```
typedef unsigned long size_t;
typedef long __off_t;
typedef long __off64_t;
struct _IO_FILE;
struct _IO_FILE;
struct _IO_FILE;
typedef struct _IO_FILE FILE;
typedef void _IO_lock_t;
struct _IO_marker {
  struct _IO_marker *_next;
  struct _IO_FILE *_sbuf;
  int _pos;
};
struct _IO_FILE {
  int _flags;
  char *_IO_read_ptr;
  char *_IO_read_end;
  char *_IO_read_base;
  char *_IO_write_base;
  char *_IO_write_ptr;
  char *_IO_write_end;
  char *_IO_buf_base;
  char *_IO_buf_end;
  char *_IO_save_base;
  char *_IO_backup_base;
  char *_IO_save_end;
  struct _IO_marker *_markers;
  struct _IO_FILE *_chain;
  int _fileno;
  int _flags2;
  __off_t _old_offset;
  unsigned short _cur_column;
  signed char _vtable_offset;
  char _shortbuf[1];
  _IO_lock_t *_lock;
  __off64_t _offset;
  void *__pad1;
  void *__pad2;
  void *__pad3;
  void *__pad4;
  size_t __pad5;
  int _mode;
  char _unused2[(15UL * sizeof(int) - 4UL * sizeof(void *)) - sizeof(size_t)];
};
typedef long __time_t;
typedef unsigned long __dev_t;
typedef unsigned int __uid_t;
typedef unsigned int __gid_t;
typedef unsigned long __ino_t;
typedef unsigned int __mode_t;
typedef unsigned long __nlink_t;
typedef long __blksize_t;
typedef long __blkcnt_t;
typedef long __syscall_slong_t;
typedef __mode_t mode_t;
struct timespec {
  __time_t tv_sec;
  __syscall_slong_t tv_nsec;
};
struct stat {
  __dev_t st_dev;
  __ino_t st_ino;
  __nlink_t st_nlink;
  __mode_t st_mode;
```

```
  __uid_t st_uid;
  __gid_t st_gid;
  int __pad0;
  __dev_t st_rdev;
  __off_t st_size;
  __blksize_t st_blksize;
  __blkcnt_t st_blocks;
  struct timespec st_atim;
  struct timespec st_mtim;
  struct timespec st_ctim;
  __syscall_slong_t __glibc_reserved[3];
};
typedef __ino_t ino_t;
typedef __dev_t dev_t;
struct hash_table;
struct hash_table;
struct hash_table;
typedef struct hash_table Hash_table;
struct F_triple {
  char *name;
  ino_t st_ino;
  dev_t st_dev;
};
typedef __off_t off_t;
typedef __builtin_va_list __gnuc_va_list;
typedef __gnuc_va_list va_list;
typedef int wchar_t;
union __anonunion___value_4 {
  unsigned int __wch;
  char __wchb[4];
};
struct __anonstruct___mbstate_t_3 {
  int __count;
  union __anonunion___value_4 __value;
};
typedef struct __anonstruct___mbstate_t_3 __mbstate_t;
typedef unsigned int wint_t;
struct hash_tuning {
  float shrink_threshold;
  float shrink_factor;
  float growth_threshold;
  float growth_factor;
  _Bool is_n_buckets;
};
typedef struct hash_tuning Hash_tuning;
typedef __mbstate_t mbstate_t;
struct mbchar {
  char const *ptr;
  size_t bytes;
  _Bool wc_valid;
  wchar_t wc;
  char buf[24];
};
struct mbuiter_multi {
  _Bool in_shift;
  mbstate_t state;
  _Bool next_done;
  struct mbchar cur;
};
typedef struct mbuiter_multi mbui_iterator_t;
typedef __gid_t gid_t;
typedef __uid_t uid_t;
typedef unsigned long uintmax_t;
struct dev_ino {
```

```
  ino_t st_ino;
  dev_t st_dev;
};
struct cycle_check_state {
  struct dev_ino dev_ino;
  uintmax_t chdir_counter;
  int magic;
};
typedef long ptrdiff_t;
struct dirent {
  __ino_t d_ino;
  __off_t d_off;
  unsigned short d_reclen;
  unsigned char d_type;
  char d_name[256];
};
struct __dirstream;
struct __dirstream;
struct __dirstream;
typedef struct __dirstream DIR;
typedef long __ssize_t;
typedef __ssize_t ssize_t;
enum quoting_style {
  literal_quoting_style = 0,
  shell_quoting_style = 1,
  shell_always_quoting_style = 2,
  c_quoting_style = 3,
  c_maybe_quoting_style = 4,
  escape_quoting_style = 5,
  locale_quoting_style = 6,
  clocale_quoting_style = 7,
  custom_quoting_style = 8
};
struct option {
  char const *name;
  int has_arg;
  int *flag;
  int val;
};
typedef __nlink_t nlink_t;
struct I_ring {
  int ir_data[4];
  int ir_default_val;
  unsigned int ir_front;
  unsigned int ir_back;
  _Bool ir_empty;
};
typedef struct I_ring I_ring;
struct _ftsent;
struct _ftsent;
struct _ftsent;
union __anonunion_fts_cycle_26 {
  struct hash_table *ht;
  struct cycle_check_state *state;
};
struct __anonstruct_FTS_25 {
  struct _ftsent *fts_cur;
  struct _ftsent *fts_child;
  struct _ftsent **fts_array;
  dev_t fts_dev;
  char *fts_path;
  int fts_rfd;
  int fts_cwd_fd;
  size_t fts_pathlen;
```

```
  size_t fts_nitems;
  int (*fts_compar)(struct _ftsent const **, struct _ftsent const **);
  int fts_options;
  struct hash_table *fts_leaf_optimization_works_ht;
  union __anonunion_fts_cycle_26 fts_cycle;
  I_ring fts_fd_ring;
};
typedef struct __anonstruct_FTS_25 FTS;
struct _ftsent {
  struct _ftsent *fts_cycle;
  struct _ftsent *fts_parent;
  struct _ftsent *fts_link;
  long fts_number;
  void *fts_pointer;
  char *fts_accpath;
  char *fts_path;
  int fts_errno;
  int fts_symfd;
  size_t fts_pathlen;
  FTS *fts_fts;
  ptrdiff_t fts_level;
  size_t fts_namelen;
  nlink_t fts_n_dirs_remaining;
  unsigned short fts_info;
  unsigned short fts_flags;
  unsigned short fts_instr;
  struct stat fts_statp[1];
  char fts_name[1];
};
typedef struct _ftsent FTSENT;
typedef unsigned long reg_syntax_t;
struct quoting_options;
struct quoting_options;
struct quoting_options;
struct quoting_options {
  enum quoting_style style;
  int flags;
  unsigned int quote_these_too[255UL / (sizeof(int) * 8UL) + 1UL];
  char const *left_quote;
  char const *right_quote;
};
struct slotvec {
  size_t size;
  char *val;
};
struct hash_entry {
  void *data;
  struct hash_entry *next;
};
struct hash_table {
  struct hash_entry *bucket;
  struct hash_entry const *bucket_limit;
  size_t n_buckets;
  size_t n_buckets_used;
  size_t n_entries;
  Hash_tuning const *tuning;
  size_t (*hasher)(void const *, size_t);
  _Bool (*comparator)(void const *, void const *);
  void (*data_freer)(void *);
  struct hash_entry *free_entry_list;
};
struct __anonstruct___fsid_t_1 {
  int __val[2];
};
```

```
typedef struct __anonstruct___fsid_t_1 __fsid_t;
typedef unsigned long __fsblkcnt_t;
typedef unsigned long __fsfilcnt_t;
typedef long __fsword_t;
struct Active_dir {
  dev_t dev;
  ino_t ino;
  FTSENT *fts_ent;
};
struct statfs {
  __fsword_t f_type;
  __fsword_t f_bsize;
  __fsblkcnt_t f_blocks;
  __fsblkcnt_t f_bfree;
  __fsblkcnt_t f_bavail;
  __fsfilcnt_t f_files;
  __fsfilcnt_t f_ffree;
  __fsid_t f_fsid;
  __fsword_t f_namelen;
  __fsword_t f_frsize;
  __fsword_t f_flags;
  __fsword_t f_spare[4];
};
struct LCO_ent {
  dev_t st_dev;
  _Bool opt_ok;
};
enum rm_interactive { RMI_ALWAYS = 3,
                      RMI_SOMETIMES = 4,
                      RMI_NEVER = 5 };
struct rm_options {
  _Bool ignore_missing_files;
  enum rm_interactive interactive;
  _Bool one_file_system;
  _Bool recursive;
  struct dev_ino *root_dev_ino;
  _Bool stdin_tty;
  _Bool verbose;
  _Bool require_restore_cwd;
};
enum RM_status {
  RM_OK = 2,
  RM_USER_DECLINED = 3,
  RM_ERROR = 4,
  RM_NONEMPTY_DIR = 5
};
enum Ternary { T_UNKNOWN = 2,
               T_NO = 3,
               T_YES = 4 };
typedef enum Ternary Ternary;
enum Prompt_action { PA_DESCEND_INTO_DIR = 2,
                     PA_REMOVE_DIR = 3 };
enum interactive_type {
  interactive_never = 0,
  interactive_once = 1,
  interactive_always = 2
};
size_t freadahead(FILE *fp);
size_t freadahead(FILE *fp) {
  long tmp;

  {
    if ((unsigned long)fp->_IO_write_ptr > (unsigned long)fp->_IO_write_base) {
      return ((size_t)0);
```

```
    }
    if (fp->_flags & 256) {
      tmp = fp->_IO_save_end - fp->_IO_save_base;
    } else {
      tmp = 0L;
    }
    return ((size_t)((fp->_IO_read_end - fp->_IO_read_ptr) + tmp));
  }
}
extern __attribute__((__nothrow__)) int(__attribute__((__leaf__))
                                        tolower)(int __c);
extern __attribute__((__nothrow__))
size_t(__attribute__((__nonnull__(1), __leaf__)) strlen)(char const *__s)
    __attribute__((__pure__));
extern int fclose(FILE *__stream);
extern __attribute__((__nothrow__)) int(__attribute__((__leaf__))
                                        fileno)(FILE *__stream);
extern __attribute__((__nothrow__)) int *(
    __attribute__((__leaf__)) __errno_location)(void)__attribute__((__const__));
extern int close(int __fd);
int dup_safer(int fd);
extern __attribute__((__nothrow__)) int(
    __attribute__((__nonnull__(1, 2), __leaf__))
    strcmp)(char const *__s1, char const *__s2) __attribute__((__pure__));
extern __attribute__((__nothrow__)) int(
    __attribute__((__nonnull__(1, 2), __leaf__))
    strncmp)(char const *__s1, char const *__s2, size_t __n)
    __attribute__((__pure__));
__attribute__((__noreturn__)) void xalloc_die(void);
extern __attribute__((__nothrow__)) void *(__attribute__((__leaf__))
                                           malloc)(size_t __size)
    __attribute__((__malloc__));
size_t base_len(char const *name);
char *last_component(char const *name);
char const *file_type(struct stat const *st);
extern __attribute__((__nothrow__)) char *(__attribute__((__leaf__))
                                           gettext)(char const *__msgid)
    __attribute__((__format_arg__(1)));
char const *file_type(struct stat const *st) {
  char *tmp;
  char *tmp___0;
  char *tmp___1;
  char const *tmp___2;
  char const *tmp___3;
  char const *tmp___4;
  char const *tmp___5;
  char const *tmp___6;
  char const *tmp___7;
  char const *tmp___8;
  char const *tmp___9;
  char const *tmp___10;
  char const *tmp___12;

  {
    if ((st->st_mode & 61440U) == 32768U) {
      if (st->st_size == 0L) {
        tmp = gettext("regular empty file");
        tmp___1 = tmp;
      } else {
        tmp___0 = gettext("regular file");
        tmp___1 = tmp___0;
      }
      return ((char const *)tmp___1);
    }
```

```
    if ((st->st_mode & 61440U) == 16384U) {
      tmp___2 = (char const *)gettext("directory");
      return (tmp___2);
    }
    if ((st->st_mode & 61440U) == 24576U) {
      tmp___3 = (char const *)gettext("block special file");
      return (tmp___3);
    }
    if ((st->st_mode & 61440U) == 8192U) {
      tmp___4 = (char const *)gettext("character special file");
      return (tmp___4);
    }
    if ((st->st_mode & 61440U) == 4096U) {
      tmp___5 = (char const *)gettext("fifo");
      return (tmp___5);
    }
    if ((st->st_mode & 61440U) == 40960U) {
      tmp___6 = (char const *)gettext("symbolic link");
      return (tmp___6);
    }
    if ((st->st_mode & 61440U) == 49152U) {
      tmp___7 = (char const *)gettext("socket");
      return (tmp___7);
    }
    if (st->st_mode - st->st_mode) {
      tmp___8 = (char const *)gettext("message queue");
      return (tmp___8);
    }
    if (st->st_mode - st->st_mode) {
      tmp___9 = (char const *)gettext("semaphore");
      return (tmp___9);
    }
    if (st->st_mode - st->st_mode) {
      tmp___10 = (char const *)gettext("shared memory object");
      return (tmp___10);
    }
    tmp___12 = (char const *)gettext("weird file");
    return (tmp___12);
  }
}
void *hash_lookup(Hash_table const *table___0, void const *entry);
void *(__attribute__((__warn_unused_result__))
       hash_insert)(Hash_table *table___0, void const *entry);
void triple_free(void *x);
void *xmalloc(size_t n) __attribute__((__malloc__));
extern int fflush(FILE *__stream);
int rpl_fflush(FILE *stream);
int(__attribute__((__nonnull__(1))) rpl_fseeko)(FILE *fp, off_t offset,
                                                int whence);
extern __attribute__((__nothrow__)) int(__attribute__((__leaf__))
                                        __freading)(FILE *__fp);
__inline static void clear_ungetc_buffer_preserving_position(FILE *fp) {

  {
    if (fp->_flags & 256) {
      rpl_fseeko(fp, (off_t)0, 1);
    }
    return;
  }
}
int rpl_fflush(FILE *stream) {
  int tmp;
  int tmp___0;
  int tmp___1;
```

```
  {
    if ((unsigned long)stream == (unsigned long)((void *)0)) {
      tmp = fflush(stream);
      return (tmp);
    } else {
      tmp___0 = __freading(stream);
      if (!(tmp___0 != 0)) {
        tmp = fflush(stream);
        return (tmp);
      }
    }
    clear_ungetc_buffer_preserving_position(stream);
    tmp___1 = fflush(stream);
    return (tmp___1);
  }
}
int fd_safer(int fd);
int fd_safer(int fd) {
  int f;
  int tmp;
  int e;
  int *tmp___0;
  int *tmp___1;

  {
    if (0 <= fd) {
      if (fd <= 2) {
        tmp = dup_safer(fd);
        f = tmp;
        tmp___0 = __errno_location();
        e = *tmp___0;
        close(fd);
        tmp___1 = __errno_location();
        *tmp___1 = e;
        fd = f;
      }
    }
    return (fd);
  }
}
extern int(__attribute__((__nonnull__(1))) open)(char const *__file,
                                                 int __oflag, ...);
extern int fcntl(int __fd, int __cmd, ...);
int rpl_fcntl(int fd, int action, ...);
static int have_dupfd_cloexec = 0;
int rpl_fcntl(int fd, int action, ...) {
  va_list arg;
  int result;
  int target;
  int tmp;
  int *tmp___0;
  int flags;
  int tmp___1;
  int saved_errno;
  int *tmp___2;
  int *tmp___3;
  int tmp___4;
  void *p;
  void *tmp___5;

  {
    result = -1;
    __builtin_va_start(arg, action);
```

```
    if (action == 1030) {
      goto case_1030;
    }
    goto switch_default;
  case_1030:
    tmp = __builtin_va_arg(arg, int);
    target = tmp;
    if (0 <= have_dupfd_cloexec) {
      result = fcntl(fd, action, target);
      if (0 <= result) {
        have_dupfd_cloexec = 1;
      } else {
        tmp___0 = __errno_location();
        if (*tmp___0 != 22) {
          have_dupfd_cloexec = 1;
        } else {
          result = rpl_fcntl(fd, 0, target);
          if (result < 0) {
            goto switch_break;
          }
          have_dupfd_cloexec = -1;
        }
      }
    } else {
      result = rpl_fcntl(fd, 0, target);
    }
    if (0 <= result) {
      if (have_dupfd_cloexec == -1) {
        tmp___1 = fcntl(result, 1);
        flags = tmp___1;
        if (flags < 0) {
          goto _L;
        } else {
          tmp___4 = fcntl(result, 2, flags | 1);
          if (tmp___4 == -1) {
          _L:
            tmp___2 = __errno_location();
            saved_errno = *tmp___2;
            close(result);
            tmp___3 = __errno_location();
            *tmp___3 = saved_errno;
            result = -1;
          }
        }
      }
    }
    goto switch_break;
  switch_default:
    tmp___5 = __builtin_va_arg(arg, void *);
    p = tmp___5;
    result = fcntl(fd, action, p);
    goto switch_break;
  switch_break:
    __builtin_va_end(arg);
    return (result);
  }
}
int volatile exit_failure;
int volatile exit_failure = (int volatile)1;
extern __attribute__((__nothrow__)) unsigned short const **(
    __attribute__((__leaf__)) __ctype_b_loc)(void)__attribute__((__const__));
extern struct _IO_FILE *stdin;
extern __attribute__((__nothrow__)) int(__attribute__((__leaf__))
                                        ferror_unlocked)(FILE *__stream);
```

```
extern __attribute__((__nothrow__))
size_t(__attribute__((__leaf__)) __ctype_get_mb_cur_max)(void);
extern __attribute__((__nothrow__)) void(__attribute__((__leaf__))
                                         free)(void *__ptr);
extern __attribute__((__nothrow__, __noreturn__)) void(__attribute__((__leaf__))
                                                        abort)(void);
extern
    __attribute__((__nothrow__)) void *(__attribute__((__nonnull__(1),
                                                       __leaf__))
                                        memset)(void *__s, int __c, size_t __n);
extern __attribute__((__nothrow__)) char *(__attribute__((__nonnull__(1),
                                                          __leaf__))
                                           strrchr)(char const *__s, int __c)
    __attribute__((__pure__));
int(__attribute__((__nonnull__(1, 2))) mbscasecmp)(char const *s1,
                                                   char const *s2);
extern __attribute__((__nothrow__))
wint_t(__attribute__((__leaf__)) towlower)(wint_t __wc);
size_t hash_string(char const *string, size_t n_buckets);
Hash_table *(__attribute__((__warn_unused_result__))
             hash_initialize)(size_t candidate, Hash_tuning const *tuning,
                              size_t (*hasher)(void const *, size_t),
                              _Bool (*comparator)(void const *, void const *),
                              void (*data_freer)(void *));
void hash_free(Hash_table *table___0);
extern __attribute__((__nothrow__, __noreturn__)) void(__attribute__((
    __leaf__)) __assert_fail)(char const *__assertion, char const *__file,
                              unsigned int __line, char const *__function);
extern __attribute__((__nothrow__)) int(__attribute__((__leaf__))
                                        mbsinit)(mbstate_t const *__ps)
    __attribute__((__pure__));
extern __attribute__((__nothrow__))
size_t(__attribute__((__leaf__))
       mbrtowc)(wchar_t *__restrict __pwc, char const *__restrict __s,
                size_t __n, mbstate_t *__restrict __p);
unsigned int const is_basic_table[8];
__inline static _Bool is_basic(char c) {

  {
    return ((_Bool)((is_basic_table[(int)((unsigned char)c) >> 5] >>
                     ((int)((unsigned char)c) & 31)) &
                    1U));
  }
}
size_t strnlen1(char const *string, size_t maxlen);
__inline static void mbuiter_multi_next(struct mbuiter_multi *iter) {
  int tmp;
  size_t tmp___0;
  size_t tmp___1;
  int tmp___2;
  _Bool tmp___3;

  {
    if (iter->next_done) {
      return;
    }
    if (iter->in_shift) {
      goto with_shift;
    }
    tmp___3 = is_basic((char)*(iter->cur.ptr));
    if (tmp___3) {
      iter->cur.bytes = (size_t)1;
      iter->cur.wc = (wchar_t) * (iter->cur.ptr);
      iter->cur.wc_valid = (_Bool)1;
```

```
    } else {
      tmp = mbsinit((mbstate_t const *)(&iter->state));
      if (!tmp) {
        __assert_fail(
            "mbsinit (&iter->state)",
            "/home/khheo/project/benchmark/coreutils-8.4/lib/mbuiter.h", 142U,
            "mbuiter_multi_next");
      }
      iter->in_shift = (_Bool)1;
    with_shift:
      tmp___0 = __ctype_get_mb_cur_max();
      tmp___1 = strnlen1(iter->cur.ptr, tmp___0);
      iter->cur.bytes =
          mbrtowc(&iter->cur.wc, iter->cur.ptr, tmp___1, &iter->state);
      if (iter->cur.bytes == 0xffffffffffffffffUL) {
        iter->cur.bytes = (size_t)1;
        iter->cur.wc_valid = (_Bool)0;
      } else {
        if (iter->cur.bytes == 0xfffffffffffffffeUL) {
          iter->cur.bytes = strlen(iter->cur.ptr);
          iter->cur.wc_valid = (_Bool)0;
        } else {
          if (iter->cur.bytes == 0UL) {
            iter->cur.bytes = (size_t)1;
            if (!((int const) * (iter->cur.ptr) == 0)) {
              __assert_fail(
                  "*iter->cur.ptr == \'\\0\'",
                  "/home/khheo/project/benchmark/coreutils-8.4/lib/mbuiter.h",
                  170U, "mbuiter_multi_next");
            }
            if (!(iter->cur.wc == 0)) {
              __assert_fail(
                  "iter->cur.wc == 0",
                  "/home/khheo/project/benchmark/coreutils-8.4/lib/mbuiter.h",
                  171U, "mbuiter_multi_next");
            }
          }
          iter->cur.wc_valid = (_Bool)1;
          tmp___2 = mbsinit((mbstate_t const *)(&iter->state));
          if (tmp___2) {
            iter->in_shift = (_Bool)0;
          }
        }
      }
    }
    iter->next_done = (_Bool)1;
    return;
  }
}
void *xrealloc(void *p, size_t n);
_Bool euidaccess_stat(struct stat const *st, int mode);
extern __attribute__((__nothrow__))
__uid_t(__attribute__((__leaf__)) geteuid)(void);
extern __attribute__((__nothrow__))
__gid_t(__attribute__((__leaf__)) getegid)(void);
extern __attribute__((__nothrow__)) int(__attribute__((__leaf__))
                                        group_member)(__gid_t __gid);
_Bool euidaccess_stat(struct stat const *st, int mode) {
  uid_t euid;
  unsigned int granted;
  int tmp___2;
  int tmp___3;
  int tmp___4;
  int tmp___5;
```

```
int tmp___6;
int tmp___7;
int tmp___8;
int tmp___9;
int tmp___10;
gid_t egid;
gid_t tmp___11;
int tmp___12;

{
  mode &= 7;
  if (mode == 0) {
    return ((_Bool)1);
  }
  euid = geteuid();
  if (euid == 0U) {
    if ((mode & 1) == 0) {
      return ((_Bool)1);
    } else {
      if (st->st_mode &
          (unsigned int const)((64 | (64 >> 3)) | ((64 >> 3) >> 3))) {
        return ((_Bool)1);
      }
    }
  }
  if (256 == 4 << 6) {
    if (128 == 2 << 6) {
      if (64 == 1 << 6) {
        if (256 >> 3 == 4 << 3) {
          if (128 >> 3 == 2 << 3) {
            if (64 >> 3 == 1 << 3) {
              if ((256 >> 3) >> 3 == 4) {
                if ((128 >> 3) >> 3 == 2) {
                  if ((64 >> 3) >> 3 == 1) {
                    granted = (unsigned int)st->st_mode;
                  } else {
                    goto _L___6;
                  }
                } else {
                  goto _L___6;
                }
              } else {
                goto _L___6;
              }
            } else {
              goto _L___6;
            }
          } else {
            goto _L___6;
          }
        } else {
          goto _L___6;
        }
      } else {
        goto _L___6;
      }
    } else {
      goto _L___6;
    }
  } else {
  _L___6:
    if (st->st_mode & 256U) {
      tmp___2 = 4 << 6;
    } else {
```

```
    tmp___2 = 0;
  }
  if (st->st_mode & 128U) {
    tmp___3 = 2 << 6;
  } else {
    tmp___3 = 0;
  }
  if (st->st_mode & 64U) {
    tmp___4 = 1 << 6;
  } else {
    tmp___4 = 0;
  }
  if (st->st_mode & (unsigned int const)(256 >> 3)) {
    tmp___5 = 4 << 3;
  } else {
    tmp___5 = 0;
  }
  if (st->st_mode & (unsigned int const)(128 >> 3)) {
    tmp___6 = 2 << 3;
  } else {
    tmp___6 = 0;
  }
  if (st->st_mode & (unsigned int const)(64 >> 3)) {
    tmp___7 = 1 << 3;
  } else {
    tmp___7 = 0;
  }
  if (st->st_mode & (unsigned int const)((256 >> 3) >> 3)) {
    tmp___8 = 4;
  } else {
    tmp___8 = 0;
  }
  if (st->st_mode & (unsigned int const)((128 >> 3) >> 3)) {
    tmp___9 = 2;
  } else {
    tmp___9 = 0;
  }
  if (st->st_mode & (unsigned int const)((64 >> 3) >> 3)) {
    tmp___10 = 1;
  } else {
    tmp___10 = 0;
  }
  granted = (unsigned int)(((((((tmp___2 + tmp___3) + tmp___4) + tmp___5) +
                               tmp___6) +
                              tmp___7) +
                             tmp___8) +
                            tmp___9) +
                           tmp___10);
}
if (euid == (uid_t)st->st_uid) {
  granted >>= 6;
} else {
  tmp___11 = getegid();
  egid = tmp___11;
  if (egid == (gid_t)st->st_gid) {
    granted >>= 3;
  } else {
    tmp___12 = group_member((__gid_t)st->st_gid);
    if (tmp___12) {
      granted >>= 3;
    }
  }
}
if (((unsigned int)mode & ~granted) == 0U) {
```

```
      return ((_Bool)1);
    }
    return ((_Bool)0);
  }
}
int dup_safer(int fd) {
  int tmp;

  {
    tmp = rpl_fcntl(fd, 0, 3);
    return (tmp);
  }
}
extern __attribute__((__nothrow__)) int(
    __attribute__((__nonnull__(1, 2), __leaf__))
    stat)(char const *__restrict __file, struct stat *__restrict __buf);
extern
    __attribute__((__nothrow__)) int(__attribute__((__nonnull__(2), __leaf__))
                                     fstat)(int __fd, struct stat *__buf);
char const diacrit_base[256];
char const diacrit_diac[256];
char const diacrit_base[256] = {
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)'A', (char const)'B', (char const)'C',
    (char const)'D', (char const)'E', (char const)'F', (char const)'G',
    (char const)'H', (char const)'I', (char const)'J', (char const)'K',
    (char const)'L', (char const)'M', (char const)'N', (char const)'O',
    (char const)'P', (char const)'Q', (char const)'R', (char const)'S',
    (char const)'T', (char const)'U', (char const)'V', (char const)'W',
