Languages for Informatics

9 – Low Level System Calls

Department of Computer Science
University of Pisa
Largo B. Pontecorvo 3
56127 Pisa

Topics

- Linux programming environment (2h)
- Introduction to C programming (12h)
- Basic system programming in Linux (10h)
  1. Signals and Error Handling
  2. Low-Level System Calls in C
  3. Multi-Tasking in C
  4. Multi-Threading in C
  5. Machine-To-Machine Communication in C

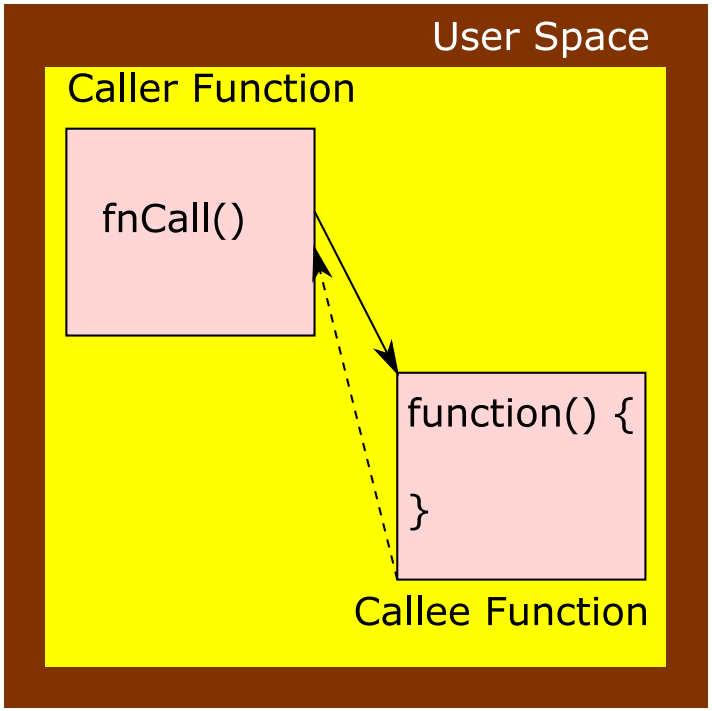
Overview

Introduction

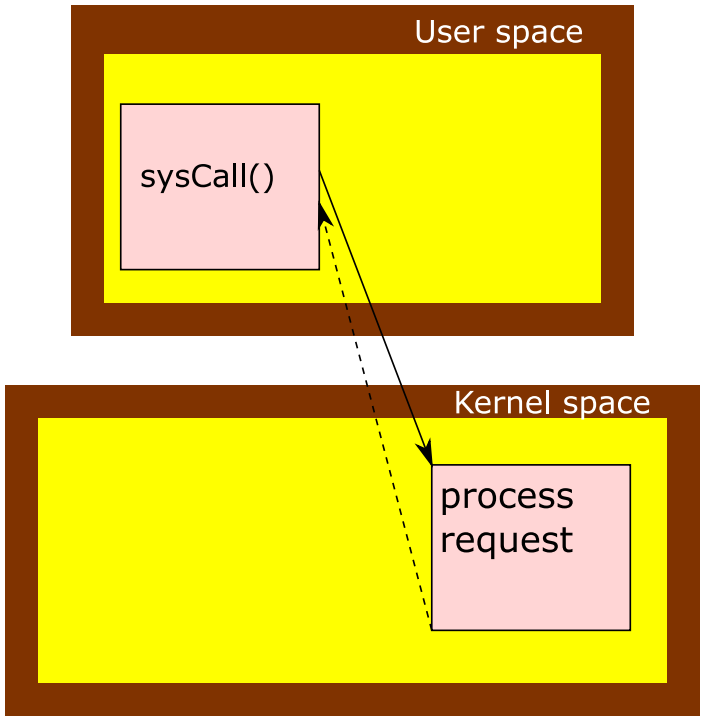
Two fundamental questions:

- Q1 How does the **OS** communicate with an **application process**?
- A1 Signals (last lecture)
- Q2 How does an **application process** communicate with the **OS**?
- A2 System Calls (this lecture)

System Calls vs. Function Calls



- Caller and callee are in the same Process
  - Same user
  - Same ”domain of trust”



- Process in user space, callee in kernel space
- OS is trusted; user is not
- OS has SU privileges; user does not.

