- `write`
- `opendir`

  - `DIR *opendir(const char *name);`
  - `#include <dirent.h>`
  - opens a directory stream
  - returns `DIR *`
  - in case of error, a null pointer is returned and errno is set

- `readdir`

  - `struct dirent *readdir(DIR *dirp);`
  - `#include <dirent.h>`
  - returns `dirent` at the current pointer
  - The order in which filenames are read by successive calls to `readdir()` depends on the filesystem implementation; it is unlikely that the names will be sorted in any fashion.
  - if entries for `.` and `..` exist, they are returned
  - The pointer returned by `readdir()` points to data which may be overwritten by another call to `readdir()` on the same directory stream. This data is not overwritten by another call to `readdir()` on a different directory stream.
  - If a file is removed from or added to the directory after the most recent call to `opendir()` or `rewinddir()`, whether a subsequent call to `readdir()` returns an entry for that file is unspecified.
  - When an error is encountered, a null pointer is returned and `errno` is set to indicate the error. When the end of the directory is encountered, a null pointer is returned and `errno` is not changed.
  - **THEREFORE**, set `errno` to zero before making a call, if you wish to check for error situations

- `closedir`

  - `int closedir(DIR *dirp);`
  - `#include <dirent.h>`
  - returns `-1` in case of error and `errno` is set

- `stat`

  - `int stat(const char *path, struct stat *buf);`
  - returns `0` on success and `-1` on error
  - These functions return information about a file. No permissions are required on the file itself, but-in the case of `stat()` and `lstat()` - execute (search) permission is required on all of the directories in path that lead to the file.

- `lstat`

  - identical to `stat` except that if path is a symbolic link, the link itself is stat-ed, not the file that it refers to.

- `getpwuid`

  - `struct passwd *getpwuid(uid_t uid);`
  - `#include <pwd.h>`
  - search the user database for an entry with a matching uid
  - Applications wishing to check for error situations should set errno to 0 before calling `getpwuid()`. If `getpwuid()` returns a null pointer and errno is set to non-zero, an error occurred.
  - The return value may point to a static area which is overwritten by a subsequent call to `getpwent()`, `getpwnam()`, or `getpwuid()`.

- `getgrgid`

  - `struct group *getgrgid(gid_t gid);`
  - `#include <grp.h>`
  - The return value may point to a static area which is overwritten by a subsequent call to `getpwent()`, `getpwnam()`, or `getpwuid()`.

- `listxattr`

  - `ssize_t listxattr(const char *path, char *list, size_t size);`
  - `#include <sys/xattr.h>`
  - retrieves the list of extended attribute names associated with the given path in the filesystem.
  - The retrieved list is placed in list, a caller-allocated buffer whose size (in bytes) is specified in the argument size. The list is the set of (null-terminated) names, one after the other. Names of extended attributes to which the calling process does not have access may be omitted from the list. The length of the attribute name list is returned.
  - A single extended attribute name is a null-terminated string. The name includes a namespace prefix; there may be several, disjoint namespaces associated with an individual inode.
  - If size is specified as zero, these calls return the current size of the list of extended attribute names (and leave list unchanged). This can be used to determine the size of the buffer that should be supplied in a subsequent call. (But, bear in mind that there is a possibility that the set of extended attributes may change between the two calls, so that it is still necessary to check the return status from the second call.)

- `getxattr`

  - `ssize_t getxattr(const char *path, const char *name, void *value, size_t size);`
  - Extended attributes are name:value pairs associated with inodes (files, directories, symbolic links, etc.). They are extensions to the normal attributes which are associated with all inodes in the system (i.e., the stat(2) data).
  - retrieves the value of the extended attribute identified by name and associated with the given path in the filesystem. The attribute value is placed in the buffer pointed to by value; size specifies the size of that buffer. The return value of the call is the number of bytes placed in value.

- `time`

  - `time_t time( time_t *second )`
  - returns the time since 00:00:00 UTC, January 1, 1970 (Unix timestamp) in seconds

- `ctime`

  - `char *ctime(const time_t *timer)`
  - returns a string representing the localtime based on the argument `timer`.
  - The returned string has the following format: `Www Mmm dd hh:mm:ss yyyy`, where `Www` is the weekday, `Mmm` the month in letters, `dd` the day of the month, `hh:mm:ss` the time, and `yyyy` the year.

