File System Calls – Part3

Directories, File Systems, and Special Files

- Directory
  - Acts as repositories for file names
  - Allows users to group together arbitrary collections of files
  - Can be nested and gives the file structure a hierarchical, tree-like form
- File systems
  - Collections of directories and files
  - Subsections of the hierarchical tree of directories and files
  - Physical sections (“partitions”) of a disk or an entire disk
- Special files
  - UNIX extends the file concept to over the peripheral devices, such as printers, disk units, etc.
  - Files that represent devices are called special files
  - Can be accessed via the file access system calls

mkdir

- #include <sys/stat.h>

  #include <sys/types.h>

  #include <fcntl.h>

  #include <unistd.h>

  int mkdir(const char *_pathname_, mode_t _mode_);
  - Attempts to create a directory named _pathname_.
  - _mode_ specifies the permissions to use.
    - Modified by the process's umask in the usual way.
  - Return value
    - zero on success, or -1 if an error occurred.

rmdir

- #include <unistd.h>

  int rmdir(const char *_pathname_);
  - Deletes a directory, which must be *empty*.
  - Return value
    - On success, zero is returned.
    - On error, -1 is returned.

opendir, closedir

- #include <sys/types.h>

  #include <dirent.h>

  DIR *opendir(const char **name*);
  - Opens a directory stream corresponding to the directory *name*, and returns a pointer to the directory stream.
  - Returns a pointer to the directory stream or NULL if an error occurred.

  int closedir(DIR **dir*);
  - Closes the directory stream associated with *dir*.
  - Returns 0 on success or -1 on failure.

readdir

- #include <sys/types.h>

  #include <dirent.h>

  struct dirent *readdir(DIR **dir*);
  - Returns a pointer to a dirent structure representing the next directory entry in the directory stream pointed to be *dir*.
  - The data returned by readdir is overwritten by subsequent calls to readdir for the same directory stream.
  - Returns a pointer to a *dirent* structure, or NULL if an error occurs or end-of-file is reached.
    - struct dirent {

      ino_t d_ino; /* inode number */

      char d_name[NAMEMAX+1]; /* null-terminated file name */

      };

rewinddir

- #include <sys/types.h>

  #include <dirent.h>

  void rewinddir(DIR *_dir_);
  - Resets the position of the directory stream *dir* to the *beginning* of the directory.
  - Example
    - Refer to 4.4.3 of the textbook

Example #9: readdir (1)

```
#include <sys/types.h>
#include <dirent.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>

void access_perm(char *perm, mode_t mode)
{
    int i;
    char permchar[] = "rwx";

    memset(perm, '-', 10);
    perm[10] = '\0';

    if (S_ISDIR(mode)) perm[0] = 'd';
    else if (S_ISCHR(mode)) perm[0] = 'c';
    else if (S_ISBLK(mode)) perm[0] = 'b';
    else if (S_ISFIFO(mode)) perm[0] = 'p';
    else if (S_ISLNK(mode)) perm[0] = 'l';
```

Example #9: readdir (2)

```
    for (i = 0; i < 9; i++)
        if ((mode << i) & 0x100)
            perm[i+1] = permchar[i%3];

    if (mode & S_ISUID) perm[3] = 's';
    if (mode & S_ISGID) perm[6] = 's';
    if (mode & S_ISVTX) perm[9] = 't';
}

int main(int argc, char *argv[])
{
    DIR *dp;
    struct stat statbuf;
    struct dirent *dent;
    char perm[11];
    char pathname[80];

    if (argc < 2) {
        fprintf(stderr, "no directory name is provided");
        exit(1);
```

Example #9: readdir (3)

```
if (access(argv[1], F_OK) == -1) {
    perror("access error");
    exit(1);
}

if (stat(argv[1], &statbuf) < 0) {
    perror("stat error");
    exit(1);
}

if (!S_ISDIR(statbuf.st_mode)) {
    fprintf(stderr, "%s is not directory\n", argv[1]);
    exit(1);
}

if ((dp = opendir(argv[1])) == NULL) {
    perror("opendir error");
    exit(1);
}
```