    (char const)'X', (char const)'Y', (char const)'Z', (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)'a', (char const)'b', (char const)'c',
    (char const)'d', (char const)'e', (char const)'f', (char const)'g',
    (char const)'h', (char const)'i', (char const)'j', (char const)'k',
    (char const)'l', (char const)'m', (char const)'n', (char const)'o',
    (char const)'p', (char const)'q', (char const)'r', (char const)'s',
    (char const)'t', (char const)'u', (char const)'v', (char const)'w',
    (char const)'x', (char const)'y', (char const)'z', (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
```

```
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)'A', (char const)'A', (char const)'A', (char const)'A',
    (char const)'A', (char const)'A', (char const)'A', (char const)'C',
    (char const)'E', (char const)'E', (char const)'E', (char const)'E',
    (char const)'I', (char const)'I', (char const)'I', (char const)'I',
    (char const)0, (char const)'N', (char const)'O', (char const)'O',
    (char const)'O', (char const)'O', (char const)'O', (char const)0,
    (char const)'O', (char const)'U', (char const)'U', (char const)'U',
    (char const)'U', (char const)'Y', (char const)0, (char const)0,
    (char const)'a', (char const)'a', (char const)'a', (char const)'a',
    (char const)'a', (char const)'a', (char const)'a', (char const)'c',
    (char const)'e', (char const)'e', (char const)'e', (char const)'e',
    (char const)'i', (char const)'i', (char const)'i', (char const)'i',
    (char const)0, (char const)'n', (char const)'o', (char const)'o',
    (char const)'o', (char const)'o', (char const)'o', (char const)0,
    (char const)'o', (char const)'u', (char const)'u', (char const)'u',
    (char const)'u', (char const)'y', (char const)0, (char const)'y'};
char const diacrit_diac[256] = {
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)4,
    (char const)0, (char const)3, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)6, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)0, (char const)0, (char const)0,
    (char const)0, (char const)0, (char const)3, (char const)2, (char const)4,
    (char const)6, (char const)5, (char const)8, (char const)1, (char const)7,
    (char const)3, (char const)2, (char const)4, (char const)5, (char const)3,
```

```
    (char const)2, (char const)4, (char const)5, (char const)0, (char const)6,
    (char const)3, (char const)2, (char const)4, (char const)6, (char const)5,
    (char const)0, (char const)9, (char const)3, (char const)2, (char const)4,
    (char const)5, (char const)2, (char const)0, (char const)0, (char const)3,
    (char const)2, (char const)4, (char const)6, (char const)5, (char const)8,
    (char const)1, (char const)7, (char const)3, (char const)2, (char const)4,
    (char const)5, (char const)3, (char const)2, (char const)4, (char const)5,
    (char const)0, (char const)6, (char const)3, (char const)2, (char const)4,
    (char const)6, (char const)5, (char const)0, (char const)9, (char const)3,
    (char const)2, (char const)4, (char const)5, (char const)2, (char const)0,
    (char const)0};
void cycle_check_init(struct cycle_check_state *state);
_Bool cycle_check(struct cycle_check_state *state, struct stat const *sb);
__inline static _Bool is_zero_or_power_of_two(uintmax_t i) {

  { return ((_Bool)((i & (i - 1UL)) == 0UL)); }
}
void cycle_check_init(struct cycle_check_state *state) {

  {
    state->chdir_counter = (uintmax_t)0;
    state->magic = 9827862;
    return;
  }
}
_Bool cycle_check(struct cycle_check_state *state, struct stat const *sb) {
  _Bool tmp;

  {
    if (!(state->magic == 9827862)) {
      __assert_fail(
          "state->magic == 9827862",
          "/home/khheo/project/benchmark/coreutils-8.4/lib/cycle-check.c", 60U,
          "cycle_check");
    }
    if (state->chdir_counter) {
      if (sb->st_ino == (__ino_t const)state->dev_ino.st_ino) {
        if (sb->st_dev == (__dev_t const)state->dev_ino.st_dev) {
          return ((_Bool)1);
        }
      }
    }
    (state->chdir_counter)++;
    tmp = is_zero_or_power_of_two(state->chdir_counter);
    if (tmp) {
      if (state->chdir_counter == 0UL) {
        return ((_Bool)1);
      }
      state->dev_ino.st_dev = (dev_t)sb->st_dev;
      state->dev_ino.st_ino = (ino_t)sb->st_ino;
    }
    return ((_Bool)0);
  }
}
extern void error(int __status, int __errnum, char const *__format, ...);
char const *quote(char const *name);
void close_stdout(void);
extern struct _IO_FILE *stdout;
extern struct _IO_FILE *stderr;
extern __attribute__((__noreturn__)) void _exit(int __status);
int close_stream(FILE *stream);
char *quotearg_colon(char const *arg);
static char const *file_name;
static _Bool ignore_EPIPE;
```

```
void close_stdout(void) {
  char const *write_error;
  char const *tmp;
  char *tmp___0;
  int *tmp___1;
  int *tmp___2;
  int tmp___3;
  int *tmp___4;
  int tmp___5;

  {
    tmp___3 = close_stream(stdout);
    if (tmp___3 != 0) {
      if (ignore_EPIPE) {
        tmp___4 = __errno_location();
        if (!(*tmp___4 == 32)) {
          goto _L;
        }
      } else {
      _L:
        tmp = (char const *)gettext("write error");
        write_error = tmp;
        if (file_name) {
          tmp___0 = quotearg_colon(file_name);
          tmp___1 = __errno_location();
          error(0, *tmp___1, "%s: %s", tmp___0, write_error);
        } else {
          tmp___2 = __errno_location();
          error(0, *tmp___2, "%s", write_error);
        }
        _exit((int)exit_failure);
      }
    }
    tmp___5 = close_stream(stderr);
    if (tmp___5 != 0) {
      _exit((int)exit_failure);
    }
    return;
  }
}
void close_stdin(void);
static char const *file_name___0;
void close_stdin(void) {
  _Bool fail;
  int tmp;
  int tmp___0;
  size_t tmp___1;
  int tmp___2;
  char const *close_error;
  char const *tmp___3;
  char *tmp___4;
  int *tmp___5;
  int *tmp___6;

  {
    fail = (_Bool)0;
    tmp___1 = freadahead(stdin);
    if (tmp___1 > 0UL) {
      tmp = rpl_fseeko(stdin, (off_t)0, 1);
      if (tmp == 0) {
        tmp___0 = rpl_fflush(stdin);
        if (tmp___0 != 0) {
          fail = (_Bool)1;
        }
```

```
      }
    }
    tmp___2 = close_stream(stdin);
    if (tmp___2 != 0) {
      fail = (_Bool)1;
    }
    if (fail) {
      tmp___3 = (char const *)gettext("error closing file");
      close_error = tmp___3;
      if (file_name___0) {
        tmp___4 = quotearg_colon(file_name___0);
        tmp___5 = __errno_location();
        error(0, *tmp___5, "%s: %s", tmp___4, close_error);
      } else {
        tmp___6 = __errno_location();
        error(0, *tmp___6, "%s", close_error);
      }
    }
    close_stdout();
    if (fail) {
      _exit((int)exit_failure);
    }
    return;
  }
}
extern __attribute__((__nothrow__))
size_t(__attribute__((__leaf__)) __fpending)(FILE *__fp);
int close_stream(FILE *stream) {
  _Bool some_pending;
  size_t tmp;
  _Bool prev_fail;
  int tmp___0;
  _Bool fclose_fail;
  int tmp___1;
  int *tmp___2;
  int *tmp___3;

  {
    tmp = __fpending(stream);
    some_pending = (_Bool)(tmp != 0UL);
    tmp___0 = ferror_unlocked(stream);
    prev_fail = (_Bool)(tmp___0 != 0);
    tmp___1 = fclose(stream);
    fclose_fail = (_Bool)(tmp___1 != 0);
    if (prev_fail) {
      goto _L___0;
    } else {
      if (fclose_fail) {
        if (some_pending) {
          goto _L___0;
        } else {
          tmp___3 = __errno_location();
          if (*tmp___3 != 9) {
          _L___0:
            if (!fclose_fail) {
              tmp___2 = __errno_location();
              *tmp___2 = 0;
            }
            return (-1);
          }
        }
      }
    }
    return (0);
```

```
  }
}
int set_cloexec_flag(int desc, _Bool value);
int set_cloexec_flag(int desc, _Bool value) {
  int flags;
  int tmp;
  int newflags;
  int tmp___0;
  int tmp___1;

  {
    tmp = rpl_fcntl(desc, 1, 0);
    flags = tmp;
    if (0 <= flags) {
      if (value) {
        tmp___0 = flags | 1;
      } else {
        tmp___0 = flags & -2;
      }
      newflags = tmp___0;
      if (flags == newflags) {
        return (0);
      } else {
        tmp___1 = rpl_fcntl(desc, 2, newflags);
        if (tmp___1 != -1) {
          return (0);
        }
      }
    }
    return (-1);
  }
}
extern __attribute__((__nothrow__)) int(__attribute__((__leaf__))
                                        fchdir)(int __fd);
extern int(__attribute__((__nonnull__(2))) openat)(int __fd, char const *__file,
                                                   int __oflag, ...);
extern __attribute__((__nothrow__)) void *(
    __attribute__((__nonnull__(1), __leaf__))
    memchr)(void const *__s, int __c, size_t __n) __attribute__((__pure__));
extern __attribute__((__nothrow__)) void *(
    __attribute__((__nonnull__(1, 2), __leaf__))
    memmove)(void *__dest, void const *__src, size_t __n);
extern __attribute__((__nothrow__)) int(
    __attribute__((__nonnull__(1, 2), __leaf__))
    lstat)(char const *__restrict __file, struct stat *__restrict __buf);
size_t triple_hash(void const *x, size_t table_size);
_Bool triple_compare_ino_str(void const *x, void const *y);
char *last_component(char const *name) {
  char const *base;
  char const *p;
  _Bool saw_slash;

  {
    base = name + 0;
    saw_slash = (_Bool)0;
    while (1) {

      if (!((int const) * base == 47)) {
        goto while_break;
      }
      base++;
    }
  while_break:
    p = base;
```

```
    while (1) {

      if (!*p) {
        goto while_break___0;
      }
      if ((int const) * p == 47) {
        saw_slash = (_Bool)1;
      } else {
        if (saw_slash) {
          base = p;
          saw_slash = (_Bool)0;
        }
      }
      p++;
    }
  while_break___0:;
    return ((char *)base);
  }
}
size_t base_len(char const *name) {
  size_t len;
  size_t prefix_len;

  {
    prefix_len = (size_t)0;
    len = strlen(name);
    while (1) {

      if (1UL < len) {
        if (!((int const) * (name + (len - 1UL)) == 47)) {
          goto while_break;
        }
      } else {
        goto while_break;
      }
      goto __Cont;
    __Cont:
      len--;
    }
  while_break:;
    return (len);
  }
}
char const *simple_backup_suffix;
void (*argmatch_die)(void);
ptrdiff_t __xargmatch_internal(char const *context, char const *arg,
                               char const *const *arglist, char const *vallist,
                               size_t valsize, void (*exit_fn)(void));
extern __attribute__((__nothrow__)) int(
    __attribute__((__nonnull__(1, 2), __leaf__))
    memcmp)(void const *__s1, void const *__s2, size_t __n)
    __attribute__((__pure__));
extern int(__attribute__((__nonnull__(1))) closedir)(DIR *__dirp);
extern struct dirent *(__attribute__((__nonnull__(1))) readdir)(DIR *__dirp);
DIR *opendir_safer(char const *name);
char const *simple_backup_suffix = "~";
ptrdiff_t argmatch(char const *arg, char const *const *arglist,
                   char const *vallist, size_t valsize);
void argmatch_invalid(char const *context, char const *value,
                      ptrdiff_t problem);
void argmatch_valid(char const *const *arglist, char const *vallist,
                    size_t valsize);
extern int fprintf(FILE *__restrict __stream, char const *__restrict __format,
                   ...);
```

```
extern int putc_unlocked(int __c, FILE *__stream);
char *quotearg_n_style(int n, enum quoting_style s, char const *arg);
char const *quote_n(int n, char const *name);
__attribute__((__noreturn__)) void usage(int status);
static void __argmatch_die(void) {

  {
    usage(1);
    return;
  }
}
void (*argmatch_die)(void) = &__argmatch_die;
ptrdiff_t argmatch(char const *arg, char const *const *arglist,
                   char const *vallist, size_t valsize) {
  size_t i;
  size_t arglen;
  ptrdiff_t matchind;
  _Bool ambiguous;
  int tmp;
  size_t tmp___0;
  int tmp___1;

  {
    matchind = (ptrdiff_t)-1;
    ambiguous = (_Bool)0;
    arglen = strlen(arg);
    i = (size_t)0;
    while (1) {

      if (!*(arglist + i)) {
        goto while_break;
      }
      tmp___1 = strncmp((char const *)*(arglist + i), arg, arglen);
      if (!tmp___1) {
        tmp___0 = strlen((char const *)*(arglist + i));
        if (tmp___0 == arglen) {
          return ((ptrdiff_t)i);
        } else {
          if (matchind == -1L) {
            matchind = (ptrdiff_t)i;
          } else {
            if ((unsigned long)vallist == (unsigned long)((void *)0)) {
              ambiguous = (_Bool)1;
            } else {
              tmp = memcmp((void const *)(vallist + valsize * (size_t)matchind),
                           (void const *)(vallist + valsize * i), valsize);
              if (tmp) {
                ambiguous = (_Bool)1;
              }
            }
          }
        }
      }
      i++;
    }
  while_break:;
    if (ambiguous) {
      return ((ptrdiff_t)-2);
    } else {
      return (matchind);
    }
  }
}
void argmatch_invalid(char const *context, char const *value,
```

```
                      ptrdiff_t problem) {
  char const *format;
  char *tmp;
  char *tmp___0;
  char *tmp___1;
  char const *tmp___2;
  char *tmp___3;

  {
    if (problem == -1L) {
      tmp = gettext("invalid argument %s for %s");
      tmp___1 = tmp;
    } else {
      tmp___0 = gettext("ambiguous argument %s for %s");
      tmp___1 = tmp___0;
    }
    format = (char const *)tmp___1;
    tmp___2 = quote_n(1, context);
    tmp___3 = quotearg_n_style(0, (enum quoting_style)6, value);
    error(0, 0, format, tmp___3, tmp___2);
    return;
  }
}
void argmatch_valid(char const *const *arglist, char const *vallist,
                    size_t valsize) {
  size_t i;
  char const *last_val;
  char *tmp;
  int tmp___0;

  {
    last_val = (char const *)((void *)0);
    tmp = gettext("Valid arguments are:");
    fprintf(stderr, (char const *)tmp);
    i = (size_t)0;
    while (1) {

      if (!*(arglist + i)) {
        goto while_break;
      }
      if (i == 0UL) {
        fprintf(stderr, "\n  - `%s\'", *(arglist + i));
        last_val = vallist + valsize * i;
      } else {
        tmp___0 = memcmp((void const *)last_val,
                         (void const *)(vallist + valsize * i), valsize);
        if (tmp___0) {
          fprintf(stderr, "\n  - `%s\'", *(arglist + i));
          last_val = vallist + valsize * i;
        } else {
          fprintf(stderr, ", `%s\'", *(arglist + i));
        }
      }
      i++;
    }
  while_break:
    putc_unlocked('\n', stderr);
    return;
  }
}
ptrdiff_t __xargmatch_internal(char const *context, char const *arg,
                               char const *const *arglist, char const *vallist,
                               size_t valsize, void (*exit_fn)(void)) {
  ptrdiff_t res;
```

```
  ptrdiff_t tmp;

  {
    tmp = argmatch(arg, arglist, vallist, valsize);
    res = tmp;
    if (res >= 0L) {
      return (res);
    }
    argmatch_invalid(context, arg, res);
    argmatch_valid(arglist, vallist, valsize);
    (*exit_fn)();
    return ((ptrdiff_t)-1);
  }
}
extern
    __attribute__((__nothrow__)) void *(__attribute__((__warn_unused_result__,
                                                       __leaf__))
                                        realloc)(void *__ptr, size_t __size);
_Bool yesno(void);
extern
    __attribute__((__nothrow__)) int(__attribute__((__nonnull__(1), __leaf__))
                                     rpmatch)(char const *__response);
extern __ssize_t getline(char **__restrict __lineptr, size_t *__restrict __n,
                         FILE *__restrict __stream);
_Bool yesno(void) {
  _Bool yes;
  char *response;
  size_t response_size;
  ssize_t response_len;
  ssize_t tmp;
  int tmp___0;

  {
    response = (char *)((void *)0);
    response_size = (size_t)0;
    tmp = getline(&response, &response_size, stdin);
    response_len = tmp;
    if (response_len <= 0L) {
      yes = (_Bool)0;
    } else {
      *(response + (response_len - 1L)) = (char)'\000';
      tmp___0 = rpmatch((char const *)response);
      yes = (_Bool)(0 < tmp___0);
    }
    free((void *)response);
    return (yes);
  }
}
__inline static void *xnmalloc(size_t n, size_t s) __attribute__((__malloc__));
__inline static void *xnmalloc(size_t n, size_t s) __attribute__((__malloc__));
__inline static void *xnmalloc(size_t n, size_t s) {
  int tmp;
  void *tmp___0;

  {
    if (sizeof(ptrdiff_t) <= sizeof(size_t)) {
      tmp = -1;
    } else {
      tmp = -2;
    }
    if ((size_t)tmp / s < n) {
      xalloc_die();
    }
    tmp___0 = xmalloc(n * s);
```

```
    return (tmp___0);
  }
}
extern
    __attribute__((__nothrow__)) void *(__attribute__((__leaf__))
                                        calloc)(size_t __nmemb, size_t __size)
        __attribute__((__malloc__));
void *xmalloc(size_t n) __attribute__((__malloc__));
void *xmalloc(size_t n) {
  void *p;
  void *tmp;

  {
    tmp = malloc(n);
    p = tmp;
    if (!p) {
      if (n != 0UL) {
        xalloc_die();
      }
    }
    return (p);
  }
}
void *xrealloc(void *p, size_t n) {

  {
    p = realloc(p, n);
    if (!p) {
      if (n != 0UL) {
        xalloc_die();
      }
    }
    return (p);
  }
}
__attribute__((__nothrow__))
FTS *(__attribute__((__warn_unused_result__, __leaf__))
      fts_open)(char *const *argv, int options,
                int (*compar)(FTSENT const **, FTSENT const **));
FTS *xfts_open(char *const *argv, int options,
               int (*compar)(FTSENT const **, FTSENT const **));
FTS *xfts_open(char *const *argv, int options,
               int (*compar)(FTSENT const **, FTSENT const **)) {
  FTS *fts;
  FTS *tmp;
  int *tmp___0;

  {
    tmp = fts_open(argv, options | 512, compar);
    fts = tmp;
    if ((unsigned long)fts == (unsigned long)((void *)0)) {
      tmp___0 = __errno_location();
      if (!(*tmp___0 != 22)) {
        __assert_fail("(*__errno_location ()) != 22",
                      "/home/khheo/project/benchmark/coreutils-8.4/lib/xfts.c",
                      41U, "xfts_open");
      }
      xalloc_die();
    }
    return (fts);
  }
}
__attribute__((__noreturn__)) void xalloc_die(void);
void xalloc_die(void) {
```

```
  char *tmp;

  {
    tmp = gettext("memory exhausted");
    error((int)exit_failure, 0, "%s", tmp);
    abort();
  }
}
_Bool can_write_any_file(void);
static _Bool initialized;
static _Bool can_write;
_Bool can_write_any_file(void) {
  _Bool can;
  __uid_t tmp;

  {
    if (!initialized) {
      can = (_Bool)0;
      tmp = geteuid();
      can = (_Bool)(tmp == 0U);
      can_write = can;
      initialized = (_Bool)1;
    }
    return (can_write);
  }
}
extern int printf(char const *__restrict __format, ...);
extern int fputs_unlocked(char const *__restrict __s,
                          FILE *__restrict __stream);
char const version_etc_copyright[47];
void version_etc_arn(FILE *stream, char const *command_name,
                     char const *package, char const *version,
                     char const *const *authors, size_t n_authors);
void version_etc_va(FILE *stream, char const *command_name, char const *package,
                    char const *version, va_list authors);
void version_etc(FILE *stream, char const *command_name, char const *package,
                 char const *version, ...) __attribute__((__sentinel__));
void version_etc_arn(FILE *stream, char const *command_name,
                     char const *package, char const *version,
                     char const *const *authors, size_t n_authors) {
  char *tmp;
  char *tmp___0;
  char *tmp___1;
  char *tmp___2;
  char *tmp___3;
  char *tmp___4;
  char *tmp___5;
  char *tmp___6;
  char *tmp___7;
  char *tmp___8;
  char *tmp___9;
  char *tmp___10;

  {
    if (command_name) {
      fprintf(stream, "%s (%s) %s\n", command_name, package, version);
    } else {
      fprintf(stream, "%s %s\n", package, version);
    }
    tmp = gettext("(C)");
    fprintf(stream, version_etc_copyright, tmp, 2010);
    tmp___0 =
        gettext("\nLicense GPLv3+: GNU GPL version 3 or later "
                "<http://gnu.org/licenses/gpl.html>.\nThis is free software: "
```

```
                "you are free to change and redistribute it.\nThere is NO "
                "WARRANTY, to the extent permitted by law.\n\n");
  fputs_unlocked((char const *)tmp___0, stream);
  if (n_authors == 0UL) {
    goto case_0;
  }
  if (n_authors == 1UL) {
    goto case_1;
  }
  if (n_authors == 2UL) {
    goto case_2;
  }
  if (n_authors == 3UL) {
    goto case_3;
  }
  if (n_authors == 4UL) {
    goto case_4;
  }
  if (n_authors == 5UL) {
    goto case_5;
  }
  if (n_authors == 6UL) {
    goto case_6;
  }
  if (n_authors == 7UL) {
    goto case_7;
  }
  if (n_authors == 8UL) {
    goto case_8;
  }
  if (n_authors == 9UL) {
    goto case_9;
  }
  goto switch_default;
case_0:
  abort();
case_1:
  tmp___1 = gettext("Written by %s.\n");
  fprintf(stream, (char const *)tmp___1, *(authors + 0));
  goto switch_break;
case_2:
  tmp___2 = gettext("Written by %s and %s.\n");
  fprintf(stream, (char const *)tmp___2, *(authors + 0), *(authors + 1));
  goto switch_break;
case_3:
  tmp___3 = gettext("Written by %s, %s, and %s.\n");
  fprintf(stream, (char const *)tmp___3, *(authors + 0), *(authors + 1),
          *(authors + 2));
  goto switch_break;
case_4:
  tmp___4 = gettext("Written by %s, %s, %s,\nand %s.\n");
  fprintf(stream, (char const *)tmp___4, *(authors + 0), *(authors + 1),
          *(authors + 2), *(authors + 3));
  goto switch_break;
case_5:
  tmp___5 = gettext("Written by %s, %s, %s,\n%s, and %s.\n");
  fprintf(stream, (char const *)tmp___5, *(authors + 0), *(authors + 1),
          *(authors + 2), *(authors + 3), *(authors + 4));
  goto switch_break;
case_6:
  tmp___6 = gettext("Written by %s, %s, %s,\n%s, %s, and %s.\n");
  fprintf(stream, (char const *)tmp___6, *(authors + 0), *(authors + 1),
          *(authors + 2), *(authors + 3), *(authors + 4), *(authors + 5));
  goto switch_break;
```

```
  case_7:
    tmp___7 = gettext("Written by %s, %s, %s,\n%s, %s, %s, and %s.\n");
    fprintf(stream, (char const *)tmp___7, *(authors + 0), *(authors + 1),
            *(authors + 2), *(authors + 3), *(authors + 4), *(authors + 5),
            *(authors + 6));
    goto switch_break;
  case_8:
    tmp___8 = gettext("Written by %s, %s, %s,\n%s, %s, %s, %s,\nand %s.\n");
    fprintf(stream, (char const *)tmp___8, *(authors + 0), *(authors + 1),
            *(authors + 2), *(authors + 3), *(authors + 4), *(authors + 5),
            *(authors + 6), *(authors + 7));
    goto switch_break;
  case_9:
    tmp___9 = gettext("Written by %s, %s, %s,\n%s, %s, %s, %s,\n%s, and %s.\n");
    fprintf(stream, (char const *)tmp___9, *(authors + 0), *(authors + 1),
            *(authors + 2), *(authors + 3), *(authors + 4), *(authors + 5),
            *(authors + 6), *(authors + 7), *(authors + 8));
    goto switch_break;
  switch_default:
    tmp___10 = gettext(
        "Written by %s, %s, %s,\n%s, %s, %s, %s,\n%s, %s, and others.\n");
    fprintf(stream, (char const *)tmp___10, *(authors + 0), *(authors + 1),
            *(authors + 2), *(authors + 3), *(authors + 4), *(authors + 5),
            *(authors + 6), *(authors + 7), *(authors + 8));
    goto switch_break;
  switch_break:;
    return;
  }
}
void version_etc_va(FILE *stream, char const *command_name, char const *package,
                    char const *version, va_list authors) {
  size_t n_authors;
  char const *authtab[10];
  char const *tmp;

  {
    n_authors = (size_t)0;
    while (1) {

      if (n_authors < 10UL) {
        tmp = __builtin_va_arg(authors, char const *);
        authtab[n_authors] = tmp;
        if (!((unsigned long)tmp != (unsigned long)((void *)0))) {
