National University of Computer and Emerging Sciences

Lahore Campus

036

Operating Systems (CS2006)

Date: September 24th 2024

Course Instructor(s)

Mr. Razi Uddin
Mr. Mubashar Hussain
Ms. Rubab Anam
Ms. Namra Absar

Sessional-I

Total Time (Hrs): 1
Total Marks: 40
Total Questions: 3

22L-7971 Roll No

BSE-5B Section

Student Signature

Do not write below this line

Attempt all the questions.

Follow the order of the questions as given. You must show all relevant work, reasoning, and steps as specified in the question statements to receive full credit.

[CLO-2] Implement solutions employing concepts of Processes and Threads

Q1: **[Marks:20]**

You are tasked with writing a C program that demonstrates inter-process communication using pipes. The goal is to create a pipeline between two child processes. The parent should create two child processes. The parent process does not participate in the data transfer between the two children but is responsible for setting up the pipe and properly managing file descriptors. Child1 should execute **Cmd1** and child 2 should execute **Cmd2.**

1. **Cmd1:** `char *cmd1[] = { "/bin/ls", "-a", NULL };`
   // **ls -a** command in Linux lists all the files in the root directory including the hidden files (with dot extension).

2. **Cmd2:** `char *cmd2[] = { "/usr/bin/sort", "-r", NULL };`
   // The **sort -r** command in Linux sorts the lines of text in reverse order.

- **Child 1** redirects its standard output to the writing end of the pipe and executes cmd1 (e.g., **ls**).
- **Child 2** redirects its standard input from the reading end of the pipe and executes cmd2 (e.g., **sort**).
- This means that the **ls** command writes its output to the pipe, and the **sort** command reads its input from the pipe. The **sort** command then sorts the output of the **ls** command in reverse order.
- The parent process closes all unnecessary file descriptors to ensure it does not interfere with the pipe communication.

Requirements:

Write C code with comments explaining the key steps. Provide a step-by-step explanation of how file descriptors are manipulated in both the parent and child processes.

```
//Write your code
char *cmd1[] = { "/bin/ls", "-a", NULL };
char *cmd2[] = { "/usr/bin/sort", "-r", NULL };
int main()
{
//code



exit(0);
}
```

[CLO-2] Implement solutions employing concepts of Processes and Threads

Q2: **[Marks:10]**

Assuming fork() never fails, draw the process tree of the following code, also write how many times "I love FAST", "I love OS", "Breaking", "Exiting" will be printed on screen. (Just write the count).

2 1 4 6

```
int main() {

    while (fork() && fork()) {
        if (fork() || fork()) {
            printf("I love FAST\n");
        } else {
            if (!fork()) {
                printf("I love OS\n");
            }
        }
        printf("Breaking\n");
        break;
    }
    printf("Exiting\n");
    wait(NULL);
    return 0;
}
```

[CLO-2] Implement solutions employing concepts of Processes and Threads

Q3: [Marks: 5]

Consider the following sample code from a simple shell program.

```
int rc1 = fork();
if(rc1 == 0) {
   exec(cmd1);
}
else
{
   int rc2 = fork();
   if(rc2 == 0)
    {
    exec(cmd2);
    }
   else {
    wait();
    wait();
   }
}
```

a. In the code shown above, do the two commands cmd1 and cmd2 execute serially (one after the other) or in parallel (concurrent)?

b. Indicate how you would modify the code above to change the mode of execution from serial to parallel (concurrent) or vice versa. That is, if you answered "serial" in part (a), then you must change the code to execute the commands in parallel (concurrent), and vice versa. Indicate your changes next to the code snippet above.

[CLO-2] Implement solutions employing concepts of Processes and Threads

Q4: [Marks: 5]

Which of the following operations by a process will definitely cause the process to move from user mode to kernel mode? Answer yes (if a change in mode happens) or no. (Support your answer with a reason)

a. A process invokes a function in a userspace library.

b. A process invokes the kill system call to send a signal to another process.