CS 332/532 Systems Programming

Lecture 10

-UNIX Files / 2-

Professor : Mahmut Unan – UAB CS

Agenda

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

Reminders

- **Class Attendance**
- Attendance is **mandatory** for both the lecture and lab. If you know you will be absent for a legitimate reason, let the instructor know. If you are sick bring a doctor's excuse or a written university excuse to resolve the absences. An absence has to be resolved as soon as possible - otherwise it will not be treated as an excused absence.

Communication Policy

- TA
  - 24 hours
- Instructor
  - 3 business days

Covid19 and Mask Requirement

- UAB is very concerned for your continued health and safety. Please consult the Students section of UAB United for up-to-date guidance, because the following information is subject to change as circumstances require.
- We strongly urge you to be fully vaccinated. Here is information on the safety of vaccines and on how to get vaccinated at UAB. There are also incentives for getting vaccinated.
- Mask-wearing has proven to be one of the most successful mitigation strategies used to combat spread of the various variants of the COVID-19 virus**. UAB requires face coverings indoors on campus- regardless of vaccine status.** Students who do not follow this requirement can be reported to Student Conduct.

Academic Integrity Code

- The University of Alabama at Birmingham expects all members of its academic community to function according to the highest ethical and professional standards. Students, faculty, and the administration of the institution must be involved to ensure this quality of academic conduct. Please review UAB's Academic Integrity Code located at
- https://www.uab.edu/one-stop/policies/academic-integrity-code
- Violations of this honor code will result in a variety of sanctions as provided in the Code. A minor offense may result in a reduced grade for the associated assignment or homework. A major offense may, at a minimum, result in the failure of the course. Repeat offenders may be expelled from UAB.
- It is important to note that
- 1) "All students are expected to be familiar with the Academic Integrity Code and abide by it. By their continued enrollment at the University, students reaffirm their pledge to adhere to the provisions of the Academic Integrity Code."
- 2) Unauthorized assistance from third parties including a commercial service such as chegg.com or engaging another person (whether paid or unpaid) constitutes an act of cheating and is interpreted as a major offense in the Academic Integrity Code, resulting in F in the course (see chart below). Further, students are prohibited from posting homework, assignments or examination questions to non-UAB web sites without explicit authorization by the instructor.
- 3) UAB requires both faculty and students uphold the standards for academic integrity. Students who witness academic misconduct have a duty to report it.

Suggested Penalties for Violations of Different Severity

(Page 21 of Academic Integrity Code)

1st offense minor	1st offense moderate 2nd offense minor	1st offense major 2nd offense moderate 3rd offense any	PROPOSED SANCTION
✓	✓	✓	Academic Integrity workshop
✓			Reduced grade on assignment
✓			Additional Course Work
✓			Opportunity to revise/repeat
✓	✓		Failure of Assignment
✓	✓		Reduced course grade
	✓	✓	F in Course
	✓	✓	Academic Probation
		✓	Academic Suspension
		✓	Academic Expulsion

Please also review UAB Student Code of Conduct at the linked below:

https://www.uab.edu/students/accountability/student-conduct-code

https://www.uab.edu/policies/content/Pages/UAB-UC-POL-0000781.html

CS332 AIC Policy

- Cite anything that is not belong to you
  - reference links, book name..etc
- We will grade your contribution

- All offenses related to the hw and exam will be considered as major offense

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

```
(base) mahmutunan@MacBook-Pro Desktop % gcc filelseek.c -o exercise2
(base) mahmutunan@MacBook-Pro Desktop % cat testfile.txt
THIS IS NEW MSG FROM LSEEK!s a test message. Do you see it?%
```

```
(base) mahmutunan@MacBook-Pro Desktop % ./exercise2 testfile.txt
```

```
(base) mahmutunan@MacBook-Pro Desktop % cat testfile.txt
THIS IS NEW MSG FROM LSEEK!s a test message. Do 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;
```

```
15
16      fprintf(stdout, "Opening file :%s\n", file_Name);
17      writeFd = open(file_Name, O_RDWR, O_NONBLOCK, O_APPEND);
18
19      if (writeFd < 0) {
20          printf("Error with file open\n");
21          exit(-1);
22      }
23
24      fprintf(stdout, "Seeking the beginning of the file \n");
```

```
25
26      if (lseek(writeFd, 0, SEEK_SET) >= 0) {
27          fprintf(stdout, "Writing the message into %s\n", file_Name);
28          char buffer[BUF_SIZE] = "Message from Exercise 3\n";
29          write(writeFd, buffer, BUF_SIZE);
30          close(writeFd);
31
32          return 0;
33      }
34  }
```

output

```
(base) mahmutunan@MacBook-Pro Desktop % gcc exercise3.c -o exercise3
(base) mahmutunan@MacBook-Pro Desktop % ./exercise3 output.txt
Opening file :output.txt
Seeking the beginning of the file
Writing the message into output.txt
(base) mahmutunan@MacBook-Pro Desktop %
```