Files and Directories

stat( ), fstat( ), lstat( )

- synopsis
  - #include <sys/stat.h>
  - #include <unistd.h>

  - int stat(const char *file_name, struct stat *buf);
  - int fstat(int filedes, struct stat *buf);
  - int lstat(const char *file_name, struct stat *buf);

stat( ), fstat( ), lstat( )

- stat
  - pathname이 가리키는 파일에 관한 정보를 buf가 가리키는 struct stat에 저장한다.
- fstat
  - stat과 동일하지만 파일을 이미 open한 파일 descriptor로 지정한다.
- lstat
  - stat과 유사하지만 파일이 symbolic link일 경우 symbolic link가 참조하는 파일이 아니라 symbolic link 그 자체의 파일에 대한 정보를 return한다

stat( ), fstat( ), lstat( )

```
struct stat {
    dev_t st_dev;     /* device */
    ino_t st_ino;     /* inode */
    mode_t st_mode;   /* protection */
    nlink_t st_nlink;    /* number of hard links */
    uid_t st_uid;     /* user ID of owner */
    gid_t st_gid;     /* group ID of owner */
    dev_t st_rdev;    /* device type (if inode device) */
    off_t st_size;    /* total size, in bytes */
    unsigned long st_blksize; /* blocksize for I/O */
    unsigned long st_blocks;  /* # of blocks allocated */
    time_t st_atime;    /* time of last access */
    time_t st_mtime;   /* time of last modification */
    time_t st_ctime;    /* time of last change */
};
```

stat( ), fstat( ), lstat( )

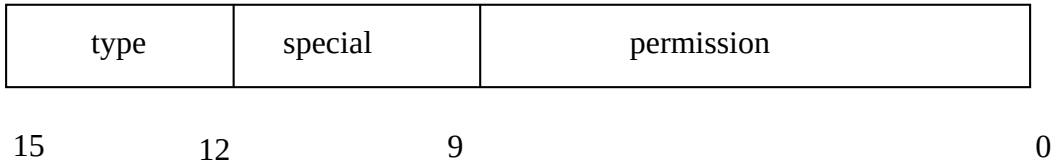
- description
  - return information about the specified file
  - do not need any access rights to the file
  - need search rights to all directories named in the path leading to the file
- all return a stat structure
  - st_blocks
    - the size of the file in 512-byte blocks
  - st_blksize
    - the "preferred" block size for efficient file system I/O
    - Writing to a file in smaller chunks may cause an inefficient read-modify-rewrite

stat( ), fstat( ), lstat( )

- st_atime
  - changed by exec(), mknod(), pipe(), utime(), read()
  - atime may not be updated
    - ex) mounting with noatime option
- st_mtime
  - changed by mknod(), truncated(), utime(), write( )
  - st_mtime of a directory is changed by creation or deletion of files in that directory.
- st_ctime
  - changed by writing or by setting inode information
  - owner, group, link count, mode, etc.

stat( ), fstat( ), lstat( )

- st_mode:
  - st_mode는 아래 그림과 같이 file type, special bit, file permission bit의 3 부분으로 나뉜다



stat( ), fstat( ), lstat( )

file type

- #define S_IFMT 00170000 /* type of file */
- #define S_IFSOCK 0140000 /* socket */
- #define S_IFLNK 0120000 /* symbolic link */
- #define S_IFREG 0100000 /* regular */
- #define S_IFBLK 0060000 /* block special */
- #define S_IFDIR 0040000 /* directory */
- #define S_IFCHR 0020000 /* character special */
- #define S_IFIFO 0010000 /* FIFO */

stat( ), fstat( ), lstat( )

- file type macro

Macro	Type of file
S_ISREG()	regular file
S_ISDIR()	directory file
S_ISCHR()	character special file
S_ISBLK()	block special file
S_ISFIFO()	pipe or FIFO
S_ISLNK()	symbolic link
S_ISSOCK()	socket

- special bits
  - #define S_ISUID 0004000 /* set uid on execution */
  - #define S_ISGID 0002000 /* set group id on execution */
  - #define S_ISVTX 0001000 /* save text(sticky bit) */

stat( ), fstat( ), lstat( )

