National University of Computer and Emerging Sciences

Operating Systems Lab (CL2006)

Date: March 12th 2025

Course Instructor(s) Miss Syeda Tehreem, Miss Fatima Gado

Sessional-I Exam (A)

Total Time: 1.5 Hours

Total Marks: 50

Total Questions: **03**

Semester: Fall-2025

Campus: Karachi

Dept: Computer Science

____________________	________	________	______________
Student Name	Roll No	Section	Student Signature

CLO # 1: Understand and Analyze Command Line tools for Linux OS and Shell scripts for system level programming to automate tasks such as file management, system backups and software installations. ***(Lab # 1 and Lab # 3)***

Q1. [= 10wtg 20 marks]

Part (A): (Manually write the commands)

1. Which command is used to print the absolute path of the current working directory?
2. How can you list all files and directories, including hidden ones, in the current location?
3. Write the command to navigate to the "Documents" directory.
4. Which command is used to create a new directory named "Project_Files"?
5. How can you remove an empty directory named "TestDir"?
6. Write the command to copy a file named "report.pdf" to the "/home/user/Backup" directory.
7. How do you rename "notes.docx" to "final_notes.docx"?
8. Which command deletes a file named "draft.txt" permanently?
9. How can you create a new empty file named "task_list.md"?
10. Write the command to view the contents of "logfile.log".

Sol:

```
1. pwd
2. ls -a
3.cd Documents
4.mkdir Project_Files
5.rmdir TestDir
6.cp report.pdf /home/user/Backup/
7.mv notes.docx final_notes.docx
8.rm draft.txt
9.touch task_list.md
10.cat logfile.log
```

Part (B)

A company requires a shell script to help employees track their daily tasks. The script should display a menu with the following options:

1. Add a new task (user inputs task name and priority: High, Medium, or Low).
2. View all tasks.

3. Display tasks sorted by priority (High → Medium → Low).
4. Mark a task as completed (remove from the list after confirmation).
5. Exit the program.

The script should validate inputs, handle errors (such as empty task names), and store tasks in a text file for persistence. The menu should run in a loop until the user chooses to exit.

Sol:

```
        #!/bin/bash

TASK_FILE="tasks.txt"

add_task() {
  echo "Enter task name:"
  read task
  if [[ -z "$task" ]]; then
    echo "Task name cannot be empty."
    return
  fi
  echo "Enter priority (High/Medium/Low):"
  read priority
  echo "$priority:$task" >> "$TASK_FILE"
  echo "Task added!"
}

view_tasks() {
  echo "All Tasks:"
  cat "$TASK_FILE"
}

sort_tasks() {
  echo "Sorted Tasks:"
  sort -r "$TASK_FILE"
}

mark_completed() {
  echo "Enter task to remove:"
  read task
  sed -i "/$task/d" "$TASK_FILE"
  echo "Task removed!"
}

while true; do
  echo "1. Add Task"
  echo "2. View Tasks"
  echo "3. Sort by Priority"
  echo "4. Mark Completed"
  echo "5. Exit"
  read choice
  case $choice in
    1) add_task ;;
    2) view_tasks ;;
    3) sort_tasks ;;
    4) mark_completed ;;
    5) exit ;;
    *) echo "Invalid choice" ;;
  esac
done
```

CLO # 2: Gain hands on experience in writing code that interacts with operating system services related process and files system, multi-thread programing and different synchronization primitives. (Lab # 4 and Lab # 6)

Q2. [=10wtg 20 marks]

Part (A)

Design a file-copying program named filecopy.c using ordinary pipes. This program will be passed two parameters: the name of the file to be copied and the name of the destination file. The program will the create an ordinary pipe and write the contents of the file to be copied to the pipe. The child process will read this file from the pipe and write it to the destination file. For example, if we invoke the program as follows:

- ./filecopy input.txt copy.txt
- The file input.txt will be written to the pipe. The child process will read the contents of this file and write it to the destination file copy.txt.

Sol:

```
            #include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
   if (argc != 3) {
      fprintf(stderr, "Usage: %s source_file destination_file\n", argv[0]);
      exit(1);
   }

   int pipefd[2];
   pipe(pipefd);

   pid_t pid = fork();
   if (pid == 0) { // Child process
      close(pipefd[1]);
      int dest = open(argv[2], O_WRONLY | O_CREAT, 0644);
      char buffer[1024];
      int bytesRead;
      while ((bytesRead = read(pipefd[0], buffer, sizeof(buffer))) > 0) {
         write(dest, buffer, bytesRead);
      }
      close(dest);
      close(pipefd[0]);
   } else { // Parent process
```

```
            close(pipefd[0]);
            int src = open(argv[1], O_RDONLY);
            char buffer[1024];
            int bytesRead;
            while ((bytesRead = read(src, buffer, sizeof(buffer))) > 0) {
                write(pipefd[1], buffer, bytesRead);
            }
            close(src);
            close(pipefd[1]);
            wait(NULL);
        }
        return 0;
    }
```

Part(B): (Manually write the complete C code)

The sum of the reciprocals of the first N even numbers is defined as:
Write a C program that performs the following operations:

$$S(N) = \frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \cdots + \frac{1}{2N}$$

- The parent process reads an integer N from the command line and passes it to the child process.
- The child process computes and prints the sum S(N) of the reciprocals of the first N even numbers.
- The parent process waits for the child process to complete before terminating.
- Implement error handling to manage invalid inputs (e.g., non-integer or negative values).

Sol:

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

    double sum_of_reciprocals(int N) {
        double sum = 0.0;
        for (int i = 1; i <= N; i++) {
            sum += 1.0 / (2 * i); // Summing reciprocals of first N even numbers
        }
        return sum;
    }

    int main(int argc, char *argv[]) {
        if (argc != 2) { // Ensure exactly one argument is provided
            fprintf(stderr, "Usage: %s <positive integer N>\n", argv[0]);
            exit(1);
        }

        int N = atoi(argv[1]); // Convert input to integer
        if (N <= 0) { // Check for valid positive integer input
            fprintf(stderr, "Error: Please enter a positive integer.\n");
            exit(1);
        }

        pid_t pid = fork(); // Create child process

        if (pid < 0) {
            perror("Fork failed");
            exit(1);
        } else if (pid == 0) { // Child process
            double result = sum_of_reciprocals(N);
            printf("Sum of reciprocals of first %d even numbers: %lf\n", N, result);
            exit(0);
        } else { // Parent process
            wait(NULL); // Wait for child to complete
        }

        return 0;
    }
```

CLO # 3: Understand how to configure and customize Linux Kernel for installations, applying patches and performance optimizations. (Lab # 2)

Q3. [= 5 wtg 10 marks]

You are developing a speed conversion system that converts speed between meters per second (m/s) and kilometers per hour (km/h). The system consists of a header file (speed.h) that declares functions for conversion, a C file (convert_speed.c) that implements the logic, and another C file (main.c) that takes user input, calls the functions, and displays the result. Write a Makefile that compiles convert_speed.c and main.c separately into object files, links them to produce an executable named speed_converter, and ensures efficient compilation by recompiling only modified files. Additionally, include a clean target to remove compiled files.

Sol:

```
speed_converter: convert_speed.o main.o
	gcc -o speed_converter convert_speed.o main.o

convert_speed.o: convert_speed.c speed.h
	gcc -c convert_speed.c

main.o: main.c speed.h
	gcc -c main.c

clean:
	rm -f speed_converter convert_speed.o main.o
```