Dhairya Chaudhary
2019035

Operating Systems - Assignment 1

TASK 1

SUMMARY

In this assignment, we have used various system calls to parse a CSV file containing dummy grades for the students of 2 sections. We used the fork system call to create a child process that computes the average for section A students, the parent process waits for the child process to exit and then does this for section B students. I have used the open, close, read and write system calls in my code and stored the marks of the section A students in a text file named sectionA.txt and for section B students in a text file named SectionB.txt. I have also printed these on the terminal (using write).

EXPLANATION OF CODE

In the main method, I have made the fork() system call to spawn a child process, used the open() call to open “file.csv” i.e the file we were given to parse, initialized a string called thisChar (used to traverse through the file character by character) and another called header that contains everything up to the first newline character (since the first line just mentions the headings).

I have used the waitpid() system call in the parent to ensure that the child executes first. The child exits with the exit() system call.

Both the parent and the child contain similar code, just that the child executes for section A students end and the parent for section B. The child opens sectionA.txt and the parent opens sectionB.txt using the open() call. Both store student ID, section, and all 4 marks in strings. In every case thisChar is used to read one character after the other using the read() system call till a “,” , “\n”, or EOF (found by checking the size of the element that was read) is encountered- this is pretty much the utility of the readattribute() function. We use the atoi() function to calculate the average of the marks in the 4 assignments. This is then again converted to a string using the tostring() function. The student ID+” “ and average+”/n” is written to the corresponding text file using the write system call.

The child process exits (using exit() system call) its loop when an element from section A is encountered and the parent ends execution when EOF is encountered. Both print

the data stored in their respective text files using the write() system call. All files are closed using the close() system call.

SYSTEM CALLS USED

I have used the following system calls:

1. fork()
   It is used to spawn a child process that runs concurrently with the parent process (the one that makes the call). The fork() syscall takes no parameters and returns an integer value. This value can be negative (denotes an error), zero (denotes the child process), or positive (denotes the parent process).

2. exit()
   This system call is used to terminate a process. We exit our child process after execution with value 0. In case of an error we use exit(1).

   exit() only accepts arguments from 0-255, otherwise the exit code will be the modulo 256 of entered value.

3. waitpid()
   waitpid() suspends the execution of the current process until a specified child has changed state. It's behaviour can be modified using the options argument.

   waitpid() accepts a pid as arguments, I have however used waitpid(-1) which means the parents will wait for any one of its children. This works in this case since we have made only one child.

4. open()
   Used to open a file for reading or writing or both. It accepts a string as path, flags, and the mode of operation as parameters.

   It makes a new entry in the file entry table and returns it. If this value is -1, this means that the file wasn't able to be opened.

5. read()
   Used to read from a file that has been successfully opened and is currently present in the file entry table. Accepts file descriptor, buffer to read data from and its length as parameters.

It returns the number of bytes read (0 in case EOF is reached). It returns -1 in case of an error or signal interrupt.

6. write()
   Used to write to a file that has been successfully opened and is currently present in the file entry table. Accepts file descriptor, the string to be added and its length as parameters.

   It returns the number of bytes written (0 in case EOF is reached). It returns -1 in case of an error or signal interrupt.

7. close()
   It closes the file by destroying the file table entry reference and setting fd to null. It takes the file descriptor as parameter.

   It returns 0 on success and -1 if an error is encountered.

ERROR HANDLING/CORNER CASES

I have handled the possibility of errors while using fork(), open(), and close() system calls in my code (based on the value they return). The code is designed to cover all cases, it can handle any integer value of student ID and marks. The last line is also covered by detecting the EOF (read() call returns 0).