CS 332/532 Systems Programming

Lecture 10

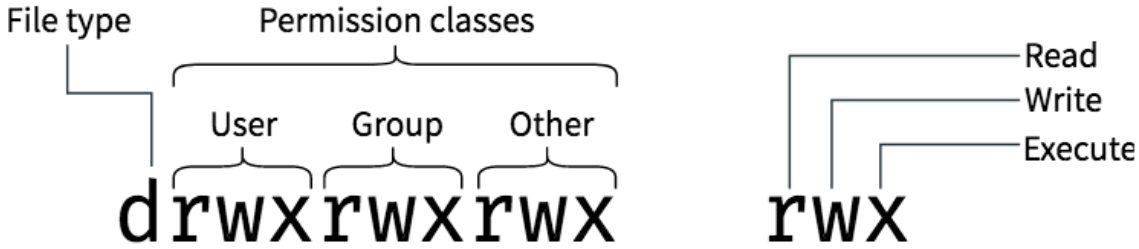
-UNIX Files / 2-

Agenda

- Open/Close/read/write a file
- lseek

File Permissions

- use `ls -l` command to display the permissions
  - Permission classes
    - user/owner(u), group (g), other (o)
  - Permission
    - read(r), write(w), execute(x), denied(-)
  - File type
    - **–** : regular file
    - **d** : directory
    - **c** : character device file
    - **b** : block device file
    - **s** : local socket file
    - **p** : named pipe
    - **l** : symbolic link




```
(common) Kai-MBA:test kzhao$ ls -l
total 0
drwxr-xr-x  2 kzhao  staff  64 Sep 14 09:47 dir
-rw-r--r--  1 kzhao  staff   0 Sep 14 09:47 text
(common) Kai-MBA:test kzhao$ chmod u-w,g+w,o+w text
(common) Kai-MBA:test kzhao$ ls -l
total 0
drwxr-xr-x  2 kzhao  staff  64 Sep 14 09:47 dir
-r--rw-rw-  1 kzhao  staff   0 Sep 14 09:47 text
```

Picture https://www.booleanworld.com/introduction-linux-file-permissions/

UNIX file I/O Functionalities

- Processes needs system calls to handle file operation
- Opening a file

```
int fd = open(path,flag)
              or
int fd = open(path,flag,mode)
```

https://man7.org/linux/man-pages/man2/open.2.html

flag

O_RDONLY

O_WRONLY

O_RDWR

O_EXEC

O_APPEND

O_CREAT

……

……

mode

- The third argument → *mode* specifies the permissions to use in case a new file is created.

The following symbolic constants are provided for *mode*:

S_IRWXU 00700 user (file owner) has read, write, and execute permission

S_IRUSR 00400 user has read permission

S_IWUSR 00200 user has write permission

S_IXUSR 00100 user has execute permission

S_IRWXG 00070 group has read, write, and execute permission

S_IRGRP 00040 group has read permission

S_IWGRP 00020 group has write permission

S_IXGRP 00010 group has execute permission

S_IRWXO 00007 others have read, write, and execute permission

S_IROTH 00004 others have read permission

S_IWOTH 00002 others have write permission

S_IXOTH 00001 others have execute permission

According to POSIX, the effect when other bits are set in *mode* is unspecified. On Linux, the following bits are also honored in *mode*:

S_ISUID 0004000 set-user-ID bit

S_ISGID 0002000 set-group-ID bit (see inode(7)).

S_ISVTX 0001000 sticky bit (see inode(7)).

https://man7.org/linux/man-pages/man2/open.2.htm

close()

- system call
- the file descriptor is returned to the pool of available descriptors

```
int close(int fd);
```

- `close()` returns zero on success. On error, -1 is returned, and errno is set appropriately.

read()

- **ssize_t read(int** *fd*, **void** **buf*, **size_t** *count*);

- **read**() attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.

- On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number.

https://man7.org/linux/man-pages/man2/read.2.html

write()

- **ssize_t write(int** *fd*, **const void** **buf*, **size_t** *count*);

- **write**() writes up to *count* bytes from the buffer starting at *buf* to the file referred to by the file descriptor *fd*.

- On success, the number of bytes written is returned. On error, -1 is returned, and *errno* is set to indicate the cause of the error.

https://www.man7.org/linux/man-pages/man2/write.2.html

Exercise

- C code to copy one file and copy the contents of that file to a new file

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

#define BUFFSIZE 4096

int main(int argc, char *argv[]) {
    int  readFileDescriptor, writeFileDescriptor;
    long int n;
    char buf[BUFFSIZE];
```

- Check if the correct numbers of the argument are given. There should be three arguments: name of the program, input file name, and output file name. If the number of arguments is not three then the program should print an error message and terminate. Also, input and output file names should not be the same.

