Referenced by strtok_init_str().

```

21 {
22     int i;
23     int slen = str_len(str);
24     int nlen = str_len(needle);
25
26     if (nlen < 1)
27         return (char *) str;
28
29     if (nlen > slen)
30         return 0;
31
32     for (i = slen - nlen + 1; i; i--) {
33         if (mem_cmp(str, needle, nlen) == 0)
34             return (char *) str;
35
36         str++;
37     }
38
39     return 0;
40 }

```

6.14.3.11 int str_len (const char * *str*)

calculate the length of a string

Parameters:

- ← *str* source string

Returns:

number of characters in str

Definition at line 19 of file `str_len.c`.

Referenced by `_lucid_vsnprintf()`, `_lucid_vsscanf()`, `addr_from_str()`, `flist32_to_str()`, `flist64_to_str()`, `str_check()`, `str_cpy()`, `str_cpyn()`, `str_dup()`, `str_path_basename()`, `str_path_concat()`, `str_path_dirname()`, `str_str()`, `stralloc_cats()`, `stralloc_copys()`, `strtok_init_str()`, `strtok_tostr()`, and `whirlpool_digest()`.

```

20 {
21     int i = 0;
22
23     while (*str++)
24         i++;
25
26     return i;
27 }
```

6.14.3.12 `char* str_path_dirname (const char * path)`

parse directory component

Parameters:

← *path* path to parse

Returns:

A pointer to the newly allocated string or NULL if insufficient memory was available.

Definition at line 20 of file `str_path_dirname.c`.

References `mem_free()`, `str_dup()`, `str_equal()`, `str_isempty`, `str_len()`, and `str_rchr()`.

Referenced by `ismount()`, and `mkdirnamep()`.

```

21 {
22     /* empty string or '..' */
23     if (str_isempty(path) || str_equal(path, ".."))
24         return str_dup(".");
25
26     /* skip prefixing '/' but preserve exactly one */
27     while (*path && *(path+1) && *path == '/' && *(path+1) == '/')
28         path++;
29
30     int found = 0;
31     char *p, *buf = str_dup(path);
32
33     while ((p = str_rchr(buf, '/', str_len(buf))) {
34         /* remove trailing slash */
35         if (p[1] == 0 && p != buf)
36             *p = 0;
37
38         /* no basename was found until yet */
39         else if (!found) {
40             *p = 0;
41             found = 1;
42         }
43
44         /* a basename was found and no trailing slash anymore */
45         else
46             break;
47     }
```

```

48
49     char *dn;
50
51     /* path consists only of basename and slashes */
52     if (str_isempty(buf))
53         dn = str_dup("/");
54
55     /* path is relative or absolute, basename was stripped */
56     else if (p)
57         dn = str_dup(buf);
58
59     /* path is relative, no basename was stripped */
60     else
61         dn = str_dup(".");
62
63     mem_free(buf);
64
65     return dn;
66 }

```

6.14.3.13 char* str_path_basename (const char * *path*)

parse basename component

Parameters:

← *path* path to parse

Returns:

A pointer to the newly allocated string or NULL if insufficient memory was available.

Definition at line 20 of file str_path_basename.c.

References mem_free(), str_dup(), str_isempty, str_len(), and str_rchr().

Referenced by log_traceme().

```

21 {
22     /* empty string */
23     if (str_isempty(path))
24         return str_dup(".");
25
26     /* skip prefixing '/' */
27     while (*path && *path == '/')
28         path++;
29
30     /* string consisting entirely of '/' */
31     if (!*path)
32         return str_dup("/");
33
34     char *p, *buf = str_dup(path);
35
36     while ((p = str_rchr(buf, '/', str_len(buf)))) {
37         /* remove trailing slash */
38         if (p[1] == 0 && p != buf)
39             *p = 0;
40
41         /* no trailing slash anymore */
42         else
43             break;
44     }
45

```

```

46     char *bn;
47
48     /* if a non-trailing slash was found, return everything after it */
49     if (p)
50         bn = str_dup(p + 1);
51
52     /* otherwise buf already contains basename */
53     else
54         bn = str_dup(buf);
55
56     mem_free(buf);
57
58     return bn;
59 }

```

6.14.3.14 char* str_path_concat (const char * *dirname*, const char * *basename*)

concatenate *dirname* and *basename*

Parameters:

- ← *dirname* directory part
- ← *basename* basename part

Returns:

A pointer to the newly allocated string or NULL if insufficient memory was available.

Definition at line 22 of file str_path_concat.c.

References `_lucid_asprintf()`, `mem_cmp()`, `str_isempty`, `str_len()`, and `str_path_isdot()`.

```

23 {
24     char *path = 0;
25
26     if (str_len(basename) > 1 && mem_cmp(basename, "./", 2) == 0)
27         basename += 2;
28
29     if (str_isempty(dirname) || str_path_isdot(basename))
30         return 0;
31
32     _lucid_asprintf(&path, "%s/%s", dirname, basename);
33
34     return path;
35 }

```

6.14.3.15 int str_path_isabs (const char * *str*)

check if path is absolute and contains no dot entries or ungraphable characters

Parameters:

- ← *str* path to check

Returns:

1 if *str* is an absolute pathname, 0 otherwise

Note:

this function does not check if the path exists

Definition at line 20 of file `str_path_isabs.c`.

References `str_equal()`, `str_isempty`, `str_isgraph`, `strtok_for_each`, `strtok_free()`, and `strtok_init_str()`.

```

21 {
22     int abs = 1;
23
24     if (str_isempty(str))
25         return 0;
26
27     if (*str != '/')
28         return 0;
29
30     strtok_t _st, *st = &_st, *p;
31
32     if (!strtok_init_str(st, str, "/", 0))
33         return -1;
34
35     strtok_for_each(st, p) {
36         if (str_equal(p->token, ".") || str_equal(p->token, "..") ||
37             !str_isgraph(p->token)) {
38             abs = 0;
39             break;
40         }
41     }
42
43     strtok_free(st);
44
45     return abs;
46 }
```

6.14.3.16 int str_path_isdot (const char * *str*)

check if given path contains . or .. entries

Parameters:

← *str* path to check

Returns:

1 if *str* has dot entries, 0 otherwise

Definition at line 20 of file `str_path_isdot.c`.

References `str_equal()`, `str_isempty`, `strtok_for_each`, `strtok_free()`, and `strtok_init_str()`.

Referenced by `mkdirp()`, and `str_path_concat()`.

```

21 {
22     int found = 0;
23
24     if (str_isempty(str))
25         return 0;
26
27     strtok_t _st, *st = &_st, *p;
```



```
28
29     if (!strtok_init_str(st, str, "/", 0))
30         return 0;
31
32     strtok_for_each(st, p) {
33         if (str_equal(p->token, ".") || str_equal(p->token, "..")) {
34             found = 1;
35             break;
36         }
37     }
38
39     strtok_free(st);
40
41     return found;
42 }
```

6.14.3.17 char* str_tolower (char * *str*)

convert string to lower-case

Parameters:

→ *str* string to convert

Returns:

pointer to *str*

Definition at line 20 of file str_tolower.c.

References char_tolower.

```
21 {
22     char *p = str;
23
24     while (*p) {
25         char_tolower(*p);
26         p++;
27     }
28
29     return str;
30 }
```

6.14.3.18 char* str_toupper (char * *str*)

convert string to upper-case

Parameters:

→ *str* string to convert

Returns:

pointer to *str*

Definition at line 20 of file str_toupper.c.

References char_toupper.

```

21 {
22     char *p = str;
23
24     while (*p) {
25         char_toupper(*p);
26         p++;
27     }
28
29     return str;
30 }

```

6.14.3.19 int str_toumax (const char * *str*, unsigned long long int * *val*, int *base*, int *n*)

convert string to integer

Parameters:

- ← *str* source string
- *val* destination integer
- ← *base* conversion base
- ← *n* convert first n bytes

Returns:

Number of bytes read from str

Definition at line 36 of file str_toumax.c.

References char_isspace.

Referenced by _lucid_vsscanf().

```

37 {
38     char c;
39     const char *p = str;
40     int d, minus = 0;
41     unsigned long long int v = 0;
42
43     while (n && char_isspace((unsigned char) *p)) {
44         p++;
45         n--;
46     }
47
48     /* Single optional + or - */
49     if (n) {
50         c = *p;
51
52         if (c == '-' || c == '+') {
53             minus = (c == '-');
54             p++;
55             n--;
56         }
57     }
58
59     if (base == 0) {
60         if (n >= 2 && p[0] == '0' && (p[1] == 'x' || p[1] == 'X')) {
61             n -= 2;
62             p += 2;
63             base = 16;

```

```

64         }
65
66         else if (n >= 1 && p[0] == '0') {
67             n--;
68             p++;
69             base = 8;
70         }
71
72         else {
73             base = 10;
74         }
75     }
76
77     else if (base == 16) {
78         if (n >= 2 && p[0] == '0' && (p[1] == 'x' || p[1] == 'X')) {
79             n -= 2;
80             p += 2;
81         }
82     }
83
84     while (n && (d = char_todigit(*p)) >= 0 && d < base) {
85         v = v * base + d;
86         n--;
87         p++;
88     }
89
90     if (p - str > 0)
91         *val = minus ? -v : v;
92
93     return p - str;
94 }

```

6.14.3.20 int str_readline (int *fd*, char ** *str*)

read a line of input

Parameters:

- ← *fd* file descriptor to read from
- *line* pointer to a string

Returns:

bytes on success, -1 on error with errno set

Note:

The caller should free obtained memory for line using free(3)

See also:

malloc(3)
free(3)
read(2)

Definition at line 22 of file str_readline.c.

References CHUNKSIZE, mem_alloc(), mem_free(), and mem_realloc().

23 {

```

24     int chunks = 1, len = 0;
25     char *buf = mem_alloc(chunks * CHUNKSIZE + 1);
26     char c;
27
28     while (1) {
29         switch(read(fd, &c, 1)) {
30             case -1:
31                 mem_free(buf);
32                 return -1;
33
34             case 0:
35                 goto out;
36
37             default:
38                 if (c == '\n' || c == '\r')
39                     goto out;
40
41                 if (len >= chunks * CHUNKSIZE) {
42                     chunks++;
43                     buf = mem_realloc(buf, chunks * CHUNKSIZE + 1);
44                 }
45
46                 buf[len++] = c;
47                 break;
48             }
49     }
50
51 out:
52     *line = buf;
53     return len;
54 }

```

6.14.3.21 int str__readfile (int *fd*, char ** *str*)

read until end of file

Parameters:

- ← *fd* file descriptor to read from
- *file* pointer to a string

Returns:

bytes on success, -1 on error with errno set

Note:

The caller should free obtained memory for file using free(3)

See also:

malloc(3)
free(3)
read(2)

Definition at line 22 of file str__readfile.c.

References CHUNKSIZE, mem_alloc(), mem_free(), and mem_realloc().

Referenced by exec_fork_pipe().

```

23 {
24     int chunks = 1, len = 0;
25     char *buf = mem_alloc(chunks * CHUNKSIZE + 1);
26
27     for (;;) {
28         int bytes_read = read(fd, buf+len, CHUNKSIZE);
29
30         if (bytes_read == -1) {
31             mem_free(buf);
32             return -1;
33         }
34
35         len += bytes_read;
36         buf[len] = '\0';
37
38         if (bytes_read == 0)
39             break;
40
41         if (bytes_read == CHUNKSIZE) {
42             chunks++;
43             buf = mem_realloc(buf, chunks * CHUNKSIZE + 1);
44         }
45     }
46
47     *str = buf;
48     return len;
49 }

```

6.14.3.22 int str_read (int *fd*, char ** *str*, int *len*)

read exact number of bytes

Parameters:

- ← *fd* file descriptor to read from
- *str* pointer to a string
- ← *len* bytes to be read

Returns:

bytes read on success, -1 on error with errno set

Note:

The caller should free obtained memory for *str* using `free(3)`

See also:

`malloc(3)`
`free(3)`
`read(2)`

Definition at line 22 of file `str_read.c`.

References `mem_alloc()`, and `mem_free()`.

```

23 {
24     int buflen;
25     char *buf = mem_alloc(len + 1);
26

```

```
27     if ((buflen = read(fd, buf, len)) == -1) {
28         mem_free(buf);
29         return -1;
30     }
31
32     *str = buf;
33     return buflen;
34 }
```

6.15 Dynamic string allocator

6.15.1 Detailed Description

A `stralloc` variable holds a byte string in dynamically allocated space. String contents are unrestricted; in particular, strings may contain `\0`. String length is limited only by memory and by the size of an unsigned int.

A `stralloc` structure has three components: `sa.s` is a pointer to the first byte of the string, or 0 if space is not allocated; `sa.len` is the number of bytes in the string, or undefined if space is not allocated; `sa.a` is the number of bytes allocated for the string, or undefined if space is not allocated.

Applications are expected to use `sa.s` and `sa.len` directly.

The `stralloc_ready()` (p. 129) function makes sure that `sa` has enough space allocated to hold `len` bytes. The `stralloc_readyplus()` (p. 129) function is like `stralloc_ready()` (p. 129) except that, if `sa` is already allocated, `stralloc_readyplus` adds the current length of `sa` to `len`.

The `stralloc_copyb()` (p. 131) function copies the string pointed to by `src` into `dst`, allocating space if necessary. The `stralloc_copys()` (p. 131) function copies a `\0`-terminated string from `src` into `dst`, without the `\0`. It is the same as `stralloc_copyb(&dst,buf,str_len(buf))`. `stralloc_copy` copies the string stored in `src` into `dst`. It is the same as `stralloc_copyb(&dst,src.s,src.len)`; `src` must already be allocated.

The `stralloc_catb()` (p. 132) adds the string pointed to by `src` to the end of the string stored in `dst`, allocating space if necessary. If `dst` is unallocated, `stralloc_catb` is the same as `stralloc_copyb`. The `stralloc_cats()` (p. 134) function is analogous to `stralloc_copys`, and `stralloc_cat` is analogous to `stralloc_copy`. The `stralloc_catf()` (p. 133) and `stralloc_catm()` (p. 133) functions are analogous to `stralloc_cats()` (p. 134) except that they take a formatted conversion or variable number of arguments, respectively, and appends these to the string stored in `dst`.

Data Structures

- struct `stralloc_t`
dynamic string allocator tracking data

Functions

- void `stralloc_init (stralloc_t *sa)`
initialize dynamic string allocator
- void `stralloc_zero (stralloc_t *sa)`
truncate string length to zero
- int `stralloc_ready (stralloc_t *sa, size_t len)`
ensure that enough memory has been allocated
- int `stralloc_readyplus (stralloc_t *sa, size_t len)`
ensure that enough memory has been allocated
- char * `stralloc_finalize (stralloc_t *sa)`
finalize dynamic string in new buffer

- `void stralloc_free (stralloc_t *sa)`
deallocate all memory
- `int stralloc_copyb (stralloc_t *dst, const char *src, size_t len)`
copy a static string to a dynamic one
- `int stralloc_copys (stralloc_t *dst, const char *src)`
copy a static string to a dynamic one
- `int stralloc_copy (stralloc_t *dst, const stralloc_t *src)`
copy one dynamic string to another
- `int stralloc_catb (stralloc_t *dst, const char *src, size_t len)`
concatenate a dynamic string and a static one
- `int stralloc_catf (stralloc_t *dst, const char *fmt,...)`
concatenate a dynamic string and a static one using formatted conversion
- `int stralloc_catm (stralloc_t *dst,...)`
concatenate a dynamic string and multiple static ones
- `int stralloc_cats (stralloc_t *dst, const char *src)`
concatenate a dynamic string and a static one
- `int stralloc_cat (stralloc_t *dst, const stralloc_t *src)`
concatenate two dynamic strings
- `int stralloc_cmp (const stralloc_t *a, const stralloc_t *b)`
compare two dynamic strings

6.15.2 Function Documentation

6.15.2.1 void stralloc_init (stralloc_t * sa)

initialize dynamic string allocator

Parameters:

→ **sa** string to initialize

Definition at line 20 of file `stralloc_init.c`.

References `stralloc_t::a`, `stralloc_t::len`, and `stralloc_t::s`.

Referenced by `flist32_to_str()`, `flist64_to_str()`, `strtok_tostr()`, and `whirlpool_digest()`.

```

21 {
22     sa->s = 0;
23     sa->len = sa->a = 0;
24 }
```


6.15.2.2 void stralloc_zero (stralloc_t * sa)

truncate string length to zero

Parameters:

→ *sa* string to truncate

Definition at line 20 of file stralloc_zero.c.

References stralloc_t::len.

```
21 {
22     sa->len = 0;
23 }
```

6.15.2.3 int stralloc_ready (stralloc_t * sa, size_t len)

ensure that enough memory has been allocated

Parameters:

← *sa* string to check

← *len* minimum length that has to be available

Returns:

0 on success, -1 on error with errno set

Definition at line 21 of file stralloc_ready.c.

References stralloc_t::a, mem_realloc(), and stralloc_t::s.

Referenced by stralloc_copyb(), and stralloc_readyplus().

```
22 {
23     size_t wanted = len + (len >> 3) + 30;
24     char *tmp;
25
26     if (!sa->s || sa->a < len) {
27         if (!(tmp = mem_realloc(sa->s, wanted)))
28             return -1;
29
30         sa->a = wanted;
31         sa->s = tmp;
32     }
33
34     return 0;
35 }
```

6.15.2.4 int stralloc_readyplus (stralloc_t * sa, size_t len)

ensure that enough memory has been allocated

Parameters:

← *sa* string to check

← *len* additional length that has to be available

Returns:

0 on success, -1 on error with `errno` set

Definition at line 22 of file `stralloc_readyplus.c`.

References `stralloc_t::len`, `stralloc_t::s`, and `stralloc_ready()`.

Referenced by `stralloc_catb()`.

```

23 {
24     if (sa->s) {
25         if (sa->len + len < len)
26             return errno = EINVAL, -1;
27
28         return stralloc_ready(sa, sa->len + len);
29     }
30
31     return stralloc_ready(sa, len);
32 }
```

6.15.2.5 `char* stralloc_finalize (stralloc_t * sa)`

finalize dynamic string in new buffer

Parameters:

← *sa* string to finalize

Returns:

Newly allocated null-terminated string on success, NULL otherwise.

Definition at line 21 of file `stralloc_finalize.c`.

References `stralloc_t::len`, `mem_alloc()`, `mem_cpy()`, and `stralloc_t::s`.

Referenced by `flist32_to_str()`, `flist64_to_str()`, `strtok_tostr()`, and `whirlpool_digest()`.

```

22 {
23     char *buf = mem_alloc(sa->len + 1);
24
25     if (!buf)
26         return 0;
27
28     mem_cpy(buf, sa->s, sa->len);
29     return buf;
30 }
```

6.15.2.6 `void stralloc_free (stralloc_t * sa)`

deallocate all memory

Parameters:

→ *sa* string to initialize

Definition at line 21 of file `stralloc_free.c`.

References `mem_free()`, and `stralloc_t::s`.

Referenced by `flist32_to_str()`, `flist64_to_str()`, `strtok_tostr()`, and `whirlpool_digest()`.

```

22 {
23     if (sa->s)
24         mem_free(sa->s);
25
26     sa->s = 0;
27 }
```

6.15.2.7 `int stralloc_copyb (stralloc_t * dst, const char * src, size_t len)`

copy a static string to a dynamic one

Parameters:

- **dst** dynamic destination string
- ← **src** static source string
- ← **len** copy at most len bytes

Returns:

- 0 on success, -1 on error with `errno` set

Definition at line 21 of file `stralloc_copyb.c`.

References `stralloc_t::len`, `mem_cpy()`, `stralloc_t::s`, and `stralloc_ready()`.

Referenced by `stralloc_copy()`, and `stralloc_copys()`.

```

22 {
23     if (stralloc_ready(dst, len) == -1)
24         return -1;
25
26     mem_cpy(dst->s, src, len);
27     dst->len = len;
28     return 0;
29 }
```

6.15.2.8 `int stralloc_copys (stralloc_t * dst, const char * src)`

copy a static string to a dynamic one

Parameters:

- **dst** dynamic destination string
- ← **src** static source string

Returns:

- 0 on success, -1 on error with `errno` set

Definition at line 21 of file `stralloc_copys.c`.