          goto while_break;
        }
      } else {
        goto while_break;
      }
      n_authors++;
    }
  while_break:
    version_etc_arn(stream, command_name, package, version,
                    (char const *const *)(authtab), n_authors);
    return;
  }
}
void version_etc(FILE *stream, char const *command_name, char const *package,
                 char const *version, ...) __attribute__((__sentinel__));
void version_etc(FILE *stream, char const *command_name, char const *package,
                 char const *version, ...) {
  va_list authors;

  {
    __builtin_va_start(authors, version);
```

```
    version_etc_va(stream, command_name, package, version, authors);
    __builtin_va_end(authors);
    return;
  }
}
char const version_etc_copyright[47] = {
    (char const)'C', (char const)'o', (char const)'p', (char const)'y',
    (char const)'r', (char const)'i', (char const)'g', (char const)'h',
    (char const)'t', (char const)' ', (char const)'%', (char const)'s',
    (char const)' ', (char const)'%', (char const)'d', (char const)' ',
    (char const)'F', (char const)'r', (char const)'e', (char const)'e',
    (char const)' ', (char const)'S', (char const)'o', (char const)'f',
    (char const)'t', (char const)'w', (char const)'a', (char const)'r',
    (char const)'e', (char const)' ', (char const)'F', (char const)'o',
    (char const)'u', (char const)'n', (char const)'d', (char const)'a',
    (char const)'t', (char const)'i', (char const)'o', (char const)'n',
    (char const)',', (char const)' ', (char const)'I', (char const)'n',
    (char const)'c', (char const)'.', (char const)'\000'};
#pragma weak pthread_key_create
#pragma weak pthread_getspecific
#pragma weak pthread_setspecific
#pragma weak pthread_key_delete
#pragma weak pthread_self
#pragma weak pthread_cancel
size_t strnlen1(char const *string, size_t maxlen) {
  char const *end;
  char const *tmp;

  {
    tmp = (char const *)memchr((void const *)string, '\000', maxlen);
    end = tmp;
    if ((unsigned long)end != (unsigned long)((void *)0)) {
      return ((size_t)((end - string) + 1L));
    } else {
      return (maxlen);
    }
  }
}
_Bool strip_trailing_slashes(char *file);
_Bool strip_trailing_slashes(char *file) {
  char *base;
  char *tmp;
  char *base_lim;
  _Bool had_slash;
  size_t tmp___0;

  {
    tmp = last_component((char const *)file);
    base = tmp;
    if (!*base) {
      base = file;
    }
    tmp___0 = base_len((char const *)base);
    base_lim = base + tmp___0;
    had_slash = (_Bool)((int)*base_lim != 0);
    *base_lim = (char)'\000';
    return (had_slash);
  }
}
int open_safer(char const *file, int flags, ...);
extern DIR *fdopendir(int __fd);
struct dev_ino *get_root_dev_ino(struct dev_ino *root_d_i);
struct dev_ino *get_root_dev_ino(struct dev_ino *root_d_i) {
  struct stat statbuf;
```

```
  int tmp;

  {
    tmp = lstat("/", &statbuf);
    if (tmp) {
      return ((struct dev_ino *)((void *)0));
    }
    root_d_i->st_ino = statbuf.st_ino;
    root_d_i->st_dev = statbuf.st_dev;
    return (root_d_i);
  }
}
reg_syntax_t rpl_re_syntax_options;
char const *const quoting_style_args[9];
enum quoting_style const quoting_style_vals[8];
int set_char_quoting(struct quoting_options *o, char c, int i);
char *quotearg_char(char const *arg, char ch);
char *quotearg_char_mem(char const *arg, size_t argsize, char ch);
__inline static char *xcharalloc(size_t n) __attribute__((__malloc__));
__inline static char *xcharalloc(size_t n) __attribute__((__malloc__));
__inline static char *xcharalloc(size_t n) {
  void *tmp;
  void *tmp___0;
  void *tmp___1;

  {
    if (sizeof(char) == 1UL) {
      tmp = xmalloc(n);
      tmp___1 = tmp;
    } else {
      tmp___0 = xnmalloc(n, sizeof(char));
      tmp___1 = tmp___0;
    }
    return ((char *)tmp___1);
  }
}
extern __attribute__((__nothrow__)) int(__attribute__((__leaf__))
                                        iswprint)(wint_t __wc);
char const *const quoting_style_args[9] = {
    "literal", "shell", "shell-always", "c", "c-maybe", "escape",
    "locale", "clocale", (char const *)0};
enum quoting_style const quoting_style_vals[8] = {
    (enum quoting_style const)0, (enum quoting_style const)1,
    (enum quoting_style const)2, (enum quoting_style const)3,
    (enum quoting_style const)4, (enum quoting_style const)5,
    (enum quoting_style const)6, (enum quoting_style const)7};
static struct quoting_options default_quoting_options;
int set_char_quoting(struct quoting_options *o, char c, int i) {
  unsigned char uc;
  unsigned int *p;
  struct quoting_options *tmp;
  int shift;
  int r;

  {
    uc = (unsigned char)c;
    if (o) {
      tmp = o;
    } else {
      tmp = &default_quoting_options;
    }
    p = tmp->quote_these_too + (unsigned long)uc / (sizeof(int) * 8UL);
    shift = (int)((unsigned long)uc % (sizeof(int) * 8UL));
    r = (int)((*p >> shift) & 1U);
```

```
    *p ^= (unsigned int)(((i & 1) ^ r) << shift);
    return (r);
  }
}
static struct quoting_options
quoting_options_from_style(enum quoting_style style) {
  struct quoting_options o;

  {
    o.style = style;
    o.flags = 0;
    memset((void *)(o.quote_these_too), 0, sizeof(o.quote_these_too));
    return (o);
  }
}
static char const *gettext_quote(char const *msgid, enum quoting_style s) {
  char const *translation;
  char const *tmp;

  {
    tmp = (char const *)gettext(msgid);
    translation = tmp;
    if ((unsigned long)translation == (unsigned long)msgid) {
      if ((unsigned int)s == 7U) {
        translation = "\"";
      }
    }
    return (translation);
  }
}
static size_t
quotearg_buffer_restyled(char *buffer, size_t buffersize, char const *arg,
                         size_t argsize, enum quoting_style quoting_style,
                         int flags, unsigned int const *quote_these_too,
                         char const *left_quote, char const *right_quote) {
  size_t i;
  size_t len;
  char const *quote_string;
  size_t quote_string_len;
  _Bool backslash_escapes;
  _Bool unibyte_locale;
  size_t tmp;
  _Bool elide_outer_quotes;
  unsigned char c;
  unsigned char esc;
  _Bool is_right_quote;
  int tmp___0;
  int tmp___1;
  size_t m;
  _Bool printable;
  unsigned short const **tmp___2;
  mbstate_t mbstate;
  wchar_t w;
  size_t bytes;
  size_t tmp___3;
  size_t j;
  int tmp___4;
  int tmp___5;
  size_t ilim;
  int tmp___6;
  size_t tmp___7;

  {
    len = (size_t)0;
```

```
  quote_string = (char const *)0;
  quote_string_len = (size_t)0;
  backslash_escapes = (_Bool)0;
  tmp = __ctype_get_mb_cur_max();
  unibyte_locale = (_Bool)(tmp == 1UL);
  elide_outer_quotes = (_Bool)((flags & 2) != 0);
  if ((unsigned int)quoting_style == 4U) {
    goto case_4;
  }
  if ((unsigned int)quoting_style == 3U) {
    goto case_3;
  }
  if ((unsigned int)quoting_style == 5U) {
    goto case_5;
  }
  if ((unsigned int)quoting_style == 6U) {
    goto case_6;
  }
  if ((unsigned int)quoting_style == 7U) {
    goto case_6;
  }
  if ((unsigned int)quoting_style == 8U) {
    goto case_6;
  }
  if ((unsigned int)quoting_style == 1U) {
    goto case_1;
  }
  if ((unsigned int)quoting_style == 2U) {
    goto case_2;
  }
  if ((unsigned int)quoting_style == 0U) {
    goto case_0;
  }
  goto switch_default;
case_4:
  quoting_style = (enum quoting_style)3;
  elide_outer_quotes = (_Bool)1;
case_3:
  if (!elide_outer_quotes) {
    while (1) {

      if (len < buffersize) {
        *(buffer + len) = (char)'\"';
      }
      len++;
      goto while_break;
    }
  while_break:;
  }
  backslash_escapes = (_Bool)1;
  quote_string = "\"";
  quote_string_len = (size_t)1;
  goto switch_break;
case_5:
  backslash_escapes = (_Bool)1;
  elide_outer_quotes = (_Bool)0;
  goto switch_break;
case_6:
  if ((unsigned int)quoting_style != 8U) {
    left_quote = gettext_quote("`", quoting_style);
    right_quote = gettext_quote("\'", quoting_style);
  }
  if (!elide_outer_quotes) {
    quote_string = left_quote;
```

```
    while (1) {

      if (!*quote_string) {
        goto while_break___0;
      }
      while (1) {

        if (len < buffersize) {
          *(buffer + len) = (char)*quote_string;
        }
        len++;
        goto while_break___1;
      }
    while_break___1:
      quote_string++;
    }
  while_break___0:;
  }
  backslash_escapes = (_Bool)1;
  quote_string = right_quote;
  quote_string_len = strlen(quote_string);
  goto switch_break;
case_1:
  quoting_style = (enum quoting_style)2;
  elide_outer_quotes = (_Bool)1;
case_2:
  if (!elide_outer_quotes) {
    while (1) {

      if (len < buffersize) {
        *(buffer + len) = (char)'\'';
      }
      len++;
      goto while_break___2;
    }
  while_break___2:;
  }
  quote_string = "\'";
  quote_string_len = (size_t)1;
  goto switch_break;
case_0:
  elide_outer_quotes = (_Bool)0;
  goto switch_break;
switch_default:
  abort();
switch_break:
  i = (size_t)0;
  while (1) {

    if (argsize == 0xffffffffffffffffUL) {
      tmp___6 = (int const) * (arg + i) == 0;
    } else {
      tmp___6 = i == argsize;
    }
    if (tmp___6) {
      goto while_break___3;
    }
    is_right_quote = (_Bool)0;
    if (backslash_escapes) {
      if (quote_string_len) {
        if (i + quote_string_len <= argsize) {
          tmp___0 = memcmp((void const *)(arg + i),
                           (void const *)quote_string, quote_string_len);
          if (tmp___0 == 0) {
```

```
        if (elide_outer_quotes) {
          goto force_outer_quoting_style;
        }
        is_right_quote = (_Bool)1;
      }
    }
  }
}
c = (unsigned char)*(arg + i);
if ((int)c == 0) {
  goto case_0___0;
}
if ((int)c == 63) {
  goto case_63;
}
if ((int)c == 7) {
  goto case_7___0;
}
if ((int)c == 8) {
  goto case_8___0;
}
if ((int)c == 12) {
  goto case_12;
}
if ((int)c == 10) {
  goto case_10;
}
if ((int)c == 13) {
  goto case_13;
}
if ((int)c == 9) {
  goto case_9;
}
if ((int)c == 11) {
  goto case_11;
}
if ((int)c == 92) {
  goto case_92;
}
if ((int)c == 123) {
  goto case_123;
}
if ((int)c == 125) {
  goto case_123;
}
if ((int)c == 35) {
  goto case_35;
}
if ((int)c == 126) {
  goto case_35;
}
if ((int)c == 32) {
  goto case_32;
}
if ((int)c == 33) {
  goto case_32;
}
if ((int)c == 34) {
  goto case_32;
}
if ((int)c == 36) {
  goto case_32;
}
if ((int)c == 38) {
```

```
  goto case_32;
}
if ((int)c == 40) {
  goto case_32;
}
if ((int)c == 41) {
  goto case_32;
}
if ((int)c == 42) {
  goto case_32;
}
if ((int)c == 59) {
  goto case_32;
}
if ((int)c == 60) {
  goto case_32;
}
if ((int)c == 61) {
  goto case_32;
}
if ((int)c == 62) {
  goto case_32;
}
if ((int)c == 91) {
  goto case_32;
}
if ((int)c == 94) {
  goto case_32;
}
if ((int)c == 96) {
  goto case_32;
}
if ((int)c == 124) {
  goto case_32;
}
if ((int)c == 39) {
  goto case_39___0;
}
if ((int)c == 37) {
  goto case_37;
}
if ((int)c == 43) {
  goto case_37;
}
if ((int)c == 44) {
  goto case_37;
}
if ((int)c == 45) {
  goto case_37;
}
if ((int)c == 46) {
  goto case_37;
}
if ((int)c == 47) {
  goto case_37;
}
if ((int)c == 48) {
  goto case_37;
}
if ((int)c == 49) {
  goto case_37;
}
if ((int)c == 50) {
  goto case_37;
```

```
}
if ((int)c == 51) {
  goto case_37;
}
if ((int)c == 52) {
  goto case_37;
}
if ((int)c == 53) {
  goto case_37;
}
if ((int)c == 54) {
  goto case_37;
}
if ((int)c == 55) {
  goto case_37;
}
if ((int)c == 56) {
  goto case_37;
}
if ((int)c == 57) {
  goto case_37;
}
if ((int)c == 58) {
  goto case_37;
}
if ((int)c == 65) {
  goto case_37;
}
if ((int)c == 66) {
  goto case_37;
}
if ((int)c == 67) {
  goto case_37;
}
if ((int)c == 68) {
  goto case_37;
}
if ((int)c == 69) {
  goto case_37;
}
if ((int)c == 70) {
  goto case_37;
}
if ((int)c == 71) {
  goto case_37;
}
if ((int)c == 72) {
  goto case_37;
}
if ((int)c == 73) {
  goto case_37;
}
if ((int)c == 74) {
  goto case_37;
}
if ((int)c == 75) {
  goto case_37;
}
if ((int)c == 76) {
  goto case_37;
}
if ((int)c == 77) {
  goto case_37;
}
```

```
if ((int)c == 78) {
  goto case_37;
}
if ((int)c == 79) {
  goto case_37;
}
if ((int)c == 80) {
  goto case_37;
}
if ((int)c == 81) {
  goto case_37;
}
if ((int)c == 82) {
  goto case_37;
}
if ((int)c == 83) {
  goto case_37;
}
if ((int)c == 84) {
  goto case_37;
}
if ((int)c == 85) {
  goto case_37;
}
if ((int)c == 86) {
  goto case_37;
}
if ((int)c == 87) {
  goto case_37;
}
if ((int)c == 88) {
  goto case_37;
}
if ((int)c == 89) {
  goto case_37;
}
if ((int)c == 90) {
  goto case_37;
}
if ((int)c == 93) {
  goto case_37;
}
if ((int)c == 95) {
  goto case_37;
}
if ((int)c == 97) {
  goto case_37;
}
if ((int)c == 98) {
  goto case_37;
}
if ((int)c == 99) {
  goto case_37;
}
if ((int)c == 100) {
  goto case_37;
}
if ((int)c == 101) {
  goto case_37;
}
if ((int)c == 102) {
  goto case_37;
}
if ((int)c == 103) {
```

```
    goto case_37;
  }
  if ((int)c == 104) {
    goto case_37;
  }
  if ((int)c == 105) {
    goto case_37;
  }
  if ((int)c == 106) {
    goto case_37;
  }
  if ((int)c == 107) {
    goto case_37;
  }
  if ((int)c == 108) {
    goto case_37;
  }
  if ((int)c == 109) {
    goto case_37;
  }
  if ((int)c == 110) {
    goto case_37;
  }
  if ((int)c == 111) {
    goto case_37;
  }
  if ((int)c == 112) {
    goto case_37;
  }
  if ((int)c == 113) {
    goto case_37;
  }
  if ((int)c == 114) {
    goto case_37;
  }
  if ((int)c == 115) {
    goto case_37;
  }
  if ((int)c == 116) {
    goto case_37;
  }
  if ((int)c == 117) {
    goto case_37;
  }
  if ((int)c == 118) {
    goto case_37;
  }
  if ((int)c == 119) {
    goto case_37;
  }
  if ((int)c == 120) {
    goto case_37;
  }
  if ((int)c == 121) {
    goto case_37;
  }
  if ((int)c == 122) {
    goto case_37;
  }
  goto switch_default___2;
case_0___0:
  if (backslash_escapes) {
    if (elide_outer_quotes) {
      goto force_outer_quoting_style;
```

```
    }
    while (1) {

      if (len < buffersize) {
        *(buffer + len) = (char)'\\';
      }
      len++;
      goto while_break___4;
    }
  while_break___4:;
    if (i + 1UL < argsize) {
      if (48 <= (int)*(arg + (i + 1UL))) {
        if ((int const) * (arg + (i + 1UL)) <= 57) {
          while (1) {

            if (len < buffersize) {
              *(buffer + len) = (char)'0';
            }
            len++;
            goto while_break___5;
          }
        while_break___5:;
          while (1) {

            if (len < buffersize) {
              *(buffer + len) = (char)'0';
            }
            len++;
            goto while_break___6;
          }
        while_break___6:;
        }
      }
    }
    c = (unsigned char)'0';
  } else {
    if (flags & 1) {
      goto __Cont;
    }
  }
  goto switch_break___0;
case_63:
  if ((unsigned int)quoting_style == 2U) {
    goto case_2___0;
  }
  if ((unsigned int)quoting_style == 3U) {
    goto case_3___0;
  }
  goto switch_default___1;
case_2___0:
  if (elide_outer_quotes) {
    goto force_outer_quoting_style;
  }
  goto switch_break___1;
case_3___0:
  if (flags & 4) {
    if (i + 2UL < argsize) {
      if ((int const) * (arg + (i + 1UL)) == 63) {
        if ((int const) * (arg + (i + 2UL)) == 33) {
          goto case_33;
        }
        if ((int const) * (arg + (i + 2UL)) == 39) {
          goto case_33;
        }
```

```
  if ((int const) * (arg + (i + 2UL)) == 40) {
    goto case_33;
  }
  if ((int const) * (arg + (i + 2UL)) == 41) {
    goto case_33;
  }
  if ((int const) * (arg + (i + 2UL)) == 45) {
    goto case_33;
  }
  if ((int const) * (arg + (i + 2UL)) == 47) {
    goto case_33;
  }
  if ((int const) * (arg + (i + 2UL)) == 60) {
    goto case_33;
  }
  if ((int const) * (arg + (i + 2UL)) == 61) {
    goto case_33;
  }
  if ((int const) * (arg + (i + 2UL)) == 62) {
    goto case_33;
  }
  goto switch_default___0;
case_33:
  if (elide_outer_quotes) {
    goto force_outer_quoting_style;
  }
  c = (unsigned char)*(arg + (i + 2UL));
  i += 2UL;
  while (1) {

    if (len < buffersize) {
      *(buffer + len) = (char)'?';
    }
    len++;
    goto while_break___7;
  }
while_break___7:;
  while (1) {

    if (len < buffersize) {
      *(buffer + len) = (char)'\"';
    }
    len++;
    goto while_break___8;
  }
while_break___8:;
  while (1) {

    if (len < buffersize) {
      *(buffer + len) = (char)'\"';
    }
    len++;
    goto while_break___9;
  }
while_break___9:;
  while (1) {

    if (len < buffersize) {
      *(buffer + len) = (char)'?';
    }
    len++;
    goto while_break___10;
  }
while_break___10:;
```

```
        goto switch_break___2;
      switch_default___0:
        goto switch_break___2;
      switch_break___2:;
      }
    }
  }
  goto switch_break___1;
switch_default___1:
  goto switch_break___1;
switch_break___1:;
  goto switch_break___0;
case_7___0:
  esc = (unsigned char)'a';
  goto c_escape;
case_8___0:
  esc = (unsigned char)'b';
  goto c_escape;
case_12:
  esc = (unsigned char)'f';
  goto c_escape;
case_10:
  esc = (unsigned char)'n';
  goto c_and_shell_escape;
case_13:
  esc = (unsigned char)'r';
  goto c_and_shell_escape;
case_9:
  esc = (unsigned char)'t';
  goto c_and_shell_escape;
case_11:
  esc = (unsigned char)'v';
  goto c_escape;
case_92:
  esc = c;
  if (backslash_escapes) {
    if (elide_outer_quotes) {
      if (quote_string_len) {
        goto store_c;
      }
    }
  }
c_and_shell_escape:
  if ((unsigned int)quoting_style == 2U) {
    if (elide_outer_quotes) {
      goto force_outer_quoting_style;
    }
  }
c_escape:
  if (backslash_escapes) {
    c = esc;
    goto store_escape;
  }
  goto switch_break___0;
case_123:
  if (argsize == 0xffffffffffffffffUL) {
    tmp___1 = (int const) * (arg + 1) == 0;
  } else {
    tmp___1 = argsize == 1UL;
  }
  if (!tmp___1) {
    goto switch_break___0;
  }
case_35:
```

```
  if (i != 0UL) {
    goto switch_break___0;
  }
case_32:
  if ((unsigned int)quoting_style == 2U) {
    if (elide_outer_quotes) {
      goto force_outer_quoting_style;
    }
  }
  goto switch_break___0;
case_39___0:
  if ((unsigned int)quoting_style == 2U) {
    if (elide_outer_quotes) {
      goto force_outer_quoting_style;
    }
    while (1) {

      if (len < buffersize) {
        *(buffer + len) = (char)'\'';
      }
      len++;
      goto while_break___11;
    }
  while_break___11:;
    while (1) {

      if (len < buffersize) {
        *(buffer + len) = (char)'\\';
      }
      len++;
      goto while_break___12;
    }
  while_break___12:;
    while (1) {

      if (len < buffersize) {
        *(buffer + len) = (char)'\'';
      }
      len++;
      goto while_break___13;
    }
  while_break___13:;
  }
  goto switch_break___0;
case_37:
  goto switch_break___0;
switch_default___2:
  if (unibyte_locale) {
    m = (size_t)1;
    tmp___2 = __ctype_b_loc();
    printable = (_Bool)(((int const) * (*tmp___2 + (int)c) & 16384) != 0);
  } else {
    memset((void *)(&mbstate), 0, sizeof(mbstate));
    m = (size_t)0;
    printable = (_Bool)1;
    if (argsize == 0xffffffffffffffffUL) {
      argsize = strlen(arg);
    }
    while (1) {
      tmp___3 = mbrtowc(&w, arg + (i + m), argsize - (i + m), &mbstate);
      bytes = tmp___3;
      if (bytes == 0UL) {
        goto while_break___14;
      } else {
```

```
  if (bytes == 0xffffffffffffffffUL) {
    printable = (_Bool)0;
    goto while_break___14;
  } else {
    if (bytes == 0xfffffffffffffffeUL) {
      printable = (_Bool)0;
      while (1) {

        if (i + m < argsize) {
          if (!*(arg + (i + m))) {
            goto while_break___15;
          }
        } else {
          goto while_break___15;
        }
        m++;
      }
    while_break___15:;
      goto while_break___14;
    } else {
      if (elide_outer_quotes) {
        if ((unsigned int)quoting_style == 2U) {
          j = (size_t)1;
          while (1) {

            if (!(j < bytes)) {
              goto while_break___16;
            }
            if ((int const) * (arg + ((i + m) + j)) == 91) {
              goto case_91___0;
            }
            if ((int const) * (arg + ((i + m) + j)) == 92) {
              goto case_91___0;
            }
            if ((int const) * (arg + ((i + m) + j)) == 94) {
              goto case_91___0;
            }
            if ((int const) * (arg + ((i + m) + j)) == 96) {
              goto case_91___0;
            }
            if ((int const) * (arg + ((i + m) + j)) == 124) {
              goto case_91___0;
            }
            goto switch_default___3;
          case_91___0:
            goto force_outer_quoting_style;
          switch_default___3:
            goto switch_break___3;
          switch_break___3:
            j++;
          }
        while_break___16:;
        }
      }
      tmp___4 = iswprint((wint_t)w);
      if (!tmp___4) {
        printable = (_Bool)0;
      }
      m += bytes;
    }
  }
}
tmp___5 = mbsinit((mbstate_t const *)(&mbstate));
if (tmp___5) {
```

```
      goto while_break___14;
    }
  }
while_break___14:;
}
if (1UL < m) {
  goto _L___0;
} else {
  if (backslash_escapes) {
    if (!printable) {
    _L___0:
      ilim = i + m;
      while (1) {

        if (backslash_escapes) {
          if (!printable) {
            if (elide_outer_quotes) {
              goto force_outer_quoting_style;
            }
            while (1) {

              if (len < buffersize) {
                *(buffer + len) = (char)'\\';
              }
              len++;
              goto while_break___18;
            }
          while_break___18:;
            while (1) {

              if (len < buffersize) {
                *(buffer + len) = (char)(48 + ((int)c >> 6));
              }
              len++;
              goto while_break___19;
            }
          while_break___19:;
            while (1) {

              if (len < buffersize) {
                *(buffer + len) = (char)(48 + (((int)c >> 3) & 7));
              }
              len++;
              goto while_break___20;
            }
          while_break___20:
            c = (unsigned char)(48 + ((int)c & 7));
          } else {
            goto _L;
          }
        } else {
        _L:
          if (is_right_quote) {
            while (1) {

              if (len < buffersize) {
                *(buffer + len) = (char)'\\';
              }
              len++;
              goto while_break___21;
            }
          while_break___21:
            is_right_quote = (_Bool)0;
          }
```

```
          }
          if (ilim <= i + 1UL) {
            goto while_break___17;
          }
          while (1) {

            if (len < buffersize) {
              *(buffer + len) = (char)c;
            }
            len++;
            goto while_break___22;
          }
        while_break___22:
          i++;
          c = (unsigned char)*(arg + i);
        }
      while_break___17:;
        goto store_c;
      }
    }
  }
switch_break___0:;
  if (backslash_escapes) {
    goto _L___3;
  } else {
    if (elide_outer_quotes) {
    _L___3:
      if (quote_these_too) {
        if (!(*(quote_these_too + (unsigned long)c / (sizeof(int) * 8UL)) &
              (unsigned int const)(1 << (unsigned long)c %
                                                (sizeof(int) * 8UL)))) {
          goto _L___2;
        }
      } else {
        goto _L___2;
      }
    } else {
    _L___2:
      if (!is_right_quote) {
        goto store_c;
      }
    }
  }
store_escape:
  if (elide_outer_quotes) {
    goto force_outer_quoting_style;
  }
  while (1) {

    if (len < buffersize) {
      *(buffer + len) = (char)'\\';
    }
    len++;
    goto while_break___23;
  }
while_break___23:;
store_c:
  while (1) {

    if (len < buffersize) {
      *(buffer + len) = (char)c;
    }
    len++;
    goto while_break___24;
```

```
      }
