System Calls and I/O Appendix

More System Calls

Directory and File System Management	
`s = mkdir(name, mode)`	Create a new directory
`s = rmdir(name)`	Remove an empty directory
`s = link(name, name)`	Create a new entry, name, pointing to name
`s = unlink(name)`	Remove a directory entry
`s = mount(special, name, flag)`	Mount a file system
`s = umount(special)`	Unmount a file system
Miscellaneous	
`s = chdir(dirname)`	Change the working directory
`s = chmod(name, mode)`	Change a file's protection bits
`s = kill(pid, signal)`	Send a signal to a process
`seconds = time(&seconds)`	Get the elapsed time since January 1, 1970

Example (`open()`)

```
#include <fcntl.h>
#include <errno.h>


int main() {
    int fd;
    fd = open("foo.txt", O_RDONLY | O_CREAT);
        if (fd == -1){
            perror("Program");
            exit(1);
    }
    printf("%d\n", fd);
}
```

Argument: string
Output: the string, a colon, and a description of the error condition stored in **`errno`**

Example (`close()`)

```
#include <fcntl.h>
main(){
    int fd1;

    if(( fd1 = open("foo.txt", O_RDONLY)) < 0){
        perror("c1");
        exit(1);
    }
    if (close(fd1) < 0) {
        perror("c1");
        exit(1);
    }
    printf("closed the fd.\n");
```


Example (`close()`)

```
#include <fcntl.h>
main(){
    int fd1;

    if(( fd1 = open("foo.txt", O_RDONLY)) < 0){
        perror("c1");
        exit(1);
    }
    if (close(fd1) < 0) {
        perror("c1");
        exit(1);
    }
    printf("closed the fd.\n");
```

After close, can you still use the file descriptor?

Why do we need to close a file?

Example (**read()**)

```
#include <fcntl.h>
main() {
    char *c;
    int fd, sz;

    c = (char *) malloc(100
              * sizeof(char));
    fd = open("foo.txt",
              O_RDONLY);
    if (fd < 0) {
        perror("r1");
        exit(1);
    }
    sz = read(fd, c, 10);
    printf("called
        read(%d, c, 10).
        returned that %d
        bytes  were
        read.\n", fd, sz);
    c[sz] = '\0';

    printf("Those bytes
        are as follows:
        %s\n", c);
    close(fd);
}
```


Example (`write()`)

```
#include <fcntl.h>
main()
{
    int fd, sz;

    fd = open("out3",
        O_RDWR | O_CREAT |
        O_APPEND, 0644);
    if (fd < 0) {
        perror("r1");
        exit(1);
    }
```

```
sz = write(fd, "cs241\n",
    strlen("cs241\n"));

printf("called write(%d,
    \"cs241\\n\", %d).
    it returned %d\n",
    fd, strlen("cs241\n"),
    sz);

close(fd);
}
```


File: Statistics

```
#include <sys/stat.h>
int stat(const char* name, struct stat* buf);
```

- Get information about a file
- Returns:
  - 0 on success
  - -1 on error, sets **`errno`**
- Parameters:
  - **`name`**: Path to file you want to use
    - Absolute paths begin with "`/`", relative paths do not
  - **`buf`**: Statistics structure
    - **`off_t st_size`**: Size in bytes
    - **`mode_t st_mode:`** protection
    - **`time_t st_mtime`**: Date of last modification
- Also

```
int fstat(int filedes, struct stat *buf);
```


Useful Macros: File types

- Is file a symbolic link

  `S_ISLNK(st_mode)`
- Is file a regular file

  `S_ISREG(st_mode)`
- Is file a character device

  `S_ISCHR(st_mode)`
- Is file a block device

  `S_ISBLK(st_mode)`
- Is file a FIFO

  `S_ISFIFO(st_mode)`
- Is file a unix socket

  `S_ISSOCK(st_mode)`

Useful Macros: File Modes

- **`S_IRWXU(st_mode)`**
  - read, write, execute, owner
- **`S_IRUSR(st_mode)`**
  - read permission, owner
- **`S_IWUSR(st_mode)`**
  - write permission, owner
- **`S_IXUSR(st_mode)`**
  - execute, owner
- **`S_IRGRP(st_mode)`**
  - read permission, group
- **`S_IRWXO(st_mode)`**
  - read, write, execute, others