References `str_len()`, and `stralloc_copyb()`.

```

22 {
23     return stralloc_copyb(dst, src, str_len(src));
24 }

```

6.15.2.9 int stralloc_copy (stralloc_t * dst, const stralloc_t * src)

copy one dynamic string to another

Parameters:

- *dst* dynamic destination string
- ← *src* dynamic source string

Returns:

0 on success, -1 on error with errno set

Definition at line 20 of file stralloc_copy.c.

References stralloc_t::len, stralloc_t::s, and stralloc_copyb().

```

21 {
22     return stralloc_copyb(dst, src->s, src->len);
23 }

```

6.15.2.10 int stralloc_catb (stralloc_t * dst, const char * src, size_t len)

concatenate a dynamic string and a static one

Parameters:

- *dst* dynamic destination string
- ← *src* static source string
- ← *len* append at most len bytes

Returns:

0 on success, -1 on error with errno set

Definition at line 21 of file stralloc_catb.c.

References stralloc_t::len, mem_cpy(), stralloc_t::s, and stralloc_readyplus().

Referenced by stralloc_cat(), and stralloc_cats().

```

22 {
23     if (stralloc_readyplus(dst, len) == -1)
24         return -1;
25
26     mem_cpy(dst->s + dst->len, src, len);
27     dst->len += len;
28     return 0;
29 }

```

6.15.2.11 `int stralloc_catf (stralloc_t * dst, const char * fmt, ...)`

concatenate a dynamic string and a static one using formatted conversion

Parameters:

- *dst* dynamic destination string
- ← *fmt* format string
- ← ... variable number of arguments

Returns:

0 on success, -1 on error with `errno` set

Definition at line 23 of file `stralloc_catf.c`.

References `_lucid_vasprintf()`, `mem_free()`, and `stralloc_cats()`.

Referenced by `flist32_to_str()`, `flist64_to_str()`, `strtok_tostr()`, and `whirlpool_digest()`.

```
24 {
25     va_list ap;
26     char *buf;
27     int rc;
28
29     va_start(ap, fmt);
30
31     if (_lucid_vasprintf(&buf, fmt, ap) == -1) {
32         va_end(ap);
33         return -1;
34     }
35
36     va_end(ap);
37
38     rc = stralloc_cats(dst, buf);
39
40     mem_free(buf);
41
42     return rc;
43 }
```

6.15.2.12 `int stralloc_catm (stralloc_t * dst, ...)`

concatenate a dynamic string and multiple static ones

Parameters:

- *dst* dynamic destination string
- ← ... variable number of source strings

Returns:

0 on success, -1 on error with `errno` set

Note:

the last argument must be `NULL`

Definition at line 22 of file `stralloc_catm.c`.

References `stralloc_cats()`.

```

23 {
24     va_list ap;
25     char *s;
26
27     va_start(ap, dst);
28
29     while ((s = va_arg(ap, char *))) {
30         if (stralloc_cats(dst, s) == -1) {
31             va_end(ap);
32             return -1;
33         }
34     }
35
36     va_end(ap);
37     return 0;
38 }

```

6.15.2.13 `int stralloc_cats (stralloc_t * dst, const char * src)`

concatenate a dynamic string and a static one

Parameters:

- *dst* dynamic destination string
- ← *src* static source string

Returns:

- 0 on success, -1 on error with `errno` set

Definition at line 21 of file `stralloc_cats.c`.

References `str_len()`, and `stralloc_catb()`.

Referenced by `stralloc_catf()`, and `stralloc_catm()`.

```

22 {
23     return stralloc_catb(dst, src, str_len(src));
24 }

```

6.15.2.14 `int stralloc_cat (stralloc_t * dst, const stralloc_t * src)`

concatenate two dynamic strings

Parameters:

- *dst* dynamic destination string
- ← *src* dynamic source string

Returns:

- 0 on success, -1 on error with `errno` set

Definition at line 20 of file `stralloc_cat.c`.

References `stralloc_t::len`, `stralloc_t::s`, and `stralloc_catb()`.

```
21 {  
22     return stralloc_catb(dst, src->s, src->len);  
23 }
```

6.15.2.15 `int stralloc_cmp (const stralloc_t * a, const stralloc_t * b)`

compare two dynamic strings

Parameters:

- ← *a* first string
- ← *b* second string

Returns:

An integer greater than, equal to, or less than 0, if the string pointed to by *a* is greater than, equal to, or less than the string pointed to by *b*, respectively.

Definition at line 20 of file `stralloc_cmp.c`.

References `stralloc_t::len`, and `stralloc_t::s`.

```
21 {  
22     size_t i, j;  
23  
24     for (i = 0;; ++i) {  
25         if (i == a->len)  
26             return i == b->len ? 0 : -1;  
27  
28         if (i == b->len)  
29             return 1;  
30  
31         if ((j = ((unsigned char)(a->s[i]) - (unsigned char)(b->s[i]))))  
32             return j;  
33     }  
34  
35     return j;  
36 }
```

6.16 String tokenizer

Data Structures

- struct **strtok_t**

Defines

- #define **strtok_for_each**(st, p) list_for_each_entry(p, &(st → list), list)
iterate through tokens

Functions

- **strtok_t * strtok_init_argv** (strtok_t *st, char *argv[], int argc, int empty)
initialize string tokenizer from argument vector
- **strtok_t * strtok_init_str** (strtok_t *st, const char *str, char *delim, int empty)
initialize string tokenizer from character array
- void **strtok_free** (strtok_t *st)
deallocate string tokenizer
- int **strtok_count** (strtok_t *st)
count number of tokens
- int **strtok_append** (strtok_t *st, const char *token)
append a token
- void **strtok_delete** (strtok_t *st, const char *token)
delete one or more tokens
- char * **strtok_prev** (strtok_t **st)
Go to the previous token.
- char * **strtok_next** (strtok_t **st)
Go to the previous token.
- int **strtok_toargv** (strtok_t *st, char **argv)
convert string tokenizer to argument vector
- int **strtok_tostr** (strtok_t *st, char **str, char *delim)
convert string tokenizer to character array

6.16.1 Define Documentation

6.16.1.1 `#define strtok_for_each(st, p) list_for_each_entry(p, &(st → list), list)`

iterate through tokens

Definition at line 112 of file strtok.h.

Referenced by `flist32_from_str()`, `flist64_from_str()`, `mkdirp()`, `str_path_isabs()`, `str_path_isdot()`, `strtok_toargv()`, and `strtok_tostr()`.

6.16.2 Function Documentation

6.16.2.1 `strtok_t* strtok_init_argv (strtok_t * st, char * argv[], int argc, int empty)`

initialize string tokenizer from argument vector

Parameters:

- *st* tokenizer to initialize
- ← *argv* argument vector
- ← *argc* argument vector size

Returns:

A pointer to *st*.

Definition at line 21 of file strtok_init_argv.c.

References `strtok_t::list`, `mem_alloc()`, `str_dup()`, `str_isempty`, and `strtok_free()`.

```

22 {
23     int i;
24     strtok_t *new;
25
26     INIT_LIST_HEAD(&(st->list));
27
28     for (i = 0; i < argc; i++) {
29         if (!empty && str_isempty(argv[i]))
30             continue;
31
32         if (!(new = mem_alloc(sizeof(strtok_t)))) {
33             strtok_free(st);
34             return 0;
35         }
36
37         if (!(new->token = str_dup(argv[i]))) {
38             strtok_free(st);
39             return 0;
40         }
41
42         list_add_tail(&(new->list), &(st->list));
43     }
44
45     return st;
46 }
```

6.16.2.2 `strtok_t* strtok_init_str (strtok_t * st, const char * str, char * delim, int empty)`

initialize string tokenizer from character array

Parameters:

- *st* tokenizer to initialize
- ← *str* pointer to a string
- ← *delim* token delimiter
- ← *empty* convert empty tokens

Returns:

A pointer to *st*.

Definition at line 21 of file `strtok_init_str.c`.

References `strtok_t::list`, `mem_alloc()`, `mem_free()`, `mem_set()`, `str_dup()`, `str_isempty`, `str_len()`, `str_str()`, and `strtok_free()`.

Referenced by `exec_fork()`, `exec_fork_background()`, `exec_fork_pipe()`, `exec_replace()`, `flist32_from_str()`, `flist64_from_str()`, `mkdirp()`, `str_path_isabs()`, and `str_path_isdot()`.

```

22 {
23     strtok_t *new;
24     char *scopy, *cur, *token;
25
26     INIT_LIST_HEAD(&(st->list));
27
28     if (!str)
29         return st;
30
31     scpy = cur = token = str_dup(str);
32
33     if (!scopy)
34         return 0;
35
36     while (token) {
37         cur = str_str(cur, delim);
38
39         if (cur) {
40             mem_set(cur, 0, str_len(delim));
41             cur += str_len(delim);
42         }
43
44         if (empty || !str_isempty(token)) {
45             if (!(new = mem_alloc(sizeof(strtok_t))))
46                 goto free;
47
48             if (!(new->token = str_dup(token)))
49                 goto free;
50
51             list_add_tail(&(new->list), &(st->list));
52         }
53
54         token = cur;
55     }
56
57     goto out;
58
59 free:

```

```

60     strtok_free(st);
61     st = 0;
62
63 out:
64     mem_free(scpy);
65     return st;
66 }

```

6.16.2.3 void strtok_free (strtok_t * st)

deallocate string tokenizer

Parameters:

→ *st* tokenizer to free

Definition at line 22 of file strtok_free.c.

References strtok_t::list, list_entry, list_for_each_safe, mem_free(), and strtok_t::token.

Referenced by exec_fork(), exec_fork_background(), exec_fork_pipe(), exec_replace(), flist32_from_str(), flist64_from_str(), mkdirp(), str_path_isabs(), str_path_isdot(), strtok_init_argv(), and strtok_init_str().

```

23 {
24     int errno_orig = errno;
25     strtok_t *p;
26     list_t *pos, *tmp;
27
28     list_for_each_safe(pos, tmp, &(st->list)) {
29         p = list_entry(pos, strtok_t, list);
30         list_del(pos);
31
32         if (p->token)
33             mem_free(p->token);
34
35         mem_free(p);
36     }
37
38     errno = errno_orig;
39 }

```

6.16.2.4 int strtok_count (strtok_t * st)

count number of tokens

Parameters:

→ *st* tokenizer to initialize

Returns:

Number of tokens in st.

Definition at line 19 of file strtok_count.c.

References strtok_t::list, and list_for_each.

Referenced by exec_fork(), exec_fork_background(), exec_fork_pipe(), exec_replace(), strtok_toargv(), and strtok_tostr().

```

20 {
21     list_t *pos;
22     int count = 0;
23
24     list_for_each(pos, &(st->list))
25         count++;
26
27     return count;
28 }

```

6.16.2.5 int strtok__append (strtok__t * *st*, const char * *token*)

append a token

Parameters:

- *st* tokenizer to append to
- ← *token* token to append

Returns:

- 0 on success, -1 on error with errno set.

Definition at line 21 of file strtok__append.c.

References strtok__t::list, mem__alloc(), mem__free(), and str__dup().

```

22 {
23     strtok_t *new;
24
25     if (!(new = mem__alloc(sizeof(strtok_t))))
26         return -1;
27
28     if (!(new->token = str__dup(token))) {
29         mem__free(new);
30         return -1;
31     }
32
33     list_add_tail(&(new->list), &(st->list));
34
35     return 0;
36 }

```

6.16.2.6 void strtok__delete (strtok__t * *st*, const char * *token*)

delete one or more tokens

Parameters:

- *st* tokenizer to delete from
- ← *token* token to delete

Definition at line 21 of file strtok__delete.c.

References strtok__t::list, list__entry, list__for_each_safe, mem__free(), str__equal(), and strtok__t::token.

```

22 {
23     strtok_t *p;
24     list_t *pos, *tmp;
25
26     list_for_each_safe(pos, tmp, &(st->list)) {
27         p = list_entry(pos, strtok_t, list);
28
29         if (str_equal(p->token, token)) {
30             list_del(pos);
31             mem_free(p->token);
32             mem_free(p);
33         }
34     }
35 }

```

6.16.2.7 char* strtok_prev (strtok_t ** st)

Go to the previous token.

Parameters:

← *st* tokenizer to iterate

Definition at line 19 of file strtok_prev.c.

References strtok_t::list, list_entry, and strtok_t::token.

```

20 {
21     strtok_t *oldhead = *st;
22     list_t *prev = (&(oldhead->list))->prev;
23     strtok_t *newhead = list_entry(prev, strtok_t, list);
24
25     *st = newhead;
26
27     return oldhead->token;
28 }

```

6.16.2.8 char* strtok_next (strtok_t ** st)

Go to the previous token.

Parameters:

← *st* tokenizer to iterate

Definition at line 19 of file strtok_next.c.

References strtok_t::list, list_entry, and strtok_t::token.

```

20 {
21     strtok_t *oldhead = *st;
22     list_t *next = (&(oldhead->list))->next;
23     strtok_t *newhead = list_entry(next, strtok_t, list);
24
25     *st = newhead;
26
27     return oldhead->token;
28 }

```

6.16.2.9 int strtok_toargv (strtok_t * *st*, char ** *argv*)

convert string tokenizer to argument vector

Parameters:

- *st* tokenizer to convert
- ← *argv* pointer to an argument vector
- ← *argc* pointer to number of elements stored in argv
- 0 on success, -1 on error with errno set.

Definition at line 19 of file strtok_toargv.c.

References strtok_count(), strtok_for_each, and strtok_t::token.

Referenced by exec_fork(), exec_fork_background(), exec_fork_pipe(), and exec_replace().

```

20 {
21     int i = 0;
22     strtok_t *p;
23
24     if (strtok_count(st) < 1)
25         return 0;
26
27     strtok_for_each(st, p)
28         argv[i++] = p->token;
29
30     argv[i] = NULL;
31
32     return i;
33 }
```

6.16.2.10 int strtok_tostr (strtok_t * *st*, char ** *str*, char * *delim*)

convert string tokenizer to character array

Parameters:

- *st* tokenizer to convert
- ← *str* pointer to a string
- 0 on success, -1 on error with errno set.

Definition at line 21 of file strtok_tostr.c.

References stralloc_t::len, str_len(), stralloc_catf(), stralloc_finalize(), stralloc_free(), stralloc_init(), strtok_count(), strtok_for_each, and strtok_t::token.

```

22 {
23     int i = 0;
24     stralloc_t _sa, *sa = &_amp;_sa;
25     strtok_t *p;
26
27     if (strtok_count(st) < 1)
28         return 0;
29
30     stralloc_init(sa);
31 }
```

```
32     strtok_for_each(st, p) {
33         if (stralloc_catf(sa, "%s%s", p->token, delim) == -1)
34             return -1;
35
36         i++;
37     }
38
39     if (sa->len > 0)
40         sa->len -= str_len(delim);
41
42     *str = stralloc_finalize(sa);
43
44     stralloc_free(sa);
45     return i;
46 }
```

6.17 TCP socket wrappers

6.17.1 Detailed Description

The tcp family of functions provide wrappers around connect(2) and listen(2) taking an IP address in the string pointed to by ip and the port number as arguments.

Functions

- int **tcp_listen** (const char *ip, int port, int backlog)
listen for incoming connections
- int **tcp_connect** (const char *ip, int port)
connect to TCP socket

6.17.2 Function Documentation

6.17.2.1 int tcp_listen (const char * ip, int port, int backlog)

listen for incoming connections

Parameters:

ip IP to listen on
port port to listen on
backlog queue backlog

Returns:

filedescriptor for the newly allocated socket, -1 on error with errno set

Definition at line 26 of file tcp_listen.c.

References `addr_from_str()`, `addr_htos()`, and `mem_set()`.

```
27 {
28     int fd;
29     struct sockaddr_in inaddr;
30
31     if (port < 1)
32         return errno = EINVAL, -1;
33
34     mem_set(&inaddr, 0, sizeof(inaddr));
35     inaddr.sin_family = AF_INET;
36     inaddr.sin_port = addr_htos(port);
37
38     if (addr_from_str(ip, &inaddr.sin_addr.s_addr, 0) == 0)
39         return errno = EINVAL, -1;
40
41     if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
42         return -1;
43
44     if (bind(fd, (struct sockaddr *) &inaddr, sizeof(struct sockaddr_in)) == -1) {
45         close(fd);
46         return -1;
```



```
47     }
48
49     if (listen(fd, backlog) == -1) {
50         close(fd);
51         return -1;
52     }
53
54     return fd;
55 }
```

6.17.2.2 int tcp_connect (const char * *ip*, int *port*)

connect to TCP socket

Parameters:

ip IP to connect to
port port to connect to

Returns:

filedescriptor for the newly allocated connection, -1 on error with errno set

Definition at line 26 of file tcp_connect.c.

References `addr_from_str()`, and `mem_set()`.

```
27 {
28     int fd;
29     struct sockaddr_in inaddr;
30
31     if (port < 1)
32         return errno = EINVAL, -1;
33
34     mem_set(&inaddr, 0, sizeof(inaddr));
35     inaddr.sin_family = AF_INET;
36     inaddr.sin_port = htons(port);
37
38     if (addr_from_str(ip, &inaddr.sin_addr.s_addr, 0) == 0)
39         return errno = EINVAL, -1;
40
41     if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
42         return -1;
43
44     if (connect(fd, (struct sockaddr *) &inaddr, sizeof(struct sockaddr_in)) == -1) {
45         close(fd);
46         return -1;
47     }
48
49     return fd;
50 }
```

6.18 Whirlpool hash function

6.18.1 Detailed Description

WHIRLPOOL is a cryptographic hash function designed after the Square block cipher. WHIRLPOOL is a Miyaguchi-Preneel construction based on a substantially modified Advanced Encryption Standard (AES). Given a message less than 2^{256} bits in length, it returns a 512-bit message digest.

The **whirlpool_init()** (p. 151) function initializes the hash context pointed to by context. After initialization input can be added to the transform routine using **whirlpool_add()** (p. 153). Once all bytes have been added the transform has to be finished by calling **whirlpool_finalize()**.

An application should not directly use the internal **whirlpool_transform()** (p. 148) function, but always use **whirlpool_add()** (p. 153).

The **whirlpool_digest()** (p. 155) function combines the procedure explained above for a single string and returns the digest in hexadecimal notation.

Data Structures

- struct **whirlpool_t**
dynamic whirlpool state data

Defines

- #define **DIGESTBYTES** 64
number of bytes in the digest
- #define **DIGESTBITS** (8*DIGESTBYTES)
number of bits in the digest
- #define **WBLOCKBYTES** 64
number of bytes in the input buffer
- #define **WBLOCKBITS** (8*WBLOCKBYTES)
number of bits in the input buffer
- #define **LENGTHBYTES** 32
number of hashed bytes
- #define **LENGTHBITS** (8*LENGTHBYTES)
number of hashed bits

Functions

- void **whirlpool_transform** (**whirlpool_t** *const context)
internal transform routine

- void **whirlpool_init** (**whirlpool_t** *const context)
initialize whirlpool state context
- void **whirlpool_finalize** (**whirlpool_t** *const context, unsigned char *const result)
finalize whirlpool transformation
- void **whirlpool_add** (**whirlpool_t** *const context, const unsigned char *const src, unsigned long bits)
add bytes to the transform routine
- char * **whirlpool_digest** (const char *str)
create digest from string

6.18.2 Define Documentation

6.18.2.1 #define DIGESTBYTES 64

number of bytes in the digest

Definition at line 45 of file whirlpool.h.

Referenced by whirlpool_digest(), and whirlpool_finalize().

6.18.2.2 #define DIGESTBITS (8*DIGESTBYTES)

number of bits in the digest

Definition at line 48 of file whirlpool.h.

Referenced by whirlpool_add().

6.18.2.3 #define WBLOCKBYTES 64

number of bytes in the input buffer

Definition at line 52 of file whirlpool.h.

Referenced by whirlpool_finalize().

6.18.2.4 #define WBLOCKBITS (8*WBLOCKBYTES)

number of bits in the input buffer

Definition at line 55 of file whirlpool.h.

6.18.2.5 #define LENGTHBYTES 32

number of hashed bytes

Definition at line 59 of file whirlpool.h.

Referenced by whirlpool_finalize(), and whirlpool_init().

6.18.2.6 `#define LENGTHBITS (8*LENGTHBYTES)`

number of hashed bits

Definition at line 62 of file whirlpool.h.

