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]

a. You have recently joined a company as a junior system administrator, and your manager has assigned you multiple critical tasks to maintain and optimize the company's Linux-based system. You need to navigate to the Desktop, create a main directory named Project Files along with subdirectories Docs, Scripts, and Backups, and list their contents to ensure they are created successfully. After that, rename an existing file oldfile.txt to newfile.txt, move it to the Docs directory, create an empty file inside the Scripts directory, and edit it using a text editor. You are also required to change ownership of all files inside Project Files to a specific user and modify permissions so only the owner can execute files in the Scripts directory. You are also required to add a new user to the system, set a password for the user, and change their ownership for specific files. To monitor system performance, you must display real-time system statistics, check memory usage, and search for a specific word inside a text file. Additionally, find all ".sh" files in the Scripts directory, remove an unnecessary file from the Backups directory, and create a compressed archive of Project Files using ZIP format. You also need to set the system date and modify the timestamp of a file or directory. Finally, search for a specific word inside a text file located in the Docs directory and find all ".sh" files in the Scripts directory. (on paper)

b. A system administrator needs a shell script to efficiently monitor and manage system processes using a menu-driven interface. The script should allow the user to view all running processes, search for a specific process by name, kill a process by entering its Process ID (PID), and check system resource usage, including CPU and memory utilization. Before performing any operation, the script should validate user input using if-else conditions to ensure correctness, and if an attempt is made to kill a non-existent process, an appropriate error message should be displayed. The script should run continuously in a loop until the user chooses to exit, with each functionality implemented in separate functions for better modularity.

National University of Computer and Emerging Sciences

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]

a) Write a C program where the parent process writes three integers (`10`, `2`, `7`) to a file `data.txt`. Use sequential forking to create three child processes:
_Child 1 reads the first integer, computes the sum of the first `n` natural numbers, and prints the result.
_Child 2 reads the second integer, computes its cube, and prints the result.
_Child 3 reads the third integer, computes its square, and prints the result.
Ensure the parent waits for each child to complete before forking the next. Use file handling and proper error checking.
Output Example:
Sum of first 10 natural numbers: 55
Cube of 2: 8
Square of 7: 49
All child processes have completed their tasks. (on paper)

b) Implement a program using pipes for two-way communication between parent and child processes. The parent sends an integer array to the child, which computes the median and sends it back. The parent then computes the factorial of the median and displays it. Use proper synchronization and explain the purpose of each pipe.

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 building a temperature converter that converts Celsius to Fahrenheit and Fahrenheit to Celsius. The system consists of a header file (temperature.h) that declares conversion functions, a C file (convert.c) that implements the conversions, and another C file (main.c) that takes user input, calls the functions, and displays the result. Write a Makefile that compiles convert.c and main.c separately into object files, links them to produce an executable named temperature_converter, and ensures efficient compilation by recompiling only modified files. Additionally, include a clean target to remove compiled files. (NO marks for conversion logic)