OS 5th Lab Assignment: Implementation of System Calls

Name: Mane Sushant Subhash

Class: TY AI-B

Roll no.: 24

System calls are functions provided by the operating system that allow user-level processes to request services or resources, such as file I/O, process creation, and network communication. They serve as an interface between applications and the kernel, ensuring controlled access to system resources.

1. write(): used to write to a file descriptor. In other words write() can be used to write to any file (all hardware are also referred as file in Linux) in the system but rather than specifying the file name, you need to specify its file descriptor.

```
 GNU nano 6.2
#include<stdio.h>
#include<unistd.h>
int main()
{
int count;
count=write(1,"Hello\n",6);
printf("Total no. of bytes written: %d/n",count);}
```

```
sushant@UbuntuVM:~$ nano write.c
sushant@UbuntuVM:~$ gcc write.c
sushant@UbuntuVM:~$ ./a.out
Hello
Total no. of bytes written: 6/nsushant@UbuntuVM:~$
```

- The bytes to be printed (third parameter) are less than the data specified in 2nd parameter

```
  GNU nano 6.2
#include<unistd.h>
int main()
{
write(1,"Hello\n",3);}
```

```
sushant@UbuntuVM:~$ nano write.c
sushant@UbuntuVM:~$ gcc write.c
sushant@UbuntuVM:~$ ./a.out
Helsushant@UbuntuVM:~$ nano write.c
```

- The file descriptor is not one of the pre-specified ones i.e., 0, 1 or 2

```
  GNU nano 6.2                    write.c
#include<stdio.h>
#include<unistd.h>
int main()
{
int count;
count=write(3,"Hello\n",3);
printf("Total bytes written: %d/n",count);}
```

```
sushant@UbuntuVM:~$ nano write.c
sushant@UbuntuVM:~$ gcc write.c
sushant@UbuntuVM:~$ ./a.out
Total bytes written: -1/nsushant@UbuntuVM:~$
```

2. read():use of *read()* system call is to read from a file descriptor. The working is same as write(), the only difference is read() will read the data from file pointed to by file descriptor.

```
  GNU nano 6.2
//read.c
#include<unistd.h>
int main()
{
char buff[20];
read(0,buff,10);
write(1,buff,10);}
```

```
sushant@UbuntuVM:~$ nano read.c
sushant@UbuntuVM:~$ gcc read.c
sushant@UbuntuVM:~$ ./a.out
0123456789
0123456789sushant@UbuntuVM:~$
```

```
  GNU nano 6.2
//read.c
#include<unistd.h>
int main()
{
int nread;
char buff[20];
nread=read(0,buff,15);
write(1,buff,nread);}
```

```
sushant@UbuntuVM:~$ nano read.c
sushant@UbuntuVM:~$ gcc read.c
sushant@UbuntuVM:~$ ./a.out
Hello World
Hello World
```

3. open(): used to open files in an operating system. It takes a file pathname and options as parameters and returns a file descriptor, allowing processes to read from or write to the specified file.

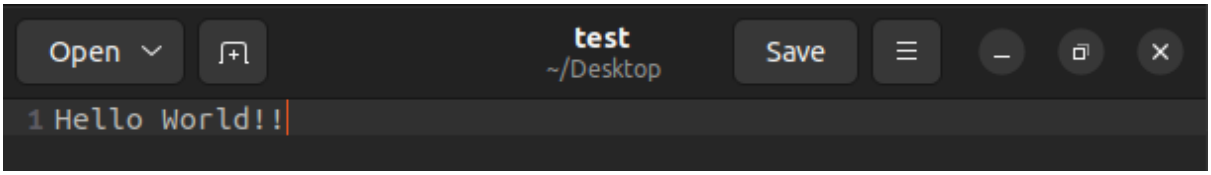
- Program using open() system call to read the first 10 characters of an existing file “test.txt” and print them on screen.