    while_break___24:;
    __Cont:
      i++;
    }
  while_break___3:;
    if (len == 0UL) {
      if ((unsigned int)quoting_style == 2U) {
        if (elide_outer_quotes) {
          goto force_outer_quoting_style;
        }
      }
    }
    if (quote_string) {
      if (!elide_outer_quotes) {
        while (1) {

          if (!*quote_string) {
            goto while_break___25;
          }
          while (1) {

            if (len < buffersize) {
              *(buffer + len) = (char)*quote_string;
            }
            len++;
            goto while_break___26;
          }
        while_break___26:
          quote_string++;
        }
      while_break___25:;
      }
    }
    if (len < buffersize) {
      *(buffer + len) = (char)'\000';
    }
    return (len);
  force_outer_quoting_style:
    tmp___7 = quotearg_buffer_restyled(
        buffer, buffersize, arg, argsize, quoting_style, flags & -3,
        (unsigned int const *)((void *)0), left_quote, right_quote);
    return (tmp___7);
  }
}
static char slot0[256];
static unsigned int nslots = 1U;
static struct slotvec slotvec0 = {sizeof(slot0), slot0};
static struct slotvec *slotvec = &slotvec0;
static char *quotearg_n_options(int n, char const *arg, size_t argsize,
                                struct quoting_options const *options) {
  int e;
  int *tmp;
  unsigned int n0;
  struct slotvec *sv;
  size_t n1;
  _Bool preallocated;
  int tmp___0;
  struct slotvec *tmp___1;
  size_t size;
  char *val;
  int flags;
  size_t qsize;
  size_t tmp___2;
```

```
  int *tmp___3;

  {
    tmp = __errno_location();
    e = *tmp;
    n0 = (unsigned int)n;
    sv = slotvec;
    if (n < 0) {
      abort();
    }
    if (nslots <= n0) {
      n1 = (size_t)(n0 + 1U);
      preallocated = (_Bool)((unsigned long)sv == (unsigned long)(&slotvec0));
      if (sizeof(ptrdiff_t) <= sizeof(size_t)) {
        tmp___0 = -1;
      } else {
        tmp___0 = -2;
      }
      if ((size_t)tmp___0 / sizeof(*sv) < n1) {
        xalloc_die();
      }
      if (preallocated) {
        tmp___1 = (struct slotvec *)((void *)0);
      } else {
        tmp___1 = sv;
      }
      sv = (struct slotvec *)xrealloc((void *)tmp___1, n1 * sizeof(*sv));
      slotvec = sv;
      if (preallocated) {
        *sv = slotvec0;
      }
      memset((void *)(sv + nslots), 0, (n1 - (size_t)nslots) * sizeof(*sv));
      nslots = (unsigned int)n1;
    }
    size = (sv + n)->size;
    val = (sv + n)->val;
    flags = (int)(options->flags | 1);
    tmp___2 = quotearg_buffer_restyled(
        val, size, arg, argsize, (enum quoting_style)options->style, flags,
        (unsigned int const *)(options->quote_these_too),
        (char const *)options->left_quote, (char const *)options->right_quote);
    qsize = tmp___2;
    if (size <= qsize) {
      size = qsize + 1UL;
      (sv + n)->size = size;
      if ((unsigned long)val != (unsigned long)(slot0)) {
        free((void *)val);
      }
      val = xcharalloc(size);
      (sv + n)->val = val;
      quotearg_buffer_restyled(val, size, arg, argsize,
                               (enum quoting_style)options->style, flags,
                               (unsigned int const *)(options->quote_these_too),
                               (char const *)options->left_quote,
                               (char const *)options->right_quote);
    }
    tmp___3 = __errno_location();
    *tmp___3 = e;
    return (val);
  }
}
char *quotearg_n_style(int n, enum quoting_style s, char const *arg) {
  struct quoting_options o;
  struct quoting_options tmp;
```

```
  char *tmp___0;

  {
    tmp = quoting_options_from_style(s);
    o = tmp;
    tmp___0 = quotearg_n_options(n, arg, (size_t)-1,
                                 (struct quoting_options const *)(&o));
    return (tmp___0);
  }
}
char *quotearg_char_mem(char const *arg, size_t argsize, char ch) {
  struct quoting_options options;
  char *tmp;

  {
    options = default_quoting_options;
    set_char_quoting(&options, ch, 1);
    tmp = quotearg_n_options(0, arg, argsize,
                             (struct quoting_options const *)(&options));
    return (tmp);
  }
}
char *quotearg_char(char const *arg, char ch) {
  char *tmp;

  {
    tmp = quotearg_char_mem(arg, (size_t)-1, ch);
    return (tmp);
  }
}
char *quotearg_colon(char const *arg) {
  char *tmp;

  {
    tmp = quotearg_char(arg, (char)':');
    return (tmp);
  }
}
char const *quote_n(int n, char const *name) {
  char const *tmp;

  {
    tmp = (char const *)quotearg_n_style(n, (enum quoting_style)6, name);
    return (tmp);
  }
}
char const *quote(char const *name) {
  char const *tmp;

  {
    tmp = quote_n(0, name);
    return (tmp);
  }
}
char const *program_name;
void set_program_name(char const *argv0);
extern char *program_invocation_name;
extern char *program_invocation_short_name;
extern int fputs(char const *__restrict __s, FILE *__restrict __stream);
char const *program_name = (char const *)((void *)0);
void set_program_name(char const *argv0) {
  char const *slash;
  char const *base;
  int tmp;
```

```
  int tmp___0;

  {
    if ((unsigned long)argv0 == (unsigned long)((void *)0)) {
      fputs("A NULL argv[0] was passed through an exec system call.\n", stderr);
      abort();
    }
    slash = (char const *)strrchr(argv0, '/');
    if ((unsigned long)slash != (unsigned long)((void *)0)) {
      base = slash + 1;
    } else {
      base = argv0;
    }
    if (base - argv0 >= 7L) {
      tmp___0 = strncmp(base - 7, "/.libs/", (size_t)7);
      if (tmp___0 == 0) {
        argv0 = base;
        tmp = strncmp(base, "lt-", (size_t)3);
        if (tmp == 0) {
          argv0 = base + 3;
          program_invocation_short_name = (char *)argv0;
        }
      }
    }
    program_name = argv0;
    program_invocation_name = (char *)argv0;
    return;
  }
}
extern DIR *(__attribute__((__nonnull__(1))) opendir)(char const *__name);
extern
    __attribute__((__nothrow__)) int(__attribute__((__nonnull__(1), __leaf__))
                                     dirfd)(DIR *__dirp);
DIR *opendir_safer(char const *name) {
  DIR *dp;
  DIR *tmp;
  int fd;
  int tmp___0;
  DIR *newdp;
  int e;
  int f;
  int tmp___1;
  int *tmp___2;
  int *tmp___3;

  {
    tmp = opendir(name);
    dp = tmp;
    if (dp) {
      tmp___0 = dirfd(dp);
      fd = tmp___0;
      if (0 <= fd) {
        if (fd <= 2) {
          tmp___1 = dup_safer(fd);
          f = tmp___1;
          newdp = fdopendir(f);
          tmp___2 = __errno_location();
          e = *tmp___2;
          if (!newdp) {
            close(f);
          }
          closedir(dp);
          tmp___3 = __errno_location();
          *tmp___3 = e;
```

```
          dp = newdp;
        }
      }
    }
    return (dp);
  }
}
int openat_safer(int fd, char const *file, int flags, ...);
int openat_safer(int fd, char const *file, int flags, ...) {
  mode_t mode;
  va_list ap;
  int tmp;
  int tmp___0;

  {
    mode = (mode_t)0;
    if (flags & 64) {
      __builtin_va_start(ap, flags);
      mode = __builtin_va_arg(ap, mode_t);
      __builtin_va_end(ap);
    }
    tmp = openat(fd, file, flags, mode);
    tmp___0 = fd_safer(tmp);
    return (tmp___0);
  }
}
int open_safer(char const *file, int flags, ...) {
  mode_t mode;
  va_list ap;
  int tmp;
  int tmp___0;

  {
    mode = (mode_t)0;
    if (flags & 64) {
      __builtin_va_start(ap, flags);
      mode = __builtin_va_arg(ap, mode_t);
      __builtin_va_end(ap);
    }
    tmp = open(file, flags, mode);
    tmp___0 = fd_safer(tmp);
    return (tmp___0);
  }
}
int(__attribute__((__nonnull__(1, 2))) mbscasecmp)(char const *s1,
                                                   char const *s2) {
  mbui_iterator_t iter1;
  mbui_iterator_t iter2;
  int cmp;
  wint_t tmp;
  wint_t tmp___0;
  int tmp___1;
  int tmp___2;
  int tmp___4;
  int tmp___5;
  int tmp___7;
  int tmp___8;
  int tmp___9;
  int tmp___10;
  int tmp___11;
  int tmp___12;
  int tmp___13;
  int tmp___14;
  int tmp___15;
```

```
int tmp___16;
unsigned char const *p1;
unsigned char const *p2;
unsigned char c1;
unsigned char c2;
int tmp___18;
unsigned short const **tmp___19;
int tmp___21;
unsigned short const **tmp___22;
size_t tmp___25;

{
  if ((unsigned long)s1 == (unsigned long)s2) {
    return (0);
  }
  tmp___25 = __ctype_get_mb_cur_max();
  if (tmp___25 > 1UL) {
    iter1.cur.ptr = s1;
    iter1.in_shift = (_Bool)0;
    memset((void *)(&iter1.state), '\000', sizeof(mbstate_t));
    iter1.next_done = (_Bool)0;
    iter2.cur.ptr = s2;
    iter2.in_shift = (_Bool)0;
    memset((void *)(&iter2.state), '\000', sizeof(mbstate_t));
    iter2.next_done = (_Bool)0;
    while (1) {
      mbuiter_multi_next(&iter1);
      if (iter1.cur.wc_valid) {
        if (iter1.cur.wc == 0) {
          tmp___13 = 0;
        } else {
          tmp___13 = 1;
        }
      } else {
        tmp___13 = 1;
      }
      if (tmp___13) {
        mbuiter_multi_next(&iter2);
        if (iter2.cur.wc_valid) {
          if (iter2.cur.wc == 0) {
            tmp___14 = 0;
          } else {
            tmp___14 = 1;
          }
        } else {
          tmp___14 = 1;
        }
        if (!tmp___14) {
          goto while_break;
        }
      } else {
        goto while_break;
      }
      if (iter1.cur.wc_valid) {
        if (iter2.cur.wc_valid) {
          tmp = towlower((wint_t)iter1.cur.wc);
          tmp___0 = towlower((wint_t)iter2.cur.wc);
          tmp___1 = (int)tmp - (int)tmp___0;
        } else {
          tmp___1 = -1;
        }
        tmp___12 = tmp___1;
      } else {
        if (iter2.cur.wc_valid) {
```

```
        tmp___11 = 1;
      } else {
        if (iter1.cur.bytes == iter2.cur.bytes) {
          tmp___2 = memcmp((void const *)iter1.cur.ptr,
                           (void const *)iter2.cur.ptr, iter1.cur.bytes);
          tmp___10 = tmp___2;
        } else {
          if (iter1.cur.bytes < iter2.cur.bytes) {
            tmp___5 = memcmp((void const *)iter1.cur.ptr,
                             (void const *)iter2.cur.ptr, iter1.cur.bytes);
            if (tmp___5 > 0) {
              tmp___4 = 1;
            } else {
              tmp___4 = -1;
            }
            tmp___9 = tmp___4;
          } else {
            tmp___8 = memcmp((void const *)iter1.cur.ptr,
                             (void const *)iter2.cur.ptr, iter2.cur.bytes);
            if (tmp___8 >= 0) {
              tmp___7 = 1;
            } else {
              tmp___7 = -1;
            }
            tmp___9 = tmp___7;
          }
          tmp___10 = tmp___9;
        }
        tmp___11 = tmp___10;
      }
      tmp___12 = tmp___11;
    }
    cmp = tmp___12;
    if (cmp != 0) {
      return (cmp);
    }
    iter1.cur.ptr += iter1.cur.bytes;
    iter1.next_done = (_Bool)0;
    iter2.cur.ptr += iter2.cur.bytes;
    iter2.next_done = (_Bool)0;
  }
while_break:
  mbuiter_multi_next(&iter1);
  if (iter1.cur.wc_valid) {
    if (iter1.cur.wc == 0) {
      tmp___15 = 0;
    } else {
      tmp___15 = 1;
    }
  } else {
    tmp___15 = 1;
  }
  if (tmp___15) {
    return (1);
  }
  mbuiter_multi_next(&iter2);
  if (iter2.cur.wc_valid) {
    if (iter2.cur.wc == 0) {
      tmp___16 = 0;
    } else {
      tmp___16 = 1;
    }
  } else {
    tmp___16 = 1;
```

```
      }
      if (tmp___16) {
        return (-1);
      }
      return (0);
    } else {
      p1 = (unsigned char const *)s1;
      p2 = (unsigned char const *)s2;
      while (1) {
        tmp___19 = __ctype_b_loc();
        if ((int const) * (*tmp___19 + (int)*p1) & 256) {
          tmp___18 = tolower((int)*p1);
          c1 = (unsigned char)tmp___18;
        } else {
          c1 = (unsigned char)*p1;
        }
        tmp___22 = __ctype_b_loc();
        if ((int const) * (*tmp___22 + (int)*p2) & 256) {
          tmp___21 = tolower((int)*p2);
          c2 = (unsigned char)tmp___21;
        } else {
          c2 = (unsigned char)*p2;
        }
        if ((int)c1 == 0) {
          goto while_break___0;
        }
        p1++;
        p2++;
        if (!((int)c1 == (int)c2)) {
          goto while_break___0;
        }
      }
    while_break___0:;
      return ((int)c1 - (int)c2);
    }
  }
}
unsigned int const is_basic_table[8] = {
    (unsigned int const)6656, (unsigned int const)4294967279U,
    (unsigned int const)4294967294U, (unsigned int const)2147483646};
extern __attribute__((__nothrow__, __noreturn__)) void(__attribute__((__leaf__))
                                                       exit)(int __status);
extern int optind;
extern __attribute__((__nothrow__)) int(__attribute__((__leaf__)) getopt_long)(
    int ___argc, char *const *___argv, char const *__shortopts,
    struct option const *__longopts, int *__longind);
#pragma weak pthread_mutex_init
#pragma weak pthread_mutex_lock
#pragma weak pthread_mutex_unlock
#pragma weak pthread_mutex_destroy
#pragma weak pthread_rwlock_init
#pragma weak pthread_rwlock_rdlock
#pragma weak pthread_rwlock_wrlock
#pragma weak pthread_rwlock_unlock
#pragma weak pthread_rwlock_destroy
#pragma weak pthread_once
#pragma weak pthread_cond_init
#pragma weak pthread_cond_wait
#pragma weak pthread_cond_signal
#pragma weak pthread_cond_broadcast
#pragma weak pthread_cond_destroy
#pragma weak pthread_mutexattr_init
#pragma weak pthread_mutexattr_settype
#pragma weak pthread_mutexattr_destroy
```

```
#pragma weak pthread_self
#pragma weak pthread_cancel
void i_ring_init(I_ring *ir, int default_val);
int i_ring_push(I_ring *ir, int val);
int i_ring_pop(I_ring *ir);
_Bool i_ring_empty(I_ring const *ir);
void i_ring_init(I_ring *ir, int default_val) {
  int i;

  {
    ir->ir_empty = (_Bool)1;
    ir->ir_front = 0U;
    ir->ir_back = 0U;
    i = 0;
    while (1) {

      if (!(i < 4)) {
        goto while_break;
      }
      ir->ir_data[i] = default_val;
      i++;
    }
  while_break:
    ir->ir_default_val = default_val;
    return;
  }
}
_Bool i_ring_empty(I_ring const *ir) {

  { return ((_Bool)ir->ir_empty); }
}
int i_ring_push(I_ring *ir, int val) {
  unsigned int dest_idx;
  int old_val;

  {
    dest_idx = (ir->ir_front + (unsigned int)(!ir->ir_empty)) % 4U;
    old_val = ir->ir_data[dest_idx];
    ir->ir_data[dest_idx] = val;
    ir->ir_front = dest_idx;
    if (dest_idx == ir->ir_back) {
      ir->ir_back = (ir->ir_back + (unsigned int)(!ir->ir_empty)) % 4U;
    }
    ir->ir_empty = (_Bool)0;
    return (old_val);
  }
}
int i_ring_pop(I_ring *ir) {
  int top_val;
  _Bool tmp;

  {
    tmp = i_ring_empty((I_ring const *)ir);
    if (tmp) {
      abort();
    }
    top_val = ir->ir_data[ir->ir_front];
    ir->ir_data[ir->ir_front] = ir->ir_default_val;
    if (ir->ir_front == ir->ir_back) {
      ir->ir_empty = (_Bool)1;
    } else {
      ir->ir_front = ((ir->ir_front + 4U) - 1U) % 4U;
    }
    return (top_val);
```

```
  }
}
_Bool(__attribute__((__warn_unused_result__))
      hash_rehash)(Hash_table *table___0, size_t candidate);
void *hash_delete(Hash_table *table___0, void const *entry);
__inline static size_t rotr_sz(size_t x, int n) {

  {
    return (((x >> n) | (x << (8UL * sizeof(x) - (unsigned long)n))) &
            0xffffffffffffffffUL);
  }
}
static struct hash_tuning const default_tuning = {
    (float)0.0, (float)1.0, (float)0.8, (float)1.414, (_Bool)0};
void *hash_lookup(Hash_table const *table___0, void const *entry) {
  struct hash_entry const *bucket;
  size_t tmp;
  struct hash_entry const *cursor;
  _Bool tmp___0;

  {
    tmp = (*(table___0->hasher))(entry, (size_t)table___0->n_buckets);
    bucket = (struct hash_entry const *)(table___0->bucket + tmp);
    if (!((unsigned long)bucket < (unsigned long)table___0->bucket_limit)) {
      abort();
    }
    if ((unsigned long)bucket->data == (unsigned long)((void *)0)) {
      return ((void *)0);
    }
    cursor = bucket;
    while (1) {

      if (!cursor) {
        goto while_break;
      }
      if ((unsigned long)entry == (unsigned long)cursor->data) {
        return ((void *)cursor->data);
      } else {
        tmp___0 = (*(table___0->comparator))(entry, (void const *)cursor->data);
        if (tmp___0) {
          return ((void *)cursor->data);
        }
      }
      cursor = (struct hash_entry const *)cursor->next;
    }
  while_break:;
    return ((void *)0);
  }
}
size_t hash_string(char const *string, size_t n_buckets) {
  size_t value;
  unsigned char ch;

  {
    value = (size_t)0;
    while (1) {
      ch = (unsigned char)*string;
      if (!ch) {
        goto while_break;
      }
      value = (value * 31UL + (size_t)ch) % n_buckets;
      string++;
    }
  while_break:;
```

```
    return (value);
  }
}
static _Bool is_prime(size_t candidate) {
  size_t divisor;
  size_t square;
  int tmp;

  {
    divisor = (size_t)3;
    square = divisor * divisor;
    while (1) {

      if (square < candidate) {
        if (!(candidate % divisor)) {
          goto while_break;
        }
      } else {
        goto while_break;
      }
      divisor++;
      square += 4UL * divisor;
      divisor++;
    }
  while_break:;
    if (candidate % divisor) {
      tmp = 1;
    } else {
      tmp = 0;
    }
    return ((_Bool)tmp);
  }
}
static size_t next_prime(size_t candidate) {
  _Bool tmp;

  {
    if (candidate < 10UL) {
      candidate = (size_t)10;
    }
    candidate |= 1UL;
    while (1) {

      if (0xffffffffffffffffUL != candidate) {
        tmp = is_prime(candidate);
        if (tmp) {
          goto while_break;
        }
      } else {
        goto while_break;
      }
      candidate += 2UL;
    }
  while_break:;
    return (candidate);
  }
}
static size_t raw_hasher(void const *data, size_t n) {
  size_t val;
  size_t tmp;

  {
    tmp = rotr_sz((size_t)data, 3);
    val = tmp;
```

```
    return (val % n);
  }
}
static _Bool raw_comparator(void const *a, void const *b) {

  { return ((_Bool)((unsigned long)a == (unsigned long)b)); }
}
static _Bool check_tuning(Hash_table *table___0) {
  Hash_tuning const *tuning;
  float epsilon;

  {
    tuning = table___0->tuning;
    if ((unsigned long)tuning == (unsigned long)(&default_tuning)) {
      return ((_Bool)1);
    }
    epsilon = 0.1f;
    if (epsilon < (float)tuning->growth_threshold) {
      if (tuning->growth_threshold < (float const)((float)1 - epsilon)) {
        if ((float)1 + epsilon < (float)tuning->growth_factor) {
          if ((float const)0 <= tuning->shrink_threshold) {
            if (tuning->shrink_threshold + (float const)epsilon <
                tuning->shrink_factor) {
              if (tuning->shrink_factor <= (float const)1) {
                if (tuning->shrink_threshold + (float const)epsilon <
                    tuning->growth_threshold) {
                  return ((_Bool)1);
                }
              }
            }
          }
        }
      }
    }
    table___0->tuning = &default_tuning;
    return ((_Bool)0);
  }
}
static size_t compute_bucket_size(size_t candidate, Hash_tuning const *tuning) {
  float new_candidate;
  int tmp;

  {
    if (!tuning->is_n_buckets) {
      new_candidate =
          (float)((float const)candidate / tuning->growth_threshold);
      if ((float)0xffffffffffffffffUL <= new_candidate) {
        return ((size_t)0);
      }
      candidate = (size_t)new_candidate;
    }
    candidate = next_prime(candidate);
    if (sizeof(ptrdiff_t) <= sizeof(size_t)) {
      tmp = -1;
    } else {
      tmp = -2;
    }
    if ((size_t)tmp / sizeof(struct hash_entry *) < candidate) {
      return ((size_t)0);
    }
    return (candidate);
  }
}
Hash_table *(__attribute__((__warn_unused_result__))
```

```
            hash_initialize)(size_t candidate, Hash_tuning const *tuning,
                             size_t (*hasher)(void const *, size_t),
                             _Bool (*comparator)(void const *, void const *),
                             void (*data_freer)(void *)) {
  Hash_table *table___0;
  _Bool tmp;

  {
    if ((unsigned long)hasher == (unsigned long)((void *)0)) {
      hasher = &raw_hasher;
    }
    if ((unsigned long)comparator == (unsigned long)((void *)0)) {
      comparator = &raw_comparator;
    }
    table___0 = (Hash_table *)malloc(sizeof(*table___0));
    if ((unsigned long)table___0 == (unsigned long)((void *)0)) {
      return ((Hash_table *)((void *)0));
    }
    if (!tuning) {
      tuning = &default_tuning;
    }
    table___0->tuning = tuning;
    tmp = check_tuning(table___0);
    if (!tmp) {
      goto fail;
    }
    table___0->n_buckets = compute_bucket_size(candidate, tuning);
    if (!table___0->n_buckets) {
      goto fail;
    }
    table___0->bucket = (struct hash_entry *)calloc(
        table___0->n_buckets, sizeof(*(table___0->bucket)));
    if ((unsigned long)table___0->bucket == (unsigned long)((void *)0)) {
      goto fail;
    }
    table___0->bucket_limit =
        (struct hash_entry const *)(table___0->bucket + table___0->n_buckets);
    table___0->n_buckets_used = (size_t)0;
    table___0->n_entries = (size_t)0;
    table___0->hasher = hasher;
    table___0->comparator = comparator;
    table___0->data_freer = data_freer;
    table___0->free_entry_list = (struct hash_entry *)((void *)0);
    return (table___0);
  fail:
    free((void *)table___0);
    return ((Hash_table *)((void *)0));
  }
}
void hash_free(Hash_table *table___0) {
  struct hash_entry *bucket;
  struct hash_entry *cursor;
  struct hash_entry *next;

  {
    if (table___0->data_freer) {
      if (table___0->n_entries) {
        bucket = table___0->bucket;
        while (1) {

          if (!((unsigned long)bucket <
                (unsigned long)table___0->bucket_limit)) {
            goto while_break;
          }
```

```
          if (bucket->data) {
            cursor = bucket;
            while (1) {

              if (!cursor) {
                goto while_break___0;
              }
              (*(table___0->data_freer))(cursor->data);
              cursor = cursor->next;
            }
          while_break___0:;
          }
          bucket++;
        }
      while_break:;
      }
    }
    bucket = table___0->bucket;
    while (1) {

      if (!((unsigned long)bucket < (unsigned long)table___0->bucket_limit)) {
        goto while_break___1;
      }
      cursor = bucket->next;
      while (1) {

        if (!cursor) {
          goto while_break___2;
        }
        next = cursor->next;
        free((void *)cursor);
        cursor = next;
      }
    while_break___2:
      bucket++;
    }
  while_break___1:
    cursor = table___0->free_entry_list;
    while (1) {

      if (!cursor) {
        goto while_break___3;
      }
      next = cursor->next;
      free((void *)cursor);
      cursor = next;
    }
  while_break___3:
    free((void *)table___0->bucket);
    free((void *)table___0);
    return;
  }
}
static struct hash_entry *allocate_entry(Hash_table *table___0) {
  struct hash_entry *new;

  {
    if (table___0->free_entry_list) {
      new = table___0->free_entry_list;
      table___0->free_entry_list = new->next;
    } else {
      new = (struct hash_entry *)malloc(sizeof(*new));
    }
    return (new);
```

```
  }
}
static void free_entry(Hash_table *table___0, struct hash_entry *entry) {

  {
    entry->data = (void *)0;
    entry->next = table___0->free_entry_list;
    table___0->free_entry_list = entry;
    return;
  }
}
static void *hash_find_entry(Hash_table *table___0, void const *entry,
                             struct hash_entry **bucket_head, _Bool delete) {
  struct hash_entry *bucket;
  size_t tmp;
  struct hash_entry *cursor;
  void *data;
  struct hash_entry *next;
  _Bool tmp___0;
  void *data___0;
  struct hash_entry *next___0;
  _Bool tmp___1;

  {
    tmp = (*(table___0->hasher))(entry, table___0->n_buckets);