6.18.3 Function Documentation

6.18.3.1 `void whirlpool_transform (whirlpool_t *const context)`

internal transform routine

Parameters:

← *context* whirlpool state context

Definition at line 26 of file whirlpool_transform.c.

References `whirlpool_t::buf`, `whirlpool_t::hash`, and `R`.

Referenced by `whirlpool_add()`, and `whirlpool_finalize()`.

```

27 {
28     int i, r;
29     uint64_t K[8];
30     uint64_t block[8];
31     uint64_t state[8];
32     uint64_t L[8];
33     uint8_t *buf = context->buf;
34
35     /* map the buffer to a block */
36     for (i = 0; i < 8; i++, buf += 8) {
37         block[i] = (((uint64_t)buf[0]          ) << 56) ^
38                    (((uint64_t)buf[1] & 0xffL) << 48) ^
39                    (((uint64_t)buf[2] & 0xffL) << 40) ^
40                    (((uint64_t)buf[3] & 0xffL) << 32) ^
41                    (((uint64_t)buf[4] & 0xffL) << 24) ^
42                    (((uint64_t)buf[5] & 0xffL) << 16) ^
43                    (((uint64_t)buf[6] & 0xffL) <<  8) ^
44                    (((uint64_t)buf[7] & 0xffL)          );
45     }
46
47     /* compute and apply K^0 to the cipher state */
48     state[0] = block[0] ^ (K[0] = context->hash[0]);
49     state[1] = block[1] ^ (K[1] = context->hash[1]);
50     state[2] = block[2] ^ (K[2] = context->hash[2]);
51     state[3] = block[3] ^ (K[3] = context->hash[3]);
52     state[4] = block[4] ^ (K[4] = context->hash[4]);
53     state[5] = block[5] ^ (K[5] = context->hash[5]);
54     state[6] = block[6] ^ (K[6] = context->hash[6]);
55     state[7] = block[7] ^ (K[7] = context->hash[7]);
56
57     /* iterate over all rounds */
58     for (r = 1; r <= R; r++) {
59         /* compute K^r from K^{r-1} */
60         L[0] = C0[(int)(K[0] >> 56)          ] ^
61               C1[(int)(K[7] >> 48) & 0xff] ^
62               C2[(int)(K[6] >> 40) & 0xff] ^
63               C3[(int)(K[5] >> 32) & 0xff] ^
64               C4[(int)(K[4] >> 24) & 0xff] ^
65               C5[(int)(K[3] >> 16) & 0xff] ^
66               C6[(int)(K[2] >>  8) & 0xff] ^
67               C7[(int)(K[1]          ) & 0xff] ^

```

```

68         rc[x];
69
70     L[1] = C0[(int)(K[1] >> 56)      ] ^
71           C1[(int)(K[0] >> 48) & 0xff] ^
72           C2[(int)(K[7] >> 40) & 0xff] ^
73           C3[(int)(K[6] >> 32) & 0xff] ^
74           C4[(int)(K[5] >> 24) & 0xff] ^
75           C5[(int)(K[4] >> 16) & 0xff] ^
76           C6[(int)(K[3] >> 8)  & 0xff] ^
77           C7[(int)(K[2]      ) & 0xff];
78
79     L[2] = C0[(int)(K[2] >> 56)      ] ^
80           C1[(int)(K[1] >> 48) & 0xff] ^
81           C2[(int)(K[0] >> 40) & 0xff] ^
82           C3[(int)(K[7] >> 32) & 0xff] ^
83           C4[(int)(K[6] >> 24) & 0xff] ^
84           C5[(int)(K[5] >> 16) & 0xff] ^
85           C6[(int)(K[4] >> 8)  & 0xff] ^
86           C7[(int)(K[3]      ) & 0xff];
87
88     L[3] = C0[(int)(K[3] >> 56)      ] ^
89           C1[(int)(K[2] >> 48) & 0xff] ^
90           C2[(int)(K[1] >> 40) & 0xff] ^
91           C3[(int)(K[0] >> 32) & 0xff] ^
92           C4[(int)(K[7] >> 24) & 0xff] ^
93           C5[(int)(K[6] >> 16) & 0xff] ^
94           C6[(int)(K[5] >> 8)  & 0xff] ^
95           C7[(int)(K[4]      ) & 0xff];
96
97     L[4] = C0[(int)(K[4] >> 56)      ] ^
98           C1[(int)(K[3] >> 48) & 0xff] ^
99           C2[(int)(K[2] >> 40) & 0xff] ^
100          C3[(int)(K[1] >> 32) & 0xff] ^
101          C4[(int)(K[0] >> 24) & 0xff] ^
102          C5[(int)(K[7] >> 16) & 0xff] ^
103          C6[(int)(K[6] >> 8)  & 0xff] ^
104          C7[(int)(K[5]      ) & 0xff];
105
106     L[5] = C0[(int)(K[5] >> 56)      ] ^
107           C1[(int)(K[4] >> 48) & 0xff] ^
108           C2[(int)(K[3] >> 40) & 0xff] ^
109           C3[(int)(K[2] >> 32) & 0xff] ^
110           C4[(int)(K[1] >> 24) & 0xff] ^
111           C5[(int)(K[0] >> 16) & 0xff] ^
112           C6[(int)(K[7] >> 8)  & 0xff] ^
113           C7[(int)(K[6]      ) & 0xff];
114
115     L[6] = C0[(int)(K[6] >> 56)      ] ^
116           C1[(int)(K[5] >> 48) & 0xff] ^
117           C2[(int)(K[4] >> 40) & 0xff] ^
118           C3[(int)(K[3] >> 32) & 0xff] ^
119           C4[(int)(K[2] >> 24) & 0xff] ^
120           C5[(int)(K[1] >> 16) & 0xff] ^
121           C6[(int)(K[0] >> 8)  & 0xff] ^
122           C7[(int)(K[7]      ) & 0xff];
123
124     L[7] = C0[(int)(K[7] >> 56)      ] ^
125           C1[(int)(K[6] >> 48) & 0xff] ^
126           C2[(int)(K[5] >> 40) & 0xff] ^
127           C3[(int)(K[4] >> 32) & 0xff] ^
128           C4[(int)(K[3] >> 24) & 0xff] ^
129           C5[(int)(K[2] >> 16) & 0xff] ^
130           C6[(int)(K[1] >> 8)  & 0xff] ^
131           C7[(int)(K[0]      ) & 0xff];
132
133     K[0] = L[0];
134     K[1] = L[1];

```

```

135         K[2] = L[2];
136         K[3] = L[3];
137         K[4] = L[4];
138         K[5] = L[5];
139         K[6] = L[6];
140         K[7] = L[7];
141
142         /* apply the r-th round transformation */
143         L[0] = C0[(int)(state[0] >> 56)      ] ^
144             C1[(int)(state[7] >> 48) & 0xff] ^
145             C2[(int)(state[6] >> 40) & 0xff] ^
146             C3[(int)(state[5] >> 32) & 0xff] ^
147             C4[(int)(state[4] >> 24) & 0xff] ^
148             C5[(int)(state[3] >> 16) & 0xff] ^
149             C6[(int)(state[2] >>  8) & 0xff] ^
150             C7[(int)(state[1]      ) & 0xff] ^
151             K[0];
152
153         L[1] = C0[(int)(state[1] >> 56)      ] ^
154             C1[(int)(state[0] >> 48) & 0xff] ^
155             C2[(int)(state[7] >> 40) & 0xff] ^
156             C3[(int)(state[6] >> 32) & 0xff] ^
157             C4[(int)(state[5] >> 24) & 0xff] ^
158             C5[(int)(state[4] >> 16) & 0xff] ^
159             C6[(int)(state[3] >>  8) & 0xff] ^
160             C7[(int)(state[2]      ) & 0xff] ^
161             K[1];
162
163         L[2] = C0[(int)(state[2] >> 56)      ] ^
164             C1[(int)(state[1] >> 48) & 0xff] ^
165             C2[(int)(state[0] >> 40) & 0xff] ^
166             C3[(int)(state[7] >> 32) & 0xff] ^
167             C4[(int)(state[6] >> 24) & 0xff] ^
168             C5[(int)(state[5] >> 16) & 0xff] ^
169             C6[(int)(state[4] >>  8) & 0xff] ^
170             C7[(int)(state[3]      ) & 0xff] ^
171             K[2];
172
173         L[3] = C0[(int)(state[3] >> 56)      ] ^
174             C1[(int)(state[2] >> 48) & 0xff] ^
175             C2[(int)(state[1] >> 40) & 0xff] ^
176             C3[(int)(state[0] >> 32) & 0xff] ^
177             C4[(int)(state[7] >> 24) & 0xff] ^
178             C5[(int)(state[6] >> 16) & 0xff] ^
179             C6[(int)(state[5] >>  8) & 0xff] ^
180             C7[(int)(state[4]      ) & 0xff] ^
181             K[3];
182
183         L[4] = C0[(int)(state[4] >> 56)      ] ^
184             C1[(int)(state[3] >> 48) & 0xff] ^
185             C2[(int)(state[2] >> 40) & 0xff] ^
186             C3[(int)(state[1] >> 32) & 0xff] ^
187             C4[(int)(state[0] >> 24) & 0xff] ^
188             C5[(int)(state[7] >> 16) & 0xff] ^
189             C6[(int)(state[6] >>  8) & 0xff] ^
190             C7[(int)(state[5]      ) & 0xff] ^
191             K[4];
192
193         L[5] = C0[(int)(state[5] >> 56)      ] ^
194             C1[(int)(state[4] >> 48) & 0xff] ^
195             C2[(int)(state[3] >> 40) & 0xff] ^
196             C3[(int)(state[2] >> 32) & 0xff] ^
197             C4[(int)(state[1] >> 24) & 0xff] ^
198             C5[(int)(state[0] >> 16) & 0xff] ^
199             C6[(int)(state[7] >>  8) & 0xff] ^
200             C7[(int)(state[6]      ) & 0xff] ^
201             K[5];

```

```

202
203         L[6] = C0[(int)(state[6] >> 56)          ] ^
204             C1[(int)(state[5] >> 48) & 0xff] ^
205             C2[(int)(state[4] >> 40) & 0xff] ^
206             C3[(int)(state[3] >> 32) & 0xff] ^
207             C4[(int)(state[2] >> 24) & 0xff] ^
208             C5[(int)(state[1] >> 16) & 0xff] ^
209             C6[(int)(state[0] >> 8) & 0xff] ^
210             C7[(int)(state[7]          ) & 0xff] ^
211             K[6];
212
213         L[7] = C0[(int)(state[7] >> 56)          ] ^
214             C1[(int)(state[6] >> 48) & 0xff] ^
215             C2[(int)(state[5] >> 40) & 0xff] ^
216             C3[(int)(state[4] >> 32) & 0xff] ^
217             C4[(int)(state[3] >> 24) & 0xff] ^
218             C5[(int)(state[2] >> 16) & 0xff] ^
219             C6[(int)(state[1] >> 8) & 0xff] ^
220             C7[(int)(state[0]          ) & 0xff] ^
221             K[7];
222
223         state[0] = L[0];
224         state[1] = L[1];
225         state[2] = L[2];
226         state[3] = L[3];
227         state[4] = L[4];
228         state[5] = L[5];
229         state[6] = L[6];
230         state[7] = L[7];
231     }
232
233     /* apply the Miyaguchi-Preneel compression function */
234     context->hash[0] ^= state[0] ^ block[0];
235     context->hash[1] ^= state[1] ^ block[1];
236     context->hash[2] ^= state[2] ^ block[2];
237     context->hash[3] ^= state[3] ^ block[3];
238     context->hash[4] ^= state[4] ^ block[4];
239     context->hash[5] ^= state[5] ^ block[5];
240     context->hash[6] ^= state[6] ^ block[6];
241     context->hash[7] ^= state[7] ^ block[7];
242 }

```

6.18.3.2 void whirlpool_init (whirlpool_t *const context)

initialize whirlpool state context

Parameters:

← *context* whirlpool state context

Definition at line 24 of file whirlpool_init.c.

References whirlpool_t::bits, whirlpool_t::buf, whirlpool_t::hash, whirlpool_t::len, LENGTHBYTES, mem_set(), and whirlpool_t::pos.

Referenced by whirlpool_digest().

```

25 {
26     int i;
27
28     mem_set(context->len, 0, LENGTHBYTES);
29
30     context->bits = context->pos = 0;

```

```

31     context->buf[0] = 0;
32
33     for (i = 0; i < 8; i++)
34         context->hash[i] = 0L;
35 }

```

6.18.3.3 void whirlpool_finalize (whirlpool_t *const context, unsigned char *const result)

finalize whirlpool transformation

Parameters:

- ← *context* whirlpool state context
- *result* string to store digest

Definition at line 24 of file whirlpool_finalize.c.

References whirlpool_t::bits, whirlpool_t::buf, DIGESTBYTES, whirlpool_t::hash, whirlpool_t::len, LENGTHBYTES, mem_cpy(), mem_set(), whirlpool_t::pos, WBLOCKBYTES, and whirlpool_transform().

Referenced by whirlpool_digest().

```

26 {
27     int i;
28     uint8_t *buf = context->buf;
29     uint8_t *len = context->len;
30     int bits = context->bits;
31     int pos = context->pos;
32     uint8_t *digest = result;
33
34     /* append a '1'-bit */
35     buf[pos] |= 0x80U >> (bits & 7);
36     pos++; /* all remaining bits on the current uint8_t are set to zero. */
37
38     /* pad with zero bits to complete (N*WBLOCKBITS - LENGTHBITS) bits */
39     if (pos > WBLOCKBYTES - LENGTHBYTES) {
40         if (pos < WBLOCKBYTES)
41             mem_set(&buf[pos], 0, WBLOCKBYTES - pos);
42
43         /* process data block */
44         whirlpool_transform(context);
45
46         /* reset buffer */
47         pos = 0;
48     }
49
50     if (pos < WBLOCKBYTES - LENGTHBYTES)
51         mem_set(&buf[pos], 0, (WBLOCKBYTES - LENGTHBYTES) - pos);
52
53     pos = WBLOCKBYTES - LENGTHBYTES;
54
55     /* append bit length of hashed data */
56     mem_cpy(&buf[WBLOCKBYTES - LENGTHBYTES], len, LENGTHBYTES);
57
58     /* process data block */
59     whirlpool_transform(context);
60
61     /* return the completed message digest */
62     for (i = 0; i < DIGESTBYTES/8; i++) {
63         digest[i] = (uint8_t)(context->hash[i] >> 56);

```



```

64         digest[1] = (uint8_t)(context->hash[i] >> 48);
65         digest[2] = (uint8_t)(context->hash[i] >> 40);
66         digest[3] = (uint8_t)(context->hash[i] >> 32);
67         digest[4] = (uint8_t)(context->hash[i] >> 24);
68         digest[5] = (uint8_t)(context->hash[i] >> 16);
69         digest[6] = (uint8_t)(context->hash[i] >> 8);
70         digest[7] = (uint8_t)(context->hash[i] );
71         digest += 8;
72     }
73
74     context->bits = bits;
75     context->pos  = pos;
76 }

```

6.18.3.4 void whirlpool_add (whirlpool_t *const context, const unsigned char *const src, unsigned long bits)

add bytes to the transform routine

Parameters:

- ← *context* whirlpool state context
- ← *src* source string
- ← *bits* number of bits in the source string

Definition at line 23 of file whirlpool_add.c.

References whirlpool_t::bits, whirlpool_t::buf, DIGESTBITS, whirlpool_t::len, whirlpool_t::pos, and whirlpool_transform().

Referenced by whirlpool_digest().

```

25 {
26     int i;
27     uint32_t b, carry;
28
29     /* index of leftmost source uint8_t containing data (1 to 8 bits). */
30     int srcpos = 0;
31
32     int gap      = (8 - ((int)srcbits & 7)) & 7; /* space on src[srcpos]. */
33     int rem      = context->bits & 7; /* occupied bits on buf[pos]. */
34
35     uint8_t *buf = context->buf;
36     uint8_t *len = context->len;
37     int bits     = context->bits;
38     int pos      = context->pos;
39
40     /* tally the length of the added data */
41     uint64_t value = srcbits;
42
43     for (i = 31, carry = 0; i >= 0 && (carry != 0 || value != 0ULL); i--) {
44         carry += len[i] + ((uint32_t)value & 0xff);
45         len[i] = (uint8_t)carry;
46         carry >>= 8;
47         value >>= 8;
48     }
49
50     /* process data in chunks of 8 bits */
51     while (srcbits > 8) {
52         /* take a byte from the source */
53         b = ((src[srcpos] << gap) & 0xff) |

```

```

54             ((src[srcpos + 1] & 0xff) >> (8 - gap));
55
56         /* process this byte */
57         buf[pos++] |= (uint8_t)(b >> rem);
58         bits += 8 - rem; /* bits = 8*pos; */
59
60         if (bits == DIGESTBITS) {
61             /* process data block */
62             whirlpool_transform(context);
63
64             /* reset buf */
65             bits = pos = 0;
66         }
67
68         buf[pos] = b << (8 - rem);
69         bits += rem;
70
71         /* proceed to remaining data */
72         srcbits -= 8;
73         srcpos++;
74     }
75
76     /* now 0 <= srcbits <= 8;
77      * furthermore, all data (if any is left) is in src[srcpos].
78      */
79     if (srcbits > 0) {
80         b = (src[srcpos] << gap) & 0xff; /* bits are left-justified on b. */
81
82         /* process the remaining bits */
83         buf[pos] |= b >> rem;
84     }
85
86     else
87         b = 0;
88
89     /* all remaining data fits on buf[pos],
90      * and there still remains some space.
91      */
92     if (rem + srcbits < 8)
93         bits += srcbits;
94
95     else {
96         /* buf[pos] is full */
97         pos++;
98         bits += 8 - rem; /* bits = 8*pos; */
99         srcbits -= 8 - rem;
100
101         /* now 0 <= srcbits < 8;
102          * furthermore, all data (if any is left) is in src[srcpos].
103          */
104         if (bits == DIGESTBITS) {
105             /* process data block */
106             whirlpool_transform(context);
107
108             /* reset buf */
109             bits = pos = 0;
110         }
111
112         buf[pos] = b << (8 - rem);
113         bits += (int)srcbits;
114     }
115
116     context->bits = bits;
117     context->pos = pos;
118 }

```

6.18.3.5 char* whirlpool_digest (const char * *str*)

create digest from string

Parameters:

← *str* source string

Returns:

digest string (memory obtained by malloc(3))

Note:

The caller should free obtained memory using free(3)

See also:

malloc(3)
free(3)

Definition at line 26 of file whirlpool_digest.c.

References DIGESTBYTES, str_len(), stralloc_catf(), stralloc_finalize(), stralloc_free(), stralloc_init(), whirlpool_add(), whirlpool_finalize(), and whirlpool_init().

```
27 {
28     whirlpool_t ctx;
29     stralloc_t sa;
30     char *buf;
31     uint8_t digest[DIGESTBYTES];
32     int i;
33
34     whirlpool_init(&ctx);
35     whirlpool_add(&ctx, (const unsigned char * const) str, str_len(str)*8);
36     whirlpool_finalize(&ctx, digest);
37
38     stralloc_init(&sa);
39
40     for (i = 0; i < DIGESTBYTES; i++)
41         stralloc_catf(&sa, "%02X", digest[i]);
42
43     buf = stralloc_finalize(&sa);
44
45     stralloc_free(&sa);
46
47     return buf;
48 }
```


Chapter 7

lucid Data Structure Documentation

7.1 `__printf_t` Struct Reference

7.1.1 Detailed Description

Definition at line 51 of file vsnprintf.c.

Data Fields

- unsigned int **f**
- int **l**
- int **p**
- int **s**
- unsigned int **w**

7.1.2 Field Documentation

7.1.2.1 `unsigned int __printf_t::f`

Definition at line 52 of file vsnprintf.c.

Referenced by `_lucid_vsnprintf()`.

7.1.2.2 `int __printf_t::l`

Definition at line 53 of file vsnprintf.c.

Referenced by `_lucid_vsnprintf()`.

7.1.2.3 `int __printf_t::p`

Definition at line 54 of file vsnprintf.c.

Referenced by `_lucid_vsnprintf()`.

7.1.2.4 int __printf_t::s

Definition at line 55 of file vsnprintf.c.

