Operating System Tasks

1. Create a multithreaded C program that sorts an array of integers using two threads. The first thread should sort the first half of the array, and the second thread should sort the second half. After both threads finish sorting, the main thread should merge the two sorted halves into a final sorted array.
2. Create a multithreaded C program that performs matrix multiplication using threads. Prompt the user to input two square matrices of size 3x3. Use a separate thread to calculate each row of the result matrix. After all threads have completed, display the final multiplied matrix.

3. Write a C program that replicates a simplified version of the \`cp\` command using system calls. It should read the source file using \`read()\` and write to a destination file using \`write()\`, with error handling.

4. Create a C program that continuously prints a message like “Running…” every 2 seconds. When the user presses Ctrl+C (SIGINT), catch the signal and display “SIGINT caught, terminating safely.” Use \`signal()\` to handle the signal.

5. Develop a C program that simulates a countdown timer from 10. If the user presses Ctrl+C, catch SIGINT and pause the timer. When the user presses Ctrl+Z, catch SIGTSTP to resume the countdown.

6. Create a C program that runs an infinite loop, printing “Working...” every 3 seconds. Set up a signal handler for SIGUSR1 such that when the signal is received, a global flag is toggled between 1 and 0. If the flag is 1, the process should print “Paused by SIGUSR1” instead of “Working...”. Sending the signal again should resume the normal output.

7. Create a C program that uses semaphores to control access to a shared counter variable updated by multiple threads. Ensure that only one thread can update the counter at a time.

8. Implement the Dining Philosophers problem using semaphores. Ensure no deadlock occurs and neighboring philosophers do not eat simultaneously.

9. Create a C program where multiple threads increment a global variable. Use a mutex to ensure the updates are synchronized and no race condition occurs.

10. Create a program where multiple threads simulate people accessing an ATM. Use a mutex to restrict access to one person at a time, printing messages when each user accesses and leaves the ATM.

Scheduling Algorithm Tasks

Task 1: First-Come First-Served (FCFS)

Given the following set of processes with their arrival and burst times, draw the Gantt chart using FCFS scheduling. Then, calculate the Waiting Time, Turnaround Time, and Average Waiting Time and Average Turnaround Time.

Process	Arrival Time	Burst Time
P1	0	5
P2	1	3
P3	2	8
P4	3	6

Task 2: Shortest Job First (SJF) – Non-Preemptive

Consider the following processes. Using Non-Preemptive SJF, prepare the Gantt chart and calculate Waiting Time, Turnaround Time, and their averages.

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Task 3: Round Robin (RR)

Given the processes below and Time Quantum = 3 ms, draw the Gantt chart using Round Robin Scheduling and calculate Waiting Time, Turnaround Time, and their averages.

Process	Arrival Time	Burst Time
P1	0	5
P2	1	4
P3	2	2
P4	3	1

Task 4: Priority Scheduling – Non-Preemptive

Perform Non-Preemptive Priority Scheduling for the following processes. Lower number indicates higher priority. Create the Gantt chart and calculate Waiting Time, Turnaround Time, and their averages.

Process	Arrival Time	Burst Time	Priority
P1	0	4	2
P2	1	3	1
P3	2	1	4
P4	3	2	3