    bucket = table___0->bucket + tmp;
    if (!((unsigned long)bucket < (unsigned long)table___0->bucket_limit)) {
      abort();
    }
    *bucket_head = bucket;
    if ((unsigned long)bucket->data == (unsigned long)((void *)0)) {
      return ((void *)0);
    }
    if ((unsigned long)entry == (unsigned long)bucket->data) {
      goto _L;
    } else {
      tmp___0 = (*(table___0->comparator))(entry, (void const *)bucket->data);
      if (tmp___0) {
      _L:
        data = bucket->data;
        if (delete) {
          if (bucket->next) {
            next = bucket->next;
            *bucket = *next;
            free_entry(table___0, next);
          } else {
            bucket->data = (void *)0;
          }
        }
        return (data);
      }
    }
    cursor = bucket;
    while (1) {

      if (!cursor->next) {
        goto while_break;
      }
      if ((unsigned long)entry == (unsigned long)(cursor->next)->data) {
        goto _L___0;
      } else {
        tmp___1 = (*(table___0->comparator))(
            entry, (void const *)(cursor->next)->data);
        if (tmp___1) {
```

```
        _L___0:
          data___0 = (cursor->next)->data;
          if (delete) {
            next___0 = cursor->next;
            cursor->next = next___0->next;
            free_entry(table___0, next___0);
          }
          return (data___0);
        }
      }
      cursor = cursor->next;
    }
  while_break:;
    return ((void *)0);
  }
}
static _Bool transfer_entries(Hash_table *dst, Hash_table *src, _Bool safe) {
  struct hash_entry *bucket;
  struct hash_entry *cursor;
  struct hash_entry *next;
  void *data;
  struct hash_entry *new_bucket;
  size_t tmp;
  size_t tmp___0;
  struct hash_entry *new_entry;
  struct hash_entry *tmp___1;

  {
    bucket = src->bucket;
    while (1) {

      if (!((unsigned long)bucket < (unsigned long)src->bucket_limit)) {
        goto while_break;
      }
      if (bucket->data) {
        cursor = bucket->next;
        while (1) {

          if (!cursor) {
            goto while_break___0;
          }
          data = cursor->data;
          tmp = (*(dst->hasher))((void const *)data, dst->n_buckets);
          new_bucket = dst->bucket + tmp;
          if (!((unsigned long)new_bucket < (unsigned long)dst->bucket_limit)) {
            abort();
          }
          next = cursor->next;
          if (new_bucket->data) {
            cursor->next = new_bucket->next;
            new_bucket->next = cursor;
          } else {
            new_bucket->data = data;
            (dst->n_buckets_used)++;
            free_entry(dst, cursor);
          }
          cursor = next;
        }
      while_break___0:
        data = bucket->data;
        bucket->next = (struct hash_entry *)((void *)0);
        if (safe) {
          goto __Cont;
        }
```

```
        tmp___0 = (*(dst->hasher))((void const *)data, dst->n_buckets);
        new_bucket = dst->bucket + tmp___0;
        if (!((unsigned long)new_bucket < (unsigned long)dst->bucket_limit)) {
          abort();
        }
        if (new_bucket->data) {
          tmp___1 = allocate_entry(dst);
          new_entry = tmp___1;
          if ((unsigned long)new_entry == (unsigned long)((void *)0)) {
            return ((_Bool)0);
          }
          new_entry->data = data;
          new_entry->next = new_bucket->next;
          new_bucket->next = new_entry;
        } else {
          new_bucket->data = data;
          (dst->n_buckets_used)++;
        }
        bucket->data = (void *)0;
        (src->n_buckets_used)--;
      }
    __Cont:
      bucket++;
    }
  while_break:;
    return ((_Bool)1);
  }
}
_Bool(__attribute__((__warn_unused_result__))
      hash_rehash)(Hash_table *table___0, size_t candidate) {
  Hash_table storage;
  Hash_table *new_table;
  size_t new_size;
  size_t tmp;
  _Bool tmp___0;
  _Bool tmp___1;
  _Bool tmp___2;

  {
    tmp = compute_bucket_size(candidate, table___0->tuning);
    new_size = tmp;
    if (!new_size) {
      return ((_Bool)0);
    }
    if (new_size == table___0->n_buckets) {
      return ((_Bool)1);
    }
    new_table = &storage;
    new_table->bucket =
        (struct hash_entry *)calloc(new_size, sizeof(*(new_table->bucket)));
    if ((unsigned long)new_table->bucket == (unsigned long)((void *)0)) {
      return ((_Bool)0);
    }
    new_table->n_buckets = new_size;
    new_table->bucket_limit =
        (struct hash_entry const *)(new_table->bucket + new_size);
    new_table->n_buckets_used = (size_t)0;
    new_table->n_entries = (size_t)0;
    new_table->tuning = table___0->tuning;
    new_table->hasher = table___0->hasher;
    new_table->comparator = table___0->comparator;
    new_table->data_freer = table___0->data_freer;
    new_table->free_entry_list = table___0->free_entry_list;
    tmp___0 = transfer_entries(new_table, table___0, (_Bool)0);
```

```
    if (tmp___0) {
      free((void *)table___0->bucket);
      table___0->bucket = new_table->bucket;
      table___0->bucket_limit = new_table->bucket_limit;
      table___0->n_buckets = new_table->n_buckets;
      table___0->n_buckets_used = new_table->n_buckets_used;
      table___0->free_entry_list = new_table->free_entry_list;
      return ((_Bool)1);
    }
    table___0->free_entry_list = new_table->free_entry_list;
    tmp___1 = transfer_entries(table___0, new_table, (_Bool)1);
    if (tmp___1) {
      tmp___2 = transfer_entries(table___0, new_table, (_Bool)0);
      if (!tmp___2) {
        abort();
      }
    } else {
      abort();
    }
    free((void *)new_table->bucket);
    return ((_Bool)0);
  }
}
void *(__attribute__((__warn_unused_result__))
       hash_insert)(Hash_table *table___0, void const *entry) {
  void *data;
  struct hash_entry *bucket;
  Hash_tuning const *tuning;
  float candidate;
  float tmp;
  _Bool tmp___0;
  void *tmp___1;
  struct hash_entry *new_entry;
  struct hash_entry *tmp___2;

  {
    if (!entry) {
      abort();
    }
    data = hash_find_entry(table___0, entry, &bucket, (_Bool)0);
    if ((unsigned long)data != (unsigned long)((void *)0)) {
      return (data);
    }
    if ((float const)table___0->n_buckets_used >
        (table___0->tuning)->growth_threshold *
            (float const)table___0->n_buckets) {
      check_tuning(table___0);
      if ((float const)table___0->n_buckets_used >
          (table___0->tuning)->growth_threshold *
              (float const)table___0->n_buckets) {
        tuning = table___0->tuning;
        if (tuning->is_n_buckets) {
          tmp = (float)((float const)table___0->n_buckets *
                        tuning->growth_factor);
        } else {
          tmp = (float)(((float const)table___0->n_buckets *
                         tuning->growth_factor) *
                        tuning->growth_threshold);
        }
        candidate = tmp;
        if ((float)0xffffffffffffffffUL <= candidate) {
          return ((void *)0);
        }
        tmp___0 = hash_rehash(table___0, (size_t)candidate);
```

```
        if (!tmp___0) {
          return ((void *)0);
        }
        tmp___1 = hash_find_entry(table___0, entry, &bucket, (_Bool)0);
        if ((unsigned long)tmp___1 != (unsigned long)((void *)0)) {
          abort();
        }
      }
    }
    if (bucket->data) {
      tmp___2 = allocate_entry(table___0);
      new_entry = tmp___2;
      if ((unsigned long)new_entry == (unsigned long)((void *)0)) {
        return ((void *)0);
      }
      new_entry->data = (void *)entry;
      new_entry->next = bucket->next;
      bucket->next = new_entry;
      (table___0->n_entries)++;
      return ((void *)entry);
    }
    bucket->data = (void *)entry;
    (table___0->n_entries)++;
    (table___0->n_buckets_used)++;
    return ((void *)entry);
  }
}
void *hash_delete(Hash_table *table___0, void const *entry) {
  void *data;
  struct hash_entry *bucket;
  Hash_tuning const *tuning;
  size_t candidate;
  float tmp;
  struct hash_entry *cursor;
  struct hash_entry *next;
  _Bool tmp___0;

  {
    data = hash_find_entry(table___0, entry, &bucket, (_Bool)1);
    if (!data) {
      return ((void *)0);
    }
    (table___0->n_entries)--;
    if (!bucket->data) {
      (table___0->n_buckets_used)--;
      if ((float const)table___0->n_buckets_used <
          (table___0->tuning)->shrink_threshold *
              (float const)table___0->n_buckets) {
        check_tuning(table___0);
        if ((float const)table___0->n_buckets_used <
            (table___0->tuning)->shrink_threshold *
                (float const)table___0->n_buckets) {
          tuning = table___0->tuning;
          if (tuning->is_n_buckets) {
            tmp = (float)((float const)table___0->n_buckets *
                          tuning->shrink_factor);
          } else {
            tmp = (float)(((float const)table___0->n_buckets *
                           tuning->shrink_factor) *
                          tuning->growth_threshold);
          }
          candidate = (size_t)tmp;
          tmp___0 = hash_rehash(table___0, candidate);
          if (!tmp___0) {
```

```
            cursor = table___0->free_entry_list;
            while (1) {

              if (!cursor) {
                goto while_break;
              }
              next = cursor->next;
              free((void *)cursor);
              cursor = next;
            }
          while_break:
            table___0->free_entry_list = (struct hash_entry *)((void *)0);
          }
        }
      }
      return (data);
    }
}
size_t hash_pjw(void const *x, size_t tablesize);
size_t triple_hash(void const *x, size_t table_size) {
  struct F_triple const *p;
  size_t tmp;
  size_t tmp___0;

  {
    p = (struct F_triple const *)x;
    tmp___0 = hash_pjw((void const *)p->name, table_size);
    tmp = tmp___0;
    return ((tmp ^ (unsigned long)p->st_ino) % table_size);
  }
}
_Bool triple_compare_ino_str(void const *x, void const *y) {
  struct F_triple const *a;
  struct F_triple const *b;
  int tmp___0;
  int tmp___1;

  {
    a = (struct F_triple const *)x;
    b = (struct F_triple const *)y;
    if (a->st_ino == b->st_ino) {
      if (a->st_dev == b->st_dev) {
        tmp___1 = strcmp((char const *)a->name, (char const *)b->name);
        if (tmp___1 == 0) {
          tmp___0 = 1;
        } else {
          tmp___0 = 0;
        }
      } else {
        tmp___0 = 0;
      }
    } else {
      tmp___0 = 0;
    }
    return ((_Bool)tmp___0);
  }
}
void triple_free(void *x) {
  struct F_triple *a;

  {
    a = (struct F_triple *)x;
    free((void *)a->name);
```

```
    free((void *)a);
    return;
  }
}
size_t hash_pjw(void const *x, size_t tablesize) {
  char const *s;
  size_t h;

  {
    h = (size_t)0;
    s = (char const *)x;
    while (1) {

      if (!*s) {
        goto while_break;
      }
      h = (unsigned long)*s + ((h << 9) | (h >> (sizeof(size_t) * 8UL - 9UL)));
      s++;
    }
  while_break:;
    return (h % tablesize);
  }
}
extern __attribute__((__nothrow__)) char *(__attribute__((__leaf__))
                                           setlocale)(int __category,
                                                      char const *__locale);
extern __attribute__((__nothrow__)) int(
    __attribute__((__nonnull__(2, 3), __leaf__))
    fstatat)(int __fd, char const *__restrict __file,
             struct stat *__restrict __buf, int __flag);
__attribute__((__nothrow__)) int(__attribute__((__warn_unused_result__,
                                                __leaf__)) fts_close)(FTS *sp);
__attribute__((__nothrow__))
FTSENT *(__attribute__((__warn_unused_result__, __leaf__)) fts_read)(FTS *sp);
__attribute__((__nothrow__)) int(__attribute__((__leaf__))
                                 fts_set)(FTS *sp __attribute__((__unused__)),
                                          FTSENT *p, int instr);
extern void(__attribute__((__nonnull__(1, 4)))
            qsort)(void *__base, size_t __nmemb, size_t __size,
                   int (*__compar)(void const *, void const *));
static FTSENT *fts_alloc(FTS *sp, char const *name, size_t namelen);
static FTSENT *fts_build(FTS *sp, int type);
static void fts_lfree(FTSENT *head);
static void fts_load(FTS *sp, FTSENT *p);
static size_t fts_maxarglen(char *const *argv);
static void fts_padjust(FTS *sp, FTSENT *head);
static _Bool fts_palloc(FTS *sp, size_t more);
static FTSENT *fts_sort(FTS *sp, FTSENT *head, size_t nitems);
static unsigned short fts_stat(FTS *sp, FTSENT *p, _Bool follow);
static int fts_safe_changedir(FTS *sp, FTSENT *p, int fd, char const *dir);
static _Bool AD_compare(void const *x, void const *y) {
  struct Active_dir const *ax;
  struct Active_dir const *ay;
  int tmp;

  {
    ax = (struct Active_dir const *)x;
    ay = (struct Active_dir const *)y;
    if (ax->ino == ay->ino) {
      if (ax->dev == ay->dev) {
        tmp = 1;
      } else {
        tmp = 0;
      }
```

```
    } else {
      tmp = 0;
    }
    return ((_Bool)tmp);
  }
}
static size_t AD_hash(void const *x, size_t table_size) {
  struct Active_dir const *ax;

  {
    ax = (struct Active_dir const *)x;
    return ((uintmax_t)ax->ino % table_size);
  }
}
static _Bool setup_dir(FTS *fts) {

  {
    if (fts->fts_options & 258) {
      fts->fts_cycle.ht =
          hash_initialize((size_t)31, (Hash_tuning const *)((void *)0),
                          &AD_hash, &AD_compare, (void (*)(void *))(&free));
      if (!fts->fts_cycle.ht) {
        return ((_Bool)0);
      }
    } else {
      fts->fts_cycle.state =
          (struct cycle_check_state *)malloc(sizeof(*(fts->fts_cycle.state)));
      if (!fts->fts_cycle.state) {
        return ((_Bool)0);
      }
      cycle_check_init(fts->fts_cycle.state);
    }
    return ((_Bool)1);
  }
}
static _Bool enter_dir(FTS *fts, FTSENT *ent) {
  struct stat const *st;
  struct Active_dir *ad;
  struct Active_dir *tmp;
  struct Active_dir *ad_from_table;
  _Bool tmp___0;

  {
    if (fts->fts_options & 258) {
      st = (struct stat const *)(ent->fts_statp);
      tmp = (struct Active_dir *)malloc(sizeof(*ad));
      ad = tmp;
      if (!ad) {
        return ((_Bool)0);
      }
      ad->dev = (dev_t)st->st_dev;
      ad->ino = (ino_t)st->st_ino;
      ad->fts_ent = ent;
      ad_from_table =
          (struct Active_dir *)hash_insert(fts->fts_cycle.ht, (void const *)ad);
      if ((unsigned long)ad_from_table != (unsigned long)ad) {
        free((void *)ad);
        if (!ad_from_table) {
          return ((_Bool)0);
        }
        ent->fts_cycle = ad_from_table->fts_ent;
        ent->fts_info = (unsigned short)2;
      }
    } else {
```

```
      tmp___0 = cycle_check(fts->fts_cycle.state,
                            (struct stat const *)(ent->fts_statp));
      if (tmp___0) {
        ent->fts_cycle = ent;
        ent->fts_info = (unsigned short)2;
      }
    }
    return ((_Bool)1);
  }
}
static void leave_dir(FTS *fts, FTSENT *ent) {
  struct stat const *st;
  struct Active_dir obj;
  void *found;
  FTSENT *parent;

  {
    st = (struct stat const *)(ent->fts_statp);
    if (fts->fts_options & 258) {
      obj.dev = (dev_t)st->st_dev;
      obj.ino = (ino_t)st->st_ino;
      found = hash_delete(fts->fts_cycle.ht, (void const *)(&obj));
      if (!found) {
        abort();
      }
      free(found);
    } else {
      parent = ent->fts_parent;
      if ((unsigned long)parent != (unsigned long)((void *)0)) {
        if (0L <= parent->fts_level) {
          while (1) {

            if ((fts->fts_cycle.state)->chdir_counter == 0UL) {
              abort();
            }
            if ((fts->fts_cycle.state)->dev_ino.st_ino == (ino_t)st->st_ino) {
              if ((fts->fts_cycle.state)->dev_ino.st_dev == (dev_t)st->st_dev) {
                (fts->fts_cycle.state)->dev_ino.st_dev =
                    parent->fts_statp[0].st_dev;
                (fts->fts_cycle.state)->dev_ino.st_ino =
                    parent->fts_statp[0].st_ino;
              }
            }
            goto while_break;
          }
        while_break:;
        }
      }
    }
    return;
  }
}
static void free_dir(FTS *sp) {

  {
    if (sp->fts_options & 258) {
      if (sp->fts_cycle.ht) {
        hash_free(sp->fts_cycle.ht);
      }
    } else {
      free((void *)sp->fts_cycle.state);
    }
    return;
  }
```

```
}
static void fd_ring_clear(I_ring *fd_ring) {
  int fd;
  int tmp;
  _Bool tmp___0;

  {
    while (1) {
      tmp___0 = i_ring_empty((I_ring const *)fd_ring);
      if (tmp___0) {
        goto while_break;
      }
      tmp = i_ring_pop(fd_ring);
      fd = tmp;
      if (0 <= fd) {
        close(fd);
      }
    }
  while_break:;
    return;
  }
}
static void fts_set_stat_required(FTSENT *p, _Bool required) {

  {
    while (1) {

      if (!((int)p->fts_info == 11)) {
        abort();
      }
      goto while_break;
    }
  while_break:;
    if (required) {
      p->fts_statp[0].st_size = (__off_t)2;
    } else {
      p->fts_statp[0].st_size = (__off_t)1;
    }
    return;
  }
}
__inline static DIR *opendirat(int fd, char const *dir) {
  int new_fd;
  int tmp;
  DIR *dirp;
  int saved_errno;
  int *tmp___0;
  int *tmp___1;

  {
    tmp = openat_safer(fd, dir, 67840);
    new_fd = tmp;
    if (new_fd < 0) {
      return ((DIR *)((void *)0));
    }
    set_cloexec_flag(new_fd, (_Bool)1);
    dirp = fdopendir(new_fd);
    if ((unsigned long)dirp == (unsigned long)((void *)0)) {
      tmp___0 = __errno_location();
      saved_errno = *tmp___0;
      close(new_fd);
      tmp___1 = __errno_location();
      *tmp___1 = saved_errno;
    }
```

```
    return (dirp);
  }
}
static void cwd_advance_fd(FTS *sp, int fd, _Bool chdir_down_one) {
  int old;
  int prev_fd_in_slot;
  int tmp;

  {
    old = sp->fts_cwd_fd;
    while (1) {

      if (!(old != fd)) {
        if (!(old == -100)) {
          abort();
        }
      }
      goto while_break;
    }
  while_break:;
    if (chdir_down_one) {
      tmp = i_ring_push(&sp->fts_fd_ring, old);
      prev_fd_in_slot = tmp;
      if (0 <= prev_fd_in_slot) {
        close(prev_fd_in_slot);
      }
    } else {
      if (!(sp->fts_options & 4)) {
        if (0 <= old) {
          close(old);
        }
      }
    }
    sp->fts_cwd_fd = fd;
    return;
  }
}
__inline static int diropen(FTS const *sp, char const *dir) {
  int open_flags;
  int tmp;
  int fd;
  int tmp___0;
  int tmp___1;
  int tmp___2;

  {
    if (sp->fts_options & 16) {
      tmp = 131072;
    } else {
      tmp = 0;
    }
    open_flags = 67840 | tmp;
    if (sp->fts_options & 512) {
      tmp___0 = openat_safer((int)sp->fts_cwd_fd, dir, open_flags);
      tmp___2 = tmp___0;
    } else {
      tmp___1 = open_safer(dir, open_flags);
      tmp___2 = tmp___1;
    }
    fd = tmp___2;
    if (0 <= fd) {
      set_cloexec_flag(fd, (_Bool)1);
    }
    return (fd);
```

```
  }
}
__attribute__((__nothrow__))
FTS *(__attribute__((__warn_unused_result__, __leaf__))
      fts_open)(char *const *argv, int options,
                int (*compar)(FTSENT const **, FTSENT const **));
FTS *(__attribute__((__warn_unused_result__, __leaf__))
      fts_open)(char *const *argv, int options,
                int (*compar)(FTSENT const **, FTSENT const **)) {
  register FTS *sp;
  register FTSENT *p;
  register FTSENT *root;
  register size_t nitems;
  FTSENT *parent;
  FTSENT *tmp;
  _Bool defer_stat;
  int *tmp___0;
  int *tmp___1;
  int *tmp___2;
  size_t maxarglen;
  size_t tmp___4;
  size_t tmp___5;
  _Bool tmp___6;
  int tmp___7;
  size_t len;
  size_t tmp___8;
  struct _ftsent *tmp___9;
  _Bool tmp___10;
  int tmp___11;

  {
    parent = (FTSENT *)((void *)0);
    tmp = (FTSENT *)((void *)0);
    if (options & -2048) {
      tmp___0 = __errno_location();
      *tmp___0 = 22;
      return ((FTS *)((void *)0));
    }
    if (options & 4) {
      if (options & 512) {
        tmp___1 = __errno_location();
        *tmp___1 = 22;
        return ((FTS *)((void *)0));
      }
    }
    if (!(options & 18)) {
      tmp___2 = __errno_location();
      *tmp___2 = 22;
      return ((FTS *)((void *)0));
    }
    sp = (FTS *)malloc(sizeof(FTS));
    if ((unsigned long)sp == (unsigned long)((void *)0)) {
      return ((FTS *)((void *)0));
    }
    memset((void *)sp, 0, sizeof(FTS));
    sp->fts_compar = compar;
    sp->fts_options = options;
    if (sp->fts_options & 2) {
      sp->fts_options |= 4;
      sp->fts_options &= -513;
    }
    sp->fts_cwd_fd = -100;
    tmp___4 = fts_maxarglen(argv);
    maxarglen = tmp___4;
```

```
if (maxarglen > 4096UL) {
  tmp___5 = maxarglen;
} else {
  tmp___5 = (size_t)4096;
}
tmp___6 = fts_palloc(sp, tmp___5);
if (!tmp___6) {
  goto mem1;
}
if ((unsigned long)*argv != (unsigned long)((void *)0)) {
  parent = fts_alloc(sp, "", (size_t)0);
  if ((unsigned long)parent == (unsigned long)((void *)0)) {
    goto mem2;
  }
  parent->fts_level = (ptrdiff_t)-1;
}
if ((unsigned long)compar == (unsigned long)((void *)0)) {
  tmp___7 = 1;
} else {
  if (sp->fts_options & 1024) {
    tmp___7 = 1;
  } else {
    tmp___7 = 0;
  }
}
defer_stat = (_Bool)tmp___7;
root = (FTSENT *)((void *)0);
nitems = (size_t)0;
while (1) {

  if (!((unsigned long)*argv != (unsigned long)((void *)0))) {
    goto while_break;
  }
  tmp___8 = strlen((char const *)*argv);
  len = tmp___8;
  p = fts_alloc(sp, (char const *)*argv, len);
  if ((unsigned long)p == (unsigned long)((void *)0)) {
    goto mem3;
  }
  p->fts_level = (ptrdiff_t)0;
  p->fts_parent = parent;
  p->fts_accpath = p->fts_name;
  if (defer_stat) {
    if ((unsigned long)root != (unsigned long)((void *)0)) {
      p->fts_info = (unsigned short)11;
      fts_set_stat_required(p, (_Bool)1);
    } else {
      p->fts_info = fts_stat(sp, p, (_Bool)0);
    }
  } else {
    p->fts_info = fts_stat(sp, p, (_Bool)0);
  }
  if (compar) {
    p->fts_link = root;
    root = p;
  } else {
    p->fts_link = (struct _ftsent *)((void *)0);
    if ((unsigned long)root == (unsigned long)((void *)0)) {
      root = p;
      tmp = root;
    } else {
      tmp->fts_link = p;
      tmp = p;
    }
```

```
      }
      argv++;
      nitems++;
    }
  while_break:;
    if (compar) {
      if (nitems > 1UL) {
        root = fts_sort(sp, root, nitems);
      }
    }
    tmp___9 = fts_alloc(sp, "", (size_t)0);
    sp->fts_cur = tmp___9;
    if ((unsigned long)tmp___9 == (unsigned long)((void *)0)) {
      goto mem3;
    }
    (sp->fts_cur)->fts_link = root;
    (sp->fts_cur)->fts_info = (unsigned short)9;
    tmp___10 = setup_dir(sp);
    if (!tmp___10) {
      goto mem3;
    }
    if (!(sp->fts_options & 4)) {
      if (!(sp->fts_options & 512)) {
        tmp___11 = diropen((FTS const *)sp, ".");
        sp->fts_rfd = tmp___11;
        if (tmp___11 < 0) {
          sp->fts_options |= 4;
        }
      }
    }
    i_ring_init(&sp->fts_fd_ring, -1);
    return (sp);
  mem3:
    fts_lfree(root);
    free((void *)parent);
  mem2:
    free((void *)sp->fts_path);
  mem1:
    free((void *)sp);
    return ((FTS *)((void *)0));
  }
}
static void fts_load(FTS *sp, FTSENT *p) {
  register size_t len;
  register char *cp;
  size_t tmp;
  char *tmp___0;

  {
    tmp = p->fts_namelen;
    p->fts_pathlen = tmp;
    len = tmp;
    memmove((void *)sp->fts_path, (void const *)(p->fts_name), len + 1UL);
    cp = strrchr((char const *)(p->fts_name), '/');
    if (cp) {
      if ((unsigned long)cp != (unsigned long)(p->fts_name)) {
        cp++;
        len = strlen((char const *)cp);
        memmove((void *)(p->fts_name), (void const *)cp, len + 1UL);
        p->fts_namelen = len;
      } else {
        if (*(cp + 1)) {
          cp++;
          len = strlen((char const *)cp);
```

```
          memmove((void *)(p->fts_name), (void const *)cp, len + 1UL);
          p->fts_namelen = len;