Referenced by `_lucid_vsnprintf()`.

7.1.2.5 unsigned int __printf_t::w

Definition at line 56 of file vsnprintf.c.

Referenced by `_lucid_vsnprintf()`.

The documentation for this struct was generated from the following file:

- `printf/vsnprintf.c`

7.2 `__scanf_t` Struct Reference

7.2.1 Detailed Description

Definition at line 48 of file `vsscanf.c`.

Data Fields

- `int f`
- `int l`
- `int s`
- `int w`

7.2.2 Field Documentation

7.2.2.1 `int __scanf_t::f`

Definition at line 49 of file `vsscanf.c`.

Referenced by `_lucid_vsscanf()`.

7.2.2.2 `int __scanf_t::l`

Definition at line 50 of file `vsscanf.c`.

Referenced by `_lucid_vsscanf()`.

7.2.2.3 `int __scanf_t::s`

Definition at line 51 of file `vsscanf.c`.

Referenced by `_lucid_vsscanf()`.

7.2.2.4 `int __scanf_t::w`

Definition at line 52 of file `vsscanf.c`.

Referenced by `_lucid_vsscanf()`.

The documentation for this struct was generated from the following file:

- `scanf/vsscanf.c`

7.3 `__mem_pool_t` Struct Reference

`#include <mem/mem_internal.h>`

Collaboration diagram for `__mem_pool_t`:

7.3.1 Detailed Description

Definition at line 22 of file `mem_internal.h`.

Data Fields

- `list_t list`
- `void * mem`
- `int len`

7.3.2 Field Documentation

7.3.2.1 `list_t __mem_pool_t::list`

Definition at line 23 of file `mem_internal.h`.

Referenced by `mem_alloc()`, `mem_free()`, and `mem_freeall()`.

7.3.2.2 `void* __mem_pool_t::mem`

Definition at line 24 of file `mem_internal.h`.

Referenced by `mem_free()`, `mem_freeall()`, and `mem_realloc()`.

7.3.2.3 `int __mem_pool_t::len`

Definition at line 25 of file `mem_internal.h`.

Referenced by `mem_realloc()`.

The documentation for this struct was generated from the following file:

- `mem/mem_internal.h`

7.4 flist32_t Struct Reference

```
#include <flist.h>
```

7.4.1 Detailed Description

32 bit list object

Definition at line 55 of file flist.h.

Data Fields

- `const char * key`
- `const uint32_t val`

7.4.2 Field Documentation

7.4.2.1 `const char* flist32_t::key`

Node key (must be unique)

Definition at line 56 of file flist.h.

Referenced by `flist32_getkey()`, `flist32_getval()`, and `flist32_to_str()`.

7.4.2.2 `const uint32_t flist32_t::val`

Node value (32-bit)

Definition at line 57 of file flist.h.

The documentation for this struct was generated from the following file:

- `flist.h`

7.5 flist64_t Struct Reference

```
#include <flist.h>
```

7.5.1 Detailed Description

64 bit list object

Definition at line 129 of file flist.h.

Data Fields

- `const char * key`
- `const uint64_t val`

7.5.2 Field Documentation

7.5.2.1 `const char* flist64_t::key`

Node key (must be unique)

Definition at line 130 of file flist.h.

Referenced by `flist64_getkey()`, `flist64_getval()`, and `flist64_to_str()`.

7.5.2.2 `const uint64_t flist64_t::val`

Node value (64-bit)

Definition at line 131 of file flist.h.

The documentation for this struct was generated from the following file:

- `flist.h`

7.6 list_head Struct Reference

```
#include <list.h>
```

Collaboration diagram for list_head:

7.6.1 Detailed Description

list head

Definition at line 62 of file list.h.

Data Fields

- list_head * next
- list_head * prev

7.6.2 Field Documentation

7.6.2.1 struct list_head* list_head::next [read]

Definition at line 63 of file list.h.

7.6.2.2 struct list_head * list_head::prev [read]

Definition at line 63 of file list.h.

The documentation for this struct was generated from the following file:

- list.h

7.7 log_options_t Struct Reference

```
#include <log.h>
```

7.7.1 Detailed Description

multiplexer configuration data

- The string pointed to by `log_ident` is prepended to every message, and is typically set to the program name.
- The `log_dest` argument specifies the log destination
- The `log_facility` argument is used to specify what type of program is logging the message; only used for the `syslog(3)` destination.
- The `log_opts` argument specifies flags which control the operation of the multiplexer.
- The `log_mask` argument is the lower level bound of messages being multiplexed.

Definition at line 76 of file `log.h`.

Data Fields

- `char * log_ident`
- `int log_dest`
- `int log_fd`
- `int log_facility`
- `int log_opts`
- `int log_mask`

7.7.2 Field Documentation

7.7.2.1 char* log_options_t::log_ident

program identifier

Definition at line 77 of file `log.h`.

Referenced by `log_init()`.

7.7.2.2 int log_options_t::log_dest

file destination switch

Definition at line 78 of file `log.h`.

Referenced by `log_close()`, and `log_init()`.

7.7.2.3 int log_options_t::log_fd

file descriptor for LOGD_FILE target

Definition at line 79 of file log.h.

Referenced by log_close(), and log_init().

7.7.2.4 int log_options_t::log_facility

program facility

Definition at line 80 of file log.h.

Referenced by log_init().

7.7.2.5 int log_options_t::log_opts

control flags

Definition at line 81 of file log.h.

Referenced by log_init().

7.7.2.6 int log_options_t::log_mask

lower log level bound

Definition at line 82 of file log.h.

Referenced by log_init().

The documentation for this struct was generated from the following file:

- log.h

7.8 stralloc_t Struct Reference

```
#include <stralloc.h>
```

7.8.1 Detailed Description

dynamic string allocator tracking data

This struct is used to keep track of the dynamic string state, i.e. its contents, its length and its additionally allocated memory.

Definition at line 69 of file stralloc.h.

Data Fields

- `char * s`
- `size_t len`
- `size_t a`

7.8.2 Field Documentation

7.8.2.1 `char* stralloc_t::s`

pointer to dynamic string

Definition at line 70 of file stralloc.h.

Referenced by `stralloc_cat()`, `stralloc_catb()`, `stralloc_cmp()`, `stralloc_copy()`, `stralloc_copyb()`, `stralloc_finalize()`, `stralloc_free()`, `stralloc_init()`, `stralloc_ready()`, and `stralloc_readyplus()`.

7.8.2.2 `size_t stralloc_t::len`

current length of s

Definition at line 71 of file stralloc.h.

Referenced by `stralloc_cat()`, `stralloc_catb()`, `stralloc_cmp()`, `stralloc_copy()`, `stralloc_copyb()`, `stralloc_finalize()`, `stralloc_init()`, `stralloc_readyplus()`, `stralloc_zero()`, and `strtok_tostr()`.

7.8.2.3 `size_t stralloc_t::a`

additional free memory in s

Definition at line 72 of file stralloc.h.

Referenced by `stralloc_init()`, and `stralloc_ready()`.

The documentation for this struct was generated from the following file:

- `stralloc.h`

7.9 strtok_t Struct Reference

```
#include <strtok.h>
```

Collaboration diagram for strtok_t:

7.9.1 Detailed Description

Definition at line 32 of file strtok.h.

Data Fields

- **list_t** **list**
- **char *** **token**

7.9.2 Field Documentation

7.9.2.1 list_t strtok_t::list

Definition at line 33 of file strtok.h.

Referenced by strtok_append(), strtok_count(), strtok_delete(), strtok_free(), strtok_init_argv(), strtok_init_str(), strtok_next(), and strtok_prev().

7.9.2.2 char* strtok_t::token

Definition at line 34 of file strtok.h.

Referenced by strtok_delete(), strtok_free(), strtok_next(), strtok_prev(), strtok_toargv(), and strtok_tostr().

The documentation for this struct was generated from the following file:

- **strtok.h**

7.10 whirlpool__t Struct Reference

```
#include <whirlpool.h>
```

7.10.1 Detailed Description

dynamic whirlpool state data

This struct is used to keep track of the whirlpool transform, i.e. its hashing state, input buffer, number of hashed bits, etc.

Definition at line 70 of file whirlpool.h.

Data Fields

- `uint8_t len` [LENGTHBYTES]
- `uint8_t buf` [WBLOCKBYTES]
- `int bits`
- `int pos`
- `uint64_t hash` [DIGESTBYTES/8]

7.10.2 Field Documentation

7.10.2.1 `uint8_t whirlpool__t::len`[LENGTHBYTES]

global number of hashed bits

Definition at line 71 of file whirlpool.h.

Referenced by `whirlpool__add()`, `whirlpool__finalize()`, and `whirlpool__init()`.

7.10.2.2 `uint8_t whirlpool__t::buf`[WBLOCKBYTES]

buffer of data to hash

Definition at line 72 of file whirlpool.h.

Referenced by `whirlpool__add()`, `whirlpool__finalize()`, `whirlpool__init()`, and `whirlpool__transform()`.

7.10.2.3 `int whirlpool__t::bits`

current number of bits on the buffer

Definition at line 73 of file whirlpool.h.

Referenced by `whirlpool__add()`, `whirlpool__finalize()`, and `whirlpool__init()`.

7.10.2.4 `int whirlpool__t::pos`

current (possibly incomplete) byte slot on the buffer

Definition at line 74 of file whirlpool.h.

Referenced by `whirlpool__add()`, `whirlpool__finalize()`, and `whirlpool__init()`.

7.10.2.5 uint64_t whirlpool_t::hash[DIGESTBYTES/8]

the hashing state

Definition at line 75 of file whirlpool.h.

Referenced by whirlpool_finalize(), whirlpool_init(), and whirlpool_transform().

The documentation for this struct was generated from the following file:

- **whirlpool.h**

Chapter 8

lucid File Documentation

8.1 addr.h File Reference

```
#include <stdint.h>
```

Include dependency graph for addr.h:

This graph shows which files directly or indirectly include this file:

Functions

- `uint16_t addr_htos (uint16_t addr)`
convert address from host to network byte order
- `uint32_t addr_hton (uint32_t addr)`
convert address from host to network byte order
- `uint32_t addr_ntoh (uint32_t addr)`
convert address from network to host byte order
- `uint16_t addr_stoh (uint16_t addr)`
convert address from network to host byte order
- `int addr_from_str (const char *str, uint32_t *ip, uint32_t *mask)`
convert string to IP address and netmask
- `char * addr_to_str (uint32_t ip, uint32_t mask)`
convert IP address and netmask to string

8.2 addr/addr_from_str.c File Reference

```
#include "addr.h"
```

```
#include "scanf.h"
```

```
#include "str.h"
```

Include dependency graph for addr_from_str.c:

Functions

- int **addr_from_str** (const char *str, uint32_t *ip, uint32_t *mask)
convert string to IP address and netmask

8.3 addr/addr_hton.c File Reference

```
#include "addr.h"
```

Include dependency graph for addr_hton.c:

Functions

- `uint32_t addr_hton (uint32_t addr)`
convert address from host to network byte order

8.4 addr/addr_htos.c File Reference

```
#include "addr.h"
```

Include dependency graph for addr_htos.c:

Functions

- `uint16_t addr_htos (uint16_t addr)`
convert address from host to network byte order

8.5 addr/addr_ntoh.c File Reference

```
#include "addr.h"
```

Include dependency graph for addr_ntoh.c:

Functions

- `uint32_t addr_ntoh (uint32_t addr)`
convert address from network to host byte order

8.6 addr/addr_stoh.c File Reference

```
#include "addr.h"
```

Include dependency graph for addr_stoh.c:

Functions

- `uint16_t addr_stoh (uint16_t addr)`
convert address from network to host byte order

8.7 addr/addr_to_str.c File Reference

```
#include "addr.h"
```

```
#include "printf.h"
```

Include dependency graph for addr_to_str.c:

Functions

- char * **addr_to_str** (uint32_t ip, uint32_t mask)
convert IP address and netmask to string

8.8 bitmap.h File Reference

`#include <stdint.h>`

Include dependency graph for `bitmap.h`:

This graph shows which files directly or indirectly include this file:

Functions

- `uint32_t i2v32 (int index)`
convert bit index to 32 bit value
- `uint64_t i2v64 (int index)`
convert bit index to 64 bit value
- `int v2i32 (uint32_t val)`
convert 32 bit value to bit index
- `int v2i64 (uint64_t val)`
convert 64 bit value to bit index

8.9 bitmap/i2v32.c File Reference

```
#include "bitmap.h"
```

Include dependency graph for i2v32.c:

Functions

- `uint32_t i2v32 (int index)`
convert bit index to 32 bit value

8.10 bitmap/i2v64.c File Reference

```
#include "bitmap.h"
```

Include dependency graph for i2v64.c:

Functions

- `uint64_t i2v64 (int index)`
convert bit index to 64 bit value

8.11 bitmap/v2i32.c File Reference

```
#include "bitmap.h"
```

Include dependency graph for v2i32.c:

Functions

- int **v2i32** (uint32_t val)
convert 32 bit value to bit index

8.12 bitmap/v2i64.c File Reference

```
#include "bitmap.h"
```

Include dependency graph for v2i64.c:

Functions

- int **v2i64** (uint64_t val)
convert 64 bit value to bit index

8.13 char.h File Reference

This graph shows which files directly or indirectly include this file:

Defines

- **#define** **char_isascii**(ch) ((unsigned int)(ch) < 128u)
check for an ASCII character
- **#define** **char_isblank**(ch) (ch == ' ' || ch == '\t')
check for a blank character (space, horizontal tab)
- **#define** **char_iscntrl**(ch) ((unsigned int)(ch) < 32u || ch == 127)
check for an ASCII control character
- **#define** **char_isdigit**(ch) ((unsigned int)(ch - '0') < 10u)
check for a digit character (0-9)
- **#define** **char_isgraph**(ch) ((unsigned int)(ch - '!') < 94u)
check for graphable characters (excluding space)
- **#define** **char_islower**(ch) ((unsigned int)(ch - 'a') < 26u)
check for a lower-case character
- **#define** **char_isprint**(ch) ((unsigned int)(ch - ' ') < 95u)
check for a printable character (including space)
- **#define** **char_isspace**(ch) ((unsigned int)(ch - '\t') < 5u || ch == ' ')
check for a whitespace character (\t, \n, \v, \f, \r)
- **#define** **char_isupper**(ch) ((unsigned int)(ch - 'A') < 26u)
check for an upper-case character
- **#define** **char_isxdigit**(ch)
check for a hexadecimal character
- **#define** **char_isalpha**(ch) (char_islower(ch) || char_isupper(ch))
check for an upper- or lower-case character
- **#define** **char_isalnum**(ch) (char_isalpha(ch) || char_isdigit(ch))
check for an upper-, lower-case or digit character
- **#define** **char_ispunct**(ch)
check for a punctuation character
- **#define** **char_tolower**(ch) do { if (char_isupper(ch)) ch += 32; } while(0)
convert character to lower-case

- `#define char_toupper(ch) do { if (char_islower(ch)) ch -= 32; } while(0)`
convert character to upper-case

8.14 chroot.h File Reference

```
#include <sys/types.h>
```

Include dependency graph for chroot.h:

This graph shows which files directly or indirectly include this file:

Functions

- int **chroot_fd** (int fd)
chroot(2) to the directory pointed to by a filedescriptor
- int **chroot_mkdirp** (const char *root, const char *dir, mode_t mode)
recursive mkdir(2) inside a secure chroot
- int **chroot_secure_chdir** (const char *root, const char *dir)
symlink-attack safe chdir(2) in chroot(2)

8.15 chroot/chroot_fd.c File Reference

```
#include <unistd.h>
```

```
#include "chroot.h"
```

Include dependency graph for chroot_fd.c:

Functions

- int **chroot_fd** (int fd)
chroot(2) to the directory pointed to by a filedescriptor

8.16 chroot/chroot__mkdirp.c File Reference

```
#include <unistd.h>
#include <errno.h>
#include "chroot.h"
#include "misc.h"
#include "open.h"
```

Include dependency graph for chroot__mkdirp.c:

Functions

- int **chroot__mkdirp** (const char *root, const char *dir, mode_t mode)
recursive mkdir(2) inside a secure chroot

8.17 chroot/chroot_secure_chdir.c File Reference

```
#include <unistd.h>
```

```
#include <errno.h>
```

```
#include "chroot.h"
```

```
#include "open.h"
```

Include dependency graph for chroot_secure_chdir.c:

Functions

- int **chroot_secure_chdir** (const char *root, const char *dir)
symlink-attack safe chdir(2) in chroot(2)

8.18 doxygen/examples.h File Reference

8.19 doxygen/license.h File Reference

8.20 doxygen/main.h File Reference

8.21 exec.h File Reference

This graph shows which files directly or indirectly include this file:

Defines

- `#define EXEC_MAX_ARGV 64`
maximum number of arguments that will be converted for `execvp(2)`

Functions

- `int exec_fork (const char *fmt,...)`
fork, `execvp` and wait
- `int exec_fork_background (const char *fmt,...)`
fork, `execvp` and ignore child
- `int exec_fork_pipe (char **out, const char *fmt,...)`
pipe, fork, `execvp` and wait
- `int exec_replace (const char *fmt,...)`
plain `execvp`

8.22 exec/exec__fork.c File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <stdarg.h>
#include <sys/wait.h>
#include "exec.h"
#include "mem.h"
#include "printf.h"
#include "strtok.h"
```

Include dependency graph for exec__fork.c:

Functions

- int **exec__fork** (const char *fmt,...)
fork, execvp and wait

8.23 exec/exec_fork_background.c File Reference

```
#include <unistd.h>
#include <stdarg.h>
#include <signal.h>
#include "exec.h"
#include "mem.h"
#include "printf.h"
#include "strtok.h"
```

Include dependency graph for exec_fork_background.c:

Functions

- **int exec_fork_background** (const char *fmt,...)
fork, execvp and ignore child

8.24 exec/exec_fork_pipe.c File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <stdarg.h>
#include <sys/wait.h>
#include "exec.h"
#include "mem.h"
#include "printf.h"
#include "str.h"
#include "strtok.h"
```

Include dependency graph for exec_fork_pipe.c:

Functions

- int **exec_fork_pipe** (char **out, const char *fmt,...)
pipe, fork, execvp and wait

8.25 exec/exec_replace.c File Reference

```
#include <unistd.h>
#include <stdarg.h>
#include "exec.h"
#include "mem.h"
#include "printf.h"
#include "strtok.h"
```

Include dependency graph for exec_replace.c:

Functions

- int **exec_replace** (const char *fmt,...)
plain execvp

8.26 flist.h File Reference

```
#include <stdint.h>
```

Include dependency graph for flist.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct **flist32_t**
32 bit list object
- struct **flist64_t**
64 bit list object

Defines

- #define **FLIST32_START**(LIST) const **flist32_t** LIST[] = {
32 bit list initialization
- #define **FLIST32_NODE**(PREFIX, NAME) { #NAME, PREFIX ## _ ## NAME
},
32 bit list node
- #define **FLIST32_NODE1**(PREFIX, NAME) { #NAME, (1 << PREFIX ## _ ##
NAME) },
32 bit list node from index
- #define **FLIST32_END** { 0, 0 } ;
32 bit list termination
- #define **FLIST64_START**(LIST) const **flist64_t** LIST[] = {
64 bit list initialization
- #define **FLIST64_NODE**(PREFIX, NAME) { #NAME, PREFIX ## _ ## NAME
},
64 bit list node
- #define **FLIST64_NODE1**(PREFIX, NAME) { #NAME, (1 << PREFIX ## _ ##
NAME) },
64 bit list node from index
- #define **FLIST64_END** { 0, 0 } ;
64 bit list termination

Functions

- `uint32_t flist32_getval (const flist32_t list[], const char *key)`
get 32 bit value by key
- `const char * flist32_getkey (const flist32_t list[], uint32_t val)`
get key from 32 bit value
- `int flist32_from_str (const char *str, const flist32_t list[], uint32_t *flags, uint32_t *mask, char clmod, char *delim)`
parse flag list string
- `char * flist32_to_str (const flist32_t list[], uint32_t val, char *delim)`
convert bit mask to flag list string
- `uint64_t flist64_getval (const flist64_t list[], const char *key)`
get 64 bit value by key
- `const char * flist64_getkey (const flist64_t list[], uint64_t val)`
get key from 64 bit value
- `int flist64_from_str (const char *str, const flist64_t list[], uint64_t *flags, uint64_t *mask, char clmod, char *delim)`
parse flag list string
- `char * flist64_to_str (const flist64_t list[], uint64_t val, char *delim)`
convert bit mask to flag list string

