|  |
| --- |
| Assignment I |
| Total Marks: 70 |
| Write appropriate programs and/or answer to solve the following problems.   * Submit all programs and question answers in a word file * For program submit screenshots of output * Word file name must be CSE325\_<section-number>\_A1\_<your-student-id> * Submission Deadline: 02 April 2021 5:05 PM |

1. a. Suppose a program infrequently invokes I/O requests. Whenever an urgent I/O request is invoked by the program then it starts to handle I/O requests frequently for moderate span of time. After executing specified number of I/O requests it starts executing large cpu bursts. While executing large cpu burst, program keeps the requested I/O calls in a queue. After finishing cpu burst it takes requests from the queue and invokes the queued I/O requests.

Now propose a processor- device communication model of your own by using the idea of two approaches such as interrupt and poling to make an efficient I/O communication for the above mentioned process. If necessary then draw a diagram that depicts the scenario and provides the work flow of your proposed model.

b. How does operating system involve in different activities to handle your proposed model. Describe the answer in your own way.

1. Write a program “**MergeFile.c**” to read all txt files in a specified directory that begins with that same name and merge them all into a new txt file. You must have to returns a file descriptor of the new file.

We all know about exec system call. To understand the functionality of exec system call, write another program **Exec.c.**  The program will be replaced by the image of “**MergeFile.c**” program. The specified name of files which you have to search in the directory need to be passed to “**MergeFile.c**” file from “**Exec.c**” file. Now do the necessary actions to implement the above mentioned scenario.

1. Let us consider your have a directory and it contains many sub-directories and files. First you have to find duplicate files name inside the directory and if found then remove the duplicate files. Write a program that will remove the duplicate files and categorize all files in the specified folder based on their file type. That is all .txt file in txt folder, all .jpg files in image folder etc. The argument to the program is a directory name.
2. a. Write a program that will create a child process that will execute a specified command the will display all the files name that contains a specified word. If child process can’t execute the command then print an error message. While child is executing, parent will wait for it, after finishing child task parent will print the child id with its returned status.

b. Now change the code of the above program in such a way that the child process will become a zombie process. This zombie process must remain in the system for at least 10 seconds. Show the zombie processes from CLI.

* 1. Why zombie process is not welcomed? What action you need to take if zombie processes are already created.
  2. Write the differences between exit() and return statement
  3. Write the differences between zombie and orphan process.
  4. Write similar program like problem no 4(b) that creates five zombie process.

1. Consider the following Code:

#include <unistd.h>

#include <sys/wait.h>

#define CMD\_LEN 120

int main(int argc, char \*\*argv) {

char cmd[CMD\_LEN];

char \*cmd\_args[2];

int n;

int child\_pid;

while (1) {

write(1, "tanni% ", 7);

n = read(0, cmd, CMD\_LEN);

if (n == 0) break; /\* EOF reached; exit program \*/

cmd[n - 1] = '\0'; /\* replace '\n' with '\0' \*/

child\_pid = fork();

if (child\_pid == 0) {

cmd\_args[0] = cmd;

cmd\_args[1] = NULL;

execvp(cmd, cmd\_args);

write(1, "Command not found\n", 19);

exit(-1);

}

waitpid(child\_pid, &n, 0);

}

return 0;

}

Now compile the above code and if you find any error then fix it. Explain the code line by line. Explain your understanding about the code in your own words.

1. Write a function foo (int fd, char\* buf, int b\_size, int n, int skip) that reads to buf from file with file descriptor fd, n blocks of size b\_size each. The last argument specifies how many bytes to skip after reading each block. Return -1 if the operation is unsuccessful. Else return total number of bytes read.