Steps for making a System Call

Procedure

- Store `sysCall` arguments in registers and switch to kernel mode
- The OS code takes control of the CPU, privileges are updated, and value of all registers is saved
- The OS examines the call parameters and type of `sysCall`
- The OS performs the requested function
- The OS restores value of all registers
- The OS returns control of the CPU to the caller

Examples of System Calls

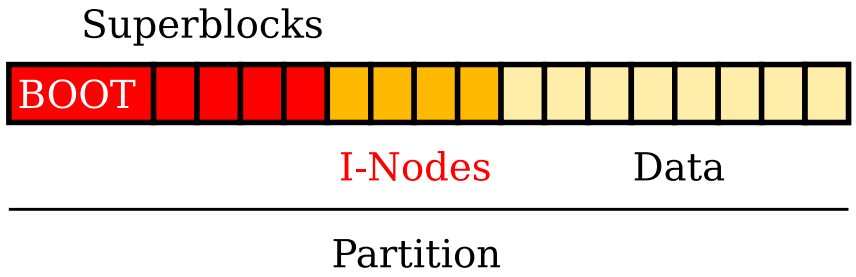
- System Calls
  - **`open()`** // opens a file
  - **`mkdir()`** // attempts to create a directory
  - **`execve()`** // executes a program
- Function Calls – in contrast –
  - **`fopen()`**
  - **`printf()`**
  - **`qsort()`**

Principles
I-Nodes (1)

- Linux is an **i-node** based file system, attached to a particular device
- The i-node is a data structure, containing
  - **I-node number**: a unique number assigned to objects when created
  - **Type**: regular file, directory, symbolic link, named pipe (FIFO), socket, ...
  - **Protection mode**: read/write/execute for Owner, Group, Others
  - **Ownership**: User and Group
  - **Timestamps**: Time of last access, modification and status change
  - **Number of hard-links**
  - **Pointers to block** of actual data

Principles
I-Nodes (2)

- Organization of a partition for a UNIX file system

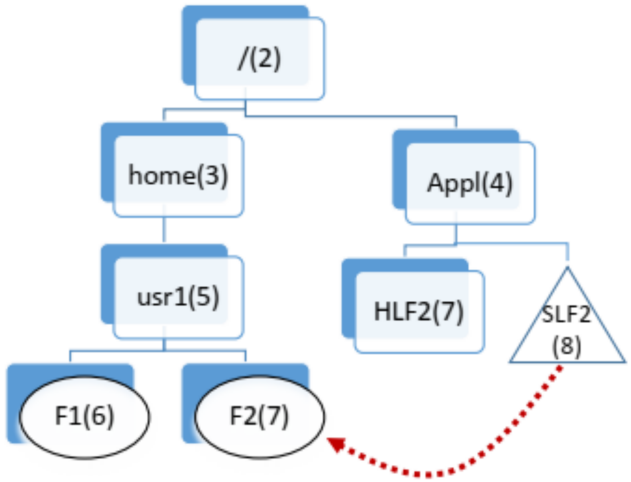


- The **superblock** contains various information on the File System.
  - File System Size
  - # free blocks, free block list, next free block
  - Inode list size, # free inodes, free inode list
  - It is crucial, and if damaged it would make the data inaccessible
  - Several copies made in various blocks (at regular intervals)

Principles
I-Nodes (3)

- Logical structure to organize files in form of **extent tree**

Directory: /		Dir: /home/usr1	
Name	I-Node	Name	I-Node
	2		3
	2		5
home	3	file1	6
Appl	4	file2	7