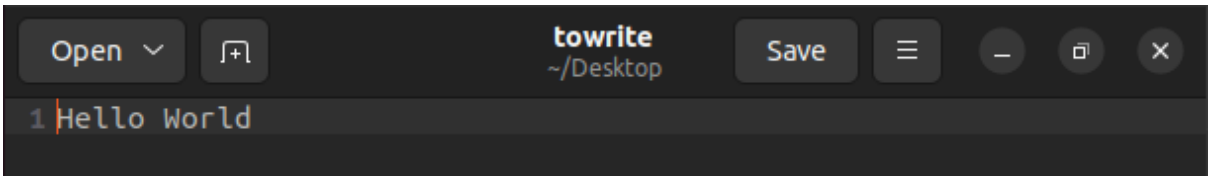
```
  GNU nano 6.2                          open.c
//open.c
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
int main()
{
int n,fd;
char buff[50];
fd=open("test",O_RDONLY);
printf("The file descripter of the file is: %d\n",fd);
n=read(fd,buff,11);
write(1,buff,n);}
```

```
sushant@UbuntuVM:~/Desktop$ cat test
Hello World!!
sushant@UbuntuVM:~/Desktop$ gcc open.c
sushant@UbuntuVM:~/Desktop$ ./a.out
The file descripter of the file is: 3
Hello Worldsushant@UbuntuVM:~/Desktop$ nano open.c
```

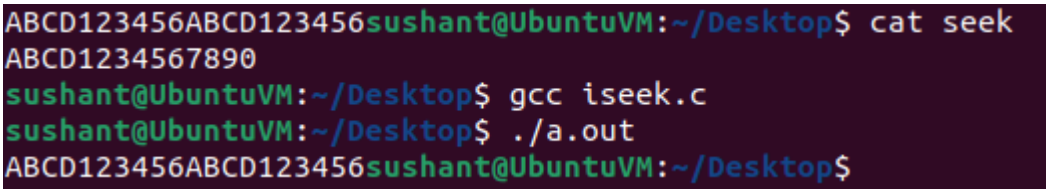
- Program to read 10 characters from file “test” and write them into non-existing file “towrite”

```
  GNU nano 6.2                          open.c
//open.c
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
int main()
{
int n,fd,fd1;
char buff[50];
fd=open("test",O_RDONLY);
n=read(fd,buff,11);
fd1=open("towrite",O_WRONLY|O_CREAT,0642);
write(fd1,buff,n);}
```





4. lseek(): used to change the current offset (position) within a file. It allows processes to move the file pointer to a specific byte location, enabling random access within the file for reading or writing operations.



5. dup(): used to duplicate an existing file descriptor in Unix-like operating systems. It creates a new file descriptor that refers to the same underlying resource as the original, allowing multiple references to the same file or resource within a process.

   - dup()

```
 GNU nano 6.2                    dup.c
//dup.c
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>
int main()
{
int old_fd, new_fd;
old_fd=open("test",O_RDWR);
printf("File descriptor is %d\n",old_fd);
new_fd=dup(old_fd);
printf("New file descriptor is %d\n",new_fd);}
```

```
sushant@UbuntuVM:~/Desktop$ nano dup.c
sushant@UbuntuVM:~/Desktop$ gcc dup.c
sushant@UbuntuVM:~/Desktop$ ./a.out
File descriptor is 3
New file descriptor is 4
```

   - dup2(): duplicates an existing file descriptor to a specified target file descriptor number, facilitating redirection or manipulation of input/output streams.

```
 GNU nano 6.2                    dup.c
//dup2.c
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>
int main()
{
int old_fd, new_fd;
old_fd=open("test",O_RDWR);
printf("File descriptor is %d\n",old_fd);
new_fd=dup2(old_fd,7);
printf("New file descriptor is %d\n",new_fd);}
```

```
sushant@UbuntuVM:~/Desktop$ nano dup.c
sushant@UbuntuVM:~/Desktop$ gcc dup.c
sushant@UbuntuVM:~/Desktop$ ./a.out
File descriptor is 3
New file descriptor is 7
```