8.27 flist/flist32_from_str.c File Reference

```
#include "flist.h"
```

```
#include "strtok.h"
```

Include dependency graph for flist32_from_str.c:

Functions

- int **flist32_from_str** (const char *str, const **flist32_t** list[], uint32_t *flags, uint32_t *mask, char clmod, char *delim)
parse flag list string

8.28 flist/flist32_getkey.c File Reference

```
#include "flist.h"
```

Include dependency graph for flist32_getkey.c:

Functions

- `const char * flist32_getkey (const flist32_t list[], uint32_t val)`
get key from 32 bit value

8.29 flist/flist32_getval.c File Reference

```
#include "flist.h"
```

```
#include "str.h"
```

Include dependency graph for flist32_getval.c:

Functions

- `uint32_t flist32_getval (const flist32_t list[], const char *key)`
get 32 bit value by key

8.30 flist/flist32_to_str.c File Reference

```
#include "flist.h"
```

```
#include "str.h"
```

```
#include "stralloc.h"
```

Include dependency graph for flist32_to_str.c:

Functions

- char * **flist32_to_str** (const **flist32_t** list[], uint32_t val, char *delim)
convert bit mask to flag list string

8.31 flist/flist64_from_str.c File Reference

```
#include "flist.h"
```

```
#include "strtok.h"
```

Include dependency graph for flist64_from_str.c:

Functions

- int **flist64_from_str** (const char *str, const **flist64_t** list[], uint64_t *flags, uint64_t *mask, char clmod, char *delim)
parse flag list string

8.32 flist/flist64_getkey.c File Reference

```
#include "flist.h"
```

Include dependency graph for flist64_getkey.c:

Functions

- `const char * flist64_getkey (const flist64_t list[], uint64_t val)`
get key from 64 bit value

8.33 flist/flist64_getval.c File Reference

```
#include "flist.h"
```

```
#include "str.h"
```

Include dependency graph for flist64_getval.c:

Functions

- `uint64_t flist64_getval (const flist64_t list[], const char *key)`
get 64 bit value by key

8.34 flist/flist64_to_str.c File Reference

```
#include "flist.h"
```

```
#include "str.h"
```

```
#include "stralloc.h"
```

Include dependency graph for flist64_to_str.c:

Functions

- char * **flist64_to_str** (const **flist64_t** list[], uint64_t val, char *delim)
convert bit mask to flag list string

8.35 list.h File Reference

```
#include <stddef.h>
```

```
#include <lucid/mem.h>
```

Include dependency graph for list.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct **list_head**
list head

Defines

- #define **container_of**(ptr, type, member) ((type *)((char *)(ptr) - offsetof(type, member)))
get container of list head
- #define **LIST_NODE_ALLOC**(NAME) NAME = mem_alloc(sizeof(*NAME))
- #define **list_entry**(ptr, type, member) container_of(ptr, type, member)
get the struct for this entry
- #define **list_for_each**(pos, head) for (pos = (head) → next; pos != (head); pos = pos → next)
iterate over a list
- #define **list_for_each_prev**(pos, head) for (pos = (head) → prev; pos != (head); pos = pos → prev)
iterate over a list backwards
- #define **list_for_each_safe**(pos, n, head)
iterate over a list safe against removal of list entry
- #define **list_for_each_entry**(pos, head, member)
iterate over list of given type
- #define **list_for_each_entry_reverse**(pos, head, member)
iterate backwards over list of given type.
- #define **list_for_each_entry_safe**(pos, n, head, member)
iterate over list of given type safe against removal of list entry
- #define **list_for_each_entry_safe_reverse**(pos, n, head, member)

iterate backwards over list of given type safe against removal of list entry

Typedefs

- typedef **list_head** **list_t**
list head

8.36 log.h File Reference

```
#include <stdarg.h>
```

Include dependency graph for log.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct **log__options_t**
multiplexer configuration data

Defines

- #define **LOGD__SYSLOG** 0x01
- #define **LOGD__FILE** 0x02
- #define **LOGD__STDERR** 0x04
- #define **LOGP__ALERT** 0
- #define **LOGP__ERROR** 1
- #define **LOGP__WARN** 2
- #define **LOGP__NOTE** 3
- #define **LOGP__INFO** 4
- #define **LOGP__DEBUG** 5
- #define **LOGP__TRACE** 6
- #define **LOGO__PID** 0x01
- #define **LOGO__TIME** 0x02
- #define **LOGO__PRIO** 0x04
- #define **LOGO__IDENT** 0x08
- #define **LOG__TRACEME** log_traceme(__FILE__, __FUNCTION__, __LINE__, __);
simple trace helper

Functions

- void **log__init** (log__options_t *options)
initialize log message mutliplexer
- int **log__alert** (const char *fmt,...)
send ALERT level message to the multiplexer
- int **log__error** (const char *fmt,...)
send ERR level message to the multiplexer

- int **log_warn** (const char *fmt,...)
send WARNING level message to the multiplexer
- int **log_notice** (const char *fmt,...)
send NOTICE level message to the multiplexer
- int **log_info** (const char *fmt,...)
send INFO level message to the multiplexer
- int **log_debug** (const char *fmt,...)
send DEBUG level message to the multiplexer
- int **log_trace** (const char *fmt,...)
send TRACE level message to the multiplexer
- int **log_traceme** (const char *file, const char *func, int line)
send TRACE level message to the multiplexer
- void **log_alert_and_die** (const char *fmt,...)
send ALERT level message to the multiplexer and exit(2)
- void **log_error_and_die** (const char *fmt,...)
send ERR level message to the multiplexer and exit(2)
- int **log_palert** (const char *fmt,...)
send ALERT level message to the multiplexer and append strerror(errno)
- int **log_perror** (const char *fmt,...)
send ERR level message to the multiplexer and append strerror(errno)
- int **log_pwarn** (const char *fmt,...)
send WARNING level message to the multiplexer and append strerror(errno)
- int **log_pnotice** (const char *fmt,...)
send NOTICE level message to the multiplexer and append strerror(errno)
- int **log_pinfo** (const char *fmt,...)
send INFO level message to the multiplexer and append strerror(errno)
- int **log_pdebug** (const char *fmt,...)
send DEBUG level message to the multiplexer and append strerror(errno)
- int **log_ptrace** (const char *fmt,...)
send TRACE level message to the multiplexer and append strerror(errno)
- void **log_palert_and_die** (const char *fmt,...)
send ALERT level message to the multiplexer, append strerror(errno) and exit(2)
- void **log_perror_and_die** (const char *fmt,...)
send ERR level message to the multiplexer, append strerror(errno) and exit(2)

- void **log_close** (void)
close connection to logging system

8.37 log/log_close.c File Reference

```
#include <unistd.h>
```

```
#include <syslog.h>
```

```
#include "mem.h"
```

```
#include "log.h"
```

Include dependency graph for log_close.c:

Functions

- void **log_close** (void)
close connection to logging system

Variables

- **log_options_t** * **_log_options**

8.37.1 Variable Documentation

8.37.1.1 log_options_t* _log_options

Definition at line 25 of file log_init.c.

Referenced by log_close(), and log_init().

8.38 log/log_init.c File Reference

```
#include <unistd.h>
#include <syslog.h>
#include <sys/stat.h>
#include "log.h"
#include "mem.h"
#include "str.h"
```

Include dependency graph for log_init.c:

Defines

- `#define MASK_PRIO(p) (1 << (p))`

Functions

- `void log_init (log_options_t *options)`
initialize log message mutliplxer

Variables

- `log_options_t * _log_options = 0`

8.38.1 Define Documentation

8.38.1.1 `#define MASK_PRIO(p) (1 << (p))`

Definition at line 27 of file log_init.c.

8.38.2 Variable Documentation

8.38.2.1 `log_options_t* _log_options = 0`

Definition at line 25 of file log_init.c.

8.39 log/log_internal.c File Reference

```
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <stdarg.h>
#include <syslog.h>
#include <string.h>
#include <time.h>
#include "log.h"
#include "mem.h"
#include "printf.h"
#include "str.h"
```

Include dependency graph for log_internal.c:

Defines

- `#define LOGFUNC(name, level, rc)`
- `#define LOGFUNC DIE(name, level)`
- `#define LOGPFUNC(name, level, rc)`
- `#define LOGPFUNC DIE(name, level)`

Functions

- `int log_traceme (const char *file, const char *func, int line)`
send TRACE level message to the multiplexer

Variables

- `log_options_t * _log_options`

8.39.1 Define Documentation

8.39.1.1 `#define LOGFUNC(name, level, rc)`

Value:

```
int log_ ## name (const char *fmt, ...) { \
    va_list ap; va_start(ap, fmt); \
    log_internal(level, 0, fmt, ap); \
    va_end(ap); \
    return rc; \
}
```

Definition at line 128 of file log_internal.c.

8.39.1.2 `#define LOGFUNCDIE(name, level)`

Value:

```
void log_ ## name ## _and_die(const char *fmt, ...) { \
    va_list ap; va_start(ap, fmt); \
    log_internal(level, 0, fmt, ap); \
    va_end(ap); \
    exit(EXIT_FAILURE); \
}
```

Definition at line 144 of file log_internal.c.

8.39.1.3 `#define LOGPFUNC(name, level, rc)`

Value:

```
int log_p ## name (const char *fmt, ...) { \
    errno_orig = errno; \
    va_list ap; va_start(ap, fmt); \
    log_internal(level, 1, fmt, ap); \
    va_end(ap); \
    return rc; \
}
```

Definition at line 155 of file log_internal.c.

8.39.1.4 `#define LOGPFUNCDIE(name, level)`

Value:

```
void log_p ## name ## _and_die(const char *fmt, ...) { \
    errno_orig = errno; \
    va_list ap; va_start(ap, fmt); \
    log_internal(level, 1, fmt, ap); \
    va_end(ap); \
    exit(EXIT_FAILURE); \
}
```

Definition at line 172 of file log_internal.c.

8.39.2 Variable Documentation

8.39.2.1 `log_options_t* _log_options`

Definition at line 25 of file log_init.c.

8.40 mem.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- void * **mem_alloc** (int n)
allocate memory
- void * **mem_cpy** (void *s1, const void *s2, int c, int n)
copy memory block until character is found
- void * **mem_chr** (const void *s, int c, int n)
find character in memory block
- int **mem_cmp** (const void *s1, const void *s2, int n)
compare two memory regions
- void * **mem_cpy** (void *s1, const void *s2, int n)
copy memory block
- void * **mem_dup** (const void *s, int n)
duplicate a memory block
- void **mem_free** (void *s)
free memory
- void **mem_freeall** (void)
free all memory
- int **mem_idx** (const void *s, int c, int n)
find character in memory block
- void * **mem_realloc** (void *s, int n)
reallocate memory
- void * **mem_set** (void *s, int c, int n)
fill memory block with character

8.41 mem/mem__alloc.c File Reference

```
#include <stdlib.h>
#include "mem.h"
#include "mem_internal.h"
```

Include dependency graph for mem__alloc.c:

Functions

- void * **mem__alloc** (int n)
allocate memory

Variables

- **_mem_pool_t** * **_mem_pool** = 0

8.41.1 Variable Documentation

8.41.1.1 _mem_pool_t* _mem_pool = 0

Definition at line 22 of file mem__alloc.c.

Referenced by mem__alloc(), mem__free(), mem__freeall(), and mem__realloc().

8.42 mem/mem_ccpy.c File Reference

```
#include "mem.h"
```

Include dependency graph for mem_ccpy.c:

Functions

- void * **mem_ccpy** (void *s1, const void *s2, int c, int n)
copy memory block until character is found

8.43 mem/mem__chr.c File Reference

```
#include "mem.h"
```

Include dependency graph for mem__chr.c:

Functions

- void * **mem__chr** (const void *s, int c, int n)
find character in memory block

8.44 mem/mem_cmp.c File Reference

```
#include "mem.h"
```

Include dependency graph for mem_cmp.c:

Functions

- int **mem_cmp** (const void *s1, const void *s2, int n)
compare two memory regions

8.45 mem/mem_cpy.c File Reference

```
#include "mem.h"
```

Include dependency graph for mem_cpy.c:

Functions

- void * **mem_cpy** (void *s1, const void *s2, int n)
copy memory block

8.46 mem/mem_dup.c File Reference

```
#include "mem.h"
```

Include dependency graph for mem_dup.c:

Functions

- void * **mem_dup** (const void *s, int n)
duplicate a memory block

8.47 mem/mem__free.c File Reference

```
#include <stdlib.h>
```

```
#include <errno.h>
```

```
#include "mem.h"
```

```
#include "mem_internal.h"
```

Include dependency graph for mem_free.c:

Functions

- void **mem__free** (void *s)
free memory

8.48 mem/mem__freeall.c File Reference

```
#include <stdlib.h>
```

```
#include "mem.h"
```

```
#include "mem_internal.h"
```

Include dependency graph for mem__freeall.c:

Functions

- void **mem__freeall** (void)
free all memory

8.49 mem/mem_idx.c File Reference

```
#include "mem.h"
```

Include dependency graph for mem_idx.c:

Functions

- int **mem_idx** (const void *s, int c, int n)
find character in memory block

8.50 mem/mem_internal.h File Reference

```
#include "list.h"
```

Include dependency graph for mem_internal.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct **_mem_pool_t**

Defines

- #define **mem_for_each**(pool, p) list_for_each_entry(p, &(pool → list), list)
- #define **mem_for_each_safe**(pool, p, tmp) list_for_each_entry_safe(p, tmp, &(pool → list), list)

Variables

- **_mem_pool_t * _mem_pool**

8.50.1 Define Documentation

8.50.1.1 #define mem_for_each(pool, p) list_for_each_entry(p, &(pool → list), list)

Definition at line 30 of file mem_internal.h.

Referenced by mem_free(), and mem_realloc().

8.50.1.2 #define mem_for_each_safe(pool, p, tmp) list_for_each_entry_safe(p, tmp, &(pool → list), list)

Definition at line 31 of file mem_internal.h.

Referenced by mem_freeall().

8.50.2 Variable Documentation

8.50.2.1 _mem_pool_t * _mem_pool

Definition at line 22 of file mem_alloc.c.

Referenced by mem_alloc(), mem_free(), mem_freeall(), and mem_realloc().

8.51 mem/mem__realloc.c File Reference

```
#include <stdlib.h>
```

```
#include <errno.h>
```

```
#include "mem.h"
```

```
#include "mem_internal.h"
```

Include dependency graph for mem__realloc.c:

Functions

- void * **mem__realloc** (void *s, int n)
reallocate memory

8.52 mem/mem_set.c File Reference

```
#include "mem.h"
```

Include dependency graph for mem_set.c:

Functions

- void * **mem_set** (void *s, int c, int n)
fill memory block with character

8.53 misc.h File Reference

```
#include <sys/types.h>
```

Include dependency graph for misc.h:

This graph shows which files directly or indirectly include this file:

Functions

- **int ispath** (const char *path)
check if given path exists
- **int isdir** (const char *path)
check if given path is a directory
- **int isfile** (const char *path)
check if given path is a regular file
- **int islink** (const char *path)
check if given path is a symbolic link
- **int ismount** (const char *path)
check if given path is a top-level mount point
- **int mkdirnamep** (const char *path, mode_t mode)
recursive mkdir(2) with dirname(3)
- **int mkdirp** (const char *path, mode_t mode)
recursive mkdir(2)
- **int rmlink** (const char *path)
recursive unlink(2) and rmdir(2)
- **char * readsymlink** (const char *path)
read contents of symlink
- **int copy_file** (int srcfd, int dstfd)
copy a file

8.54 misc/copy_file.c File Reference

```
#include <unistd.h>
#include <errno.h>
#include <fcntl.h>
#include <signal.h>
#include <setjmp.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include "misc.h"
#include "open.h"
```

Include dependency graph for copy_file.c:

Defines

- `#define CHUNKSIZE (16*1024*1024)`

Functions

- `int copy_file (int srcfd, int dstfd)`
copy a file

8.54.1 Define Documentation

8.54.1.1 `#define CHUNKSIZE (16*1024*1024)`

Definition at line 64 of file copy_file.c.

Referenced by `copy_file()`, `readsymlink()`, `str_readfile()`, and `str_readline()`.

8.55 misc/isdir.c File Reference

```
#include <sys/stat.h>
```

```
#include "misc.h"
```

Include dependency graph for isdir.c:

Functions

- int **isdir** (const char *path)
check if given path is a directory

8.56 misc/isfile.c File Reference

```
#include <sys/stat.h>
```

```
#include "misc.h"
```

Include dependency graph for isfile.c:

Functions

- int **isfile** (const char *path)
check if given path is a regular file

8.57 misc/islink.c File Reference

```
#include <sys/stat.h>
```

```
#include "misc.h"
```

Include dependency graph for islink.c:

Functions

- int **islink** (const char *path)
check if given path is a symbolic link

8.58 misc/ismount.c File Reference

```
#include <sys/stat.h>
```

```
#include "mem.h"
```

```
#include "misc.h"
```

```
#include "str.h"
```

Include dependency graph for ismount.c:

Functions

- int **ismount** (const char *path)
check if given path is a top-level mount point

8.59 misc/ispath.c File Reference

```
#include <sys/stat.h>
```

```
#include "misc.h"
```

Include dependency graph for ispath.c:

Functions

- int **ispath** (const char *path)
check if given path exists

8.60 misc/mkdirnamep.c File Reference

```
#include <errno.h>
```

```
#include "mem.h"
```

```
#include "misc.h"
```

```
#include "str.h"
```

Include dependency graph for mkdirnamep.c:

Functions

- int **mkdirnamep** (const char *path, mode_t mode)
recursive mkdir(2) with dirname(3)

8.61 misc/mkdirp.c File Reference

```
#include <unistd.h>
#include <errno.h>
#include <sys/stat.h>
#include "misc.h"
#include "open.h"
#include "str.h"
#include "strtok.h"
```

Include dependency graph for mkdirp.c:

Functions

- int **mkdirp** (const char *path, mode_t mode)
recursive mkdir(2)

8.62 misc/readsymlink.c File Reference

```
#include <unistd.h>
```

```
#include "mem.h"
```

```
#include "misc.h"
```

Include dependency graph for readsymlink.c:

Defines

- `#define CHUNKSIZE 128`

Functions

- `char * readsymlink (const char *path)`
read contents of symlink

8.62.1 Define Documentation

8.62.1.1 `#define CHUNKSIZE 128`

Definition at line 22 of file readsymlink.c.