Note

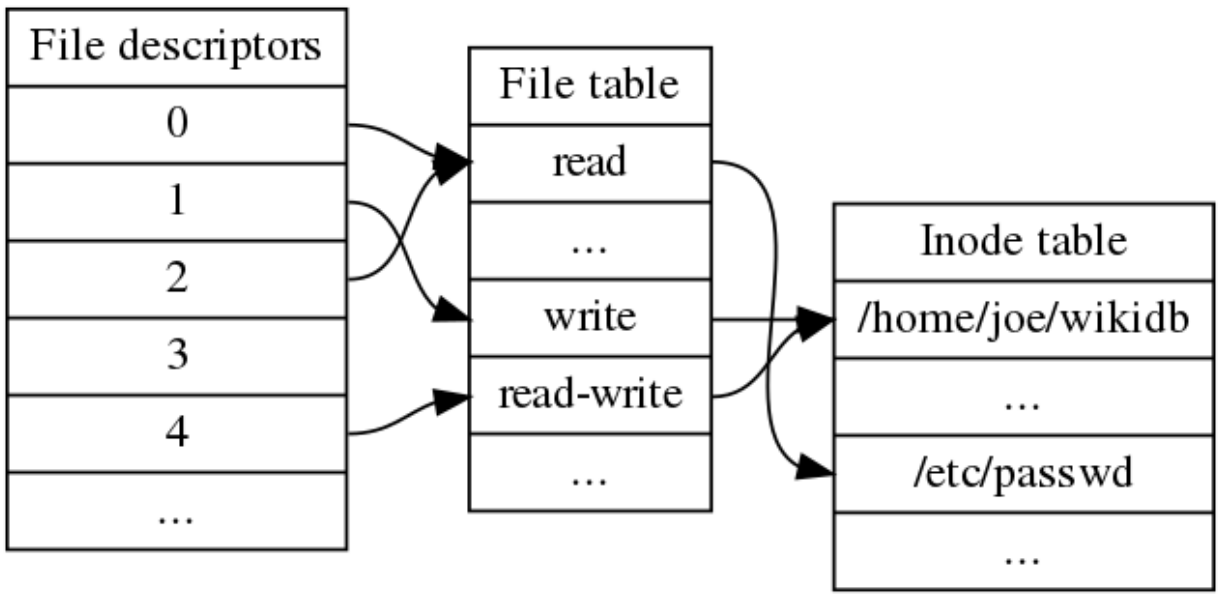
Soft-links are **pointers** to another file/directory.

Principles
File Descriptors

- A **file descriptor** (FD) is an abstract indicator (handle) used to access an open stream (file, directory, pipe, socket, ...)
- **maintained by the kernel**
- organized as **index array of open files** in contrast to an i-node [which is a filesystem structure representing files]
- **Range of possible values** of file descriptors is from 0 to 1023 for (32-bit or 64-bit) Linux system.

Principles
File Offset

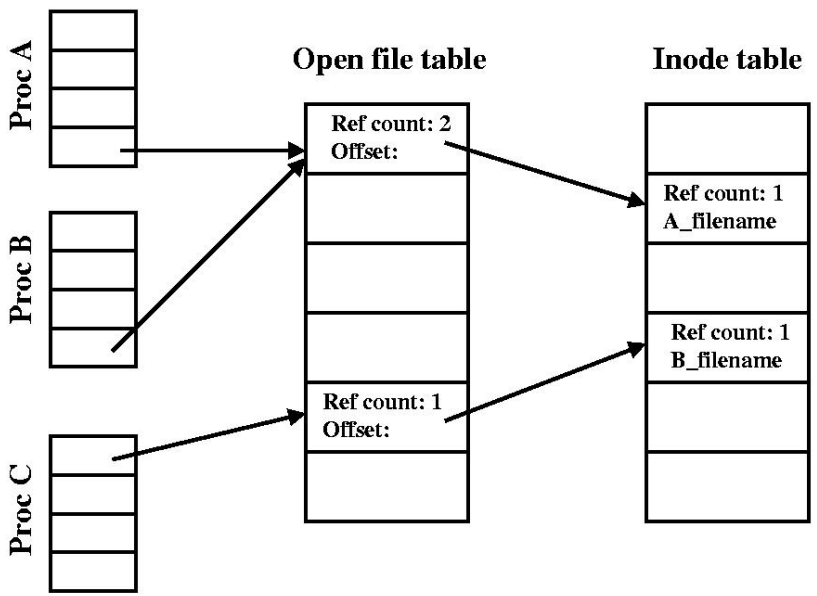
- One table of file descriptors **per process**
- Table of open files (status, including opening mode and current position in the file (=offset)
- I-node table for all open files



