**CSE231 - Operating Systems**

**Assignment 1**

**Exercise 1.1**

**Name**: Ansh Arora

**Roll Number**: 2019022

**Section**: B

**Branch**: CSE

**Task: Process Creation and Termination using system calls for calculating averages of students from given class marks in a CSV file.**

To run the program, run the following make command in the terminal –

make run

To clear out the files generated and reset the program for a new run, run the following command in the terminal –

make clear

Text

Description automatically generated

*Makefile*

On running the command, the following output is made to generate –

Text

Description automatically generated

*Section A compiled by Child process*

A close up of text on a black background

Description automatically generated

*Section B compiled by Parent process*

**How the program works –**

The program works by first creating a child process to perform processing on the Section A records. This is done by using the fork() function. The pid of the fork() function is passed into the pid\_id variable. The fork() function returns an integer 0 value if the id belongs to a child process, which is what we observe here. The if condition allots the work for the child process to pid\_id 0.

In the child process, the open() function is used to open the csv file in Read Only mode. A character array buf is created to store the values that will be read from the csv file. The read() command is then used to read in the elements from the csv file into the character array. I then use a dataset array of character arrays, which will now be used to store individual entries of the database. These entries are seperated by performing the strtok() operation on the buf array, seperating out the elements by the ‘\n’ character. We have now generated the 400 entry dataset in the form of an array of character arrays. I now perform strtok operation on all 400 elements of the array, seperating them by the ‘,’ delimiter. The elements generated – student id, section and marks in the 4 assignments are stored in individual variables. The check for section is performed. If the section of the students is “A”, the average marks operation is performed and that too are stored in specific variables. This data is then finally printed out onto the screen using the write syscall, in which the parameter 1 is passed as the file descriptor for printing out to the stdout. After the end of the child process, it is terminated by using the exit(0) command, indicating a successful run.

The parent process waits for the child process’s termination by the use of the wait() command. After the termination of the child process, it begins its processing on the section B in the exact same manner as section A, this time just processing and printing out the details of the students of section A.

It again reads and prints out the necessary details by using the earlier defined system calls.

**System Call commands used and Error Handling-**

1. **fork()** – The fork system call is used to create a child process. It runs along side the process that makes for its call, which is known as the parent process. It takes no parameters and returns an integer, which gives the status of the process generation –

* Negative Value – Unsuccesful operation
* Zero – Value given to child process
* Positive Value – Pid of child process

**Error Handling** –

In the program, the program checks whether the pid\_id generated is negative, in which case it prints and error message and exits from the process.

1. **wait()** – This system call is used to halt the parent process until the child process succesfully exits. After the termination of the child process, the parent is able to continue with its execution. It returns the value -1 if there are no child processes.

**Error handling** –

The parent process only runs after the child process has succesfully exited, thus taking care of anomalous behaviour.

1. **exit()** – This system call terminates the process which it is running, disregarding the code that follows the command call. It is used to exit from the child process and signal the parent’s wait command that it can resume its process. It takes two values in my program, exit(0) which refers to the succesful termination of the code, and exit(1), which refers to the unsuccessful termination of the code caused by an error.

**Error handling** –

This command is used to depict errors as given above and is called whenever we run into one cause by other syscalls.

1. **open()** – This sytem call is used to open a file for reading, writing or doing both at the same time. The mode can be specified using one of the flag specifiers. The path is also passed for the file to be opened, on which operations will be performed afterwards. It returns an integer value, aka the file descriptor which can henceforth be used to reference the open file.

**Error handling -**

The open command returns a -1 value if it runs unsuccessfully due to any reason. This has been handled in the code by providing an if condition and an exit(1).

1. **read()** – This system call is used to read given amount of bytes from the opened file into the user-defined array. It takes in the file descriptor value, array to read into and length of array to read as parameters. It returns -1 on and unsuccessful run and the number of bytes read on successful termination.

**Error handling** –

I have used the value returned to check if there is in fact an error generated during the syscall, on account of which I show an error and call an exit(1) command.

1. **write()** – This system call is used to write given const character arrays onto a given file/stdout. It takes in the file descriptor value, array to write out and its size as its parameters. The file descriptor value can be passed as 1(as I have done so in my case) to print out the values on stdout. It returns number of bytes written as return value, and a -1 in case of an unsuccessful run.

**Error Handling** -

I have used the value returned and stored in werror variable to check if there is in fact an error generated during the syscall, on account of which I show an error and call an exit(1) command.

1. **close()** – This system call is used to tell an OS that we have used a particular file descriptor and hence close the file that we had opened using open(). It takes in the file descriptor as its only parameter, and returns 0 on success and -1 on failure.

**Error Handling** –

In case of a -1 return value observed, I display an error message and call an exit(1) command.