```
12
13      if (argc != 3){
14          printf("Usage: %s <source_filename> <destination_filename>\n", argv[0]);
15          exit (-1);
16      }
17
```

- Use the *open* function in *read only mode* to read the input file.
- The open function takes the name of the file as the first argument and the open flag as the second argument.
- The open flag specifies if the file should be opened in read only mode (O_RDONLY), write only mode (O_WRONLY), or read-write mode (O_RDWR).
- There is an optional third argument that specifies the file permissions called *mode*, we will not use the optional third argument here. The third argument specifies the file permissions of the new file created for writing. Note that the UNIX file uses {*read, write, execute*} (*rwx)* permissions for the user, group, and everyone

```
18        readFileDescriptor = open(argv[1], O_RDONLY);
```

```
18        readFileDescriptor = open(argv[1], O_RDONLY);
```

- The function returns a file descriptor which is typically a non-negative integer.
- If there is an error opening the file, the function returns -1.
- Note that most programs have access to three standard file descriptors: standard input (stdin) - 0, standard output (stdout) - 1, and standard error (stderr) - 2.
- Instead of using the values 0, 1, and 2 we can also use the POSIX name STDIN_FILENO, STDOUT_FILENO, and STDERR_FILENO, respectively.

- Use the open function to create a new write for writing the output such that if the file does not exist it will create a new file and if a file with the given name exists it will overwrite the existing file.
- This is accomplished by ORing the different open flags: O_CREAT, O_WRONLY, and O_TRUNC. O_CREAT specifies to open a new empty file if the file does not exist and requires the third file permission argument to be provided.
- O_WRONLY specifies that the file should be open for writing only and the O_TRUNC flag specifies that if the file already exists, then it must be truncated to zero length (i.e., destroy any previous data).

```
19      writeFileDescriptor = open(argv[2], O_CREAT|O_WRONLY|O_TRUNC, 0700);
```

- Check the file descriptor to see if there is a problem or not

```
18      readFileDescriptor = open(argv[1], O_RDONLY);
19      writeFileDescriptor = open(argv[2], O_CREAT|O_WRONLY|O_TRUNC, 0700);
20
21      if (readFileDescriptor == -1 || writeFileDescriptor == -1){
22          printf("Error with file open\n");
23          exit (-1);
24      }
```

- Now read the file by reading fixed chunks of data using the *read* function by providing the file descriptor, input buffer address, and the maximum size of the buffer provided .
- A successful read returns the number of bytes read, 0 if the end-of-file is reached, or -1 if there is an error.
- After you read the data in to the buffer write the buffer to the new file using the *write* function.
- The *write* function takes the file descriptor, buffer to write, and the number of bytes to write. If the write is successful, it will return the number of bytes actually written.

```
26      while ((n = read(readFileDescriptor, buf, BUFFSIZE)) > 0){
27          if (write(writeFileDescriptor, buf, n) != n){
28              printf("Error writing to output file\n");
29              exit (-1);
30          }
31      }
```

- check the error condition

```
32      if (n < 0){
33          printf("Error reading input file\n");
34          exit (-1);
35      }
36
```

- After completing the copy process use the *close* function to close both file descriptors. The close function takes the file descriptor as the argument and returns 0 on success and -1 in case of an error.

```
38      close(readFileDescriptor);
39      close(writeFileDescriptor);
40      return 0;
41  }
```

Run the example

gcc filecopy.c -o exercise1

./exercise1 smallTale.txt outputFile.txt

```
MacBook-Pro:Desktop mahmutunan$ gcc filecopy.c -o exercise1
MacBook-Pro:Desktop mahmutunan$ ./exercise1 smallTale.txt outputFile.txt
MacBook-Pro:Desktop mahmutunan$
```

lseek()

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

- 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:

https://man7.org/linux/man-pages/man2/lseek.2.html

- **SEEK_SET** The file offset is set to *offset* bytes.
- **SEEK_CUR** The file offset is set to its current location plus *offset* bytes.
- **SEEK_END** The file offset is set to the size of the file plus *offset* bytes.

Example 2

- Now, we will use *lseek* function to move to particular location in the file and modify it by performing following steps;

1. You can use the output file of Example #1.

Or, you can use any txt file.

I will create a new txt file and put some text

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

#define BUFFSIZE 4096
#define SEEKSIZE -10

int main(int argc, char *argv[]) {
    int  RWFileDescriptor;
    long int n;
    char buf[BUFFSIZE];
    const char lseekMSG[] = "THIS IS NEW MSG FROM LSEEK!\0";

    if (argc != 2){
        printf("Usage: %s <filename>\n", argv[0]);
        exit (-1);
    }

    RWFileDescriptor = open(argv[1], O_RDONLY);

    if (RWFileDescriptor == -1){
        printf("Error with file open\n");
        exit (-1);
    }