Source: Wikiwand

Principles
File Offset and File Sharing

- **File sharing in UNIX**
  - When two or more processes open **the same** file for reading, there are **more** entries for the Active Process Table and the Table of Open files but **one** in the i-node table
  - Entry in Table of Open files is shared between processes if FD are **`dup()`**ed or **`fork()`**ed.



Principles
Example

Relation between FD and actual stream.

Example

```
$ ls -al /proc/45973/fd
lrwx------ 1 pc pc 64 nov 10 19:35 0 -> /dev/pts/1
lrwx------ 1 pc pc 64 nov 10 19:35 1 -> /dev/pts/1
lrwx------ 1 pc pc 64 nov 10 19:35 2 -> /dev/pts/1
lr-x------ 1 pc pc 64 nov 10 19:35 3 ->
'/tmp/sh-thd.iuwbku (deleted)'
lrwx------ 1 pc pc 64 nov 10 19:35 4 ->
'socket:[482834]'
```

Remember that `stdin` (FD = 0), `stdout` (FD = 1) and `stderr` (FD = 2).

Principles
Overview

Inode Manipulation

- **`stat()` `access()`**, **`link()`**, **`unlink()`**, **`chown()`**, **`chmod()`**, **`mknod()`**, ...
- many of these system calls have l-prefixed variants (e.g., **`lstat()`**) that **do not** follow soft links
- many of these system calls have f-prefixed variants (e.g., **`fstat()`**) operating on file descriptors

File descriptor manipulation

- **`open()`**, **`creat()`**, **`close()`**, **`read()`**, **`write()`**, **`lseek()`**, **`fcntl()`**, ...
- **`open()`** may also create a new file (hence a new inode)
- Use **`fdopen()`** and **`fileno()`** to get a file C library `FILE*` from a file descriptor and reciprocally.

Open and possibly create a file (1)

- Syntax

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int flags[, mode_t
    mode]);
```

- **Path** to the file which you want to use
  - use absolute or relative path w.r.t. root "/" or current directory "./", resp.
- **flags**
  - set to **one** of `O_RDONLY`: read only; `O_WRONLY`: write only; `O_RDWR`: read and write; `O_APPEND`: append; **possibly combined** with `O_CREAT`: create file if it doesn't exist; `O_EXCL`: prevents creation by `O_CREAT` if it already exists.
  - more flags can be combined by logical OR "|".

Open and possibly create a file (2)

- Syntax

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int flags[, mode_t
    mode]);
```

- The **mode** argument must be supplied if `O_CREAT` or `O_TMPFILE` is specified in flags; if it is not supplied, some arbitrary bytes from the stack will be applied as the file mode.
  - For example, the mode `0644` gives `u+rw`, `g+r`, `o+r` permission.