8.63 misc/runlink.c File Reference

```
#include <unistd.h>
#include <errno.h>
#include <dirent.h>
#include <sys/stat.h>
#include "mem.h"
#include "misc.h"
#include "printf.h"
```

Include dependency graph for runlink.c:

Functions

- int **runlink** (const char *path)
recursive unlink(2) and rmdir(2)

8.64 open.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- int **open_append** (const char *filename)
open file in append mode
- int **open_excl** (const char *filename)
open file exclusively
- int **open_read** (const char *filename)
open file for reading
- int **open_rw** (const char *filename)
open file for reading and writing
- int **open_trunc** (const char *filename)
open and truncate file for reading and writing
- int **open_write** (const char *filename)
open file for writing

8.65 open/open_append.c File Reference

```
#include <fcntl.h>
```

```
#include "open.h"
```

Include dependency graph for open_append.c:

Functions

- int **open_append** (const char *filename)
open file in append mode

8.66 open/open_excl.c File Reference

```
#include <fcntl.h>
```

```
#include "open.h"
```

Include dependency graph for open_excl.c:

Functions

- int **open_excl** (const char *filename)
open file exclusively

8.67 open/open_read.c File Reference

```
#include <fcntl.h>
```

```
#include "open.h"
```

Include dependency graph for open_read.c:

Functions

- int **open_read** (const char *filename)
open file for reading

8.68 open/open_rw.c File Reference

```
#include <fcntl.h>
```

```
#include "open.h"
```

Include dependency graph for open_rw.c:

Functions

- int **open_rw** (const char *filename)
open file for reading and writing

8.69 open/open_trunc.c File Reference

```
#include <fcntl.h>
```

```
#include "open.h"
```

Include dependency graph for open_trunc.c:

Functions

- int **open_trunc** (const char *filename)
open and truncate file for reading and writing

8.70 open/open_write.c File Reference

```
#include <fcntl.h>
```

```
#include "open.h"
```

Include dependency graph for open_write.c:

Functions

- int **open_write** (const char *filename)
open file for writing

8.71 printf.h File Reference

`#include <stdarg.h>`

Include dependency graph for printf.h:

This graph shows which files directly or indirectly include this file:

Functions

- `int __lucid__vsnprintf (char *str, int size, const char *fmt, va_list ap)`
write conversion to string using va_list
- `int __lucid__snprintf (char *str, int size, const char *fmt,...)`
write conversion to string using variable number of arguments
- `int __lucid__vasprintf (char **ptr, const char *fmt, va_list ap)`
write conversion to allocated string using va_list
- `int __lucid__asprintf (char **ptr, const char *fmt,...)`
write conversion to allocated string using variable number of arguments
- `int __lucid__vdprintf (int fd, const char *fmt, va_list ap)`
write conversion to file descriptor using va_list
- `int __lucid__dprintf (int fd, const char *fmt,...)`
write conversion to file descriptor using variable number of arguments
- `int __lucid__vprintf (const char *fmt, va_list ap)`
write conversion to stdout using va_list
- `int __lucid__printf (const char *fmt,...)`
write conversion to stdout using variable number of arguments

8.72 printf/asprintf.c File Reference

`#include "printf.h"`

Include dependency graph for asprintf.c:

Functions

- `int __lucid_asprintf (char **ptr, const char *fmt,...)`
write conversion to allocated string using variable number of arguments

8.73 printf/dprintf.c File Reference

```
#include "printf.h"
```

Include dependency graph for dprintf.c:

Functions

- `int __lucid_dprintf (int fd, const char *fmt,...)`
write conversion to file descriptor using variable number of arguments

8.74 printf/printf.c File Reference

```
#include "printf.h"
```

Include dependency graph for printf.c:

Functions

- `int __lucid_printf (const char *fmt,...)`
write conversion to stdout using variable number of arguments

8.75 printf/snprintf.c File Reference

#include "printf.h"

Include dependency graph for snprintf.c:

Functions

- int **__lucid_snprintf** (char *str, int size, const char *fmt,...)
write conversion to string using variable number of arguments

8.76 printf/vasprintf.c File Reference

```
#include "mem.h"
```

```
#include "printf.h"
```

Include dependency graph for vasprintf.c:

Functions

- int **__lucid_vasprintf** (char **ptr, const char *fmt, va_list ap)
write conversion to allocated string using va_list

8.77 printf/vdprintf.c File Reference

```
#include <unistd.h>
```

```
#include "mem.h"
```

```
#include "printf.h"
```

Include dependency graph for vdprintf.c:

Functions

- `int __lucid_vdprintf (int fd, const char *fmt, va_list ap)`
write conversion to file descriptor using va_list

8.78 printf/vprintf.c File Reference

```
#include "printf.h"
```

Include dependency graph for vprintf.c:

Functions

- `int __lucid_vprintf (const char *fmt, va_list ap)`
write conversion to stdout using va_list

8.79 printf/vsnprintf.c File Reference

```
#include "mem.h"
```

```
#include "printf.h"
```

```
#include "str.h"
```

Include dependency graph for vsnprintf.c:

Data Structures

- struct `__printf_t`

Defines

- `#define PFR_MIN PFR_CHAR`
- `#define PFR_MAX PFR_LLONG`
- `#define EMIT(C) { if (idx < size - 1) { *str++ = C; } idx++; }`

Enumerations

- enum `__printf_flags` {
 PFL_ALT = 0x01, **PFL_ZERO** = 0x02, **PFL_LEFT** = 0x04, **PFL_BLANK** = 0x08,
 PFL_SIGN = 0x10, **PFL_UPPER** = 0x20, **PFL_SIGNED** = 0x40 }
- enum `__printf_rank` {
 PFR_CHAR, **PFR_SHORT**, **PFR_INT**, **PFR_LONG**,
 PFR_LLONG }
- enum `__printf_state` {
 PFS_NORMAL, **PFS_FLAGS**, **PFS_WIDTH**, **PFS_PREC**,
 PFS_MOD, **PFS_CONV** }

Functions

- int `__lucid_vsnprintf` (char *str, int size, const char *fmt, va_list _ap)
 write conversion to string using va_list

8.79.1 Define Documentation

8.79.1.1 `#define PFR_MIN PFR_CHAR`

Definition at line 39 of file vsnprintf.c.

Referenced by `__lucid_vsnprintf()`.

8.79.1.2 `#define PFR_MAX PFR_LLONG`

Definition at line 40 of file vsnprintf.c.

Referenced by `_lucid_vsnprintf()`.

8.79.1.3 `#define EMIT(C) { if (idx < size - 1) { *str++ = C; } idx++; }`

Definition at line 59 of file vsnprintf.c.

Referenced by `_lucid_vsnprintf()`.

8.79.2 Enumeration Type Documentation

8.79.2.1 `enum __printf_flags`

Enumerator:

PFL_ALT
PFL_ZERO
PFL_LEFT
PFL_BLANK
PFL_SIGN
PFL_UPPER
PFL_SIGNED

Definition at line 21 of file vsnprintf.c.

```
21          {  
22      PFL_ALT    = 0x01,  
23      PFL_ZERO   = 0x02,  
24      PFL_LEFT   = 0x04,  
25      PFL_BLANK  = 0x08,  
26      PFL_SIGN   = 0x10,  
27      PFL_UPPER  = 0x20,  
28      PFL_SIGNED = 0x40,  
29  };
```

8.79.2.2 `enum __printf_rank`

Enumerator:

PFR_CHAR
PFR_SHORT
PFR_INT
PFR_LONG
PFR_LLONG

Definition at line 31 of file vsnprintf.c.


```
31         {
32             PFR_CHAR,
33             PFR_SHORT,
34             PFR_INT,
35             PFR_LONG,
36             PFR_LLONG,
37     };
```

8.79.2.3 enum __printf_state

Enumerator:

PFS_NORMAL
PFS_FLAGS
PFS_WIDTH
PFS_PREC
PFS_MOD
PFS_CONV

Definition at line 42 of file vsnprintf.c.

```
42         {
43             PFS_NORMAL,
44             PFS_FLAGS,
45             PFS_WIDTH,
46             PFS_PREC,
47             PFS_MOD,
48             PFS_CONV,
49     };
```

8.80 scanf.h File Reference

```
#include <stdarg.h>
```

Include dependency graph for scanf.h:

This graph shows which files directly or indirectly include this file:

Functions

- int **__lucid_vsscanf** (const char *str, const char *fmt, va_list ap)
read conversion from string using va_list
- int **__lucid_sscanf** (const char *str, const char *fmt,...)
read conversion from string using variable number of arguments

8.81 scanf/sscanf.c File Reference

`#include "scanf.h"`

Include dependency graph for sscanf.c:

Functions

- `int __lucid_sscanf` (const char *str, const char *fmt,...)
read conversion from string using variable number of arguments

8.82 scanf/vsscanf.c File Reference

```
#include "char.h"
#include "scanf.h"
#include "str.h"
```

Include dependency graph for vsscanf.c:

Data Structures

- struct `__scanf_t`

Defines

- `#define SFR_MIN SFR_CHAR`
- `#define SFR_MAX SFR_LLONG`

Enumerations

- enum `__scanf_flags` { `SFL_NOOP` = 0x01, `SFL_WIDTH` = 0x02 }
- enum `__scanf_rank` {
 `SFR_CHAR`, `SFR_SHORT`, `SFR_INT`, `SFR_LONG`,
 `SFR_LLONG` }
- enum `__scanf_state` {
 `SFS_NORMAL`, `SFS_FLAGS`, `SFS_WIDTH`, `SFS_MOD`,
 `SFS_CONV`, `SFS_STORE`, `SFS_EOF`, `SFS_ERR` }

Functions

- int `__lucid_vsscanf` (const char *str, const char *fmt, va_list _ap)
 read conversion from string using va_list

8.82.1 Define Documentation

8.82.1.1 `#define SFR_MIN SFR_CHAR`

Definition at line 34 of file vsscanf.c.

Referenced by `__lucid_vsscanf()`.

8.82.1.2 `#define SFR_MAX SFR_LLONG`

Definition at line 35 of file vsscanf.c.

Referenced by `__lucid_vsscanf()`.

8.82.2 Enumeration Type Documentation

8.82.2.1 enum __scanf_flags

Enumerator:

SFL_NOOP
SFL_WIDTH

Definition at line 21 of file vsscanf.c.

```
21          {  
22          SFL_NOOP = 0x01,  
23          SFL_WIDTH = 0x02,  
24  };
```

8.82.2.2 enum __scanf_rank

Enumerator:

SFR_CHAR
SFR_SHORT
SFR_INT
SFR_LONG
SFR_LLONG

Definition at line 26 of file vsscanf.c.

```
26          {  
27          SFR_CHAR,  
28          SFR_SHORT,  
29          SFR_INT,  
30          SFR_LONG,  
31          SFR_LLONG,  
32  };
```

8.82.2.3 enum __scanf_state

Enumerator:

SFS_NORMAL
SFS_FLAGS
SFS_WIDTH
SFS_MOD
SFS_CONV
SFS_STORE
SFS_EOF
SFS_ERR

Definition at line 37 of file vsscanf.c.

```
37         {
38         SFS_NORMAL,
39         SFS_FLAGS,
40         SFS_WIDTH,
41         SFS_MOD,
42         SFS_CONV,
43         SFS_STORE,
44         SFS_EOF,
45         SFS_ERR,
46     };
```

8.83 str.h File Reference

This graph shows which files directly or indirectly include this file:

Defines

- `#define CC__ALNUM (1 << 1)`
class for alpha-numerical characters
- `#define CC__ALPHA (1 << 2)`
class for upper- or lower-case characters
- `#define CC__ASCII (1 << 3)`
class for ASCII characters
- `#define CC__BLANK (1 << 4)`
class for blank characters
- `#define CC__CNTRL (1 << 5)`
class for ASCII control characters
- `#define CC__DIGIT (1 << 6)`
class for digit characters
- `#define CC__GRAPH (1 << 7)`
class for graphable characters
- `#define CC__LOWER (1 << 8)`
class for lower-case characters
- `#define CC__PRINT (1 << 9)`
class for printable characters
- `#define CC__PUNCT (1 << 10)`
class for punctuation characters
- `#define CC__SPACE (1 << 11)`
class for white space characters
- `#define CC__UPPER (1 << 12)`
class for upper-case characters
- `#define CC__XDIGIT (1 << 13)`
class for hexadecimal characters
- `#define str__isempty(str) (!str || str__check(str, CC__BLANK))`
check if string is empty

- `#define str_isalnum(str) str_check(str, CC_ALNUM)`
check string for alpha-numerical characters
- `#define str_isalpha(str) str_check(str, CC_ALPHA)`
check string for upper- or lower-case characters
- `#define str_iscii(str) str_check(str, CC_ASCII)`
check string for ASCII characters
- `#define str_isdigit(str) str_check(str, CC_DIGIT)`
check string for digit characters
- `#define str_isgraph(str) str_check(str, CC_GRAPH)`
check string for graphable characters
- `#define str_islower(str) str_check(str, CC_LOWER)`
check string for lower-case characters
- `#define str_isprint(str) str_check(str, CC_PRINT)`
check string for printable characters
- `#define str_isupper(str) str_check(str, CC_UPPER)`
check string for upper-case characters
- `#define str_isxdigit(str) str_check(str, CC_XDIGIT)`
check string for hexadecimal characters
- `#define CHUNKSIZE 4096`

Functions

- `int str_check (const char *str, int allowed)`
check string against classes of allowed characters
- `int str_cmp (const char *str1, const char *str2)`
compare two strings
- `int str_cmpn (const char *str1, const char *str2, int n)`
compare two strings
- `int str_equal (const char *str1, const char *str2)`
compare two strings
- `char * str_cpy (char *dst, const char *src)`
copy a string
- `char * str_cpyn (char *dst, const char *src, int n)`
copy a string
- `char * str_dup (const char *str)`

duplicate a string

- char * **str_chr** (const char *str, int c, int n)
scan string for character
- char * **str_rchr** (const char *str, int c, int n)
scan string for character beginning at the end
- char * **str_str** (const char *str, const char *needle)
locate a substring
- int **str_len** (const char *str)
calculate the length of a string
- char * **str_path_dirname** (const char *path)
parse directory component
- char * **str_path_basename** (const char *path)
parse basename component
- char * **str_path_concat** (const char *dirname, const char *basename)
concatenate dirname and basename
- int **str_path_isabs** (const char *str)
check if path is absolute and contains no dot entries or ungraphable characters
- int **str_path_isdot** (const char *str)
check if given path contains . or .. entries
- char * **str_tolower** (char *str)
convert string to lower-case
- char * **str_toupper** (char *str)
convert string to upper-case
- int **str_toumax** (const char *str, unsigned long long int *val, int base, int n)
convert string to integer
- int **str_readline** (int fd, char **str)
read a line of input
- int **str_readfile** (int fd, char **str)
read until end of file
- int **str_read** (int fd, char **str, int len)
read exact number of bytes

8.84 str/str_check.c File Reference

```
#include "char.h"
```

```
#include "str.h"
```

Include dependency graph for str_check.c:

Functions

- int **str_check** (const char *str, int allowed)
check string against classes of allowed characters

8.85 str/str_chr.c File Reference

#include "str.h"

Include dependency graph for str_chr.c:

Functions

- char * **str_chr** (const char *str, int c, int n)
scan string for character

8.86 str/str_cmp.c File Reference

#include "str.h"

Include dependency graph for str_cmp.c:

Functions

- int **str_cmp** (const char *str1, const char *str2)
compare two strings

8.87 str/str_cmpn.c File Reference

```
#include "str.h"
```

Include dependency graph for str_cmpn.c:

Functions

- int **str_cmpn** (const char *str1, const char *str2, int n)
compare two strings

8.88 str/str_cpy.c File Reference

```
#include "mem.h"
```

```
#include "str.h"
```

Include dependency graph for str_cpy.c:

Functions

- char * **str_cpy** (char *dst, const char *src)
copy a string

8.89 str/str_cpyn.c File Reference

```
#include "mem.h"
```

```
#include "str.h"
```

Include dependency graph for str_cpyn.c:

Functions

- char * **str_cpyn** (char *dst, const char *src, int n)
copy a string

8.90 str/str_dup.c File Reference

```
#include "mem.h"
```

```
#include "str.h"
```

Include dependency graph for str_dup.c:

Functions

- char * **str_dup** (const char *str)
duplicate a string

8.91 str/str_equal.c File Reference

```
#include "str.h"
```

Include dependency graph for str_equal.c:

Functions

- int **str_equal** (const char *str1, const char *str2)
compare two strings

8.92 str/str_len.c File Reference

`#include "str.h"`

Include dependency graph for str_len.c:

Functions

- int **str_len** (const char *str)
calculate the length of a string

8.93 str/str_path_basename.c File Reference

```
#include "mem.h"
```

```
#include "str.h"
```

Include dependency graph for str_path_basename.c:

Functions

- char * **str_path_basename** (const char *path)
parse basename component

8.94 str/str_path_concat.c File Reference

```
#include "mem.h"
```

```
#include "printf.h"
```

```
#include "str.h"
```

Include dependency graph for str_path_concat.c:

Functions

- char * **str_path_concat** (const char *dirname, const char *basename)
concatenate dirname and basename

8.95 str/str_path_dirname.c File Reference

```
#include "mem.h"
```

```
#include "str.h"
```

Include dependency graph for str_path_dirname.c:

Functions

- char * **str_path_dirname** (const char *path)
parse directory component

8.96 str/str_path_isabs.c File Reference

```
#include "str.h"
```

```
#include "strtok.h"
```

Include dependency graph for str_path_isabs.c:

Functions

- int **str_path_isabs** (const char *str)
check if path is absolute and contains no dot entries or ungraphable characters

8.97 str/str_path_isdot.c File Reference

```
#include "str.h"
```

```
#include "strtok.h"
```

Include dependency graph for str_path_isdot.c:

Functions

- int **str_path_isdot** (const char *str)
check if given path contains . or .. entries

8.98 str/str_rchr.c File Reference

#include "str.h"

Include dependency graph for str_rchr.c:

Functions

- char * **str_rchr** (const char *str, int c, int n)
scan string for character beginning at the end

8.99 str/str_read.c File Reference

```
#include <unistd.h>
```

```
#include "mem.h"
```

```
#include "str.h"
```

Include dependency graph for str_read.c:

Functions

- int **str_read** (int fd, char **str, int len)
read exact number of bytes

8.100 str/str_readfile.c File Reference

```
#include <unistd.h>
```

```
#include "mem.h"
```

```
#include "str.h"
```

Include dependency graph for str_readfile.c:

Functions

- int **str_readfile** (int fd, char **str)
read until end of file

8.101 str/str_readline.c File Reference

```
#include <unistd.h>
```

```
#include "mem.h"
```

```
#include "str.h"
```

Include dependency graph for str_readline.c:

Functions

- int **str_readline** (int fd, char **line)
read a line of input

8.102 str/str_str.c File Reference

```
#include "mem.h"
```

```
#include "str.h"
```

Include dependency graph for str_str.c:

Functions

- char * **str_str** (const char *str, const char *needle)
locate a substring

8.103 str/str_tolower.c File Reference

```
#include "char.h"
```

```
#include "str.h"
```

Include dependency graph for str_tolower.c:

Functions

- char * **str_tolower** (char *str)
convert string to lower-case

8.104 str/str_toumax.c File Reference

```
#include "char.h"
```

```
#include "str.h"
```

Include dependency graph for str_toumax.c:

Functions

- int **str_toumax** (const char *str, unsigned long long int *val, int base, int n)
convert string to integer

8.105 str/str_toupper.c File Reference

```
#include "char.h"
```

```
#include "str.h"
```

Include dependency graph for str_toupper.c:

Functions

- char * **str_toupper** (char *str)
convert string to upper-case

8.106 stralloc.h File Reference

```
#include <sys/types.h>
```

Include dependency graph for stralloc.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct **stralloc__t**
dynamic string allocator tracking data

Functions

- void **stralloc__init** (stralloc__t *sa)
initialize dynamic string allocator
- void **stralloc__zero** (stralloc__t *sa)
truncate string length to zero
- int **stralloc__ready** (stralloc__t *sa, size_t len)
ensure that enough memory has been allocated
- int **stralloc__readyplus** (stralloc__t *sa, size_t len)
ensure that enough memory has been allocated
- char * **stralloc__finalize** (stralloc__t *sa)
finalize dynamic string in new buffer
- void **stralloc__free** (stralloc__t *sa)
deallocate all memory
- int **stralloc__copyb** (stralloc__t *dst, const char *src, size_t len)
copy a static string to a dynamic one
- int **stralloc__copys** (stralloc__t *dst, const char *src)
copy a static string to a dynamic one
- int **stralloc__copy** (stralloc__t *dst, const stralloc__t *src)
copy one dynamic string to another
- int **stralloc__catb** (stralloc__t *dst, const char *src, size_t len)
concatenate a dynamic string and a static one

- int **stralloc_catf** (**stralloc_t** *dst, const char *fmt,...)
concatenate a dynamic string and a static one using formatted conversion
- int **stralloc_catm** (**stralloc_t** *dst,...)
concatenate a dynamic string and multiple static ones
- int **stralloc_cats** (**stralloc_t** *dst, const char *src)
concatenate a dynamic string and a static one
- int **stralloc_cat** (**stralloc_t** *dst, const **stralloc_t** *src)
concatenate two dynamic strings
- int **stralloc_cmp** (const **stralloc_t** *a, const **stralloc_t** *b)
compare two dynamic strings

8.107 stralloc/stralloc_cat.c File Reference

#include "stralloc.h"

Include dependency graph for stralloc_cat.c:

Functions

- int **stralloc_cat** (**stralloc_t** *dst, const **stralloc_t** *src)
concatenate two dynamic strings

8.108 stralloc/stralloc_catb.c File Reference

```
#include "mem.h"
```

```
#include "stralloc.h"
```

Include dependency graph for stralloc_catb.c:

