Lab# 6
Input-Output System Calls in C

CS311L

Lab Engineer: Eisha ter raazia Mir

Learning Objectives

- To open and read a file in C using gcc compiler.
- To write in a file and close the file using gcc compiler.

System Calls

- System calls are the mechanism through which a process requests services from the operating system (OS).
- Services related to
  - file operations (e.g., open, close, read, write, stat) for managing files and directories
  - process management (e.g., fork, exec, exit) for managing processes

Open() System Call

- Used to provide access to a file/resource.
- Allocates resources to a file.

#include <fcntl.h>

int open(const char *pathname, int flags);

File Access modes

- O_RDONLY Open for read-only
- O_WRONLY Open for write-only
- O_RDWR Open for reading and writing

int fd = open("example.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);

Open() system call parameters

- The call may also include a combination (using a bitwise OR) of the following optional modes in the flag parameter:
  - O_APPEND: Place written data at the end of the file
  - O_TRUNC: Set the length of the file to zero, discarding existing contents
  - O_CREAT: Creates the file, if necessary, with permissions given in mode
  - O_EXCL: Used with O_CREAT, ensures that the caller creates the file. The open is atomic; it’s performed with just one function call. This protects against two programs creating the file at the same time. If the file already exists, open will fail.

Open() System Call

- To create a file named ***new_file***, not present yet for write only purpose

open (“new_file”, O_WRONLY|O_CREAT);

Close System Call

close(int fd);

- Used to terminate access to a file.
- Files no longer required by process
- Buffers are flushed.
- The file descriptor becomes available for reuse
- It returns 0 if successful and –1 on error

```c
// C program to illustrate
// open system call

#include<stdio.h>

#include<fcntl.h>

#include<errno.h>

int main()
{
    // if file does not have in directory
    // then file first.txt is created.

    int fd = open("foo.txt", O_RDONLY | O_CREAT, 0644);

    printf("fd = %d\n", fd);

    if (fd ==-1)
    {
        // print which type of error have in a code

        printf("Error Number % d\n", errno);

        // print program detail "Success or failure"

        perror("Program");
    }
      close(fd);

     return 0;
}
```

```
eisha@eisha-virtual-machine:~$ nano lab6
eisha@eisha-virtual-machine:~$ gcc lab6.c -o lab6
eisha@eisha-virtual-machine:~$ ./lab6
fd = 3
eisha@eisha-virtual-machine:~$
```

Write() System Call

#include <unistd.h>

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

- The write() system call is used to write data from a buffer to a file or file descriptor.
- It takes three arguments: the file descriptor to which to write, the buffer containing the data to be written, and the number of bytes to write.
- It returns the number of bytes written
- This may be less than count if there has been an error

```c
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>  // for write() and close()

int main()
{
    int sz;

    // Open file with write-only, create if not exists,
and truncate (clear old data)
    int fd = open("foo.txt", O_WRONLY | O_CREAT | O_TRUNC,
0644);

    if (fd < 0)
    {
        perror("r1");
        exit(1);
    }

    // Write data to the file
    sz = write(fd, "hello geeks\n", strlen("hello
geeks\n"));

    // Print number of bytes written
    printf("Size of the data written in file is: %d\n",
sz);

    // Close the file
    close(fd);

    return 0;}
```

```
eisha@eisha-virtual-machine:~$ gcc lab666.c -o lab666
eisha@eisha-virtual-machine:~$ ./lab666
Size of the data written in file is: 12
eisha@eisha-virtual-machine:~$ nano foo.txt
eisha@eisha-virtual-machine:~$
```

read() System Call

#include <unistd.h>

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

- The read() system call is used to read data from a file or file descriptor into a buffer.
- It takes three arguments: the file descriptor from which to read, the buffer where the data will be stored, and the number of bytes to read.
- If a read call returns 0, it had nothing to read; it reached the end of the file

```c
#include<stdio.h>
#include<string.h>
#include<unistd.h>
#include<fcntl.h>
int main(void){
int fd[2];
char buf1[12]="hello world";
char buf2[12];
fd[0]=open("foobar.txt",O_RDWR);
fd[1]=open("foobar.txt",O_RDWR);
if(fd[0]<0||fd[1]<0){perror("open");return 1;}
write(fd[0],buf1,strlen(buf1));
write(1,buf2,read(fd[1],buf2,12));
close(fd[0]);
close(fd[1]);
return 0;
}
```

```
eisha@eisha-virtual-machine:~$ touch foobar.txt
eisha@eisha-virtual-machine:~$ gcc lab62.c -o lab62
eisha@eisha-virtual-machine:~$ ./lab62
hello worldeisha@eisha-virtual-machine:~$
```

More IO Functions

- The read()/write() system calls are too low-level
- The C standard library provides relatively high-level functionality for doing file IO

C fprintf() function

- To write a file in C, the fprintf() function is used.
- To read a file in C fscanf() function is used.

C fseek() function

- fseek() is used to move the file pointer associated with a given file to a specific position
- **int fseek(FILE *pointer, long int offset, int position)**

  pointer: pointer to a FILE object that identifies the stream

  offset: number of bytes to offset from position

  position: position from where offset is added

  returns: zero if successful, or else it returns a non-zero value

C fseek() function

- Position defines the point with respect to which the file pointer needs to be moved. It has three values:
  SEEK_END : It denotes end of the file
  SEEK_SET : It denotes starting of the file
  SEEK_CUR : It denotes file pointer’s current position

```c
#include <stdio.h>

int main()
{
    FILE *fp;

    fp = fopen("test.txt", "r");

    fseek(fp, 12, SEEK_SET);

    // Printing position of pointer

    printf("%d\n", ftell(fp));

    return 0;
}
```

Tasks

1. Write a C program to take maximum of 20 characters from the keyboard as input and display it on screen using system call
2. Write a C program which reads from user entered file and write exactly the same data at the terminal both by system call and fprintf and fscanf functions
3. Write a C program which takes the average of all the numbers present in a file name ***Average*** and display it on terminal
4. Write a code in C which fixes run time error of memory outflow