Open and possibly create a file (3)

- Syntax

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int open(const char *pathname, int flags[, mode_t
    mode]);
```

Return Value

- On success, the system call returns a (non-negative) `int` **file descriptor** (i.e. the *lowest-numbered file descriptor of the process not currently open*)
- Return `-1` on error

File Access

Open a stream through a File Descriptor

- To associate a standard I/O stream with an existing file descriptor (the filedesc argument),

```
FILE *fp = fdopen(int fd, const char *mode)
```

  File descriptors are obtained from the **system calls** **`open()`**, **`dup()`**, or **`pipe()`**, which open files but do not return pointers to a `FILE` structure.

Close a File Descriptor

- Syntax

```
#include <fcntl.h>
int close(int fd);
```

Return Value

- On success, the system call returns `0`
- On error, `-1` is returned.
- When closing the last descriptor to a file that has been removed using `unlink()`, the file is effectively deleted.

Example

```c
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>    // file control options
#include<errno.h>
#include<unistd.h>    // close

int main()
{
    int fd1 = open("foo.txt", O_RDWR);   //read/write
    if (fd1 < 0)
    {
        printf("Error Number %d\n", errno);
        perror("main");
        exit(1);
    }
    printf("I opened fd = %d\n", fd1);
```

Example (cont'd)

```
    if (close(fd1) < 0)
    {
        printf("Error Number %d\n", errno);
        perror("main");
        exit(1);
    }
    printf("I closed the fd.\n");
}
```

Shell

```
$ ./a.out
Error Number 2
main:  No such file or directory
$ echo "" > foo.txt && ./a.out
I opened fd = 3
I closed the fd.
```

Read from File Descriptor (1)

- Syntax

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

- Attempts to read **up to** `count` bytes from file descriptor `fd` into the buffer starting at `buf`
  - can return fewer bytes than you asked for due to EOF, signal, reading from socket etc.

Return value

- On success, the number of bytes read is returned
- On EOF, **0**
- On error, **-1**.

Read from File Descriptor (2)

- Syntax

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

- Important **`errno`** codes
  - `EINTR`: The call was interrupted by a signal before any data was read
  - `EAGAIN`: The file descriptor `fd` refers to a socket and has been marked nonblocking (`O_NONBLOCK`), and the read would block.

Write to File Descriptor (1)

- Syntax

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

- Attempts to write **up to** `count` bytes to the file descriptor `fd` from the buffer starting at `buf`
  - If the file was opened with `O_APPEND`, the file offset is first set to the end of the file before writing.
  - may write less than `count` bytes due to insufficient disk space, signal, writing to socket etc.

Return value

- On success, the number of bytes **effectively written** is returned
- When nothing was written, return **0**.
- On error, return **-1**.

Write to File Descriptor (2)

- Syntax

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

- Important **`errno`** code
  - `ENOSPC`: no space left on device containing the file.

Unlink File Descriptor (1)

- **`unlink()`** - Delete a name and possibly the file it refers to.
- Syntax

```
#include <unistd.h>
int unlink(const char *pathname);
```

Note

- Having unlinked it, the file **will remain in existence** until the last file descriptor referring to it is closed.
- This is **also true for socket, fifo or device** when active processes still have the object open.

Unlink File Descriptor (2)

- **`unlink()`** - Delete a name and possibly the file it refers to.
- Syntax

```
#include <unistd.h>
int unlink(const char *pathname);
```

Error Conditions

- The system call returns **0** on success, **-1** on error
- The **`errno`** code `EISDIR` indicates that the `pathname` refers to a directory.

open() - read()

Example for appending text

```
int my_read(char *pathname, int count, char *buf) {
  int fd;
  if ((fd = open(pathname, O_RDWR|O_APPEND|O_CREAT, 0644)) ==
    -1) {
     perror("open()");
     exit(1);
  }

  int progress, remaining = count;  // Read count bytes

  while ((progress = read(fd, buf, remaining)) != 0) { //
    Iterate while progess or recoverable error
    if (progress == -1) {
      if (errno == EINTR)
        continue; // Interrupted by signal, retry
      perror("read()");
      exit(1);
    }
  buf += progress; // points to the last element.
  remaining -= progress;
  }
```

write() - close()

Example for appending text