- permission bits

```
#define S_IRUSR 00400     /* read permission: owner */
#define S_IWUSR 00200     /* write permission: owner */
#define S_IXUSR 00100     /* execute permission: owner */
#define S_IRGRP 00040     /* read permission: group */
#define S_IWGRP 00020     /* write permission: group */
#define S_IXGRP 00010     /* execute permission: group */
#define S_IROTH 00004     /* read permission: other */
#define S_IWOTH 00002     /* write permission: other */
#define S_IXOTH 00001     /* execute permission: other */
```

stat( ), fstat( ), lstat( )

- return value
  - On success, zero is returned
  - On error, -1 is returned
- cf) “stat” shell command
- example

```
struct stat buf;
stat(“/etc/passwd”, &buf);
if (buf.st_mode & S_IRUSR) {
    ...
```

statfs(), fstatfs()

- synopsis
  - #include <sys/vfs.h>
  - int statfs(const char *path, struct statfs *buf);
  - int fstatfs(int fs, struct statfs *buf);
- description
  - returns information about a mounted file system
  - path is the path name of any file within the mounted file system

statfs(), fstatfs()

```
struct statfs {
    long    f_type; /* type of filesystem (magic number, EXT2: 0xEF53) */
    long    f_bsize;    /* optimal transfer block size */
    long    f_blocks;   /* total data blocks in file system */
    long    f_bfree;    /* free blocks in fs */
    long    f_bavail;   /* free blocks avail to non-superuser */
    long    f_files;    /* total file nodes in file system */
    long    f_ffree;    /* free file nodes in fs */
    fsid_t  f_fsid;     /* file system id */
    long    f_namelen;  /* maximum length of filenames */
    long    f_spare[6]; /* spare for later */
};
```

File Access Permissions

- real ID and effective ID
  - real user(group) ID
    - the user(group) ID who executed the process
  - effective user(group) ID
    - user(group) ID used to assign ownership of newly created file or to check access permission when system call is done
  - real user(group) ID can be different from the real user(group), in the case of programs with the setuid(setgid) bit set

File Access Permissions

- example

  -rwxr-xr-x student bitgrp 43753 a.out

  chmod u+s a.out

  -rwsr-xr-x student bitgrp 43753 a.out

  - the effective user id of the process which is executing “a.out” is “student”, and the real user id is “guest”

File Access Permissions

- regular file
  - read permission
    - can open with O_RDONLY and O_RDWR flags
  - write permission
    - can open with O_WRONLY and O_RDWR flags
  - must have write permission to specify the O_TRUNC
- example

  user = stud, group = bitgrp

  --wxr-x--- stud bitgrp testfile

File Access Permissions

- directory
  - whenever want to open any type of file by name, must have execute permission in each directory mentioned in the name
  - read permission for directory
    - lets us read the directory, obtaining a list of all the filenames in the directory
  - execution permission for directory
    - lets us pass through the directory when it is a component of a pathname that we are trying to access

File Access Permissions

- cannot create a new file in a directory unless we have write permission and execute permission in the directory
- to delete a file, we need write permission and execute permission in the directory containing the file
  - do not need read permission or write permission for the file itself

- ownership of new files
  - the effective user ID and group ID of the creating process

access( )

- synopsis
  - #include <unistd.h>
  - int access(const char *pathname, int mode);
- description
  - checks whether the process would be allowed to read, write or test for existence of the file whose name is pathname
  - If pathname is a symbolic link permissions of the file referred to by this symbolic link are tested

access( )

- mode
  - R_OK
    - file exist, read permission
  - W_OK
    - file exist, write permission
  - X_OK
    - file exist, execution permission
  - F_OK
    - file exist
- The tests depend on the permissions of the directories occurring in the path to the file

access( )

- The check is done with the process's real uid and gid
  - This is to allow set-UID programs to easily determine the invoking user's authority
- return value
  - On success zero is returned
  - On error -1 is returned
  - returns an error if any of the access types in the requested call fails, even if other types might be successful
  - access("bit", R_OK|W_OK) : check read and write permission

umask( )

- synopsis
  - #include <sys/types.h>
  - #include <sys/stat.h>
  - mode_t umask(mode_t mask);
- description
  - set file creation mask
    - sets the umask to mask & 0777
  - permissions in the umask are turned off from the mode argument to open( )

umask( )