Example - (`stat()`)

```
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
int main(int argc, char **argv) {
    struct stat fileStat;
    if(argc != 2)
        return 1;
    if(stat(argv[1], &fileStat) < 0)
        return 1;
    printf("Information for %s\n",argv[1]);
    printf("---------------------------\n");
    printf("File Size: \t\t%d bytes\n", fileStat.st_size);
    printf("Number of hard links: \t%d\n", fileStat.st_nlink);
    printf("File inode number: \t\t%d\n", fileStat.st_ino);
```


Example - (stat())

```
    printf("File Permissions: \t");
    printf( (S_ISDIR(fileStat.st_mode)) ? "d" : "-");
    printf( (fileStat.st_mode & S_IRUSR) ? "r" : "-");
    printf( (fileStat.st_mode & S_IWUSR) ? "w" : "-");
    printf( (fileStat.st_mode & S_IXUSR) ? "x" : "-");
    printf( (fileStat.st_mode & S_IRGRP) ? "r" : "-");
    printf( (fileStat.st_mode & S_IWGRP) ? "w" : "-");
    printf( (fileStat.st_mode & S_IXGRP) ? "x" : "-");
    printf( (fileStat.st_mode & S_IROTH) ? "r" : "-");
    printf( (fileStat.st_mode & S_IWOTH) ? "w" : "-");
    printf( (fileStat.st_mode & S_IXOTH) ? "x" : "-");
    printf("\n\n"); printf("The file %s a symbolic link\n",
    (S_ISLNK(fileStat.st_mode)) ? "is" : "is not");
    return 0;
}
```


Example - (`stat()`)

```
Information for testfile.sh
---------------------------
File Size: 36 bytes
Number of hard links: 1
File inode number: 180055
File Permissions: -rwxr-xr-x

The file is not a symbolic link
```


File: Seek

```
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
```

- Explicitly set the file offset for the open file
- Return: Where the file pointer is
  - the new offset, in bytes, from the beginning of the file
  - -1 on error, sets **`errno`**, file pointer remains unchanged
- Parameters:
  - **`fd`**: file descriptor
  - **`offset`**: indicates relative or absolute location
  - **`whence`**: How you would like to use **`lseek`**
    - **`SEEK_SET`**, set file pointer to **`offset`** bytes from the beginning of the file
    - **`SEEK_CUR`**, set file pointer to **`offset`** bytes from current location
    - **`SEEK_END`**, set file pointer to **`offset`** bytes from the end of the file

File: Seek Examples

- Random access
  - Jump to any byte in a file
- Move to byte #16

```
newpos = lseek(fd, 16, SEEK_SET);
```

- Move forward 4 bytes

```
newpos = lseek(fd, 4, SEEK_CUR);
```

- Move to 8 bytes from the end

```
newpos = lseek(fd, -8, SEEK_END);
```


Example (`lseek()`)

```
c = (char *) malloc(100 *
    sizeof(char));
fd = open("foo.txt", O_RDONLY);
if (fd < 0) {
    perror("r1");
    exit(1);
}

sz = read(fd, c, 10);
printf("We have opened in1, and
    called read(%d, c, 10).\n",
    fd);
c[sz] = '\0';
printf("Those bytes are as
    follows: %s\n", c);
```

```
i = lseek(fd, 0, SEEK_CUR);
printf("lseek(%d, 0, SEEK_CUR)
    returns that the current
    offset is %d\n\n", fd, i);

printf("now, we seek to the
    beginning of the file and
    call read(%d, c, 10)\n",
    fd);
lseek(fd, 0, SEEK_SET);
sz = read(fd, c, 10);
c[sz] = '\0';
printf("The read returns the
    following bytes: %s\n", c);
…
```


Stream Processing - `fscanf()`

```
int scanf(const char *format, ... );
```

- Read from the standard input stream **`stdin`**
  - Stores read characters in buffer pointed to by *s*.
- Return
  - Number of successfully matched and assigned input items
  - **`EOF`** on error

```
int fscanf(FILE *stream, const char *fmt, ... );
```

  - Read from the named input **`stream`**

```
int sscanf(const char *s, const char *fmt, ... );
```

  - Read from the string **`s`**

Example: (`scanf()`)

- Input: `56789 56a72`

```
#include <stdio.h>
int main() {
    int i;
    float x;
    char name[50];
    scanf("%2d%f %[0123456789]", &i, &x, name);
}
```

What are **`i`**, **`x`**, and **`name`** after the call to **`scanf()`**?