```
void my_write(int fd, int count, char *buf) {
  int progress, remaining = count;  // Read count bytes
  while ((progress = write(fd, buf, remaining)) != 0) {
    if (progress == -1) {
      perror("write()");
      exit(1);
    }
  buf += progress; // Pointer arithmetic
  remaining -= progress;
  }

  if ((fd = close(fd)) == -1) {
     perror("close()");
     exit(1);
  }
}
```

main()
Example for appending text

```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<errno.h>
#include<unistd.h>
#include<string.h>

int my_read(char *pathname, int count, char *buf);
void my_write(int fd, int count, char *buf);

int main() {
  int BUF_SIZE = 1000;
  int fd;
  char *buf = (char *) malloc(BUF_SIZE* sizeof(char));
  char append[] = "Hello World! \n";

  char pathname[] = "foo.txt";
  fd = my_read(pathname, BUF_SIZE, buf);
  my_write(fd, strlen(append), append); //append new string
  return 0;
}
```

All together
Example

Compile with

Shell

```
bash~$ gcc append.c my_read.c my_write.c -Wall
```

Benchmark
glibc vs. system call

glibc vs. system call.

glibc	System Call
`fopen()`, high-level C call	`open()`, low-level sys call
`fread()` - **buffered** I	`read()` - unbuffered I, **POSIX**
`fwrite()`- buffered O	`write()` - unbuffered O, POSIX
`fclose()`	`close()`, POSIX
`remove()`	`unlink()`

Changing the read/write pointer: `lseek()` (1)

- Syntax:

```
#include <sys/types.h>
#include <unistd.h>

off_t lseek(int fd, off_t offset, int whence);
```

- **`lseek()`** repositions the file offset of the open file description associated with the file descriptor `fd` to the argument `offset` according to the directive `whence` as follows:
  - `SEEK_SET`: The offset is set to offset bytes.
    `SEEK_CUR`: The offset is set to its current location plus offset bytes.
    `SEEK_END`: The offset is set to the size of the file plus offset bytes.

Changing the read/write pointer: `lseek()` (2)

Examples:

```
/* begin or EOF */
lseek(fd, 0, SEEK_SET);
lseek(fd, 0, SEEK_END); /* (*) */
/* where am I? */
pos = lseek(fd, 0, SEEK_CUR);
/* go back by one 1 Byte */
lseek(fd, -1, SEEK_CUR);
/* at absolute position k */
lseek(fd, k, SEEK_SET);
```

Note

`lseek()` allows the file offset to be set beyond the end of the file. If data is later written at this point, gap is filled with `\0`.

Get file status: `stat()`, `lstat()`, `fstat()`

- Syntax

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

int stat(const char *path, struct stat *statbuf);
int fstat(int FD, struct stat *statbuf);
int lstat(const char *path, struct stat *statbuf);
```

- These functions return information about a file, in the buffer pointed to by `statbuf`.
- **`stat()`** and **`fstat()`** retrieve information about the file pointed to by `path`; but the latter accepts FD, only.
- if `path` is a symbolic link, then **`lstat()`** returns information about the link itself, not the file that the link refers to.

The `stat` structure

```
struct stat {
    ...
    ino_t     st_ino;         /* Inode number */
    mode_t    st_mode;        /* File type and mode */
    nlink_t   st_nlink;       /* Number of hard links */
    uid_t     st_uid;         /* User ID of owner */
    gid_t     st_gid;         /* Group ID of owner */
    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 */
    time_t    st_atime;   /* time of last access */
    time_t    st_mtime;   /* time of last modification */
    time_t    st_ctime;   /* time of last status change */
}
```

POSIX Macros to check the file type

```
struct stat info;

if (S_ISLNK(info.st_mode)){ /* symbolic link */ }
else if (S_ISREG(info.st_mode)){ /* regular file */ }
else if (S_ISDIR(info.st_mode)){ /* directory */ }
else if (S_ISCHR(info.st_mode)){ /* character device */
    }
else (S_ISBLK(info.st_mode)){ /* block device */ }
```

