Department of Computer Systems Engineering
University of Engineering and Technology
Peshawar, Pakistan

CSE-204 Operating Systems, Spring 2023
Assignment# 01 and 02

Total Marks: 20 **Submission Deadline: April 9th, 2023**

INSTRUCTIONS

1. Attempt **ALL** questions in a precise and to-the-point manner.
2. Extra details will not add any weight to the attained marks.
3. Plagiarism is strongly discouraged.
4. You can take help from available resources but do not just copy them and also do not forget to refer to the resources used.

ASSIGNMENT 01 (CHAPTERS 01 AND 02)

Question: (Topic: OS types) **[CLO-1]**

1. Differentiate between multiprogramming systems and multiprocessing systems.
2. Differentiate between a multicore, multi-kernel, and multiple processor system. Discuss in terms of efficiency.

Question: (Topic: OS modes) **[CLO-1]**

1. Some computer systems do not provide a privileged mode of operation in hardware. Is it possible to construct a secure operating system for these computer systems? Give arguments both that it is and that it is not possible.
2. What is the purpose of system calls, and how do system calls relate to the OS and to the concept of dual-mode (kernel-mode and user-mode) operation?
3. To a programmer, a system call looks just like a function call. Explain the difference in the underlying implementation. How does OS pass arguments to system calls?
4. Discuss which of the following instructions should be privileged and why?
    I. Set the value of the timer.
    II. Read the clock.
    III. Clear memory.
    IV. Issue a trap instruction.
    V. Turn off interrupts.
    VI. Modify entries in the device status table.

VII. Switch from user to kernel mode.

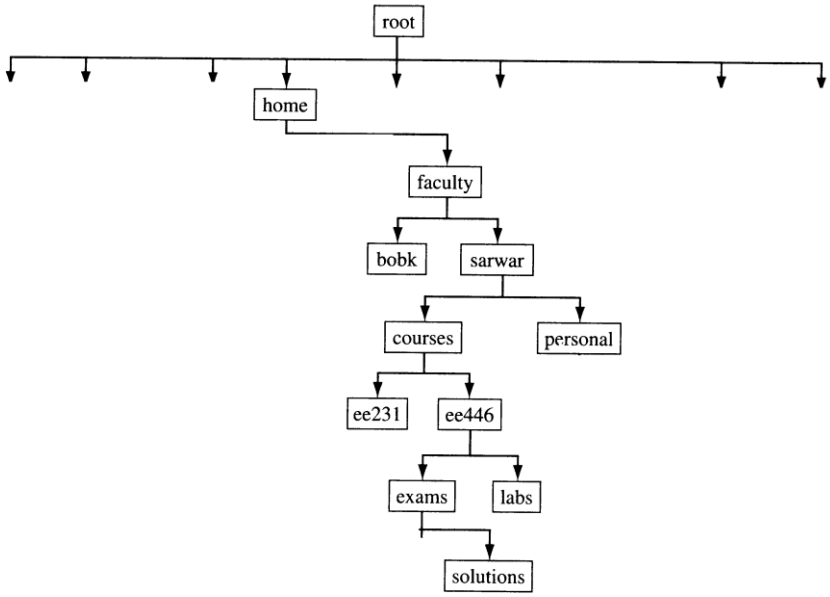
VIII. Access I/O device

Question: (Topic: Interrupts and Traps) **[CLO-1]**

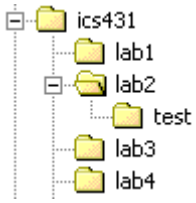
1. What is the purpose of interrupts? What are the differences between a trap and an interrupt? Can traps be generated intentionally by a user program? If so, for what purpose?
2. Describe various events where interrupts, traps and exception signals are generated. Does all of these even invoke kernel? What are the events that do not invoke kernel?

Question (Topic: UNIX/ LINUX file Hierarchies) **[CLO-1]**

1. Write the set of commands to create the following file hierarchy in UNIX bash.



2. Write the set of commands for moving the file test to **lab4** considering the directory **ics431** has root as its parent directory and **lab1** is your current directory.



ASSIGNMENT 02 (CHAPTER 03)

Question: (Topic: Processes, Context Switching) **[CLO-2]**

1. In a system context switching occurs between processes (and/or threads with a process) via dispatcher in short-term scheduling of the ready queue. List and discuss the items that are loaded or saved to/from a processor register during a process (and/or threads within a process) context switch.
2. Explain different sections of the process image.

Question: (Topic Processes, Schedulers) **[CLO-2]**

In most priority schedulers, a process priority is increased and decreased dynamically over time. List the reasons:

1. Why would a typical scheduler increase the priority of an executing process? What problem(s) would be solved by doing so?
2. Why would a typical scheduler decrease the priority of an executing process? What problem(s) would be solved by doing so?

Question: (Topic: Process Creation and Termination) **[CLO-2]**

1. The following code is known as **fork bomb.** What is the potential danger in running this code? What should be done in the kernel to limit this problem?

```
while (1) fork();
```

2. Practice the following code in which we will see how a **zombie process** is created.

```
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
        // fork returns process id in the parent process

   pid_t child_pid = fork();

        // Parent process
        if (child_pid > 0)
                sleep(60);

        // Child process
        elseif (child_pid==0)
                exit(0);
```

```
        else
        exit (EXIT_FAILURE)

        return 0;
}
```

The child process completes its execution by using an **exit()** system call.

So when the child finishes its execution **'SIGCHLD'** signal is delivered to the parent process by the kernel. Parents should, ideally, read the child's status from the process table and then delete the child's entry. But here the parent does not wait for the child to terminate, rather it does its own subsequent job, i.e. here sleeping for 60 seconds.

So the child's exit status is never read by the parent and the child's entry continues to remain there in the process table even when the child has died.

Note: Kernel sends a **SIGCHLD** signal to the parent process to indicate that the child process has ended.

3. Write a code for generating an **orphan child** process using the following steps:
   Call the fork system call and save the pid in PID_C. The parent process sleeps for 20 seconds while the child process sleeps for 30 seconds. So after sleeping for 20 seconds the parent completes its execution while the child process is still there for at least 30 seconds.
   Note: When the child process becomes an orphan process, the kernel reassigns the parent process to the child process. As a result the parent process id of the child process before and after sleep() will be different.