- return value
  - This system call always succeeds and the previous value of the mask is returned

- cf) “umask” shell command
- example

```
umask(022);
fd = creat(“tmp”, 0666);
...
```

  - permission of “tmp” is -rw-r--r-- (0644)

chmod( ) and fchmod( )

- synopsis
  - #include <sys/types.h>
  - #include <sys/stat.h>
  - int chmod(const char *path, mode_t mode);
  - int fchmod(int fildes, mode_t mode);
- description
  - change permissions of a file
  - The mode of the file given by path or referenced by fildes is changed

chmod( ) and fchmod( )

- mode is specified by or'ing the following
  - S_ISUID, S_ISGID, S_ISVTX, S_I{R,W,X}{USR,GRP,OTH}, S_IRWX{U,G,O}
- effective UID of the process must be zero or must match the owner of the file
- SGID bit is turned off when
  - effective UID of the process is not zero
  - the group of the file does not match the effective group ID of the process or one of its supplementary group IDs

chmod( ) and fchmod( )

- return value
  - On success, zero is returned
  - On error, -1 is returned

Sticky bit

- S_ISVTX bit
  - saved text
  - set for an executable program file
    - the first time the program was executed a copy of the program's text was saved in the swap area when the process terminated
    - causes the program to load into memory faster
    - today's Unix systems have a virtual memory system and a faster file system, the need for this technique has disappeared

Sticky bit

- set for a directory
  - a file in the directory can be removed or renamed only if
    - the user has write permission for the directory and
    - the user owns the file or owns the directory or the user is superuser

- example

  - chmod("/tmp", S_ISVTX|0777);
  - permission of "/tmp" is drwxrwxrwt

chown( ), fchown( ), lchown( )

- synopsis
  - #include <sys/types.h>
  - #include <unistd.h>

  - int chown(const char *path, uid_t owner, gid_t group);
  - int fchown(int fd, uid_t owner, gid_t group);
  - int lchown(const char *path, uid_t owner, gid_t group);

chown( ), fchown( ), lchown( )

- description
  - The owner of the file specified by path or by fd is changed
  - Only the super-user may change the owner of a file
  - The owner of a file may change the group of the file to any group of which that owner is a member
  - when the group of an executable file is changed by a non-super-user, setgid bit is cleared.

chown( ), fchown( ), lchown( )

- If the owner or group is specified as -1, then that ID is not changed
- return value
  - On success, zero is returned
  - On error, -1 is returned

truncate( ) and ftruncate( )

- synopsis
  - #include <unistd.h>
  - int truncate(const char *path, off_t length);
  - int ftruncate(int fd, off_t length);
- description
  - truncate a file to a specified length
  - If the file previously was larger than length, the extra data is lost
  - If the file previously was smaller than length, the hole is created

truncate( ) and ftruncate( )

- With ftruncate, the file must be open for writing
- return value
  - On success, zero is returned
  - On error, -1 is returned

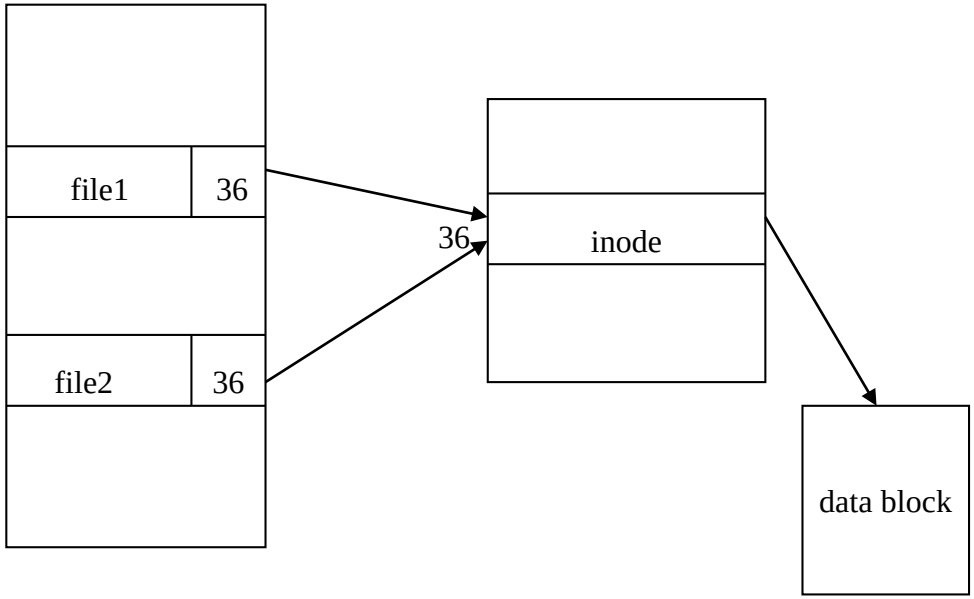
link( )