Example #9: readdir (4)

```
    printf("Lists of Directory(%s):\n", argv[1]);
    while((dent = readdir(dp)) != NULL) {
        sprintf(pathname, "%s/%s", argv[1], dent->d_name);
        if (stat(pathname, &statbuf) < 0) {
            perror("stat error");
            exit(1);
        }
        access_perm(perm, statbuf.st_mode);
        printf("%s %8ld %s\n", perm, statbuf.st_size, dent->d_name);
    }

    closedir(dp);
    exit(0);
}
```

chdir, fchdir

- #include <unistd.h>

  int chdir(const char *_path_);

  int fchdir(int _fd_);
  - Changes the current directory to that specified in _path_.
  - fchdir( ) uses an open file descriptor as an argument.
  - Return value
    - On success, zero is returned.
    - On error, -1 is returned.
  - Example
    - Refer to 4.4.5 of textbook

Example #10: chdir

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    if (chdir("/tmp") <0) {
        perror("chdir failed");
        exit(1);
    }

    printf("chdir to /tmp succeeded\n");
    exit(0);
}
```

getcwd

- #include <unistd.h>

  char *getcwd(char **buf*, size_t *size*);
  - Get current working directory.
    - Copies the *absolute* pathname of the current working directory to the array pointed to by *buf*, which is of length *size.*
    - If the current absolute path name would require a buffer longer than *size* elements, NULL is returned.
    - Allocates the buffer dynamically using malloc if buf is NULL on call.
  - Return value
    - NULL on failure
    - Address of *buf* on success

Example #11: getcwd

```
#include <stdio.h>
#include <unistd.h>

int main(void)
{
    char *ptr;

    if (chdir("/usr/spool/uucppublic") < 0) {
        perror("chdir failed");
        exit(1);
    }

    if (ptr = getcwd(NULL, 0) == NULL) {
        perror("getcwd failed");
        exit(1);
    }

    printf("cwd = %s\n", ptr);
    exit(0);
}
```

sync, fsync

- #include <unistd.h>

  void sync(void);

  int fsync(int *fd*);
  - To *flush out* all main memory buffers to disk, making sure things are up to date.
  - Typically called by programs that need to examine a file system at a low level via the file system's special file name, or programs that want to preserve data integrity across a machine crash.
  - fsync( ) refers only one file specified by the file descriptor *fd*, and returns 0 if OK, -1 on error.

statvfs, fstatvfs

- #include <sys/statvfs.h>

  int statvfs(const char *_path_, struct statvfs *_buf_);

  int fstatvfs(int _fd_, struct statvfs *_buf_)
  - Return information about the file system which holds the file referenced by the path name or the file descriptor
  - Member of the *struct statvfs*
    - Refer to 4.6.3 of the textbook
  - Example
    - Refer to 4.6.3 of the textbook

pathconf, fpathconf

- #include <unistd.h>

  long int pathconf(const char *_path_, int _name_);

  long int fpathconf(int _fd_, int _name_)
  - Get configuration values of files
    - Return the values of certain system limits for a particular system on a file or directory
  - Second parameter is the symbolic name of the variable to be requested
  - Return value
    - The value for requested system limits on success
    - -1 on failure
  - cf. sysconf()

Example #13: pathconf (1)

```
#include <unistd.h>
#include <stdio.h>

typedef struct{
      int val;
      char *name;
} Table;

main()
{
      Table *tb;
      static Table options[] = {
                { _PC_LINK_MAX, "Maximum number of links"},
                { _PC_NAME_MAX, "Maximum length of a filename"},
                { _PC_PATH_MAX, "Maximum length of pathname"},
               {-1, NULL} };

for (tb=options; tb->name != NULL; tb++)
       printf ("%-28.28s\t%ld\n", tb->name,
                                  pathconf ("/tmp", tb->val));
}
```

Example #13: pathconf (2)

```
dhlee72@cse:~>a.out
Maximum number of links          32000
Maximum length of a filename      255
Maximum length of pathname         4096
```