Functions

- int **stralloc_catb** (**stralloc_t** *dst, const char *src, size_t len)
concatenate a dynamic string and a static one

8.109 stralloc/stralloc_catf.c File Reference

```
#include <stdarg.h>
```

```
#include "mem.h"
```

```
#include "printf.h"
```

```
#include "stralloc.h"
```

Include dependency graph for stralloc_catf.c:

Functions

- int **stralloc_catf**(stralloc_t *dst, const char *fmt,...)
concatenate a dynamic string and a static one using formatted conversion

8.110 stralloc/stralloc_catm.c File Reference

```
#include <stdarg.h>
```

```
#include "stralloc.h"
```

Include dependency graph for stralloc_catm.c:

Functions

- int **stralloc_catm** (stralloc_t *dst,...)
concatenate a dynamic string and multiple static ones

8.111 stralloc/stralloc_cats.c File Reference

```
#include "str.h"
```

```
#include "stralloc.h"
```

Include dependency graph for stralloc_cats.c:

Functions

- int **stralloc_cats** (**stralloc_t** *dst, const char *src)
concatenate a dynamic string and a static one

8.112 stralloc/stralloc__cmp.c File Reference

```
#include "stralloc.h"
```

Include dependency graph for stralloc__cmp.c:

Functions

- int **stralloc__cmp** (const **stralloc__t** *a, const **stralloc__t** *b)
compare two dynamic strings

8.113 stralloc/stralloc__copy.c File Reference

```
#include "stralloc.h"
```

Include dependency graph for stralloc__copy.c:

Functions

- int **stralloc__copy** (stralloc__t *dst, const stralloc__t *src)
copy one dynamic string to another

8.114 stralloc/stralloc__copyb.c File Reference

```
#include "mem.h"
```

```
#include "stralloc.h"
```

Include dependency graph for stralloc__copyb.c:

Functions

- int **stralloc__copyb**(stralloc__t *dst, const char *src, size_t len)
copy a static string to a dynamic one

8.115 stralloc/stralloc_copys.c File Reference

```
#include "str.h"
```

```
#include "stralloc.h"
```

Include dependency graph for stralloc_copys.c:

Functions

- int **stralloc_copys** (**stralloc_t** *dst, const char *src)
copy a static string to a dynamic one

8.116 stralloc/stralloc__finalize.c File Reference

```
#include "mem.h"
```

```
#include "stralloc.h"
```

Include dependency graph for stralloc__finalize.c:

Functions

- char * **stralloc__finalize** (stralloc__t *sa)
finalize dynamic string in new buffer

8.117 stralloc/stralloc_free.c File Reference

```
#include "mem.h"
```

```
#include "stralloc.h"
```

Include dependency graph for stralloc_free.c:

Functions

- void **stralloc_free** (stralloc_t *sa)
deallocate all memory

8.118 stralloc/stralloc__init.c File Reference

```
#include "stralloc.h"
```

Include dependency graph for stralloc__init.c:

Functions

- void **stralloc__init** (stralloc__t *sa)
initialize dynamic string allocator

8.119 stralloc/stralloc_ready.c File Reference

```
#include "mem.h"
```

```
#include "stralloc.h"
```

Include dependency graph for stralloc_ready.c:

Functions

- int **stralloc_ready** (**stralloc_t** *sa, size_t len)
ensure that enough memory has been allocated

8.120 stralloc/stralloc__readyplus.c File Reference

```
#include <errno.h>
```

```
#include "stralloc.h"
```

Include dependency graph for stralloc__readyplus.c:

Functions

- int **stralloc__readyplus** (stralloc__t *sa, size_t len)
ensure that enough memory has been allocated

8.121 stralloc/stralloc_zero.c File Reference

#include "stralloc.h"

Include dependency graph for stralloc_zero.c:

Functions

- void **stralloc_zero** (**stralloc_t** *sa)
truncate string length to zero

8.122 strtok.h File Reference

#include <lucid/list.h>

Include dependency graph for strtok.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct **strtok_t**

Defines

- #define **strtok_for_each**(st, p) list_for_each_entry(p, &(st → list), list)
iterate through tokens

Functions

- **strtok_t * strtok_init_argv** (strtok_t *st, char *argv[], int argc, int empty)
initialize string tokenizer from argument vector
- **strtok_t * strtok_init_str** (strtok_t *st, const char *str, char *delim, int empty)
initialize string tokenizer from character array
- void **strtok_free** (strtok_t *st)
deallocate string tokenizer
- int **strtok_count** (strtok_t *st)
count number of tokens
- int **strtok_append** (strtok_t *st, const char *token)
append a token
- void **strtok_delete** (strtok_t *st, const char *token)
delete one or more tokens
- char * **strtok_prev** (strtok_t **st)
Go to the previous token.
- char * **strtok_next** (strtok_t **st)
Go to the previous token.
- int **strtok_toargv** (strtok_t *st, char **argv)

convert string tokenizer to argument vector

- `int strtok_tostr (strtok_t *st, char **str, char *delim)`

convert string tokenizer to character array

8.123 strtok/strtok_append.c File Reference

```
#include "mem.h"
```

```
#include "str.h"
```

```
#include "strtok.h"
```

Include dependency graph for strtok_append.c:

Functions

- `int strtok_append (strtok_t *st, const char *token)`
append a token

8.124 strtok/strtok_count.c File Reference

```
#include "strtok.h"
```

Include dependency graph for strtok_count.c:

Functions

- int **strtok_count** (strtok_t *st)
count number of tokens

8.125 strtok/strtok_delete.c File Reference

```
#include "mem.h"
```

```
#include "str.h"
```

```
#include "strtok.h"
```

Include dependency graph for strtok_delete.c:

Functions

- void **strtok_delete** (**strtok_t** *st, const char *token)
delete one or more tokens

8.126 strtok/strtok_free.c File Reference

```
#include <errno.h>
```

```
#include "mem.h"
```

```
#include "strtok.h"
```

Include dependency graph for strtok_free.c:

Functions

- void **strtok_free** (strtok_t *st)
deallocate string tokenizer

8.127 strtok/strtok_init_argv.c File Reference

```
#include "mem.h"
```

```
#include "str.h"
```

```
#include "strtok.h"
```

Include dependency graph for strtok_init_argv.c:

Functions

- **strtok_t * strtok_init_argv** (strtok_t *st, char *argv[], int argc, int empty)
initialize string tokenizer from argument vector

8.128 strtok/strtok_init_str.c File Reference

```
#include "mem.h"
```

```
#include "str.h"
```

```
#include "strtok.h"
```

Include dependency graph for strtok_init_str.c:

Functions

- **strtok_t * strtok_init_str** (strtok_t *st, const char *str, char *delim, int empty)
initialize string tokenizer from character array

8.129 strtok/strtok_next.c File Reference

```
#include "strtok.h"
```

Include dependency graph for strtok_next.c:

Functions

- char * **strtok_next** (strtok_t **st)
Go to the previous token.

8.130 strtok/strtok_prev.c File Reference

```
#include "strtok.h"
```

Include dependency graph for strtok_prev.c:

Functions

- char * **strtok_prev** (strtok_t **st)
Go to the previous token.

8.131 strtok/strtok_toargv.c File Reference

```
#include "strtok.h"
```

Include dependency graph for strtok_toargv.c:

Functions

- int **strtok_toargv** (strtok_t *st, char **argv)
convert string tokenizer to argument vector

8.132 strtok/strtok_tostr.c File Reference

```
#include "str.h"
```

```
#include "stralloc.h"
```

```
#include "strtok.h"
```

Include dependency graph for strtok_tostr.c:

Functions

- int **strtok_tostr**(**strtok_t** *st, char **str, char *delim)
convert string tokenizer to character array

8.133 tcp.h File Reference

This graph shows which files directly or indirectly include this file:

Functions

- int **tcp_listen** (const char *ip, int port, int backlog)
listen for incoming connections
- int **tcp_connect** (const char *ip, int port)
connect to TCP socket

8.134 tcp/tcp_connect.c File Reference

```
#include <unistd.h>
#include <errno.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include "addr.h"
#include "mem.h"
#include "tcp.h"
```

Include dependency graph for tcp_connect.c:

Functions

- int **tcp_connect** (const char *ip, int port)
connect to TCP socket

8.135 tcp/tcp_listen.c File Reference

```
#include <unistd.h>
#include <errno.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include "addr.h"
#include "mem.h"
#include "tcp.h"
```

Include dependency graph for tcp_listen.c:

Functions

- int **tcp_listen** (const char *ip, int port, int backlog)
listen for incoming connections

8.136 whirlpool.h File Reference

`#include <stdint.h>`

Include dependency graph for whirlpool.h:

This graph shows which files directly or indirectly include this file:

Data Structures

- struct **whirlpool_t**
dynamic whirlpool state data

Defines

- `#define DIGESTBYTES 64`
number of bytes in the digest
- `#define DIGESTBITS (8*DIGESTBYTES)`
number of bits in the digest
- `#define WBLOCKBYTES 64`
number of bytes in the input buffer
- `#define WBLOCKBITS (8*WBLOCKBYTES)`
number of bits in the input buffer
- `#define LENGTHBYTES 32`
number of hashed bytes
- `#define LENGTHBITS (8*LENGTHBYTES)`
number of hashed bits

Functions

- void **whirlpool_transform** (**whirlpool_t** *const context)
internal transform routine
- void **whirlpool_init** (**whirlpool_t** *const context)
initialize whirlpool state context
- void **whirlpool_finalize** (**whirlpool_t** *const context, unsigned char *const result)
finalize whirlpool transformation

- void **whirlpool_add** (**whirlpool_t** *const context, const unsigned char *const src, unsigned long bits)
add bytes to the transform routine
- char * **whirlpool_digest** (const char *str)
create digest from string

8.137 whirlpool/whirlpool_add.c File Reference

```
#include "whirlpool.h"
```

Include dependency graph for whirlpool_add.c:

Functions

- void **whirlpool_add** (**whirlpool_t** *const context, const unsigned char *const src, unsigned long srcbits)
add bytes to the transform routine

8.138 whirlpool/whirlpool__digest.c File Reference

```
#include "mem.h"
```

```
#include "str.h"
```

```
#include "stralloc.h"
```

```
#include "whirlpool.h"
```

Include dependency graph for whirlpool__digest.c:

Functions

- char * **whirlpool__digest** (const char *str)
create digest from string

8.139 whirlpool/whirlpool_finalize.c File Reference

```
#include "mem.h"
```

```
#include "whirlpool.h"
```

Include dependency graph for whirlpool_finalize.c:

Functions

- void **whirlpool_finalize** (**whirlpool_t** *const context, unsigned char *const result)
finalize whirlpool transformation

8.140 whirlpool/whirlpool_init.c File Reference

```
#include "mem.h"
```

```
#include "whirlpool.h"
```

Include dependency graph for whirlpool_init.c:

Functions

- void **whirlpool_init** (**whirlpool_t** *const context)
initialize whirlpool state context

8.141 whirlpool/whirlpool_tables.h File Reference

```
#include <stdint.h>
```

Include dependency graph for whirlpool_tables.h:

This graph shows which files directly or indirectly include this file:

Defines

- `#define R 10`

8.141.1 Define Documentation

8.141.1.1 `#define R 10`

Definition at line 26 of file whirlpool_tables.h.

Referenced by whirlpool_transform().

8.142 whirlpool/whirlpool__transform.c File Reference

```
#include <stdint.h>
```

```
#include "whirlpool.h"
```

```
#include "whirlpool_tables.h"
```

Include dependency graph for whirlpool__transform.c:

Functions

- void **whirlpool__transform** (whirlpool__t *const context)
internal transform routine

Chapter 9

lucid Page Documentation

9.1 Examples

To be done ...

9.2 License

9.2.1 GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

9.2.1.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

9.2.1.2 TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means

either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be

distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right

claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Index

- __printf_flags
 - vsnprintf.c, 256
- __printf_rank
 - vsnprintf.c, 256
- __printf_state
 - vsnprintf.c, 257
- __printf_t, 157
 - f, 157
 - l, 157
 - p, 157
 - s, 157
 - w, 158
- __scanf_flags
 - vsscanf.c, 261
- __scanf_rank
 - vsscanf.c, 261
- __scanf_state
 - vsscanf.c, 261
- __scanf_t, 159
 - f, 159
 - l, 159
 - s, 159
 - w, 159
- _log_options
 - log_close.c, 212
 - log_init.c, 213
 - log_internal.c, 215
- _lucid_asprintf
 - printf, 93
- _lucid_dprintf
 - printf, 94
- _lucid_printf
 - printf, 95
- _lucid_snprintf
 - printf, 91
- _lucid_sscanf
 - scanf, 103
- _lucid_vasprintf
 - printf, 92
- _lucid_vdprintf
 - printf, 93
- _lucid_vprintf
 - printf, 94
- _lucid_vsnprintf
 - printf, 86
- _lucid_vsscanf
 - scanf, 98
- _mem_pool
 - mem_alloc.c, 217
 - mem_internal.h, 226
- _mem_pool_t, 160
 - len, 160
 - list, 160
 - mem, 160
- a
 - stralloc_t, 166
- addr
 - addr_from_str, 15
 - addr_hton, 14
 - addr_htos, 14
 - addr_ntoh, 14
 - addr_stoh, 15
 - addr_to_str, 16
- addr.h, 171
- addr/addr_from_str.c, 172
- addr/addr_hton.c, 173
- addr/addr_htos.c, 174
- addr/addr_ntoh.c, 175
- addr/addr_stoh.c, 176
- addr/addr_to_str.c, 177
- addr_from_str
 - addr, 15
- addr_hton
 - addr, 14
- addr_htos
 - addr, 14
- addr_ntoh
 - addr, 14
- addr_stoh
 - addr, 15
- addr_to_str
 - addr, 16
- bitmap
 - i2v32, 18
 - i2v64, 18
 - v2i32, 19
 - v2i64, 19
- Bitmap conversion, 18

- bitmap.h, 178
- bitmap/i2v32.c, 179
- bitmap/i2v64.c, 180
- bitmap/v2i32.c, 181
- bitmap/v2i64.c, 182
- bits
 - whirlpool_t, 168
- buf
 - whirlpool_t, 168
- CC_ALNUM
 - str, 108
- CC_ALPHA
 - str, 108
- CC_ASCII
 - str, 108
- CC_BLANK
 - str, 108
- CC_CNTRL
 - str, 109
- CC_DIGIT
 - str, 109
- CC_GRAPH
 - str, 109
- CC_LOWER
 - str, 109
- CC_PRINT
 - str, 109
- CC_PUNCT
 - str, 109
- CC_SPACE
 - str, 109
- CC_UPPER
 - str, 110
- CC_XDIGIT
 - str, 110
- char
 - char_isalnum, 24
 - char_isalpha, 24
 - char_isascii, 22
 - char_isblank, 22
 - char_iscntrl, 22
 - char_isdigit, 23
 - char_isgraph, 23
 - char_islower, 23
 - char_isprint, 23
 - char_ispunct, 24
 - char_isspace, 23
 - char_isupper, 23
 - char_isxdigit, 23
 - char_tolower, 24
 - char_toupper, 24
- char.h, 183
- char_isalnum
 - char, 24
- char_isalpha
 - char, 24
- char_isascii
 - char, 22
- char_isblank
 - char, 22
- char_iscntrl
 - char, 22
- char_isdigit
 - char, 23
- char_isgraph
 - char, 23
- char_islower
 - char, 23
- char_isprint
 - char, 23
- char_ispunct
 - char, 24
- char_isspace
 - char, 23
- char_isupper
 - char, 23
- char_isxdigit
 - char, 23
- char_tolower
 - char, 24
- char_toupper
 - char, 24
- Character classification and manipulation, 21
- chroot
 - chroot_fd, 26
 - chroot_mkdirp, 27
 - chroot_secure_chdir, 28
- chroot.h, 185
- chroot/chroot_fd.c, 186
- chroot/chroot_mkdirp.c, 187
- chroot/chroot_secure_chdir.c, 188
- chroot_fd
 - chroot, 26
- chroot_mkdirp
 - chroot, 27
- chroot_secure_chdir
 - chroot, 28
- CHUNKSIZE
 - copy_file.c, 230
 - readsymlink.c, 238
 - str, 111
- Command execution wrappers, 30
- container_of
 - list, 47
- copy_file
 - misc, 77
- copy_file.c

- CHUNKSIZE, 230
- Create or open files, 80
- DIGESTBITS
 - whirlpool, 147
- DIGESTBYTES
 - whirlpool, 147
- doxygen/examples.h, 189
- doxygen/license.h, 190
- doxygen/main.h, 191
- Dynamic string allocator, 127
- EMIT
 - vsnprintf.c, 256
- exec
 - exec_fork, 30
 - exec_fork_background, 32
 - exec_fork_pipe, 33
 - EXEC_MAX_ARGV, 30
 - exec_replace, 35
- exec.h, 192
- exec/exec_fork.c, 193
- exec/exec_fork_background.c, 194
- exec/exec_fork_pipe.c, 195
- exec/exec_replace.c, 196
- exec_fork
 - exec, 30
- exec_fork_background
 - exec, 32
- exec_fork_pipe
 - exec, 33
- EXEC_MAX_ARGV
 - exec, 30
- exec_replace
 - exec, 35
- f
 - __printf_t, 157
 - __scanf_t, 159
- Flag list conversion, 37
- flist
 - FLIST32_END, 39
 - flist32_from_str, 40
 - flist32_getkey, 40
 - flist32_getval, 39
 - FLIST32_NODE, 38
 - FLIST32_NODE1, 39
 - FLIST32_START, 38
 - flist32_to_str, 41
 - FLIST64_END, 39
 - flist64_from_str, 43
 - flist64_getkey, 43
 - flist64_getval, 42
 - FLIST64_NODE, 39
 - FLIST64_NODE1, 39
 - FLIST64_START, 39
 - flist64_t, 162
 - key, 162
 - val, 162
 - flist64_to_str
 - flist, 44
 - Formatted input conversion, 96
 - Formatted output conversion, 83
 - hash
 - FLIST64_NODE1, 39
 - FLIST64_START, 39
 - flist64_to_str, 44
 - flist.h, 197
 - flist/flist32_from_str.c, 199
 - flist/flist32_getkey.c, 200
 - flist/flist32_getval.c, 201
 - flist/flist32_to_str.c, 202
 - flist/flist64_from_str.c, 203
 - flist/flist64_getkey.c, 204
 - flist/flist64_getval.c, 205
 - flist/flist64_to_str.c, 206
 - FLIST32_END
 - flist, 39
 - flist32_from_str
 - flist, 40
 - flist32_getkey
 - flist, 40
 - flist32_getval
 - flist, 39
 - FLIST32_NODE
 - flist, 38
 - FLIST32_NODE1
 - flist, 39
 - FLIST32_START
 - flist, 38
 - flist32_t, 161
 - key, 161
 - val, 161
 - flist32_to_str
 - flist, 41
 - FLIST64_END
 - flist, 39
 - flist64_from_str
 - flist, 43
 - flist64_getkey
 - flist, 43
 - flist64_getval
 - flist, 42
 - FLIST64_NODE
 - flist, 39
 - FLIST64_NODE1
 - flist, 39
 - FLIST64_START
 - flist, 39
 - flist64_t, 162
 - key, 162
 - val, 162
 - flist64_to_str
 - flist, 44
 - Formatted input conversion, 96
 - Formatted output conversion, 83
 - hash

