## University of Mauritius

## Faculty of Engineering

## Department of Computer Science and Engineering

1. **BSc Computer Science**

### Operating Systems (CSE 2022Y)

**Labsheet 4 (Two Weeks)**

This labsheet introduces you to the use of Linux ***System calls***. A system call is a function that resides within the Operating System kernel, that is used either to obtain information about data managed by the kernel or to request the OS to do some work. In the first part of this first labsheet on *system calls*, you will get access to some attributes of the process and also to obtain information about time. In the second part, you will use system calls for performing files I/O.

The prototype for each system call is usually defined in a header file. To use a system call, you need to include the appropriate header file in your program and you also need to know what the function header is like so that you can call the function, i.e you need to know the return type as well as the type and purpose of each parameter. These information are available in the built-in manual **man**. To see the online manual for a *system call* **funct1**, for example, You simply type ***man funct1.*** Since with Ubuntu Linux, man does not automatically include all system calls and commands available, you can use the following address to obtain these information: <http://linux.about.com/od/commands/l/blcmdl_2a.htm>.

**Part 1 – Introduction to System calls**

**Question 1**

The *system calls* **getpid()** and **gettppid()** respectively return the identifier of a process and the identifier of its parent. The synopsis is as given below. The type pid\_t can be treated as integers. Write a program to display the pid value of a process as well as that of its parent. Use the **ps –a** command to check if the **pid** displayed by the process is the same as that displayed by the **ps –a** command. You can use the **sleep()** system call to cause the process to sleep for some time to allow you to use the **ps -a** command.

Note: Use two terminals: One to run your program and another to type in the command **ps –a**.

## SYNOPSIS

**#include <**[**sys/types.h**](file:///\\usr\include\sys\types.h)**>**   
 **#include <[unistd.h](file:///\\\\usr\\include\\unistd.h)>**

**pid\_t getpid(void);**   
 **pid\_t getppid(void);**

**Question 2**

Study the system call **gettimeofday()**. Write a program that contains the following:

* A function **Power**, defined by you to calculate Xn, where X is of type float and n is an integer. You can use iteration or recursion, but **do not** make use of the pre-defined function **pow()**.
* A **main()** function that calculates the value of a polynomial, such as

X100 +X 99+ X98, for some chosen value X.

* Use **gettimeofday()** to find out for how long the program runs. (Time should be displayed in number of seconds and milliseconds).

**Question 3**

Write a program that executes a loop causing it to sleep for 2 seconds (using the sleep() system call), wakes up and displays its pid as well as the total number of milliseconds elapsed since it started. It can run the loop any number of times you choose. Your output can be in the form:

**Program with pid <pid> has run for <number of milliseconds> milliseconds**.

**Note**: Make use of sleep() and gettimeofday() system calls.

**Part 2 – File I/O System calls**

In this part of the labsheet you will work with file I/O system calls. Make use of the following system calls.

* open
* close
* read
* write
* lseek

For the include statements, use the following:

**#include <sys/types.h>**

**#include <sys/stat.h>**

**#include <fcntl.h>**

To use a file you first need to open it through the **open()** system call. The prototypes for opening a file is:

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

**int open(const char \****pathname***, int** *flags***, mode\_t** *mode***);**

Note: The open call requires: The pathname of the file, the flags which indicate whether the file is opened for reading, writing, read & write etc., and the mode, which specifies the different types of access rights to the file.

To open a file for reading, use the first prototype. To open a file for writing, use the second one. Note that open() returns an integer. The returned value is called a file descriptor and is used for accessing the file.

In this labsheet, you have been requested to use open(), read(), write()…etc. **Do not use** fopen(), fread(), fwrite().

The **read()** and **write()** system callsuse the file descriptor returned by open(). They both use pointers to void types, which allow any type of data (including structs) to be cast.

The include file for read and write are: <unistd.h>

**#include <[unistd.h](file:///\\\\usr\\include\\unistd.h)>**

**ssize\_t read(int** *fd***, void \****buf***, size\_t** *count***);**

**ssize\_t write(int** *fd***, const void \****buf***, size\_t** *count***);**

You can take **ssize\_t** to be **int**.

An example for opening the file file1.dat for reading 20 bytes is:

void main(){

int fd;

char buf[25];

fd=open(“file1.dat”,O\_RDONLY);

n=read(fd,buf,20);

// code for using the data should be inserted here……..

…….

…….

close(fd);

}

When opening a file for writing, you want the file to be created if it doesn’t exist. For that you should use the flag O\_WRONLY, ored with O\_CREAT as follows:

fd=open(“file1.dat”,O\_WRONLY|O\_CREAT);

You should also give the access rights, through the mode:

Eg.: fd=open(“file1.dat”,O\_WRONLY|O\_CREAT,S\_IRWXU);

An example

void main(){

int fd;

char buf[]=”Hello”;

fd=open(“file1.dat”,O\_WRONLY|O\_CREAT,S\_IRWXU);

n=write(fd,buf,10);

…….

……

close(fd);

}

You are required to explore the uses of the system calls, to find out how to specify the size of data to use in read() and write() calls and to find out the values of n in each case. You will also need to find out how to read a file until we reach the end of the file.

**Question 4**

Using a text editor, create a file called “lab44.dat”. Write a few sentences in it. Write a program to read the text from the file and display on the screen.

**Question 5**

Write a program that opens a file called “lab45.dat” for writing. If the file does not exist, the program creates it. If it exists already, the program will simply overwrite it. Your program should then read data for 5 students from the terminal and write them into the file. Your data should consist of the names of students (20 characters), a student id (10 characters), date of birth (10 characters), gender (7 characters (male/female)) and marital status (10 characters(single/married)). Write the data in the file as individual fields.

**Question 6**

Write a program to open the file in question 5, read all the data and display them on the screen.

**Question 7**

Write a program to open the file in question 5 in Append mode and add information for 5 more students. Use the program in question 3 to read the information from the file and display on the screen.

**Question 8**

Modify the programs in questions 5 and 6, so that they use structs for storing students information. Each write or read should write/read one struct. Note: you have to make use of sizeof() to obtain the required number of bytes to write/read.

**Question 9**

Write a program that opens the above file, allows you to enter an integer value, (eg. 5) on the terminal, go to that particular student in the file and displays the information for that student. (Hint: use lseek).

**Question 10**

Write a program that creates a file called “input13.dat”. It then allows you to enter the names and address of a number of persons. For each person it allows you to input the name and address, it computes the length of the names and address and writes in the file “input13.dat”, the length of the name, the name, the length of the address and the address.

**Question 11**

Write a program that reads the file “input13.dat” (created in question 10), displays the name, the size of the name, the address and the size of the address for the person with the longest name as well as for the person with the longest address. The program should also display the average size of names and average size of addresses in the file.

O. Moonian