- synopsis
  - #include <unistd.h>
  - int link(const char *oldpath, const char *newpath);
- description
  - make a new name for a file
    - creates a new link (also known as a hard link) to an existing file
  - If newpath exists it will not be overwritten

link( )

- This new name may be used exactly as the old one for any operation
  - both names refer to the same file
  - have the same permissions and ownership
  - it is impossible to tell which name was the original
- Hard links, as created by link, cannot span file systems
- Cannot create a link to a directory

link( )



link(“file1”, “file2”);

link( )

- link count
  - regular file
    - the link count is the number of the names
  - directory
    - all the directories have at least two links (“.” and directory name)
    - all the directories have a link to the parent directory
    - the link count of a directory is 2 + # of sub directories

unlink( )

- synopsis
  - #include <unistd.h>
  - int unlink(const char *pathname);
- description
  - delete a name and possibly the file it refers to
    - deletes a name from the file system
    - If that name was the last link to a file and no processes have the file open the file is deleted

unlink( )

- If the name was the last link to a file but any processes still have the file open the file will remain in existence until the last file descriptor referring to it is closed
- If the name referred to a symbolic link the link is removed

- return value
  - On success, zero is returned. On error, -1 is returned

remove( )

- synopsis
  - #include <stdio.h>
  - int remove(const char *pathname);
- description
  - library function(not system call)
  - delete a name and possibly the file it refers to
    - It calls unlink for files, and rmdir for directories
  - return value
    - On success, zero is returned. On error, -1 is returned

rename( )

- synopsis
  - #include <stdio.h>
  - int rename(const char *oldpath, const char *newpath);
- description
  - renames a file or a directory, moving it between directories if required
  - oldpath and newpath should be in the same file system

rename( )

- If newpath already exists it will be atomically replaced
  - if oldpath is not a directory, newpath should not be a directory
  - if oldpath is a directory and newpath exists, newpath should be an empty directory
- If oldpath refers to a symbolic link the link is renamed
- if newpath refers to a symbolic link the link will be overwritten

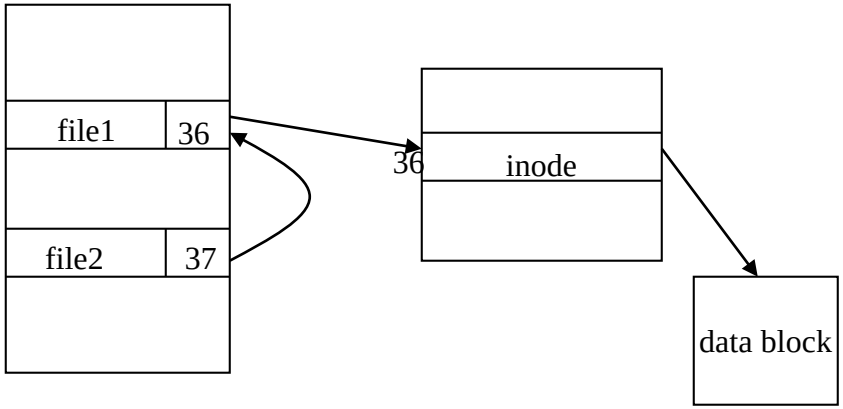
Symbolic Link

- symbolic link is indirect pointer to a file
  - hard link points directly to the inode of the file
  - limitation of hard links
    - require that the link and the file reside in the same file system
    - a hard link to a directory can't be created
  - there are no file system limitations on a symbolic link
  - anyone can create a symbolic link to a directory

symlink( )