- whirlpool_t, 168
- i2v32
 - bitmap, 18
- i2v64
 - bitmap, 18
- Internet address conversion, 13
- isdir
 - misc, 72
- isfile
 - misc, 72
- islink
 - misc, 73
- ismount
 - misc, 73
- ispath
 - misc, 72
- key
 - flist32_t, 161
 - flist64_t, 162
- l
 - __printf_t, 157
 - __scanf_t, 159
- len
 - _mem_pool_t, 160
 - stralloc_t, 166
 - whirlpool_t, 168
- LENGTHBITS
 - whirlpool, 147
- LENGTHBYTES
 - whirlpool, 147
- list
 - _mem_pool_t, 160
 - container_of, 47
 - list_entry, 47
 - list_for_each, 47
 - list_for_each_entry, 48
 - list_for_each_entry_reverse, 48
 - list_for_each_entry_safe, 49
 - list_for_each_entry_safe_reverse, 49
 - list_for_each_prev, 47
 - list_for_each_safe, 48
 - LIST_NODE_ALLOC, 47
 - list_t, 50
 - strtok_t, 167
- list.h, 207
- list_entry
 - list, 47
- list_for_each
 - list, 47
- list_for_each_entry
 - list, 48
- list_for_each_entry_reverse
 - list, 48
- list_for_each_entry_safe
 - list, 49
- list_for_each_entry_safe_reverse
 - list, 49
- list_for_each_prev
 - list, 47
- list_for_each_safe
 - list, 48
- list_head, 163
 - next, 163
 - prev, 163
- LIST_NODE_ALLOC
 - list, 47
- list_t
 - list, 50
- log
 - log_alert, 55
 - log_alert_and_die, 58
 - log_close, 62
 - log_debug, 57
 - log_error, 56
 - log_error_and_die, 59
 - log_info, 57
 - log_init, 55
 - log_notice, 56
 - log_palert, 59
 - log_palert_and_die, 61
 - log_pdebug, 60
 - log_perror, 59
 - log_perror_and_die, 61
 - log_pinfo, 60
 - log_pnotice, 60
 - log_ptrace, 61
 - log_pwarn, 59
 - log_trace, 57
 - LOG_TRACEME, 54
 - log_traceme, 58
 - log_warn, 56
 - LOGD_FILE, 53
 - LOGD_STDERR, 53
 - LOGD_SYSLOG, 53
 - LOGO_IDENT, 54
 - LOGO_PID, 54
 - LOGO_PRIO, 54
 - LOGO_TIME, 54
 - LOGP_ALERT, 53
 - LOGP_DEBUG, 54
 - LOGP_ERROR, 53
 - LOGP_INFO, 54
 - LOGP_NOTE, 53
 - LOGP_TRACE, 54
 - LOGP_WARN, 53

- Log system multiplexer, 51
- log.h, 209
- log/log_close.c, 212
- log/log_init.c, 213
- log/log_internal.c, 214
- log_alert
 - log, 55
- log_alert_and_die
 - log, 58
- log_close
 - log, 62
- log_close.c
 - _log_options, 212
- log_debug
 - log, 57
- log_dest
 - log_options_t, 164
- log_error
 - log, 56
- log_error_and_die
 - log, 59
- log_facility
 - log_options_t, 165
- log_fd
 - log_options_t, 164
- log_ident
 - log_options_t, 164
- log_info
 - log, 57
- log_init
 - log, 55
- log_init.c
 - _log_options, 213
 - MASK_PRIO, 213
- log_internal.c
 - _log_options, 215
 - LOGFUNC, 214
 - LOGFUNC DIE, 215
 - LOGPFUNC, 215
 - LOGPFUNC DIE, 215
- log_mask
 - log_options_t, 165
- log_notice
 - log, 56
- log_options_t, 164
 - log_dest, 164
 - log_facility, 165
 - log_fd, 164
 - log_ident, 164
 - log_mask, 165
 - log_opts, 165
- log_opts
 - log_options_t, 165
- log_palert
 - log, 59
- log_palert_and_die
 - log, 61
- log_pdebug
 - log, 60
- log_perror
 - log, 59
- log_perror_and_die
 - log, 61
- log_pinfo
 - log, 60
- log_pnotice
 - log, 60
- log_ptrace
 - log, 61
- log_pwarn
 - log, 59
- log_trace
 - log, 57
- LOG_TRACEME
 - log, 54
- log_traceme
 - log, 58
- log_warn
 - log, 56
- LOGD_FILE
 - log, 53
- LOGD_STDERR
 - log, 53
- LOGD_SYSLOG
 - log, 53
- LOGFUNC
 - log_internal.c, 214
- LOGFUNC DIE
 - log_internal.c, 215
- LOGO_IDENT
 - log, 54
- LOGO_PID
 - log, 54
- LOGO_PRIO
 - log, 54
- LOGO_TIME
 - log, 54
- LOGP_ALERT
 - log, 53
- LOGP_DEBUG
 - log, 54
- LOGP_ERROR
 - log, 53
- LOGP_INFO
 - log, 54
- LOGP_NOTE
 - log, 53
- LOGP_TRACE

- log, 54
- LOGP_WARN
 - log, 53
- LOGPFUNC
 - log_internal.c, 215
- LOGPFUNC DIE
 - log_internal.c, 215
- MASK_PRIO
 - log_init.c, 213
- mem
 - _mem_pool_t, 160
 - mem_alloc, 63
 - mem_ccpy, 64
 - mem_chr, 65
 - mem_cmp, 65
 - mem_cpy, 66
 - mem_dup, 66
 - mem_free, 67
 - mem_freeall, 67
 - mem_idx, 68
 - mem_realloc, 68
 - mem_set, 69
- mem.h, 216
- mem/mem_alloc.c, 217
- mem/mem_ccpy.c, 218
- mem/mem_chr.c, 219
- mem/mem_cmp.c, 220
- mem/mem_cpy.c, 221
- mem/mem_dup.c, 222
- mem/mem_free.c, 223
- mem/mem_freeall.c, 224
- mem/mem_idx.c, 225
- mem/mem_internal.h, 226
- mem/mem_realloc.c, 227
- mem/mem_set.c, 228
- mem_alloc
 - mem, 63
- mem_alloc.c
 - _mem_pool, 217
- mem_ccpy
 - mem, 64
- mem_chr
 - mem, 65
- mem_cmp
 - mem, 65
- mem_cpy
 - mem, 66
- mem_dup
 - mem, 66
- mem_for_each
 - mem_internal.h, 226
- mem_for_each_safe
 - mem_internal.h, 226
- mem_free
 - mem, 67
- mem_freeall
 - mem, 67
- mem_idx
 - mem, 68
- mem_internal.h
 - _mem_pool, 226
 - mem_for_each, 226
 - mem_for_each_safe, 226
- mem_realloc
 - mem, 68
- mem_set
 - mem, 69
- Memory area manipulation, 63
- misc
 - copy_file, 77
 - isdir, 72
 - isfile, 72
 - islink, 73
 - ismount, 73
 - ispath, 72
 - mkdirnamep, 74
 - mkdirp, 74
 - readsymlink, 77
 - runlink, 75
- misc.h, 229
- misc/copy_file.c, 230
- misc/isdir.c, 231
- misc/isfile.c, 232
- misc/islink.c, 233
- misc/ismount.c, 234
- misc/ispath.c, 235
- misc/mkdirnamep.c, 236
- misc/mkdirp.c, 237
- misc/readsymlink.c, 238
- misc/runlink.c, 239
- Miscellaneous helpers, 71
- mkdirnamep
 - misc, 74
- mkdirp
 - misc, 74
- next
 - list_head, 163
- open
 - open_append, 80
 - open_excl, 80
 - open_read, 81
 - open_rw, 81
 - open_trunc, 82
 - open_write, 82
- open.h, 240

- open/open_append.c, 241
- open/open_excl.c, 242
- open/open_read.c, 243
- open/open_rw.c, 244
- open/open_trunc.c, 245
- open/open_write.c, 246
- open_append
 - open, 80
- open_excl
 - open, 80
- open_read
 - open, 81
- open_rw
 - open, 81
- open_trunc
 - open, 82
- open_write
 - open, 82
- P
 - __printf_t, 157
- PFL_ALT
 - vsnprintf.c, 256
- PFL_BLANK
 - vsnprintf.c, 256
- PFL_LEFT
 - vsnprintf.c, 256
- PFL_SIGN
 - vsnprintf.c, 256
- PFL_SIGNED
 - vsnprintf.c, 256
- PFL_UPPER
 - vsnprintf.c, 256
- PFL_ZERO
 - vsnprintf.c, 256
- PFR_CHAR
 - vsnprintf.c, 256
- PFR_INT
 - vsnprintf.c, 256
- PFR_LLONG
 - vsnprintf.c, 256
- PFR_LONG
 - vsnprintf.c, 256
- PFR_MAX
 - vsnprintf.c, 255
- PFR_MIN
 - vsnprintf.c, 255
- PFR_SHORT
 - vsnprintf.c, 256
- PFS_CONV
 - vsnprintf.c, 257
- PFS_FLAGS
 - vsnprintf.c, 257
- PFS_MOD
 - vsnprintf.c, 257
- PFS_NORMAL
 - vsnprintf.c, 257
- PFS_PREC
 - vsnprintf.c, 257
- PFS_WIDTH
 - vsnprintf.c, 257
- pos
 - whirlpool_t, 168
- prev
 - list_head, 163
- printf
 - _lucid_asprintf, 93
 - _lucid_dprintf, 94
 - _lucid_printf, 95
 - _lucid_snprintf, 91
 - _lucid_vasprintf, 92
 - _lucid_vdprintf, 93
 - _lucid_vprintf, 94
 - _lucid_vsnprintf, 86
- printf.h, 247
- printf/asprintf.c, 248
- printf/dprintf.c, 249
- printf/printf.c, 250
- printf/snprintf.c, 251
- printf/vasprintf.c, 252
- printf/vdprintf.c, 253
- printf/vprintf.c, 254
- printf/vsnprintf.c, 255
- R
 - whirlpool_tables.h, 326
- readsymlink
 - misc, 77
- readsymlink.c
 - CHUNKSIZE, 238
- runlink
 - misc, 75
- s
 - __printf_t, 157
 - __scanf_t, 159
 - stralloc_t, 166
- scanf
 - _lucid_sscanf, 103
 - _lucid_vsscanf, 98
- scanf.h, 258
- scanf/sscanf.c, 259
- scanf/vsscanf.c, 260
- Secure chroot wrappers, 26
- SFL_NOOP
 - vsscanf.c, 261
- SFL_WIDTH
 - vsscanf.c, 261

- SFR_CHAR
 - vsscanf.c, 261
- SFR_INT
 - vsscanf.c, 261
- SFR_LLONG
 - vsscanf.c, 261
- SFR_LONG
 - vsscanf.c, 261
- SFR_MAX
 - vsscanf.c, 260
- SFR_MIN
 - vsscanf.c, 260
- SFR_SHORT
 - vsscanf.c, 261
- SFS_CONV
 - vsscanf.c, 261
- SFS_EOF
 - vsscanf.c, 261
- SFS_ERR
 - vsscanf.c, 261
- SFS_FLAGS
 - vsscanf.c, 261
- SFS_MOD
 - vsscanf.c, 261
- SFS_NORMAL
 - vsscanf.c, 261
- SFS_STORE
 - vsscanf.c, 261
- SFS_WIDTH
 - vsscanf.c, 261
- Simple doubly linked lists, 46
- str
 - CC_ALNUM, 108
 - CC_ALPHA, 108
 - CC_ASCII, 108
 - CC_BLANK, 108
 - CC_CNTRL, 109
 - CC_DIGIT, 109
 - CC_GRAPH, 109
 - CC_LOWER, 109
 - CC_PRINT, 109
 - CC_PUNCT, 109
 - CC_SPACE, 109
 - CC_UPPER, 110
 - CC_XDIGIT, 110
 - CHUNKSIZE, 111
 - str_check, 111
 - str_chr, 115
 - str_cmp, 112
 - str_cmpn, 113
 - str_cpy, 113
 - str_cpyn, 114
 - str_dup, 114
 - str_equal, 113
 - str_isalnum, 110
 - str_isalpha, 110
 - str_isascii, 110
 - str_isdigit, 110
 - str_isempty, 110
 - str_isgraph, 111
 - str_islower, 111
 - str_isprint, 111
 - str_isupper, 111
 - str_isxdigit, 111
 - str_len, 116
 - str_path_basename, 118
 - str_path_concat, 119
 - str_path_dirname, 117
 - str_path_isabs, 119
 - str_path_isdot, 120
 - str_rchr, 115
 - str_read, 125
 - str_readfile, 124
 - str_readline, 123
 - str_str, 116
 - str_tolower, 121
 - str_toumax, 122
 - str_toupper, 121
- str.h, 263
- str/str_check.c, 266
- str/str_chr.c, 267
- str/str_cmp.c, 268
- str/str_cmpn.c, 269
- str/str_cpy.c, 270
- str/str_cpyn.c, 271
- str/str_dup.c, 272
- str/str_equal.c, 273
- str/str_len.c, 274
- str/str_path_basename.c, 275
- str/str_path_concat.c, 276
- str/str_path_dirname.c, 277
- str/str_path_isabs.c, 278
- str/str_path_isdot.c, 279
- str/str_rchr.c, 280
- str/str_read.c, 281
- str/str_readfile.c, 282
- str/str_readline.c, 283
- str/str_str.c, 284
- str/str_tolower.c, 285
- str/str_toumax.c, 286
- str/str_toupper.c, 287
- str_check
 - str, 111
- str_chr
 - str, 115
- str_cmp
 - str, 112
- str_cmpn

- str, 113
- str_cpy
 - str, 113
- str_cpyn
 - str, 114
- str_dup
 - str, 114
- str_equal
 - str, 113
- str_isalnum
 - str, 110
- str_isalpha
 - str, 110
- str_isascii
 - str, 110
- str_isdigit
 - str, 110
- str_isempty
 - str, 110
- str_isgraph
 - str, 111
- str_islower
 - str, 111
- str_isprint
 - str, 111
- str_isupper
 - str, 111
- str_isxdigit
 - str, 111
- str_len
 - str, 116
- str_path_basename
 - str, 118
- str_path_concat
 - str, 119
- str_path_dirname
 - str, 117
- str_path_isabs
 - str, 119
- str_path_isdot
 - str, 120
- str_rchr
 - str, 115
- str_read
 - str, 125
- str_readfile
 - str, 124
- str_readline
 - str, 123
- str_str
 - str, 116
- str_tolower
 - str, 121
- str_toumax
 - str, 122
- str_toupper
 - str, 121
- stralloc
 - stralloc_cat, 134
 - stralloc_catb, 132
 - stralloc_catf, 132
 - stralloc_catm, 133
 - stralloc_cats, 134
 - stralloc_cmp, 135
 - stralloc_copy, 132
 - stralloc_copyb, 131
 - stralloc_copys, 131
 - stralloc_finalize, 130
 - stralloc_free, 130
 - stralloc_init, 128
 - stralloc_ready, 129
 - stralloc_readyplus, 129
 - stralloc_zero, 128
- stralloc.h, 288
- stralloc/stralloc_cat.c, 290
- stralloc/stralloc_catb.c, 291
- stralloc/stralloc_catf.c, 292
- stralloc/stralloc_catm.c, 293
- stralloc/stralloc_cats.c, 294
- stralloc/stralloc_cmp.c, 295
- stralloc/stralloc_copy.c, 296
- stralloc/stralloc_copyb.c, 297
- stralloc/stralloc_copys.c, 298
- stralloc/stralloc_finalize.c, 299
- stralloc/stralloc_free.c, 300
- stralloc/stralloc_init.c, 301
- stralloc/stralloc_ready.c, 302
- stralloc/stralloc_readyplus.c, 303
- stralloc/stralloc_zero.c, 304
- stralloc_cat
 - stralloc, 134
- stralloc_catb
 - stralloc, 132
- stralloc_catf
 - stralloc, 132
- stralloc_catm
 - stralloc, 133
- stralloc_cats
 - stralloc, 134
- stralloc_cmp
 - stralloc, 135
- stralloc_copy
 - stralloc, 132
- stralloc_copyb
 - stralloc, 131
- stralloc_copys
 - stralloc, 131
- stralloc_finalize

- stralloc, 130
- stralloc_free
 - stralloc, 130
- stralloc_init
 - stralloc, 128
- stralloc_ready
 - stralloc, 129
- stralloc_readyplus
 - stralloc, 129
- stralloc_t, 166
 - a, 166
 - len, 166
 - s, 166
- stralloc_zero
 - stralloc, 128
- String classification and conversion, 105
- String tokenizer, 136
- strtok
 - strtok_append, 140
 - strtok_count, 139
 - strtok_delete, 140
 - strtok_for_each, 137
 - strtok_free, 139
 - strtok_init_argv, 137
 - strtok_init_str, 137
 - strtok_next, 141
 - strtok_prev, 141
 - strtok_toargv, 141
 - strtok_tostr, 142
- strtok.h, 305
- strtok/strtok_append.c, 307
- strtok/strtok_count.c, 308
- strtok/strtok_delete.c, 309
- strtok/strtok_free.c, 310
- strtok/strtok_init_argv.c, 311
- strtok/strtok_init_str.c, 312
- strtok/strtok_next.c, 313
- strtok/strtok_prev.c, 314
- strtok/strtok_toargv.c, 315
- strtok/strtok_tostr.c, 316
- strtok_append
 - strtok, 140
- strtok_count
 - strtok, 139
- strtok_delete
 - strtok, 140
- strtok_for_each
 - strtok, 137
- strtok_free
 - strtok, 139
- strtok_init_argv
 - strtok, 137
- strtok_init_str
 - strtok, 137
- strtok_next
 - strtok, 141
- strtok_prev
 - strtok, 141
- strtok_t, 167
 - list, 167
 - token, 167
- strtok_toargv
 - strtok, 141
- strtok_tostr
 - strtok, 142
- tcp
 - tcp_connect, 145
 - tcp_listen, 144
- TCP socket wrappers, 144
- tcp.h, 317
- tcp/tcp_connect.c, 318
- tcp/tcp_listen.c, 319
- tcp_connect
 - tcp, 145
- tcp_listen
 - tcp, 144
- token
 - strtok_t, 167
- v2i32
 - bitmap, 19
- v2i64
 - bitmap, 19
- val
 - flist32_t, 161
 - flist64_t, 162
- vsprintf.c
 - __printf_flags, 256
 - __printf_rank, 256
 - __printf_state, 257
 - EMIT, 256
 - PFL_ALT, 256
 - PFL_BLANK, 256
 - PFL_LEFT, 256
 - PFL_SIGN, 256
 - PFL_SIGNED, 256
 - PFL_UPPER, 256
 - PFL_ZERO, 256
 - PFR_CHAR, 256
 - PFR_INT, 256
 - PFR_LLONG, 256
 - PFR_LONG, 256
 - PFR_MAX, 255
 - PFR_MIN, 255
 - PFR_SHORT, 256
 - PFS_CONV, 257
 - PFS_FLAGS, 257

- PFS_MOD, 257
- PFS_NORMAL, 257
- PFS_PREC, 257
- PFS_WIDTH, 257
- vsscanf.c
 - __scanf_flags, 261
 - __scanf_rank, 261
 - __scanf_state, 261
 - SFL_NOOP, 261
 - SFL_WIDTH, 261
 - SFR_CHAR, 261
 - SFR_INT, 261
 - SFR_LLONG, 261
 - SFR_LONG, 261
 - SFR_MAX, 260
 - SFR_MIN, 260
 - SFR_SHORT, 261
 - SFS_CONV, 261
 - SFS_EOF, 261
 - SFS_ERR, 261
 - SFS_FLAGS, 261
 - SFS_MOD, 261
 - SFS_NORMAL, 261
 - SFS_STORE, 261
 - SFS_WIDTH, 261
- w
 - __printf_t, 158
 - __scanf_t, 159
- WBLOCKBITS
 - whirlpool, 147
- WBLOCKBYTES
 - whirlpool, 147
- whirlpool
 - DIGESTBITS, 147
 - DIGESTBYTES, 147
 - LENGTHBITS, 147
 - LENGTHBYTES, 147
 - WBLOCKBITS, 147
 - WBLOCKBYTES, 147
 - whirlpool_add, 153
 - whirlpool_digest, 154
 - whirlpool_finalize, 152
 - whirlpool_init, 151
 - whirlpool_transform, 148
- Whirlpool hash function, 146
- whirlpool.h, 320
- whirlpool/whirlpool_add.c, 322
- whirlpool/whirlpool_digest.c, 323
- whirlpool/whirlpool_finalize.c, 324
- whirlpool/whirlpool_init.c, 325
- whirlpool/whirlpool_tables.h, 326
- whirlpool/whirlpool_transform.c, 327
- whirlpool_add
 - whirlpool, 153
- whirlpool_digest
 - whirlpool, 154
- whirlpool_finalize
 - whirlpool, 152
- whirlpool_init
 - whirlpool, 151
- whirlpool_t, 168
 - bits, 168
 - buf, 168
 - hash, 168
 - len, 168
 - pos, 168
- whirlpool_tables.h
 - R, 326
- whirlpool_transform
 - whirlpool, 148