Assignment Number:9

Title:

Implement an assignment using File Handling System Calls (Low level system calls like open, read, write, etc)

Theory:

argc and argv[]:

It is possible to pass some values from the command line to your C programs when they are executed. These values are called **command line arguments** and many times they are important for your program especially when you want to control your program from outside instead of hard coding those values inside the code.

The command line arguments are handled using main() function arguments where **argc** refers to the number of arguments passed, and **argv[]** is a pointer array which points to each argument passed to the program. Following is a simple example which checks if there is any argument supplied from the command line and take action accordingly

It should be noted that **argv[0]** holds the name of the program itself and **argv[1]** is a pointer to the first command line argument supplied, and *argv[n] is the last argument. If no arguments are supplied, argc will be one, and if you pass one argument then **argc** is set at 2.

For example, to open file "tmp.txt" in the current working directory for reading and writing:

System calls	Function
open	open an existing file or create a new file
read	Read data from a file
write	Write data to a file

fd = open("tmp.txt", O_RDWR);

To open "sample.txt" in the current working directory for appending or create it, if it does not exist, with read, write and execute permissions for owner only:

fd = open("tmp.txt", O_WRONLY|O_APPEND|O_CREAT, S_IRWXU);

A file may be opened or created outside the current working directory. In this case, an absolute path and relative path may prefix the file name. For example, to create a file in /tmp directory:

open("/tmp/tmp.txt", O_RDWR);

Header File

```
#include <unistd.h>
ssize_t read(int fildes, void *buf, size_t nbyte);
```

DESCRIPTION

read attempts to read nbyte bytes from the file associated with fildes into the buffer pointed to by buf. If nbyte is zero, read returns zero and has no other results.

write

```
#include <unistd.h>
ssize_t write(int fildes, const void *buf, size_t nbyte);
```

DESCRIPTION

write attempts to write nbyte bytes from the buffer pointed to by buf to the file associated with fildes. If nbyte is zero and the file is a regular file, write returns zero and has no other results. fildes is a file descriptor.

close

```
#include <unistd.h>
int close(int fildes);
```

DESCRIPTION

close closes the file descriptor indicated by fildes.

stat

```
#include <sys/types.h>
#include <sys/stat.h>
```

int stat(const char *path, struct stat *buf);

DESCRIPTION
stat obtains information about the named file. For more information, please read Section 2.8 in Interprocess Communications in UNIX:
The Nooks & Crannies.

Read from files

The system call for reading from a file is ***read***. Its syntax is

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

The first parameter *fd* is the file descriptor of the file you want to read from, it is normally returned from *open*. The second parameter *buf* is a pointer pointing the memory location where the input data should be stored. The last parameter *nbytes* specifies the maximum number of bytes you want to read. The system call returns the number of bytes it actually read, and normally this number is either smaller or equal to *nbytes*. The following segment of code reads up to 1024 bytes from file tmp.txt:

```
int actual_count = 0;
int fd = open("tmp.txt", O_RDONLY);
void *buf = (char*) malloc(1024);

actual_count = read(fd, buf, 1024);
```

Each file has a pointer, normally called read/write offset, indicating where next *read* will start from. This pointer is incremented by the number of bytes actually read by the *read* call. For the above example, if the offset was zero before the *read* and it actually read 1024 bytes, the offset will be 1024 when the *read* returns. This offset may be changed by the system call *lseek*, which will be covered shortly.

Write to files

The system call ***write*** is to write data to a file. Its syntax is

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

It writes *nbytes* of data to the file referenced by file descriptor *fd* from the buffer pointed by *buf*. The *write* starts at the position pointed by the offset of the file. Upon returning from *write*, the offset is advanced by the number of bytes which were successfully written. The function returns the number of bytes that were actually written, or it returns the value -1 if failed.

Close files

The ***close*** system call closes a file. Its syntax is

int close(int fd);

It returns the value 0 if successful; otherwise the value -1 is returned.

Files to be included for file-related system calls.

```
#include    <unistd.h>
#include    <fcntl.h>
#include    <sys/types.h>
#include    <sys/uio.h>
#include    <sys/stat.h>
```

Conclusion: We conclude from this practical assignment that we can handle the file by using the system calls of the operating system.