        }
      }
    }
    tmp___0 = sp->fts_path;
    p->fts_path = tmp___0;
    p->fts_accpath = tmp___0;
    return;
  }
}
__attribute__((__nothrow__)) int(__attribute__((__warn_unused_result__,
                                                __leaf__)) fts_close)(FTS *sp);
int(__attribute__((__warn_unused_result__, __leaf__)) fts_close)(FTS *sp) {
  register FTSENT *freep;
  register FTSENT *p;
  int saved_errno;
  int *tmp;
  int tmp___0;
  int *tmp___1;
  int tmp___2;
  int *tmp___3;
  int tmp___4;
  int *tmp___5;

  {
    saved_errno = 0;
    if (sp->fts_cur) {
      p = sp->fts_cur;
      while (1) {

        if (!(p->fts_level >= 0L)) {
          goto while_break;
        }
        freep = p;
        if ((unsigned long)p->fts_link != (unsigned long)((void *)0)) {
          p = p->fts_link;
        } else {
          p = p->fts_parent;
        }
        free((void *)freep);
      }
    while_break:
      free((void *)p);
    }
    if (sp->fts_child) {
      fts_lfree(sp->fts_child);
    }
    free((void *)sp->fts_array);
    free((void *)sp->fts_path);
    if (sp->fts_options & 512) {
      if (0 <= sp->fts_cwd_fd) {
        tmp___0 = close(sp->fts_cwd_fd);
        if (tmp___0) {
          tmp = __errno_location();
          saved_errno = *tmp;
        }
      }
    } else {
      if (!(sp->fts_options & 4)) {
        tmp___2 = fchdir(sp->fts_rfd);
        if (tmp___2) {
          tmp___1 = __errno_location();
          saved_errno = *tmp___1;
```

```
        }
        tmp___4 = close(sp->fts_rfd);
        if (tmp___4) {
          if (saved_errno == 0) {
            tmp___3 = __errno_location();
            saved_errno = *tmp___3;
          }
        }
      }
    }
    fd_ring_clear(&sp->fts_fd_ring);
    if (sp->fts_leaf_optimization_works_ht) {
      hash_free(sp->fts_leaf_optimization_works_ht);
    }
    free_dir(sp);
    free((void *)sp);
    if (saved_errno) {
      tmp___5 = __errno_location();
      *tmp___5 = saved_errno;
      return (-1);
    }
    return (0);
  }
}
extern
    __attribute__((__nothrow__)) int(__attribute__((__nonnull__(2), __leaf__))
                                     fstatfs)(int __fildes,
                                              struct statfs *__buf);
static _Bool dirent_inode_sort_may_be_useful(int dir_fd) {
  struct statfs fs_buf;
  int tmp;

  {
    tmp = fstatfs(dir_fd, &fs_buf);
    if (tmp != 0) {
      return ((_Bool)1);
    }
    if (fs_buf.f_type == 16914836L) {
      goto case_16914836;
    }
    if (fs_buf.f_type == 26985L) {
      goto case_16914836;
    }
    goto switch_default;
  case_16914836:
    return ((_Bool)0);
  switch_default:
    return ((_Bool)1);

    return ((_Bool)0);
  }
}
static _Bool leaf_optimization_applies(int dir_fd) {
  struct statfs fs_buf;
  int tmp;

  {
    tmp = fstatfs(dir_fd, &fs_buf);
    if (tmp != 0) {
      return ((_Bool)0);
    }
    if (fs_buf.f_type == 1382369651L) {
      goto case_1382369651;
    }
```

```
    goto switch_default;
  case_1382369651:
    return ((_Bool)1);
  switch_default:
    return ((_Bool)0);

    return ((_Bool)0);
  }
}
static size_t LCO_hash(void const *x, size_t table_size) {
  struct LCO_ent const *ax;

  {
    ax = (struct LCO_ent const *)x;
    return ((uintmax_t)ax->st_dev % table_size);
  }
}
static _Bool LCO_compare(void const *x, void const *y) {
  struct LCO_ent const *ax;
  struct LCO_ent const *ay;

  {
    ax = (struct LCO_ent const *)x;
    ay = (struct LCO_ent const *)y;
    return ((_Bool)(ax->st_dev == ay->st_dev));
  }
}
static _Bool link_count_optimize_ok(FTSENT const *p) {
  FTS *sp;
  Hash_table *h;
  struct LCO_ent tmp;
  struct LCO_ent *ent;
  _Bool opt_ok;
  struct LCO_ent *t2;
  struct hash_table *tmp___0;

  {
    sp = (FTS *)p->fts_fts;
    h = sp->fts_leaf_optimization_works_ht;
    if (!(sp->fts_options & 512)) {
      return ((_Bool)0);
    }
    if ((unsigned long)h == (unsigned long)((void *)0)) {
      tmp___0 =
          hash_initialize((size_t)13, (Hash_tuning const *)((void *)0),
                          &LCO_hash, &LCO_compare, (void (*)(void *))(&free));
      sp->fts_leaf_optimization_works_ht = tmp___0;
      h = tmp___0;
      if ((unsigned long)h == (unsigned long)((void *)0)) {
        return ((_Bool)0);
      }
    }
    tmp.st_dev = (dev_t)p->fts_statp[0].st_dev;
    ent = (struct LCO_ent *)hash_lookup((Hash_table const *)h,
                                        (void const *)(&tmp));
    if (ent) {
      return (ent->opt_ok);
    }
    t2 = (struct LCO_ent *)malloc(sizeof(*t2));
    if ((unsigned long)t2 == (unsigned long)((void *)0)) {
      return ((_Bool)0);
    }
    opt_ok = leaf_optimization_applies(sp->fts_cwd_fd);
    t2->opt_ok = opt_ok;
```

```
    t2->st_dev = (dev_t)p->fts_statp[0].st_dev;
    ent = (struct LCO_ent *)hash_insert(h, (void const *)t2);
    if ((unsigned long)ent == (unsigned long)((void *)0)) {
      free((void *)t2);
      return ((_Bool)0);
    }
    while (1) {

      if (!((unsigned long)ent == (unsigned long)t2)) {
        abort();
      }
      goto while_break;
    }
  while_break:;
    return (opt_ok);
  }
}
__attribute__((__nothrow__))
FTSENT *(__attribute__((__warn_unused_result__, __leaf__)) fts_read)(FTS *sp);
FTSENT *(__attribute__((__warn_unused_result__, __leaf__)) fts_read)(FTS *sp) {
  register FTSENT *p;
  register FTSENT *tmp;
  register unsigned short instr;
  register char *t;
  int *tmp___0;
  int tmp___1;
  int *tmp___2;
  int tmp___3;
  struct _ftsent *tmp___4;
  int tmp___5;
  int tmp___6;
  int tmp___7;
  int tmp___8;
  int tmp___9;
  int *tmp___10;
  int tmp___11;
  size_t tmp___12;
  char *tmp___13;
  FTSENT *parent;
  _Bool tmp___14;
  int *tmp___15;
  _Bool tmp___16;
  int *tmp___17;
  struct _ftsent *tmp___18;
  int *tmp___19;
  int tmp___20;
  int tmp___21;
  int tmp___22;
  int tmp___23;
  int tmp___24;
  int saved_errno;
  int *tmp___25;
  int *tmp___26;
  int *tmp___27;
  int tmp___28;
  int tmp___29;
  int *tmp___30;
  int tmp___31;
  FTSENT *tmp___32;

  {
    if ((unsigned long)sp->fts_cur == (unsigned long)((void *)0)) {
      return ((FTSENT *)((void *)0));
    } else {
```

```
  if (sp->fts_options & 8192) {
    return ((FTSENT *)((void *)0));
  }
}
p = sp->fts_cur;
instr = p->fts_instr;
p->fts_instr = (unsigned short)3;
if ((int)instr == 1) {
  p->fts_info = fts_stat(sp, p, (_Bool)0);
  return (p);
}
if ((int)instr == 2) {
  if ((int)p->fts_info == 12) {
    goto _L;
  } else {
    if ((int)p->fts_info == 13) {
    _L:
      p->fts_info = fts_stat(sp, p, (_Bool)1);
      if ((int)p->fts_info == 1) {
        if (!(sp->fts_options & 4)) {
          tmp___1 = diropen((FTS const *)sp, ".");
          p->fts_symfd = tmp___1;
          if (tmp___1 < 0) {
            tmp___0 = __errno_location();
            p->fts_errno = *tmp___0;
            p->fts_info = (unsigned short)7;
          } else {
            p->fts_flags = (unsigned short)((int)p->fts_flags | 2);
          }
        }
      }
      goto check_for_dir;
    }
  }
}
if ((int)p->fts_info == 1) {
  if ((int)instr == 4) {
    goto _L___0;
  } else {
    if (sp->fts_options & 64) {
      if (p->fts_statp[0].st_dev != sp->fts_dev) {
      _L___0:
        if ((int)p->fts_flags & 2) {
          close(p->fts_symfd);
        }
        if (sp->fts_child) {
          fts_lfree(sp->fts_child);
          sp->fts_child = (struct _ftsent *)((void *)0);
        }
        p->fts_info = (unsigned short)6;
        while (1) {
          leave_dir(sp, p);
          goto while_break;
        }
      while_break:;
        return (p);
      }
    }
  }
  if ((unsigned long)sp->fts_child != (unsigned long)((void *)0)) {
    if (sp->fts_options & 4096) {
      sp->fts_options &= -4097;
      fts_lfree(sp->fts_child);
      sp->fts_child = (struct _ftsent *)((void *)0);
```

```
      }
    }
    if ((unsigned long)sp->fts_child != (unsigned long)((void *)0)) {
      tmp___3 = fts_safe_changedir(sp, p, -1, (char const *)p->fts_accpath);
      if (tmp___3) {
        tmp___2 = __errno_location();
        p->fts_errno = *tmp___2;
        p->fts_flags = (unsigned short)((int)p->fts_flags | 1);
        p = sp->fts_child;
        while (1) {

          if (!((unsigned long)p != (unsigned long)((void *)0))) {
            goto while_break___0;
          }
          p->fts_accpath = (p->fts_parent)->fts_accpath;
          p = p->fts_link;
        }
      while_break___0:;
      }
    } else {
      tmp___4 = fts_build(sp, 3);
      sp->fts_child = tmp___4;
      if ((unsigned long)tmp___4 == (unsigned long)((void *)0)) {
        if (sp->fts_options & 8192) {
          return ((FTSENT *)((void *)0));
        }
        if (p->fts_errno) {
          if ((int)p->fts_info != 4) {
            p->fts_info = (unsigned short)7;
          }
        }
        while (1) {
          leave_dir(sp, p);
          goto while_break___1;
        }
      while_break___1:;
        return (p);
      }
    }
    p = sp->fts_child;
    sp->fts_child = (struct _ftsent *)((void *)0);
    goto name;
  }
next:
  tmp = p;
  p = p->fts_link;
  if ((unsigned long)p != (unsigned long)((void *)0)) {
    sp->fts_cur = p;
    free((void *)tmp);
    if (p->fts_level == 0L) {
      fd_ring_clear(&sp->fts_fd_ring);
      if (!(sp->fts_options & 4)) {
        if (sp->fts_options & 512) {
          if (sp->fts_options & 512) {
            tmp___5 = -100;
          } else {
            tmp___5 = sp->fts_rfd;
          }
          cwd_advance_fd(sp, tmp___5, (_Bool)1);
          tmp___8 = 0;
        } else {
          if (sp->fts_options & 512) {
            tmp___6 = -100;
          } else {
```

```
          tmp___6 = sp->fts_rfd;
        }
        tmp___7 = fchdir(tmp___6);
        tmp___8 = tmp___7;
      }
      if (tmp___8) {
        tmp___9 = 1;
      } else {
        tmp___9 = 0;
      }
    } else {
      tmp___9 = 0;
    }
    if (tmp___9) {
      sp->fts_options |= 8192;
      return ((FTSENT *)((void *)0));
    }
    free_dir(sp);
    fts_load(sp, p);
    setup_dir(sp);
    goto check_for_dir;
  }
  if ((int)p->fts_instr == 4) {
    goto next;
  }
  if ((int)p->fts_instr == 2) {
    p->fts_info = fts_stat(sp, p, (_Bool)1);
    if ((int)p->fts_info == 1) {
      if (!(sp->fts_options & 4)) {
        tmp___11 = diropen((FTS const *)sp, ".");
        p->fts_symfd = tmp___11;
        if (tmp___11 < 0) {
          tmp___10 = __errno_location();
          p->fts_errno = *tmp___10;
          p->fts_info = (unsigned short)7;
        } else {
          p->fts_flags = (unsigned short)((int)p->fts_flags | 2);
        }
      }
    }
    p->fts_instr = (unsigned short)3;
  }
name:
  if ((int)*((p->fts_parent)->fts_path +
             ((p->fts_parent)->fts_pathlen - 1UL)) == 47) {
    tmp___12 = (p->fts_parent)->fts_pathlen - 1UL;
  } else {
    tmp___12 = (p->fts_parent)->fts_pathlen;
  }
  t = sp->fts_path + tmp___12;
  tmp___13 = t;
  t++;
  *tmp___13 = (char)'/';
  memmove((void *)t, (void const *)(p->fts_name), p->fts_namelen + 1UL);
check_for_dir:
  sp->fts_cur = p;
  if ((int)p->fts_info == 11) {
    if (p->fts_statp[0].st_size == 2L) {
      parent = p->fts_parent;
      if (0L < p->fts_level) {
        if (parent->fts_n_dirs_remaining == 0UL) {
          if (sp->fts_options & 8) {
            if (sp->fts_options & 16) {
              tmp___14 = link_count_optimize_ok((FTSENT const *)parent);
```

```
              if (!tmp___14) {
                goto _L___4;
              }
            } else {
              goto _L___4;
            }
          } else {
            goto _L___4;
          }
        } else {
          goto _L___4;
        }
      } else {
      _L___4:
        p->fts_info = fts_stat(sp, p, (_Bool)0);
        if ((p->fts_statp[0].st_mode & 61440U) == 16384U) {
          if (p->fts_level != 0L) {
            if (parent->fts_n_dirs_remaining) {
              (parent->fts_n_dirs_remaining)--;
            }
          }
        }
      }
    } else {
      while (1) {

        if (!(p->fts_statp[0].st_size == 1L)) {
          abort();
        }
        goto while_break___2;
      }
    while_break___2:;
    }
  }
  if ((int)p->fts_info == 1) {
    if (p->fts_level == 0L) {
      sp->fts_dev = p->fts_statp[0].st_dev;
    }
    tmp___16 = enter_dir(sp, p);
    if (!tmp___16) {
      tmp___15 = __errno_location();
      *tmp___15 = 12;
      return ((FTSENT *)((void *)0));
    }
  }
  return (p);
}
p = tmp->fts_parent;
sp->fts_cur = p;
free((void *)tmp);
if (p->fts_level == -1L) {
  free((void *)p);
  tmp___17 = __errno_location();
  *tmp___17 = 0;
  tmp___18 = (struct _ftsent *)((void *)0);
  sp->fts_cur = tmp___18;
  return (tmp___18);
}
while (1) {

  if (!((int)p->fts_info != 11)) {
    abort();
  }
  goto while_break___3;
```

```
  }
while_break___3:
  *(sp->fts_path + p->fts_pathlen) = (char)'\000';
  if (p->fts_level == 0L) {
    fd_ring_clear(&sp->fts_fd_ring);
    if (!(sp->fts_options & 4)) {
      if (sp->fts_options & 512) {
        if (sp->fts_options & 512) {
          tmp___20 = -100;
        } else {
          tmp___20 = sp->fts_rfd;
        }
        cwd_advance_fd(sp, tmp___20, (_Bool)1);
        tmp___23 = 0;
      } else {
        if (sp->fts_options & 512) {
          tmp___21 = -100;
        } else {
          tmp___21 = sp->fts_rfd;
        }
        tmp___22 = fchdir(tmp___21);
        tmp___23 = tmp___22;
      }
      if (tmp___23) {
        tmp___24 = 1;
      } else {
        tmp___24 = 0;
      }
    } else {
      tmp___24 = 0;
    }
    if (tmp___24) {
      tmp___19 = __errno_location();
      p->fts_errno = *tmp___19;
      sp->fts_options |= 8192;
    }
  } else {
    if ((int)p->fts_flags & 2) {
      if (!(sp->fts_options & 4)) {
        if (sp->fts_options & 512) {
          cwd_advance_fd(sp, p->fts_symfd, (_Bool)1);
          tmp___29 = 0;
        } else {
          tmp___28 = fchdir(p->fts_symfd);
          tmp___29 = tmp___28;
        }
        if (tmp___29) {
          tmp___25 = __errno_location();
          saved_errno = *tmp___25;
          close(p->fts_symfd);
          tmp___26 = __errno_location();
          *tmp___26 = saved_errno;
          tmp___27 = __errno_location();
          p->fts_errno = *tmp___27;
          sp->fts_options |= 8192;
        }
      }
      close(p->fts_symfd);
    } else {
      if (!((int)p->fts_flags & 1)) {
        tmp___31 = fts_safe_changedir(sp, p->fts_parent, -1, "..");
        if (tmp___31) {
          tmp___30 = __errno_location();
          p->fts_errno = *tmp___30;
```

```
            sp->fts_options |= 8192;
          }
        }
      }
    }
    if (p->fts_errno) {
      p->fts_info = (unsigned short)7;
    } else {
      p->fts_info = (unsigned short)6;
    }
    if (p->fts_errno == 0) {
      while (1) {
        leave_dir(sp, p);
        goto while_break___4;
      }
    while_break___4:;
    }
    if (sp->fts_options & 8192) {
      tmp___32 = (FTSENT *)((void *)0);
    } else {
      tmp___32 = p;
    }
    return (tmp___32);
  }
}
__attribute__((__nothrow__)) int(__attribute__((__leaf__))
                                 fts_set)(FTS *sp __attribute__((__unused__)),
                                          FTSENT *p, int instr);
int(__attribute__((__leaf__)) fts_set)(FTS *sp __attribute__((__unused__)),
                                       FTSENT *p, int instr) {
  int *tmp;

  {
    if (instr != 0) {
      if (instr != 1) {
        if (instr != 2) {
          if (instr != 3) {
            if (instr != 4) {
              tmp = __errno_location();
              *tmp = 22;
              return (1);
            }
          }
        }
      }
    }
    p->fts_instr = (unsigned short)instr;
    return (0);
  }
}
static int fts_compare_ino(struct _ftsent const **a, struct _ftsent const **b) {
  int tmp;
  int tmp___0;

  {
    if ((*(a + 0))->fts_statp[0].st_ino < (*(b + 0))->fts_statp[0].st_ino) {
      tmp___0 = -1;
    } else {
      if ((*(b + 0))->fts_statp[0].st_ino < (*(a + 0))->fts_statp[0].st_ino) {
        tmp = 1;
      } else {
        tmp = 0;
      }
      tmp___0 = tmp;
```

```
    }
    return (tmp___0);
  }
}
static void set_stat_type(struct stat *st, unsigned int dtype) {
  mode_t type;

  {
    if (dtype == 6U) {
      goto case_6;
    }
    if (dtype == 2U) {
      goto case_2;
    }
    if (dtype == 4U) {
      goto case_4;
    }
    if (dtype == 1U) {
      goto case_1;
    }
    if (dtype == 10U) {
      goto case_10;
    }
    if (dtype == 8U) {
      goto case_8;
    }
    if (dtype == 12U) {
      goto case_12;
    }
    goto switch_default;
  case_6:
    type = (mode_t)24576;
    goto switch_break;
  case_2:
    type = (mode_t)8192;
    goto switch_break;
  case_4:
    type = (mode_t)16384;
    goto switch_break;
  case_1:
    type = (mode_t)4096;
    goto switch_break;
  case_10:
    type = (mode_t)40960;
    goto switch_break;
  case_8:
    type = (mode_t)32768;
    goto switch_break;
  case_12:
    type = (mode_t)49152;
    goto switch_break;
  switch_default:
    type = (mode_t)0;
  switch_break:
    st->st_mode = type;
    return;
  }
}
static FTSENT *fts_build(FTS *sp, int type) {
  register struct dirent *dp;
  register FTSENT *p;
  register FTSENT *head;
  register size_t nitems;
  FTSENT *cur;
```

```
FTSENT *tail;
DIR *dirp;
void *oldaddr;
int saved_errno;
_Bool descend;
_Bool doadjust;
ptrdiff_t level;
nlink_t nlinks;
_Bool nostat;
size_t len;
size_t maxlen;
size_t new_len;
char *cp;
int *tmp;
DIR *tmp___0;
DIR *tmp___1;
int *tmp___2;
_Bool tmp___3;
int tmp___4;
int dir_fd;
int tmp___5;
int *tmp___6;
int tmp___7;
char *tmp___8;
_Bool is_dir;
size_t tmp___9;
int *tmp___10;
int *tmp___11;
size_t tmp___12;
_Bool tmp___13;
size_t tmp___14;
size_t tmp___15;
int *tmp___16;
_Bool skip_stat;
int tmp___17;
int tmp___18;
int tmp___19;
int tmp___20;
int tmp___21;
int tmp___22;
int tmp___23;
int tmp___24;
int tmp___25;
int tmp___26;
_Bool tmp___27;

{
  cur = sp->fts_cur;
  if (!(sp->fts_options & 4)) {
    if (sp->fts_options & 512) {
      tmp___0 = opendirat(sp->fts_cwd_fd, (char const *)cur->fts_accpath);
      dirp = tmp___0;
    } else {
      tmp___1 = opendir_safer((char const *)cur->fts_accpath);
      dirp = tmp___1;
    }
  } else {
    tmp___1 = opendir_safer((char const *)cur->fts_accpath);
    dirp = tmp___1;
  }
  if ((unsigned long)dirp == (unsigned long)((void *)0)) {
    if (type == 3) {
      cur->fts_info = (unsigned short)4;
      tmp = __errno_location();
```

```
    cur->fts_errno = *tmp;
  }
  return ((FTSENT *)((void *)0));
}
if ((int)cur->fts_info == 11) {
  cur->fts_info = fts_stat(sp, cur, (_Bool)0);
} else {
  if (sp->fts_options & 256) {
    while (1) {
      leave_dir(sp, cur);
      goto while_break;
    }
  while_break:
    fts_stat(sp, cur, (_Bool)0);
    tmp___3 = enter_dir(sp, cur);
    if (!tmp___3) {
      tmp___2 = __errno_location();
      *tmp___2 = 12;
      return ((FTSENT *)((void *)0));
    }
  }
}
if (type == 2) {
  nlinks = (nlink_t)0;
  nostat = (_Bool)0;
} else {
  if (sp->fts_options & 8) {
    if (sp->fts_options & 16) {
      if (sp->fts_options & 32) {
        tmp___4 = 0;
      } else {
        tmp___4 = 2;
      }
      nlinks = cur->fts_statp[0].st_nlink - (__nlink_t)tmp___4;
      nostat = (_Bool)1;
    } else {
      nlinks = (nlink_t)-1;
      nostat = (_Bool)0;
    }
  } else {
    nlinks = (nlink_t)-1;
    nostat = (_Bool)0;
  }
}
if (nlinks) {
  goto _L___0;
} else {
  if (type == 3) {
  _L___0:
    tmp___5 = dirfd(dirp);
    dir_fd = tmp___5;
    if (sp->fts_options & 512) {
      if (0 <= dir_fd) {
        dir_fd = dup_safer(dir_fd);
        set_cloexec_flag(dir_fd, (_Bool)1);
      }
    }
    if (dir_fd < 0) {
      goto _L;
    } else {
      tmp___7 =
          fts_safe_changedir(sp, cur, dir_fd, (char const *)((void *)0));
      if (tmp___7) {
      _L:
```

```
        if (nlinks) {
          if (type == 3) {
            tmp___6 = __errno_location();
            cur->fts_errno = *tmp___6;
          }
        }
        cur->fts_flags = (unsigned short)((int)cur->fts_flags | 1);
        descend = (_Bool)0;
        closedir(dirp);
        if (sp->fts_options & 512) {
          if (0 <= dir_fd) {
            close(dir_fd);
          }
        }
        dirp = (DIR *)((void *)0);
      } else {
        descend = (_Bool)1;
      }
    }
  } else {
    descend = (_Bool)0;
  }
}
if ((int)*(cur->fts_path + (cur->fts_pathlen - 1UL)) == 47) {
  len = cur->fts_pathlen - 1UL;
} else {
  len = cur->fts_pathlen;
}
if (sp->fts_options & 4) {
  cp = sp->fts_path + len;
  tmp___8 = cp;
  cp++;
  *tmp___8 = (char)'/';
} else {
  cp = (char *)((void *)0);
}
len++;
maxlen = sp->fts_pathlen - len;
level = cur->fts_level + 1L;
doadjust = (_Bool)0;
tail = (FTSENT *)((void *)0);
head = tail;
nitems = (size_t)0;
while (1) {

  if (dirp) {
    dp = readdir(dirp);
    if (!dp) {
      goto while_break___0;
    }
  } else {
    goto while_break___0;
  }
  if (!(sp->fts_options & 32)) {
    if ((int)dp->d_name[0] == 46) {
      if (!dp->d_name[1]) {
        goto __Cont;
      } else {
        if ((int)dp->d_name[1] == 46) {
          if (!dp->d_name[2]) {
            goto __Cont;
          }
        }
      }
```

```
  }
}
tmp___9 = strlen((char const *)(dp->d_name));
p = fts_alloc(sp, (char const *)(dp->d_name), tmp___9);
if ((unsigned long)p == (unsigned long)((void *)0)) {
  goto mem1;
}
tmp___14 = strlen((char const *)(dp->d_name));
if (tmp___14 >= maxlen) {
  oldaddr = (void *)sp->fts_path;
  tmp___12 = strlen((char const *)(dp->d_name));
  tmp___13 = fts_palloc(sp, (tmp___12 + len) + 1UL);
  if (!tmp___13) {
  mem1:
    tmp___10 = __errno_location();
    saved_errno = *tmp___10;
    free((void *)p);
    fts_lfree(head);
    closedir(dirp);
    cur->fts_info = (unsigned short)7;
    sp->fts_options |= 8192;
    tmp___11 = __errno_location();
    *tmp___11 = saved_errno;
    return ((FTSENT *)((void *)0));
  }
  if ((unsigned long)oldaddr != (unsigned long)sp->fts_path) {
    doadjust = (_Bool)1;
    if (sp->fts_options & 4) {
      cp = sp->fts_path + len;
    }
  }
  maxlen = sp->fts_pathlen - len;
}
tmp___15 = strlen((char const *)(dp->d_name));
new_len = len + tmp___15;
if (new_len < len) {
  free((void *)p);
  fts_lfree(head);
  closedir(dirp);
  cur->fts_info = (unsigned short)7;
  sp->fts_options |= 8192;
  tmp___16 = __errno_location();
  *tmp___16 = 36;
  return ((FTSENT *)((void *)0));
}
p->fts_level = level;
p->fts_parent = sp->fts_cur;
p->fts_pathlen = new_len;
p->fts_statp[0].st_ino = dp->d_ino;
if (sp->fts_options & 4) {
  p->fts_accpath = p->fts_path;
  memmove((void *)cp, (void const *)(p->fts_name), p->fts_namelen + 1UL);
} else {
  p->fts_accpath = p->fts_name;
}
if ((unsigned long)sp->fts_compar == (unsigned long)((void *)0)) {
  goto _L___1;
} else {
  if (sp->fts_options & 1024) {
  _L___1:
    if (sp->fts_options & 16) {
      if (sp->fts_options & 8) {
        if ((int)dp->d_type != 0) {
          if (!((int)dp->d_type == 4)) {
```

```
              tmp___17 = 1;
            } else {
              tmp___17 = 0;
            }
          } else {
            tmp___17 = 0;
          }
        } else {
          tmp___17 = 0;
        }
      } else {
        tmp___17 = 0;
      }
      skip_stat = (_Bool)tmp___17;
      p->fts_info = (unsigned short)11;
      set_stat_type(p->fts_statp, (unsigned int)dp->d_type);
      fts_set_stat_required(p, (_Bool)(!skip_stat));
      if (sp->fts_options & 16) {
        if ((int)dp->d_type == 4) {
          tmp___18 = 1;
        } else {
          tmp___18 = 0;
        }
      } else {
        tmp___18 = 0;
      }
      is_dir = (_Bool)tmp___18;
    } else {
      p->fts_info = fts_stat(sp, p, (_Bool)0);