- synopsis
  - #include <unistd.h>
  - int symlink(const char *oldpath, const char *newpath);
- description
  - creates a symbolic link named newpath which contains the string oldpath
  - Symbolic links are interpreted at run-time
  - dangling link
    - may point to an non-existing file

symlink( )



$ ln -s testfile newfile
$ ls -l newfile
 1 lrwxrwxrwx 1 yhshin users 8 Aug 27 20:02 newfile -> testfile
$ rm testfile
$ cat newfile
cat: newfile: No such file or directory
하드링크와 달리 원래 파일을 지우면 reference할 수 없다.

symlink( )

- The permissions of a symbolic link are irrelevant
  - the ownership is ignored when following the link
  - permission is checked when removal or renaming of the link is requested and the link is in a directory with the sticky bit set
- If newpath exists it will not be overwritten
- return value
  - On success, zero is returned. On error, -1 is returned

readlink( )

- synopsis
  - #include <unistd.h>
  - int readlink(const char *path, char *buf, size_t bufsiz);
- description
  - read value of a symbolic link
    - places the contents of the symbolic link path in the buffer buf, which has size bufsiz
    - does not append a NULL character to buf

readlink( )

- return value
  - the count of characters placed in the buffer if it succeeds
  - -1 if an error occurs

utime( )

- synopsis
  - #include <sys/types.h>
  - #include <utime.h>
  - int utime(const char *filename, struct utimbuf *buf);
- description
  - change access and/or modification times of an inode
    - changes the access and modification times of the inode specified by filename to the actime and modtime fields of buf respectively

utime( )

- If buf is NULL, then the access and modification times of the file are set to the current time

```
struct utimbuf {
   time_t actime;  /* access time */
   time_t modtime; /* modification time */
};
```

- return value
  - On success, zero is returned. On error, -1 is returned

utime( ) - example

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

int main(int argc, char *argv[])
{
    int    i;
    struct stat      statbuf;
    struct utimbuf           timebuf;
```

utime( ) - example

```
for (i = 1; i < argc; i++) {
        if (stat(argv[i], &statbuf) < 0) {
                fprintf(stderr, "%s: stat error", argv[i]);
                continue;
        }
        if (open(argv[i], O_RDWR | O_TRUNC) < 0) {
                fprintf(stderr, "%s: open error", argv[i]);
                continue;
        }
```

utime( ) - example

```
            timebuf.actime  = statbuf.st_atime;
            timebuf.modtime = statbuf.st_mtime;
            if (utime(argv[i], &timebuf) < 0) {
                        fprintf(stderr, "%s: utime error", argv[i]);
                        continue;
            }
    }
}
```

mkdir( )

- synopsis
  - #include <sys/stat.h>
  - #include <sys/types.h>
  - #include <fcntl.h>
  - #include <unistd.h>

  - int mkdir(const char *pathname, mode_t mode);

mkdir( )

- description
  - attempts to create a directory named pathname
  - mode specifies the permissions to use
    - modified by the process's umask in the usual way
  - ownership
    - The newly created directory will be owned by the effective uid of the process
    - If the directory containing the file has the set group id bit set, the new directory will inherit the group ownership from its parent

mkdir( )

- otherwise it will be owned by the effective gid of the process

- set group id bit
  - If the parent directory has the set group id bit set then so will the newly created directory
- return value
  - zero on success, or -1 if an error occurred

rmdir( )

- synopsis
  - #include <unistd.h>
  - int rmdir(const char *pathname);
- description
  - deletes a directory, which must be empty
  - return value
    - On success, zero is returned. On error, -1 is returned

Reading Directories

- synopsis
  - #include <sys/types.h>
  - #include <dirent.h>
  - DIR *opendir(const char *name);
  - struct dirent *readdir(DIR *dir);
  - off_t telldir(DIR *dir);
  - void seekdir(DIR *dir, off_t offset);
  - void rewinddir(DIR *dir);
  - int closedir(DIR *dir);

Reading Directories

- 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
- 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

Reading Directories

- 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;
   char d_name[NAMEMAX+1];
}
```

Reading Directories

- off_t telldir(DIR *dir);
  - returns the current location associated with the directory stream dir.
  - returns –1 if an error occurs
- void seekdir(DIR *dir, off_t offset);
  - sets the location in the directory stream from which the next readdir( ) call will start.
  - should be used with an offset returned by telldir( ).