- `readlink`

  - `ssize_t readlink(const char *pathname, char *buf, size_t bufsiz);`
  - `readlink()` places the contents of the symbolic link pathname in the buffer `buf`, which has size `bufsiz`.
  - `readlink()` does NOT append a `null` byte to `buf`.
  - On success, these calls return the number of bytes placed in `buf`. (If the returned value equals `bufsiz`, then truncation may have occurred.) On error, -1 is returned and `errno` is set to indicate the error.

- `malloc`
- `free`
- `perror`
- `strerror`
- `exit`

Structs and typedefs

1. `dirent`

   - Only the fields `d_name` and (as an XSI extension) `d_ino` are specified in POSIX.1. Other than Linux, the `d_type` field is available mainly only on BSD systems. The remaining fields are available on many, but not all systems. Under glibc, programs can check for the availability of the fields not defined in POSIX.1 by testing whether the macros `_DIRENT_HAVE_D_NAMLEN`, `_DIRENT_HAVE_D_RECLEN`, `_DIRENT_HAVE_D_OFF`, or `_DIRENT_HAVE_D_TYPE` are defined.
   - `char d_name[]` – null-terminated file name
   - `ino_t d_fileno` – file serial number

     - the same as `st_ino` returned by `stat`
     - unspecified for symbolic links

   - `unsigned char d_namelen` - length of the file name
   - `unsigned char d_type` - type of the file

        - `DT_UNKNOWN` - (BSD extension) the type is unknown or the filesystem doesn't have support to return the type of the file
        - `DT_REG` - regular file
        - `DT_DIR` - directory
        - `DT_FIFO` - named pipe
        - `DT_SOCK` - local-domain socket
        - `DT_CHR` - character device
        - `DT_BLK` - block device
        - `DT_LNK` - symbolic link

2. `stat`

    - `dev_t st_dev` - id of device containing file
    - `ino_t st_ino` - inode number
    - `mode_t st_mode` - file type and mode

        - `#include <string.h>; strmode(st_mode mode, char *buf);`

    - `nlink_t st_nlink` - number of hard links

        - `short unsigned int`

    - `uid_t st_uid` - user ID of owner

        - `unsigned int`

    - `gid_t st_gid` - group ID of owner
    - `dev_t st_rdev` - device ID (if special file)
    - `off_t st_size` - total size (in bytes)
    - `blksize_t st_blksize` - block size for filesystem I/O
    - `blkcnt_t st_blocks` - number of 512B blocks allocated
    - `struct timespec st_atimespec` - time of last access
    - `struct timespec st_mtimespec` - time of last modification
    - `struct timespec st_ctimespec` - time of last status change

3. `timespec`

    - `#include <time.h>`
    - `time_t tv_sec` - number of whole seconds elapsed since the epoch (for a simple calendar time) or since some other starting point (for an elapsed time)
    - `long int tv_nsec` - number of nanoseconds elapsed since the time given by `tv_spec`

        - is and must be supplied in the range greater than or equal to zero, and less than 1'000'000'000

4. `passwd`

    - `char *pw_name` - User's login name.
    - `uid_t pw_uid` - Numerical user ID.

    - `gid_t pw_gid` - Numerical group ID.
    - `char *pw_dir` - Initial working directory.
    - `char *pw_shell` - Program to use as shell.

5. `group`

    - `char *gr_name;` - Group name.
    - `char *gr_passwd;` - Encrypted password.
    - `gid_t gr_gid;` - Group ID.
    - `char **gr_mem;` - List of group members.

6. `time_t`

    - ISO C defines time_t as an arithmetic type, but does not specify any particular type, range, resolution, or encoding for it. Also unspecified are the meanings of arithmetic operations applied to time values.
    - implement `time_t` as an integer or real-floating type (typically a 32- or 64-bit integer) which represents the number of seconds since the start of the Unix epoch: midnight UTC of January 1, 1970 (not counting leap seconds). Some systems correctly handle negative time values, while others do not. Systems using a signed 32-bit `time_t` type are susceptible to the Year 2038 problem.

Errors

File Name Errors opendir Errors

Requirements

1.