      if ((int)p->fts_info == 1) {
        tmp___19 = 1;
      } else {
        if ((int)p->fts_info == 2) {
          tmp___19 = 1;
        } else {
          if ((int)p->fts_info == 5) {
            tmp___19 = 1;
          } else {
            tmp___19 = 0;
          }
        }
      }
      is_dir = (_Bool)tmp___19;
    }
  }
  if (nlinks > 0UL) {
    if (is_dir) {
      nlinks -= (nlink_t)nostat;
    }
  }
  p->fts_link = (struct _ftsent *)((void *)0);
  if ((unsigned long)head == (unsigned long)((void *)0)) {
    tail = p;
    head = tail;
  } else {
    tail->fts_link = p;
    tail = p;
  }
  nitems++;
__Cont:;
  }
while_break___0:;
  if (dirp) {
    closedir(dirp);
```

```
}
if (doadjust) {
  fts_padjust(sp, head);
}
if (sp->fts_options & 4) {
  if (len == sp->fts_pathlen) {
    cp--;
  } else {
    if (nitems == 0UL) {
      cp--;
    }
  }
  *cp = (char)'\000';
}
if (descend) {
  if (type == 1) {
    goto _L___2;
  } else {
    if (!nitems) {
    _L___2:
      if (cur->fts_level == 0L) {
        fd_ring_clear(&sp->fts_fd_ring);
        if (!(sp->fts_options & 4)) {
          if (sp->fts_options & 512) {
            if (sp->fts_options & 512) {
              tmp___20 = -100;
            } else {
              tmp___20 = sp->fts_rfd;
            }
            cwd_advance_fd(sp, tmp___20, (_Bool)1);
            tmp___23 = 0;
          } else {
            if (sp->fts_options & 512) {
              tmp___21 = -100;
            } else {
              tmp___21 = sp->fts_rfd;
            }
            tmp___22 = fchdir(tmp___21);
            tmp___23 = tmp___22;
          }
          if (tmp___23) {
            tmp___24 = 1;
          } else {
            tmp___24 = 0;
          }
        } else {
          tmp___24 = 0;
        }
        tmp___26 = tmp___24;
      } else {
        tmp___25 = fts_safe_changedir(sp, cur->fts_parent, -1, "..");
        tmp___26 = tmp___25;
      }
      if (tmp___26) {
        cur->fts_info = (unsigned short)7;
        sp->fts_options |= 8192;
        fts_lfree(head);
        return ((FTSENT *)((void *)0));
      }
    }
  }
}
if (!nitems) {
  if (type == 3) {
```

```
        cur->fts_info = (unsigned short)6;
      }
      fts_lfree(head);
      return ((FTSENT *)((void *)0));
    }
    if (nitems > 10000UL) {
      if (!sp->fts_compar) {
        if (sp->fts_options & 512) {
          tmp___27 = dirent_inode_sort_may_be_useful(sp->fts_cwd_fd);
          if (tmp___27) {
            sp->fts_compar = &fts_compare_ino;
            head = fts_sort(sp, head, nitems);
            sp->fts_compar = (int (*)(struct _ftsent const **, struct _ftsent
const **))((void *)0);
          }
        }
      }
    }
    if (sp->fts_compar) {
      if (nitems > 1UL) {
        head = fts_sort(sp, head, nitems);
      }
    }
    return (head);
  }
}
static unsigned short fts_stat(FTS *sp, FTSENT *p, _Bool follow) {
  struct stat *sbp;
  int saved_errno;
  int *tmp;
  int *tmp___0;
  int *tmp___1;
  int tmp___2;
  int tmp___3;
  int *tmp___4;
  int tmp___5;
  int tmp___6;
  int tmp___7;

  {
    sbp = p->fts_statp;
    if (p->fts_level == 0L) {
      if (sp->fts_options & 1) {
        follow = (_Bool)1;
      }
    }
    if (sp->fts_options & 2) {
      goto _L;
    } else {
      if (follow) {
      _L:
        tmp___3 = stat((char const *)p->fts_accpath, sbp);
        if (tmp___3) {
          tmp = __errno_location();
          saved_errno = *tmp;
          tmp___1 = __errno_location();
          if (*tmp___1 == 2) {
            tmp___2 = lstat((char const *)p->fts_accpath, sbp);
            if (tmp___2 == 0) {
              tmp___0 = __errno_location();
              *tmp___0 = 0;
              return ((unsigned short)13);
            }
          }
```

```
          p->fts_errno = saved_errno;
          goto err;
        }
      } else {
        tmp___5 =
            fstatat(sp->fts_cwd_fd, (char const *)p->fts_accpath, sbp, 256);
        if (tmp___5) {
          tmp___4 = __errno_location();
          p->fts_errno = *tmp___4;
        err:
          memset((void *)sbp, 0, sizeof(struct stat));
          return ((unsigned short)10);
        }
      }
    }
    if ((sbp->st_mode & 61440U) == 16384U) {
      if (sp->fts_options & 32) {
        tmp___6 = 0;
      } else {
        tmp___6 = 2;
      }
      p->fts_n_dirs_remaining = sbp->st_nlink - (__nlink_t)tmp___6;
      if ((int)p->fts_name[0] == 46) {
        if (!p->fts_name[1]) {
          goto _L___0;
        } else {
          if ((int)p->fts_name[1] == 46) {
            if (!p->fts_name[2]) {
            _L___0:
              if (p->fts_level == 0L) {
                tmp___7 = 1;
              } else {
                tmp___7 = 5;
              }
              return ((unsigned short)tmp___7);
            }
          }
        }
      }
      return ((unsigned short)1);
    }
    if ((sbp->st_mode & 61440U) == 40960U) {
      return ((unsigned short)12);
    }
    if ((sbp->st_mode & 61440U) == 32768U) {
      return ((unsigned short)8);
    }
    return ((unsigned short)3);
  }
}
static int fts_compar(void const *a, void const *b) {
  FTSENT const **pa;
  FTSENT const **pb;
  int tmp;

  {
    pa = (FTSENT const **)a;
    pb = (FTSENT const **)b;
    tmp = (*(((*(pa + 0))->fts_fts)->fts_compar))(pa, pb);
    return (tmp);
  }
}
static FTSENT *fts_sort(FTS *sp, FTSENT *head, size_t nitems) {
  register FTSENT **ap;
```

```
register FTSENT *p;
FTSENT *dummy = 0;
int (*compare)(void const *, void const *);
int (*tmp)(void const *, void const *);
FTSENT **a;
FTSENT **tmp___0;

{
  if (sizeof(&dummy) == sizeof(void *)) {
    if ((long)(&dummy) == (long)((void *)(&dummy))) {
      tmp = (int (*)(void const *, void const *))sp->fts_compar;
    } else {
      tmp = &fts_compar;
    }
  } else {
    tmp = &fts_compar;
  }
  compare = tmp;
  if (nitems > sp->fts_nitems) {
    sp->fts_nitems = nitems + 40UL;
    if (0xffffffffffffffffUL / sizeof(*a) < sp->fts_nitems) {
      free((void *)sp->fts_array);
      sp->fts_array = (struct _ftsent **)((void *)0);
      sp->fts_nitems = (size_t)0;
      return (head);
    } else {
      a = (FTSENT **)realloc((void *)sp->fts_array,
                             sp->fts_nitems * sizeof(*a));
      if (!a) {
        free((void *)sp->fts_array);
        sp->fts_array = (struct _ftsent **)((void *)0);
        sp->fts_nitems = (size_t)0;
        return (head);
      }
    }
    sp->fts_array = a;
  }
  ap = sp->fts_array;
  p = head;
  while (1) {

    if (!p) {
      goto while_break;
    }
    tmp___0 = ap;
    ap++;
    *tmp___0 = p;
    p = p->fts_link;
  }
while_break:
  qsort((void *)sp->fts_array, nitems, sizeof(FTSENT *), compare);
  ap = sp->fts_array;
  head = *ap;
  while (1) {
    nitems--;
    if (!nitems) {
      goto while_break___0;
    }
    (*(ap + 0))->fts_link = *(ap + 1);
    ap++;
  }
while_break___0:
  (*(ap + 0))->fts_link = (struct _ftsent *)((void *)0);
  return (head);
```

```
  }
}
static FTSENT *fts_alloc(FTS *sp, char const *name, size_t namelen) {
  register FTSENT *p;
  size_t len;

  {
    len = sizeof(FTSENT) + namelen;
    p = (FTSENT *)malloc(len);
    if ((unsigned long)p == (unsigned long)((void *)0)) {
      return ((FTSENT *)((void *)0));
    }
    memmove((void *)(p->fts_name), (void const *)name, namelen);
    p->fts_name[namelen] = (char)'\000';
    p->fts_namelen = namelen;
    p->fts_fts = sp;
    p->fts_path = sp->fts_path;
    p->fts_errno = 0;
    p->fts_flags = (unsigned short)0;
    p->fts_instr = (unsigned short)3;
    p->fts_number = 0L;
    p->fts_pointer = (void *)0;
    return (p);
  }
}
static void fts_lfree(FTSENT *head) {
  register FTSENT *p;

  {
    while (1) {
      p = head;
      if (!p) {
        goto while_break;
      }
      head = head->fts_link;
      free((void *)p);
    }
  while_break:;
    return;
  }
}
static _Bool fts_palloc(FTS *sp, size_t more) {
  char *p;
  size_t new_len;
  int *tmp;

  {
    new_len = (sp->fts_pathlen + more) + 256UL;
    if (new_len < sp->fts_pathlen) {
      free((void *)sp->fts_path);
      sp->fts_path = (char *)((void *)0);
      tmp = __errno_location();
      *tmp = 36;
      return ((_Bool)0);
    }
    sp->fts_pathlen = new_len;
    p = (char *)realloc((void *)sp->fts_path, sp->fts_pathlen);
    if ((unsigned long)p == (unsigned long)((void *)0)) {
      free((void *)sp->fts_path);
      sp->fts_path = (char *)((void *)0);
      return ((_Bool)0);
    }
    sp->fts_path = p;
    return ((_Bool)1);
```

```
  }
}
static void fts_padjust(FTS *sp, FTSENT *head) {
  FTSENT *p;
  char *addr;

  {
    addr = sp->fts_path;
    p = sp->fts_child;
    while (1) {

      if (!p) {
        goto while_break;
      }
      while (1) {

        if ((unsigned long)p->fts_accpath != (unsigned long)(p->fts_name)) {
          p->fts_accpath = addr + (p->fts_accpath - p->fts_path);
        }
        p->fts_path = addr;
        goto while_break___0;
      }
    while_break___0:
      p = p->fts_link;
    }
  while_break:
    p = head;
    while (1) {

      if (!(p->fts_level >= 0L)) {
        goto while_break___1;
      }
      while (1) {

        if ((unsigned long)p->fts_accpath != (unsigned long)(p->fts_name)) {
          p->fts_accpath = addr + (p->fts_accpath - p->fts_path);
        }
        p->fts_path = addr;
        goto while_break___2;
      }
    while_break___2:;
      if (p->fts_link) {
        p = p->fts_link;
      } else {
        p = p->fts_parent;
      }
    }
  while_break___1:;
    return;
  }
}
static size_t fts_maxarglen(char *const *argv) {
  size_t len;
  size_t max;

  {
    max = (size_t)0;
    while (1) {

      if (!*argv) {
        goto while_break;
      }
      len = strlen((char const *)*argv);
      if (len > max) {
```

```
        max = len;
      }
      argv++;
    }
  while_break:;
    return (max + 1UL);
  }
}
static int fts_safe_changedir(FTS *sp, FTSENT *p, int fd, char const *dir) {
  int ret;
  _Bool is_dotdot;
  int tmp;
  int tmp___0;
  int newfd;
  int parent_fd;
  _Bool tmp___1;
  struct stat sb;
  int tmp___2;
  int *tmp___3;
  int tmp___4;
  int oerrno;
  int *tmp___5;
  int *tmp___6;

  {
    if (dir) {
      tmp = strcmp(dir, "..");
      if (tmp == 0) {
        tmp___0 = 1;
      } else {
        tmp___0 = 0;
      }
    } else {
      tmp___0 = 0;
    }
    is_dotdot = (_Bool)tmp___0;
    if (sp->fts_options & 4) {
      if (sp->fts_options & 512) {
        if (0 <= fd) {
          close(fd);
        }
      }
      return (0);
    }
    if (fd < 0) {
      if (is_dotdot) {
        if (sp->fts_options & 512) {
          tmp___1 = i_ring_empty((I_ring const *)(&sp->fts_fd_ring));
          if (!tmp___1) {
            parent_fd = i_ring_pop(&sp->fts_fd_ring);
            is_dotdot = (_Bool)1;
            if (0 <= parent_fd) {
              fd = parent_fd;
              dir = (char const *)((void *)0);
            }
          }
        }
      }
    }
    newfd = fd;
    if (fd < 0) {
      newfd = diropen((FTS const *)sp, dir);
      if (newfd < 0) {
        return (-1);
```

```
      }
    }
    if (sp->fts_options & 2) {
      goto _L;
    } else {
      if (dir) {
        tmp___4 = strcmp(dir, "..");
        if (tmp___4 == 0) {
        _L:
          tmp___2 = fstat(newfd, &sb);
          if (tmp___2) {
            ret = -1;
            goto bail;
          }
          if (p->fts_statp[0].st_dev != sb.st_dev) {
            tmp___3 = __errno_location();
            *tmp___3 = 2;
            ret = -1;
            goto bail;
          } else {
            if (p->fts_statp[0].st_ino != sb.st_ino) {
              tmp___3 = __errno_location();
              *tmp___3 = 2;
              ret = -1;
              goto bail;
            }
          }
        }
      }
    }
    if (sp->fts_options & 512) {
      cwd_advance_fd(sp, newfd, (_Bool)(!is_dotdot));
      return (0);
    }
    ret = fchdir(newfd);
  bail:
    if (fd < 0) {
      tmp___5 = __errno_location();
      oerrno = *tmp___5;
      close(newfd);
      tmp___6 = __errno_location();
      *tmp___6 = oerrno;
    }
    return (ret);
  }
}
extern int fseeko(FILE *__stream, __off_t __off, int __whence);
extern __attribute__((__nothrow__))
__off_t(__attribute__((__leaf__)) lseek)(int __fd, __off_t __offset,
                                         int __whence);
int(__attribute__((__nonnull__(1))) rpl_fseeko)(FILE *fp, off_t offset,
                                                int whence) {
  off_t pos;
  off_t tmp;
  int tmp___0;
  off_t tmp___1;
  int tmp___2;

  {
    if ((unsigned long)fp->_IO_read_end == (unsigned long)fp->_IO_read_ptr) {
      if ((unsigned long)fp->_IO_write_ptr ==
          (unsigned long)fp->_IO_write_base) {
        if ((unsigned long)fp->_IO_save_base == (unsigned long)((void *)0)) {
          if (whence == 2) {
```

```
            if (offset > 0L) {
              tmp = (off_t)0;
            } else {
              tmp = offset;
            }
          } else {
            tmp = offset;
          }
          tmp___0 = fileno(fp);
          tmp___1 = lseek(tmp___0, tmp, whence);
          pos = tmp___1;
          if (pos == -1L) {
            return (-1);
          }
          fp->_flags &= -17;
          if (whence == 2) {
            if (!(offset > 0L)) {
              return (0);
            }
          } else {
            return (0);
          }
        }
      }
    }
    tmp___2 = fseeko(fp, offset, whence);
    return (tmp___2);
  }
}
#pragma weak pthread_key_create
#pragma weak pthread_getspecific
#pragma weak pthread_setspecific
#pragma weak pthread_key_delete
#pragma weak pthread_self
#pragma weak pthread_cancel
#pragma weak pthread_mutex_init
#pragma weak pthread_mutex_lock
#pragma weak pthread_mutex_unlock
#pragma weak pthread_mutex_destroy
#pragma weak pthread_rwlock_init
#pragma weak pthread_rwlock_rdlock
#pragma weak pthread_rwlock_wrlock
#pragma weak pthread_rwlock_unlock
#pragma weak pthread_rwlock_destroy
#pragma weak pthread_once
#pragma weak pthread_cond_init
#pragma weak pthread_cond_wait
#pragma weak pthread_cond_signal
#pragma weak pthread_cond_broadcast
#pragma weak pthread_cond_destroy
#pragma weak pthread_mutexattr_init
#pragma weak pthread_mutexattr_settype
#pragma weak pthread_mutexattr_destroy
#pragma weak pthread_self
#pragma weak pthread_cancel
char const *Version = "8.4";
extern
    __attribute__((__nothrow__)) int(__attribute__((__nonnull__(1), __leaf__))
                                     euidaccess)(char const *__name,
                                                 int __type);
extern
    __attribute__((__nothrow__)) int(__attribute__((__nonnull__(2), __leaf__))
                                     faccessat)(int __fd, char const *__file,
                                                int __type, int __flag);
```

```
extern
    __attribute__((__nothrow__)) int(__attribute__((__nonnull__(2), __leaf__))
                                     unlinkat)(int __fd, char const *__name,
                                               int __flag);
__inline static int lstatat(int fd, char const *name, struct stat *st) {
  int tmp;

  {
    tmp = fstatat(fd, name, st, 256);
    return (tmp);
  }
}
__inline static _Bool dot_or_dotdot(char const *file_name___3) {
  char sep;
  int tmp;

  {
    if ((int const) * (file_name___3 + 0) == 46) {
      sep = (char)*(file_name___3 +
                    (((int const) * (file_name___3 + 1) == 46) + 1));
      if (!sep) {
        tmp = 1;
      } else {
        if ((int)sep == 47) {
          tmp = 1;
        } else {
          tmp = 0;
        }
      }
      return ((_Bool)tmp);
    } else {
      return ((_Bool)0);
    }
  }
}
__inline static struct dirent const *
readdir_ignoring_dot_and_dotdot(DIR *dirp) {
  struct dirent const *dp;
  struct dirent const *tmp;
  _Bool tmp___0;

  {
    while (1) {
      tmp = (struct dirent const *)readdir(dirp);
      dp = tmp;
      if ((unsigned long)dp == (unsigned long)((void *)0)) {
        return (dp);
      } else {
        tmp___0 = dot_or_dotdot((char const *)(dp->d_name));
        if (!tmp___0) {
          return (dp);
        }
      }
    }

    return ((struct dirent const *)0);
  }
}
__inline static _Bool is_empty_dir(int fd_cwd, char const *dir) {
  DIR *dirp;
  struct dirent const *dp;
  int saved_errno;
  int fd;
  int tmp;
```

```
  int *tmp___0;
  int *tmp___1;
  int tmp___2;

  {
    tmp = openat(fd_cwd, dir, 198912);
    fd = tmp;
    if (fd < 0) {
      return ((_Bool)0);
    }
    dirp = fdopendir(fd);
    if ((unsigned long)dirp == (unsigned long)((void *)0)) {
      close(fd);
      return ((_Bool)0);
    }
    tmp___0 = __errno_location();
    *tmp___0 = 0;
    dp = readdir_ignoring_dot_and_dotdot(dirp);
    tmp___1 = __errno_location();
    saved_errno = *tmp___1;
    closedir(dirp);
    if ((unsigned long)dp != (unsigned long)((void *)0)) {
      return ((_Bool)0);
    }
    if (saved_errno == 0) {
      tmp___2 = 1;
    } else {
      tmp___2 = 0;
    }
    return ((_Bool)tmp___2);
  }
}
enum RM_status rm(char *const *file, struct rm_options const *x);
static int cache_fstatat(int fd, char const *file, struct stat *st, int flag) {
  int *tmp;
  int tmp___0;
  int *tmp___1;

  {
    if (st->st_size == -1L) {
      tmp___0 = fstatat(fd, file, st, flag);
      if (tmp___0 != 0) {
        st->st_size = (__off_t)-2;
        tmp = __errno_location();
        st->st_ino = (__ino_t)*tmp;
      }
    }
    if (0L <= st->st_size) {
      return (0);
    }
    tmp___1 = __errno_location();
    *tmp___1 = (int)st->st_ino;
    return (-1);
  }
}
__inline static struct stat *cache_stat_init(struct stat *st) {

  {
    st->st_size = (__off_t)-1;
    return (st);
  }
}
static int write_protected_non_symlink(int fd_cwd, char const *file,
                                       char const *full_name,
```

```
                                        struct stat *buf___1) {
  _Bool tmp;
  int tmp___0;
  int tmp___1;
  size_t file_name_len;
  size_t tmp___2;
  _Bool tmp___3;
  int tmp___4;
  int tmp___5;
  int *tmp___6;
  int *tmp___7;

  {
    tmp = can_write_any_file();
    if (tmp) {
      return (0);
    }
    tmp___0 = cache_fstatat(fd_cwd, file, buf___1, 256);
    if (tmp___0 != 0) {
      return (-1);
    }
    if ((buf___1->st_mode & 61440U) == 40960U) {
      return (0);
    }
    tmp___1 = faccessat(fd_cwd, file, 2, 512);
    if (tmp___1 == 0) {
      return (0);
    }
    tmp___2 = strlen(full_name);
    file_name_len = tmp___2;
    if (4096UL <= file_name_len) {
      tmp___3 = euidaccess_stat((struct stat const *)buf___1, 2);
      if (tmp___3) {
        tmp___4 = 0;
      } else {
        tmp___4 = 1;
      }
      return (tmp___4);
    }
    tmp___5 = euidaccess(full_name, 2);
    if (tmp___5 == 0) {
      return (0);
    }
    tmp___7 = __errno_location();
    if (*tmp___7 == 13) {
      tmp___6 = __errno_location();
      *tmp___6 = 0;
      return (1);
    }
    return (-1);
  }
}
static enum RM_status prompt(FTS const *fts, FTSENT const *ent, _Bool is_dir,
                             struct rm_options const *x,
                             enum Prompt_action mode, Ternary *is_empty_p) {
  int fd_cwd;
  char const *full_name;
  char const *filename;
  struct stat st;
  struct stat *sbuf;
  int dirent_type;
  int tmp;
  int write_protected;
  int wp_errno;
```

```
int *tmp___0;
int *tmp___1;
int tmp___2;
char const *quoted_name;
char const *tmp___3;
char *tmp___4;
_Bool is_empty;
char *tmp___5;
char *tmp___6;
char *tmp___7;
char *tmp___8;
int *tmp___9;
int tmp___10;
char const *tmp___11;
char *tmp___12;
char *tmp___13;
char *tmp___14;
_Bool tmp___15;

{
  fd_cwd = (int)fts->fts_cwd_fd;
  full_name = (char const *)ent->fts_path;
  filename = (char const *)ent->fts_accpath;
  if (is_empty_p) {
    *is_empty_p = (Ternary)2;
  }
  sbuf = &st;
  cache_stat_init(sbuf);
  if (is_dir) {
    tmp = 4;
  } else {
    tmp = 0;
  }
  dirent_type = tmp;
  write_protected = 0;
  if (ent->fts_number) {
    return ((enum RM_status)3);
  }
  if ((unsigned int const)x->interactive == 5U) {
    return ((enum RM_status)2);
  }
  wp_errno = 0;
  if (!x->ignore_missing_files) {
    if ((unsigned int const)x->interactive == 3U) {
      goto _L;
    } else {
      if (x->stdin_tty) {
      _L:
        if (dirent_type != 10) {
          write_protected =
              write_protected_non_symlink(fd_cwd, filename, full_name, sbuf);
          tmp___0 = __errno_location();
          wp_errno = *tmp___0;
        }
      }
    }
  }
  if (write_protected) {
    goto _L___2;
  } else {
    if ((unsigned int const)x->interactive == 3U) {
    _L___2:
      if (0 <= write_protected) {
        if (dirent_type == 0) {
```

```
    tmp___2 = cache_fstatat(fd_cwd, filename, sbuf, 256);
    if (tmp___2 == 0) {
      if ((sbuf->st_mode & 61440U) == 40960U) {
        dirent_type = 10;
      } else {
        if ((sbuf->st_mode & 61440U) == 16384U) {
          dirent_type = 4;
        }
      }
    } else {
      write_protected = -1;
      tmp___1 = __errno_location();
      wp_errno = *tmp___1;
    }
  }
}
if (0 <= write_protected) {
  if (dirent_type == 10) {
    goto case_10;
  }
  if (dirent_type == 4) {
    goto case_4;
  }
  goto switch_break;
case_10:
  if ((unsigned int const)x->interactive != 3U) {
    return ((enum RM_status)2);
  }
  goto switch_break;
case_4:
  if (!x->recursive) {
    write_protected = -1;
    wp_errno = 21;
  }
  goto switch_break;
switch_break:;
}
tmp___3 = quote(full_name);
quoted_name = tmp___3;
if (write_protected < 0) {
  tmp___4 = gettext("cannot remove %s");
  error(0, wp_errno, (char const *)tmp___4, quoted_name);
  return ((enum RM_status)4);
}
if (is_empty_p) {
  is_empty = is_empty_dir(fd_cwd, filename);
  if (is_empty) {
    *is_empty_p = (Ternary)4;
  } else {
    *is_empty_p = (Ternary)3;
  }
} else {
  is_empty = (_Bool)0;
}
if (dirent_type == 4) {
  if ((unsigned int)mode == 2U) {
    if (!is_empty) {
      if (write_protected) {
        tmp___5 =
            gettext("%s: descend into write-protected directory %s? ");
        tmp___7 = tmp___5;
      } else {
        tmp___6 = gettext("%s: descend into directory %s? ");
        tmp___7 = tmp___6;
```

```
              }
              fprintf(stderr, (char const *)tmp___7, program_name, quoted_name);
            } else {
              goto _L___1;
            }
          } else {
            goto _L___1;
          }
        } else {
        _L___1:
          tmp___10 = cache_fstatat(fd_cwd, filename, sbuf, 256);
          if (tmp___10 != 0) {
            tmp___8 = gettext("cannot remove %s");
            tmp___9 = __errno_location();
            error(0, *tmp___9, (char const *)tmp___8, quoted_name);
            return ((enum RM_status)4);
          }
          tmp___11 = file_type((struct stat const *)sbuf);
          if (write_protected) {
            tmp___12 = gettext("%s: remove write-protected %s %s? ");
            tmp___14 = tmp___12;
          } else {
            tmp___13 = gettext("%s: remove %s %s? ");
            tmp___14 = tmp___13;
          }
          fprintf(stderr, (char const *)tmp___14, program_name, tmp___11,
                  quoted_name);
        }
        tmp___15 = yesno();
        if (!tmp___15) {
          return ((enum RM_status)3);
        }
      }
    }
    return ((enum RM_status)2);
  }
}
__inline static _Bool nonexistent_file_errno(int errnum) {

  {
    if (errnum == 2) {
      goto case_2;
    }
    if (errnum == 20) {
      goto case_2;
    }
    goto switch_default;
  case_2:
    return ((_Bool)1);
  switch_default:
    return ((_Bool)0);

    return ((_Bool)0);
  }
}
__inline static _Bool ignorable_missing(struct rm_options const *x,
                                        int errnum) {