```

- 2. Use lseek to read last 10 bytes of file and print it on console.
  - The *lseek* function takes three arguments: file descriptor, the file offset, and the base address from which the offset is to be implemented (often referred to as *whence*).
  - In this example, we are trying to read the last 10 bytes in the file, so we set the whence to the end of the file using SEEK_END and specify the offset as 10.
  - You can look at the man page for lseek to find out other predefined whence values: SEEK_SET, SEEK_CUR, etc.
  - The lseek function returns the new file offset on a successful and returns -1 when there is an error.

```
28      if (lseek(RWFileDescriptor, SEEKSIZE, SEEK_END) >= 0){
29          if((n = read(RWFileDescriptor, buf, BUFFSIZE)) > 0){
30              if (write(STDOUT_FILENO, buf, n) != n) {
31                  printf("Error writing to file\n");
32                  exit (-1);
33              }
34          } else {
35              printf("Error reading file\n");
36              exit (-1);
37          }
38      } else {
39          printf("lseek error (Part 1)\n");
40          exit (-1);
41      }
42      close(RWFileDescriptor);
```

- 3. Use lseek to write a string character “THIS IS NEW MSG FROM LSEEK!” at the beginning of the file.

```
43
44      RWFileDescriptor = open(argv[1], O_WRONLY);
45      if (lseek(RWFileDescriptor, 0, SEEK_SET) >= 0){
46          if (write(RWFileDescriptor, lseekMSG, strlen(lseekMSG)) != strlen(lseekMSG)) {
47              printf("Error writing to file\n");
48          }
49      } else {
50          printf("lseek error (Part 2)\n");
51      }
52
53      close(RWFileDescriptor);
54
55      return 0;
56  }
57
```

Run the exercise 2

```
(common) Kai-MBA:lecture10 kzhao$ gcc filelseek.c -o exercise2
(common) Kai-MBA:lecture10 kzhao$ cat testfile.txt
This is a test message. Do you see it?
```

```
(common) Kai-MBA:lecture10 kzhao$ ./exercise2 testfile.txt
u see it?
```

```
(common) Kai-MBA:lecture10 kzhao$ cat testfile.txt
THIS IS NEW MSG FROM LSEEK!you see it?
```

Lab 4 - exercise

- Check out what values are printed if you change the *offset* and *whence* to the following values when you are using the *lseek* function with the *readFileDescriptor*:

offset	**whence**
0	SEEK_SET
0	SEEK_END
-1	SEEK_END
-10	SEEK_CUR

Exercise 3

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#define BUF_SIZE 1024

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <filename>\n", argv[0]);
        exit(-1);
    }
    char *file_Name = argv[1];

    int writeFd;
```

```
fprintf(stdout, "Opening file :%s\n", file_Name);
writeFd = open(file_Name, O_RDWR|O_NONBLOCK);

if (writeFd < 0) {
    printf("Error with file open\n");
    exit(-1);
}

fprintf(stdout, "Seeking the end of the file \n");
```

```
if (lseek(writeFd, 0, SEEK_END) >= 0) {
    fprintf(stdout, "Writing the message into %s\n", file_Name);
    char buffer[BUF_SIZE] = "Message from Exercise 3\n";
    write(writeFd, buffer, BUF_SIZE);
    close(writeFd);

    return 0;
}
```

output

```
(common) Kai-MBA:lecture10 kzhao$ gcc exercise3.c -o exercise3
```

```
(common) Kai-MBA:lecture10 kzhao$ cat testfile.txt
This is a test message. you see it?
(common) Kai-MBA:lecture10 kzhao$ ./exercise3 testfile.txt
Opening file :testfile.txt
Seeking the end of the file
Writing the message into testfile.txt
(common) Kai-MBA:lecture10 kzhao$ cat testfile.txt
This is a test message. you see it?
Message from Exercise 3
```