Example

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

void printattr(char * path) {
struct stat info;

if ( stat(path,&info)== -1) { perror("printattr"); }

printf("Type: \t\t\t");
if (S_ISREG(info.st_mode)) printf("regular");
else if (S_ISDIR(info.st_mode)) printf("directory");
else if (S_ISLNK(info.st_mode)) printf("symbolic link");
else if (S_ISCHR(info.st_mode)) printf("character device");
else if (S_ISBLK(info.st_mode)) printf("block device");
else if (S_ISFIFO(info.st_mode)) printf("pipe");
else if (S_ISSOCK(info.st_mode)) printf("socket");
else printf("unknown");
```

Example (cont'd)

```
printf("\nFile Size: \t\t%ld bytes\n", info.st_size);
printf("i-node number \t\t %ld \n", (long)info.st_ino);
printf("Permissions \t\t");
/* user */
if (S_IRUSR & info.st_mode) putchar('r'); else putchar('-');
if (S_IWUSR & info.st_mode) putchar('w'); else putchar('-');
if (S_IXUSR & info.st_mode) putchar('x'); else putchar('-');
/* group */
if (S_IRGRP & info.st_mode) putchar('r'); else putchar('-');
if (S_IWGRP & info.st_mode) putchar('w'); else putchar('-');
if (S_IXGRP & info.st_mode) putchar('x'); else putchar('-');
/* others */
if (S_IROTH & info.st_mode) putchar('r'); else putchar('-');
if (S_IWOTH & info.st_mode) putchar('w'); else putchar('-');
if (S_IXOTH & info.st_mode) putchar('x'); else putchar('-');
printf("\nlast access: \t\t%s",ctime(&info.st_atime));
printf("last modification: \t%s",ctime(&info.st_mtime));
printf("last status change: \t%s",ctime(&info.st_ctime));
printf("uid \t\t\t%d\n", info.st_uid);
printf("gid \t\t\t%d\n", info.st_gid);
}
```

Example (cont'd)

```
printf("\nFile Size: \t\t%ld bytes\n", info.st_size);
printf("i-node number \t\t %ld \n", (long)info.st_ino);
printf("Permissions \t\t");
/* user */
if (S_IRUSR & info.st_mode) putchar('r'); else putchar('-');
if (S_IWUSR & info.st_mode) putchar('w'); else putchar('-');
if (S_IXUSR & info.st_mode) putchar('x'); else putchar('-');
/* group */
if (S_IRGRP & info.st_mode) putchar('r'); else putchar('-');
if (S_IWGRP & info.st_mode) putchar('w'); else putchar('-');
if (S_IXGRP & info.st_mode) putchar('x'); else putchar('-');
/* others */
if (S_IROTH & info.st_mode) putchar('r'); else putchar('-');
if (S_IWOTH & info.st_mode) putchar('w'); else putchar('-');
if (S_IXOTH & info.st_mode) putchar('x'); else putchar('-');
printf("\nlast access: \t\t%s",ctime(&info.st_atime));
printf("last modification: \t%s",ctime(&info.st_mtime));
printf("last status change: \t%s",ctime(&info.st_ctime));
printf("uid \t\t\t%d\n", info.st_uid);
printf("gid \t\t\t%d\n", info.st_gid);
}
```

Example (cont'd)

```
int main(int argc, char **argv)
{
    if(argc != 2)    {
       fprintf(stderr, "Usage: %s <filename>\n", argv[0]);
       return 1;
     }
    char *pathname = argv[1];
    printf("Information for %s:\n",pathname);
    printattr(pathname);
    return 0;
}
```

Information for foo.txt

```
Type:                regular
File Size:           1938 bytes
i-node number        790456
Permissions          rw-r--r--
last access:         Fri Nov 13 12:38:38 2020
last modification:   Fri Nov 13 12:38:35 2020
last status change:  Fri Nov 13 12:38:35 2020
uid                  1001
gid                  1002
```

Open directory

- **`opendir()`, `fopendir()`**
- Syntax

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *name);
DIR *fdopendir(int fd);
```

- **`opendir()`** opens a directory stream corresponding to the directory `name`, and returns a pointer to the directory stream
- **`fopendir()`** works on file descriptor `fd` instead.