  _Bool tmp;
  int tmp___0;

  {
    if (x->ignore_missing_files) {
      tmp = nonexistent_file_errno(errnum);
      if (tmp) {
```

```
        tmp___0 = 1;
      } else {
        tmp___0 = 0;
      }
    } else {
      tmp___0 = 0;
    }
    return ((_Bool)tmp___0);
  }
}
static void fts_skip_tree(FTS *fts, FTSENT *ent) {

  {
    fts_set(fts, ent, 4);
    ent = fts_read(fts);
    return;
  }
}
static void mark_ancestor_dirs(FTSENT *ent) {
  FTSENT *p;

  {
    p = ent->fts_parent;
    while (1) {

      if (!(0L <= p->fts_level)) {
        goto while_break;
      }
      if (p->fts_number) {
        goto while_break;
      }
      p->fts_number = 1L;
      p = p->fts_parent;
    }
  while_break:;
    return;
  }
}
static enum RM_status excise(FTS *fts, FTSENT *ent, struct rm_options const *x,
                             _Bool is_dir) {
  int flag;
  int tmp;
  char const *tmp___0;
  char *tmp___1;
  char *tmp___2;
  char *tmp___3;
  int tmp___4;
  struct stat st;
  int *tmp___5;
  int tmp___6;
  int *tmp___7;
  int *tmp___8;
  int *tmp___9;
  _Bool tmp___10;
  int *tmp___11;
  char const *tmp___12;
  char *tmp___13;
  int *tmp___14;

  {
    if (is_dir) {
      tmp = 512;
    } else {
      tmp = 0;
```

```
    }
    flag = tmp;
    tmp___4 = unlinkat(fts->fts_cwd_fd, (char const *)ent->fts_accpath, flag);
    if (tmp___4 == 0) {
      if (x->verbose) {
        tmp___0 = quote((char const *)ent->fts_path);
        if (is_dir) {
          tmp___1 = gettext("removed directory: %s\n");
          tmp___3 = tmp___1;
        } else {
          tmp___2 = gettext("removed %s\n");
          tmp___3 = tmp___2;
        }
        printf((char const *)tmp___3, tmp___0);
      }
      return ((enum RM_status)2);
    }
    tmp___8 = __errno_location();
    if (*tmp___8 == 30) {
      tmp___6 = lstatat(fts->fts_cwd_fd, (char const *)ent->fts_accpath, &st);
      if (tmp___6) {
        tmp___7 = __errno_location();
        if (!(*tmp___7 == 2)) {
          tmp___5 = __errno_location();
          *tmp___5 = 30;
        }
      } else {
        tmp___5 = __errno_location();
        *tmp___5 = 30;
      }
    }
    tmp___9 = __errno_location();
    tmp___10 = ignorable_missing(x, *tmp___9);
    if (tmp___10) {
      return ((enum RM_status)2);
    }
    if ((int)ent->fts_info == 4) {
      tmp___11 = __errno_location();
      *tmp___11 = ent->fts_errno;
    }
    tmp___12 = quote((char const *)ent->fts_path);
    tmp___13 = gettext("cannot remove %s");
    tmp___14 = __errno_location();
    error(0, *tmp___14, (char const *)tmp___13, tmp___12);
    mark_ancestor_dirs(ent);
    return ((enum RM_status)4);
  }
}
static enum RM_status rm_fts(FTS *fts, FTSENT *ent,
                             struct rm_options const *x) {
  char const *tmp;
  char *tmp___0;
  _Bool tmp___1;
  char const *tmp___2;
  char *tmp___3;
  char *tmp___4;
  _Bool tmp___5;
  char const *tmp___6;
  char *tmp___7;
  char const *tmp___8;
  char const *tmp___9;
  char *tmp___10;
  int tmp___11;
  char *tmp___12;
```

```
Ternary is_empty_directory;
enum RM_status s;
enum RM_status tmp___13;
char const *tmp___14;
char *tmp___15;
_Bool is_dir;
int tmp___16;
enum RM_status s___0;
enum RM_status tmp___17;
enum RM_status tmp___18;
char const *tmp___19;
char *tmp___20;
char const *tmp___21;
char *tmp___22;
char const *tmp___23;
char *tmp___24;

{
  if ((int)ent->fts_info == 1) {
    goto case_1;
  }
  if ((int)ent->fts_info == 8) {
    goto case_8;
  }
  if ((int)ent->fts_info == 10) {
    goto case_8;
  }
  if ((int)ent->fts_info == 12) {
    goto case_8;
  }
  if ((int)ent->fts_info == 13) {
    goto case_8;
  }
  if ((int)ent->fts_info == 6) {
    goto case_8;
  }
  if ((int)ent->fts_info == 4) {
    goto case_8;
  }
  if ((int)ent->fts_info == 11) {
    goto case_8;
  }
  if ((int)ent->fts_info == 3) {
    goto case_8;
  }
  if ((int)ent->fts_info == 2) {
    goto case_2;
  }
  if ((int)ent->fts_info == 7) {
    goto case_7;
  }
  goto switch_default;
case_1:
  if (!x->recursive) {
    tmp = quote((char const *)ent->fts_path);
    tmp___0 = gettext("cannot remove %s");
    error(0, 21, (char const *)tmp___0, tmp);
    mark_ancestor_dirs(ent);
    fts_skip_tree(fts, ent);
    return ((enum RM_status)4);
  }
  if (ent->fts_level == 0L) {
    tmp___1 = strip_trailing_slashes(ent->fts_path);
    if (tmp___1) {
```

```
      ent->fts_pathlen = strlen((char const *)ent->fts_path);
    }
    tmp___4 = last_component((char const *)ent->fts_accpath);
    tmp___5 = dot_or_dotdot((char const *)tmp___4);
    if (tmp___5) {
      tmp___2 = quote((char const *)ent->fts_path);
      tmp___3 = gettext("cannot remove directory: %s");
      error(0, 0, (char const *)tmp___3, tmp___2);
      fts_skip_tree(fts, ent);
      return ((enum RM_status)4);
    }
    if (x->root_dev_ino) {
      if (ent->fts_statp[0].st_ino == (x->root_dev_ino)->st_ino) {
        if (ent->fts_statp[0].st_dev == (x->root_dev_ino)->st_dev) {
          while (1) {
            tmp___11 = strcmp((char const *)ent->fts_path, "/");
            if (tmp___11 == 0) {
              tmp___6 = quote((char const *)ent->fts_path);
              tmp___7 =
                  gettext("it is dangerous to operate recursively on %s");
              error(0, 0, (char const *)tmp___7, tmp___6);
            } else {
              tmp___8 = quote_n(1, "/");
              tmp___9 = quote_n(0, (char const *)ent->fts_path);
              tmp___10 = gettext("it is dangerous to operate recursively on "
                                 "%s (same as %s)");
              error(0, 0, (char const *)tmp___10, tmp___9, tmp___8);
            }
            tmp___12 =
                gettext("use --no-preserve-root to override this failsafe");
            error(0, 0, (char const *)tmp___12);
            goto while_break;
          }
        while_break:
          fts_skip_tree(fts, ent);
          return ((enum RM_status)4);
        }
      }
    }
  }
  tmp___13 = prompt((FTS const *)fts, (FTSENT const *)ent, (_Bool)1, x,
                    (enum Prompt_action)2, &is_empty_directory);
  s = tmp___13;
  if ((unsigned int)s == 2U) {
    if ((unsigned int)is_empty_directory == 4U) {
      s = excise(fts, ent, x, (_Bool)1);
      fts_skip_tree(fts, ent);
    }
  }
  if ((unsigned int)s != 2U) {
    mark_ancestor_dirs(ent);
    fts_skip_tree(fts, ent);
  }
  return (s);
case_8:
  if ((int)ent->fts_info == 6) {
    if (x->one_file_system) {
      if (0L < ent->fts_level) {
        if (ent->fts_statp[0].st_dev != fts->fts_dev) {
          mark_ancestor_dirs(ent);
          tmp___14 = quote((char const *)ent->fts_path);
          tmp___15 =
              gettext("skipping %s, since it\'s on a different device");
          error(0, 0, (char const *)tmp___15, tmp___14);
```

```
            return ((enum RM_status)4);
          }
        }
      }
    }
    if ((int)ent->fts_info == 6) {
      tmp___16 = 1;
    } else {
      if ((int)ent->fts_info == 4) {
        tmp___16 = 1;
      } else {
        tmp___16 = 0;
      }
    }
    is_dir = (_Bool)tmp___16;
    tmp___17 = prompt((FTS const *)fts, (FTSENT const *)ent, is_dir, x,
                      (enum Prompt_action)3, (Ternary *)((void *)0));
    s___0 = tmp___17;
    if ((unsigned int)s___0 != 2U) {
      return (s___0);
    }
    tmp___18 = excise(fts, ent, x, is_dir);
    return (tmp___18);
  case_2:
    while (1) {
      tmp___19 = quote((char const *)ent->fts_path);
      tmp___20 = gettext(
          "WARNING: Circular directory structure.\nThis almost certainly means "
          "that you have a corrupted file system.\nNOTIFY YOUR SYSTEM "
          "MANAGER.\nThe following directory is part of the cycle:\n  %s\n");
      error(0, 0, (char const *)tmp___20, tmp___19);
      goto while_break___0;
    }
  while_break___0:
    fts_skip_tree(fts, ent);
    return ((enum RM_status)4);
  case_7:
    tmp___21 = quote((char const *)ent->fts_path);
    tmp___22 = gettext("traversal failed: %s");
    error(0, ent->fts_errno, (char const *)tmp___22, tmp___21);
    fts_skip_tree(fts, ent);
    return ((enum RM_status)4);
  switch_default:
    tmp___23 = quote((char const *)ent->fts_path);
    tmp___24 =
        gettext("unexpected failure: fts_info=%d: %s\nplease report to %s");
    error(0, 0, (char const *)tmp___24, (int)ent->fts_info, tmp___23,
          "bug-coreutils@gnu.org");
    abort();

    return ((enum RM_status)0);
  }
}
enum RM_status rm(char *const *file, struct rm_options const *x) {
  enum RM_status rm_status;
  int bit_flags;
  FTS *fts;
  FTS *tmp;
  FTSENT *ent;
  char *tmp___0;
  int *tmp___1;
  int *tmp___2;
  enum RM_status s;
  enum RM_status tmp___3;
```

```
char *tmp___4;
int *tmp___5;
int tmp___6;

{
  rm_status = (enum RM_status)2;
  if (*file) {
    bit_flags = 536;
    if (x->one_file_system) {
      bit_flags |= 64;
    }
    tmp = xfts_open(file, bit_flags,
                    (int (*)(FTSENT const **, FTSENT const **))((void *)0));
    fts = tmp;
    while (1) {
      ent = fts_read(fts);
      if ((unsigned long)ent == (unsigned long)((void *)0)) {
        tmp___2 = __errno_location();
        if (*tmp___2 != 0) {
          tmp___0 = gettext("fts_read failed");
          tmp___1 = __errno_location();
          error(0, *tmp___1, (char const *)tmp___0);
          rm_status = (enum RM_status)4;
        }
        goto while_break;
      }
      tmp___3 = rm_fts(fts, ent, x);
      s = tmp___3;
      if (!((unsigned int)s == 2U)) {
        if (!((unsigned int)s == 3U)) {
          if (!((unsigned int)s == 4U)) {
            __assert_fail(
                "((s) == RM_OK || (s) == RM_USER_DECLINED || (s) == "
                "RM_ERROR)",
                "/home/khheo/project/benchmark/coreutils-8.4/src/remove.c",
                624U, "rm");
          }
        }
      }
      while (1) {

        if ((unsigned int)s == 4U) {
          rm_status = s;
        } else {
          if ((unsigned int)s == 3U) {
            if ((unsigned int)rm_status == 2U) {
              rm_status = s;
            }
          }
        }
        goto while_break___0;
      }
    while_break___0:;
    }
  while_break:
    tmp___6 = fts_close(fts);
    if (tmp___6 != 0) {
      tmp___4 = gettext("fts_close failed");
      tmp___5 = __errno_location();
      error(0, *tmp___5, (char const *)tmp___4);
      rm_status = (enum RM_status)4;
    }
  }
  return (rm_status);
```

```
  }
}
extern char *optarg;
extern __attribute__((__nothrow__)) int(__attribute__((__leaf__))
                                        isatty)(int __fd);
extern
    __attribute__((__nothrow__)) int(__attribute__((__nonnull__(1), __leaf__))
                                     atexit)(void (*__func)(void));
extern
    __attribute__((__nothrow__)) char *(__attribute__((__leaf__))
                                        textdomain)(char const *__domainname);
extern __attribute__((__nothrow__)) char *(__attribute__((
    __leaf__)) bindtextdomain)(char const *__domainname, char const *__dirname);
__inline static void emit_ancillary_info(void) {
  char *tmp;
  char *tmp___0;
  char *tmp___1;
  char *tmp___2;
  char const *lc_messages;
  char const *tmp___3;
  char *tmp___4;
  char *tmp___5;
  int tmp___6;
  char *tmp___7;
  char *tmp___8;

  {
    tmp = last_component(program_name);
    tmp___0 = gettext("\nReport %s bugs to %s\n");
    printf((char const *)tmp___0, tmp, "bug-coreutils@gnu.org");
    tmp___1 = gettext("%s home page: <http://www.gnu.org/software/%s/>\n");
    printf((char const *)tmp___1, "GNU coreutils", "coreutils");
    tmp___2 = gettext(
        "General help using GNU software: <http://www.gnu.org/gethelp/>\n");
    fputs_unlocked((char const *)tmp___2, stdout);
    tmp___3 = (char const *)setlocale(5, (char const *)((void *)0));
    lc_messages = tmp___3;
    if (lc_messages) {
      tmp___6 = strncmp(lc_messages, "en_", (size_t)3);
      if (tmp___6) {
        tmp___4 = last_component(program_name);
        tmp___5 = gettext("Report %s translation bugs to "
                          "<http://translationproject.org/team/>\n");
        printf((char const *)tmp___5, tmp___4);
      }
    }
    tmp___7 = last_component(program_name);
    tmp___8 = gettext(
        "For complete documentation, run: info coreutils \'%s invocation\'\n");
    printf((char const *)tmp___8, tmp___7);
    return;
  }
}
__inline static int priv_set_remove_linkdir(void) {

  { return (-1); }
}
static struct option const long_opts[12] = {
    {"directory", 0, (int *)((void *)0), 'd'},
    {"force", 0, (int *)((void *)0), 'f'},
    {"interactive", 2, (int *)((void *)0), 128},
    {"one-file-system", 0, (int *)((void *)0), 129},
    {"no-preserve-root", 0, (int *)((void *)0), 130},
    {"preserve-root", 0, (int *)((void *)0), 131},
```

```
    {"-presume-input-tty", 0, (int *)((void *)0), 132},
    {"recursive", 0, (int *)((void *)0), 'r'},
    {"verbose", 0, (int *)((void *)0), 'v'},
    {"help", 0, (int *)((void *)0), -130},
    {"version", 0, (int *)((void *)0), -131},
    {(char const *)((void *)0), 0, (int *)((void *)0), 0}};
static char const *const interactive_args[7] = {
    "never", "no", "none", "once", "always", "yes", (char const *)((void *)0)};
static enum interactive_type const interactive_types[6] = {
    (enum interactive_type const)0, (enum interactive_type const)0,
    (enum interactive_type const)0, (enum interactive_type const)1,
    (enum interactive_type const)2, (enum interactive_type const)2};
static void diagnose_leading_hyphen(int argc, char **argv) {
  int i;
  char const *arg;
  struct stat st;
  char const *tmp;
  char *tmp___0;
  char *tmp___1;
  int tmp___2;

  {
    i = 1;
    while (1) {

      if (!(i < argc)) {
        goto while_break;
      }
      arg = (char const *)*(argv + i);
      if ((int const) * (arg + 0) == 45) {
        if (*(arg + 1)) {
          tmp___2 = lstat(arg, &st);
          if (tmp___2 == 0) {
            tmp = quote(arg);
            tmp___0 = quotearg_n_style(1, (enum quoting_style)1, arg);
            tmp___1 = gettext("Try `%s ./%s\' to remove the file %s.\n");
            fprintf(stderr, (char const *)tmp___1, *(argv + 0), tmp___0, tmp);
            goto while_break;
          }
        }
      }
      i++;
    }
  while_break:;
    return;
  }
}
__attribute__((__noreturn__)) void usage(int status);
void usage(int status) {
  char *tmp;
  char *tmp___0;
  char *tmp___1;
  char *tmp___2;
  char *tmp___3;
  char *tmp___4;
  char *tmp___5;
  char *tmp___6;
  char *tmp___7;
  char *tmp___8;
  char *tmp___9;

  {
    if (status != 0) {
      tmp = gettext("Try `%s --help\' for more information.\n");
```

```
      fprintf(stderr, (char const *)tmp, program_name);
    } else {
      tmp___0 = gettext("Usage: %s [OPTION]... FILE...\n");
      printf((char const *)tmp___0, program_name);
      tmp___1 = gettext("Remove (unlink) the FILE(s).\n\n  -f, --force         "
                        "  ignore nonexistent files, never prompt\n  -i        "
                        "            prompt before every removal\n");
      fputs_unlocked((char const *)tmp___1, stdout);
      tmp___2 = gettext(
          "  -I                    prompt once before removing more than three "
          "files, or\n                           when removing recursively.  "
          "Less intrusive than -i,\n                           while still "
          "giving protection against most mistakes\n      --interactive[=WHEN] "
          " prompt according to WHEN: never, once (-I), or\n                    "
          "       always (-i).  Without WHEN, prompt always\n");
      fputs_unlocked((char const *)tmp___2, stdout);
      tmp___3 =
          gettext("      --one-file-system  when removing a hierarchy "
                  "recursively, skip any\n                           directory "
                  "that is on a file system different from\n                    "
                  "       that of the corresponding command line argument\n");
      fputs_unlocked((char const *)tmp___3, stdout);
      tmp___4 = gettext(
          "      --no-preserve-root  do not treat `/\' specially\n      "
          "--preserve-root   do not remove `/\' (default)\n  -r, -R, "
          "--recursive   remove directories and their contents recursively\n  "
          "-v, --verbose         explain what is being done\n");
      fputs_unlocked((char const *)tmp___4, stdout);
      tmp___5 = gettext("      --help     display this help and exit\n");
      fputs_unlocked((char const *)tmp___5, stdout);
      tmp___6 =
          gettext("      --version  output version information and exit\n");
      fputs_unlocked((char const *)tmp___6, stdout);
      tmp___7 =
          gettext("\nBy default, rm does not remove directories.  Use the "
                  "--recursive (-r or -R)\noption to remove each listed "
                  "directory, too, along with all of its contents.\n");
      fputs_unlocked((char const *)tmp___7, stdout);
      tmp___8 = gettext("\nTo remove a file whose name starts with a `-\', for "
                        "example `-foo\',\nuse one of these commands:\n  %s -- "
                        "-foo\n\n  %s ./-foo\n");
      printf((char const *)tmp___8, program_name, program_name);
      tmp___9 = gettext("\nNote that if you use rm to remove a file, it is "
                        "usually possible to recover\nthe contents of that "
                        "file.  If you want more assurance that the contents "
                        "are\ntruly unrecoverable, consider using shred.\n");
      fputs_unlocked((char const *)tmp___9, stdout);
      emit_ancillary_info();
    }
    exit(status);
  }
}
static void rm_option_init(struct rm_options *x) {

  {
    x->ignore_missing_files = (_Bool)0;
    x->interactive = (enum rm_interactive)4;
    x->one_file_system = (_Bool)0;
    x->recursive = (_Bool)0;
    x->root_dev_ino = (struct dev_ino *)((void *)0);
    x->stdin_tty = (_Bool)isatty(0);
    x->verbose = (_Bool)0;
    x->require_restore_cwd = (_Bool)0;
    return;
```

```
  }
}
static struct dev_ino dev_ino_buf;
int main(int argc, char **argv) {
  _Bool preserve_root;
  struct rm_options x;
  _Bool prompt_once;
  int c;
  int i;
  ptrdiff_t tmp;
  char *tmp___0;
  char const *tmp___1;
  char *tmp___2;
  int *tmp___3;
  size_t n_files;
  char **file;
  char *tmp___4;
  char *tmp___5;
  char *tmp___6;
  _Bool tmp___7;
  enum RM_status status;
  enum RM_status tmp___8;
  int tmp___9;

  {
    preserve_root = (_Bool)1;
    prompt_once = (_Bool)0;
    set_program_name((char const *)*(argv + 0));
    setlocale(6, "");
    bindtextdomain("coreutils", "/usr/local/share/locale");
    textdomain("coreutils");
    atexit(&close_stdin);
    rm_option_init(&x);
    priv_set_remove_linkdir();
    while (1) {
      c = getopt_long(argc, (char *const *)argv, "dfirvIR", long_opts,
                      (int *)((void *)0));
      if (!(c != -1)) {
        goto while_break;
      }
      if (c == 100) {
        goto case_100;
      }
      if (c == 102) {
        goto case_102;
      }
      if (c == 105) {
        goto case_105;
      }
      if (c == 73) {
        goto case_73;
      }
      if (c == 114) {
        goto case_114;
      }
      if (c == 82) {
        goto case_114;
      }
      if (c == 128) {
        goto case_128;
      }
      if (c == 129) {
        goto case_129;
      }
```

```
  if (c == 130) {
    goto case_130;
  }
  if (c == 131) {
    goto case_131;
  }
  if (c == 132) {
    goto case_132;
  }
  if (c == 118) {
    goto case_118;
  }
  if (c == -130) {
    goto case_neg_130;
  }
  if (c == -131) {
    goto case_neg_131;
  }
  goto switch_default;
case_100:
  goto switch_break;
case_102:
  x.interactive = (enum rm_interactive)5;
  x.ignore_missing_files = (_Bool)1;
  prompt_once = (_Bool)0;
  goto switch_break;
case_105:
  x.interactive = (enum rm_interactive)3;
  x.ignore_missing_files = (_Bool)0;
  prompt_once = (_Bool)0;
  goto switch_break;
case_73:
  x.interactive = (enum rm_interactive)5;
  x.ignore_missing_files = (_Bool)0;
  prompt_once = (_Bool)1;
  goto switch_break;
case_114:
  x.recursive = (_Bool)1;
  goto switch_break;
case_128:
  if (optarg) {
    tmp = __xargmatch_internal("--interactive", (char const *)optarg,
                               interactive_args,
                               (char const *)(interactive_types),
                               sizeof(interactive_types[0]), argmatch_die);
    i = (int)interactive_types[tmp];
  } else {
    i = 2;
  }
  if (i == 0) {
    goto case_0;
  }
  if (i == 1) {
    goto case_1;
  }
  if (i == 2) {
    goto case_2;
  }
  goto switch_break___0;
case_0:
  x.interactive = (enum rm_interactive)5;
  prompt_once = (_Bool)0;
  goto switch_break___0;
case_1:
```

```
    x.interactive = (enum rm_interactive)4;
    x.ignore_missing_files = (_Bool)0;
    prompt_once = (_Bool)1;
    goto switch_break___0;
  case_2:
    x.interactive = (enum rm_interactive)3;
    x.ignore_missing_files = (_Bool)0;
    prompt_once = (_Bool)0;
    goto switch_break___0;
  switch_break___0:;
    goto switch_break;
  case_129:
    x.one_file_system = (_Bool)1;
    goto switch_break;
  case_130:
    preserve_root = (_Bool)0;
    goto switch_break;
  case_131:
    preserve_root = (_Bool)1;
    goto switch_break;
  case_132:
    x.stdin_tty = (_Bool)1;
    goto switch_break;
  case_118:
    x.verbose = (_Bool)1;
    goto switch_break;
  case_neg_130:
    usage(0);
    goto switch_break;
  case_neg_131:
    version_etc(stdout, "rm", "GNU coreutils", Version, "Paul Rubin",
                "David MacKenzie", "Richard M. Stallman", "Jim Meyering",
                (char *)((void *)0));
    exit(0);
    goto switch_break;
  switch_default:
    diagnose_leading_hyphen(argc, argv);
    usage(1);
  switch_break:;
  }
while_break:;
  if (argc <= optind) {
    if (x.ignore_missing_files) {
      exit(0);
    } else {
      tmp___0 = gettext("missing operand");
      error(0, 0, (char const *)tmp___0);
      usage(1);
    }
  }
  if (x.recursive) {
    if (preserve_root) {
      x.root_dev_ino = get_root_dev_ino(&dev_ino_buf);
      if ((unsigned long)x.root_dev_ino == (unsigned long)((void *)0)) {
        tmp___1 = quote("/");
        tmp___2 = gettext("failed to get attributes of %s");
        tmp___3 = __errno_location();
        error(1, *tmp___3, (char const *)tmp___2, tmp___1);
      }
    }
  }
  n_files = (size_t)(argc - optind);
  file = argv + optind;
  if (prompt_once) {
```

```
      if (x.recursive) {
        goto _L;
      } else {
        if (3UL < n_files) {
        _L:
          if (x.recursive) {
            tmp___4 = gettext("%s: remove all arguments recursively? ");
            tmp___6 = tmp___4;
          } else {
            tmp___5 = gettext("%s: remove all arguments? ");
            tmp___6 = tmp___5;
          }
          fprintf(stderr, (char const *)tmp___6, program_name);
          tmp___7 = yesno();
          if (!tmp___7) {
            exit(0);
          }
        }
      }
    }
    tmp___8 = rm((char *const *)file, (struct rm_options const *)(&x));
    status = tmp___8;
    if (!((unsigned int)status == 2U)) {
      if (!((unsigned int)status == 3U)) {
        if (!((unsigned int)status == 4U)) {
          __assert_fail("((status) == RM_OK || (status) == RM_USER_DECLINED || "
                        "(status) == RM_ERROR)",
                        "/home/khheo/project/benchmark/coreutils-8.4/src/rm.c",
                        352U, "main");
        }
      }
    }
    if ((unsigned int)status == 4U) {
      tmp___9 = 1;
    } else {
      tmp___9 = 0;
    }
    exit(tmp___9);
  }
}
```