Reading Directories

- void rewinddir(DIR *dir);
  - resets the position of the directory stream dir to the beginning of the directory
- int closedir(DIR *dir);
  - closes the directory stream associated with dir
  - returns 0 on success or -1 on failure

scandir( )

- synopsis
  - #include <dirent.h>
  - int scandir(const char *dir,

    struct dirent ***namelist,

    int (*select)(const struct dirent *),

    int (*compar)(const struct dirent **, const struct dirent **));
- return value
  - the number of directory entries selected or –1 (error)

scandir( )

- description
  - scans the directory dir, calling select() on each directory entry.
  - Entries for which select() returns non-zero are stored in strings allocated via malloc(), sorted using comparison function compar(), and collected in array namelist which is allocated via malloc().
  - If select is NULL, all entries are selected.

scandir( )

- example

```
#include <dirent.h>
#include <string.h>
int selfile(const struct dirent *p)
{
    if (strstr(p->d_name, ".c"))
      return 1;
    return 0;
}
int mysort(const struct dirent **a, const struct dirent **b)
{
    return strlen((*a)->d_name) - strlen((*b)->d_name);
}
```

scandir( )

```
main(int argc, char *argv[])
{
    struct dirent **namelist;
    int n, i;
    if ((n = scandir(argv[1], &namelist, selfile, mysort)) < 0) {
      perror("scandir");
      exit(1);
    }
    for (i = 0; i < n; i++) {
      printf("%s\n", namelist[i]->d_name);
    }
}
```

Example

```
#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’;
```

Example

```
    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';
    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';
}
```

Example

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

    if (argc < 2) exit(1);
    if (access(argv[1], F_OK) == -1) {
            perror("Error");
            exit(1);
    }
```

Example

```
stat(argv[1], &statbuf);
if (!S_ISDIR(statbuf.st_mode)) {
        fprintf(stderr, "%s is not directory\n",
                argv[1]);
        exit(1);
}
if ((dp = opendir(argv[1])) == NULL) {
        perror("Error:");
        exit(1);
}
printf("Lists of Directory(%s):\n", argv[1]);
```

Example

```
    while((dent = readdir(dp)) != NULL) {
            sprintf(pathname, "%s/%s", argv[1],
                    dent->d_name);
            stat(pathname, &statbuf);
            access_perm(perm, statbuf.st_mode);
            printf("%s %8ld %s\n", perm,statbuf.st_size,
                    dent->d_name);
    }
    closedir(dp);
}
```

chdir( ) and fchdir( )

- synopsis
  - #include <unistd.h>
  - int chdir(const char *path);
  - int fchdir(int fd);
- description
  - changes the current directory to that specified in path
  - fchdir uses an open file descriptor as an argument

chdir( ) and fchdir( )

- example

```
#include <unistd.h>
#include <stdio.h>
main(void)
{
        if (chdir("/tmp") < 0) {
                perror("chdir");
                exit(1);
        }
        printf("chdir to /tmp succeeded\n");
}
```

getcwd( )

- synopsis
  - #include <unistd.h>
  - char *getcwd(char *buf, size_t size);

- description
  - 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

getcwd( )

- 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
  - buf on success

chroot( )

- synopsis
  - #include <unistd.h>
  - int chroot(const char *path);
- description
  - changes the root directory to that specified in path.
  - only the super-user may change the root directory.
- return value
  - On success, zero is returned.
  - On error, -1 is returned.

sync( )

- synopsis
  - #include <unistd.h>
  - int sync(void);
- description
  - first commits inodes to buffers, and then buffers to disk.
- return value
  - always returns 0.

fsync( )

- synopsis
  - #include <unistd.h>
  - int fsync(int fd);
- description
  - copies all in-core parts of a file to disk.
- return value
  - On success, zero is returned.
  - On error, -1 is returned

fdatasync( )

- synopsis
  - #include <unistd.h>
  - int fdatasync(int fd);
- description
  - flushes all data buffers of a file to disk.
  - does not update the metadata.
- return value
  - On success, zero is returned.
  - On error, -1 is returned