Open directory

- **`opendir()`, `fopendir()`**
- Syntax

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *name);
DIR *fdopendir(int fd);
```

Return value

- On success: a pointer to the directory stream.
- On error, NULL is returned, and `errno` is set appropriately.

Close directory

- **`closedir()`**
- Syntax

```
#include <sys/types.h>
#include <dirent.h>

int closedir(DIR *dirp);
```

- The **`closedir()`** function closes the directory stream associated with the directory pointer `dirp`.

Return value

- On success: **0**.
- On error, **-1**, and `errno` is set appropriately.

opendir() - closedir() Skeleton

```
DIR * d;
/* example for opening directory */
if ((d = opendir(".")) == NULL){
perror("opening cwd");
exit(EXIT_FAILURE);
}
/* insert your code here */

/* close directory */
if (( closedir(d) == -1) ){
perror("closing cwd");
exit(EXIT_FAILURE);
}
```

Read a directory

- **`readdir()`**
- Syntax

```
#include <dirent.h>

struct dirent *readdir(DIR *dirp);
```

- **`readdir()`** returns a pointer to a `dirent` structure representing the **the next** directory entry in the directory stream pointed to by `dirp`, **only**.
- **returns** `NULL` on reaching the end of the directory stream or if an error occurred.

`readdir()` Skeleton for reading files and directories

```c
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <string.h>
#include <errno.h>
#include <dirent.h>  //readdir
int main() {
  DIR * d;
  struct dirent* file;
  if ((d = opendir(".")) == NULL){ perror("opening cwd");
    exit(EXIT_FAILURE); }

  while ( (errno = 0, file = readdir(d))!=NULL) {
    printf("%s \n",file->d_name);
  }
  if (errno != 0) { /* handle error */ }
  if (( closedir(d) == -1) ){ perror("closing cwd");
  exit(EXIT_FAILURE);}
  exit(EXIT_SUCCESS);
}
```

Get current working directory

- **`getcwd()`**
- Syntax

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

- returns a null-terminated **string containing an absolute pathname** that is the current working directory of the calling process via the argument `buf`.
- Return value
  - On Success: pointer to a string containing the pathname
  - On Error: `NULL`, and `errno` is set to indicate the error.

Note

When the buffer is not long enough, **`getcwd()`** returns `NULL` with error `ERANGE`.

Change directory

```c
#include <unistd.h>
int chdir(const char* path)  /* path new cwd*/
int fchdir(int fd) /* file descriptor new cwd*/

/*return (0) success, (-1) error (set errno)*/
```

A scholary Example

```c
#include <stdio.h>
#include <unistd.h>
#include <limits.h> //path_max

int main(int argc, char **argv)
{
    char buf[PATH_MAX];
    if(argc != 2)    {
        fprintf(stderr, "Usage: %s <directory>\n", argv[0]);
        return 1;
     }
     char *dir = argv[1];
     if (chdir(dir) == -1) { /* change cwd to path */
         perror("chdir");
         return 1;
     }
     getcwd(buf, PATH_MAX);
     printf("We are in: %s\n", buf); /* print cwd as obtained
     from getcwd() */
     return 0;
}
```

A scholary example (cont'd)

shell

```
$ ./mydir Desktop
We are in:  /home/usr1/Desktop
$ pwd
/home/usr1
```

Apparently we are still in the same directory.

Note

The environment of one process cannot be changed by another process. That includes the current working directory.

Quiz

Choose the correct statement.

1. C programs can directly make system calls
2. Library functions use system call
3. Both (1) and (2)
4. Library functions don't use system calls.