**Assignment No. 03**

A logo of a university of engineering and technology

Description automatically generated

**Spring 2024**

**CSE-204**

**Operating System**

Submitted by: **NAVEED AHMAD**

Registration No.: **22PWCSE2165**

Class Section: **B**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Student Signature: A blue line on a white surface

Description automatically generated



Submitted to:

Dr. Madiha Sher

24/05/2024

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

**Question 1:**

*Write a multithreaded program to calculate the sum of all elements in an array. Considering a system with 4 cores and using a 1-to-1 threading model, how many threads would you create? Explain your reasoning*.

**Answer:**

In a system with 4 cores and using a 1-to-1 threading model, we can create 4 threads.

* One-to-One Threading Model: Each thread is assigned to a single core. With 4 cores available, we create 4 threads to utilize all cores simultaneously.
* Efficient Utilization of Resources: By creating 4 threads, each core can handle a portion of the computation independently, maximizing performance through parallel execution.
* Balanced Workload Distribution: Splitting the array into chunks and assigning each chunk to a thread ensures that the workload is evenly distributed among threads, allowing for efficient computation**.**

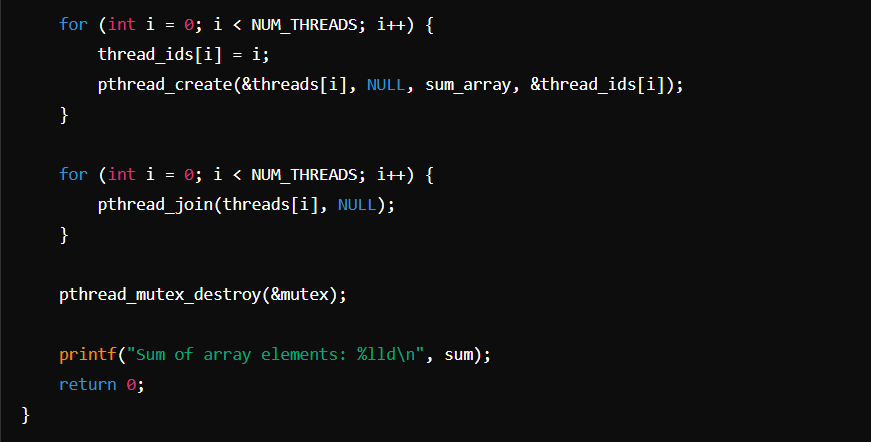
**Code:**

**PTHREAD IMPLEMENTATION:**

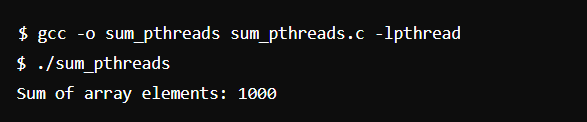
****

**A screen shot of a computer program

Description automatically generated**

****

**Output:**

****

**OPEN-MP IMPLEMENTATION:**

**A computer screen shot of text

Description automatically generated**

**Output:**

**A black screen with white text

Description automatically generated**

**Question No. 2:**

Write a multithreading program to determine the frequency of a specific

number in a large array. Given a system with 8 cores and a 1-to-1 threading model, how many threads would you use? Explain your reasoning.

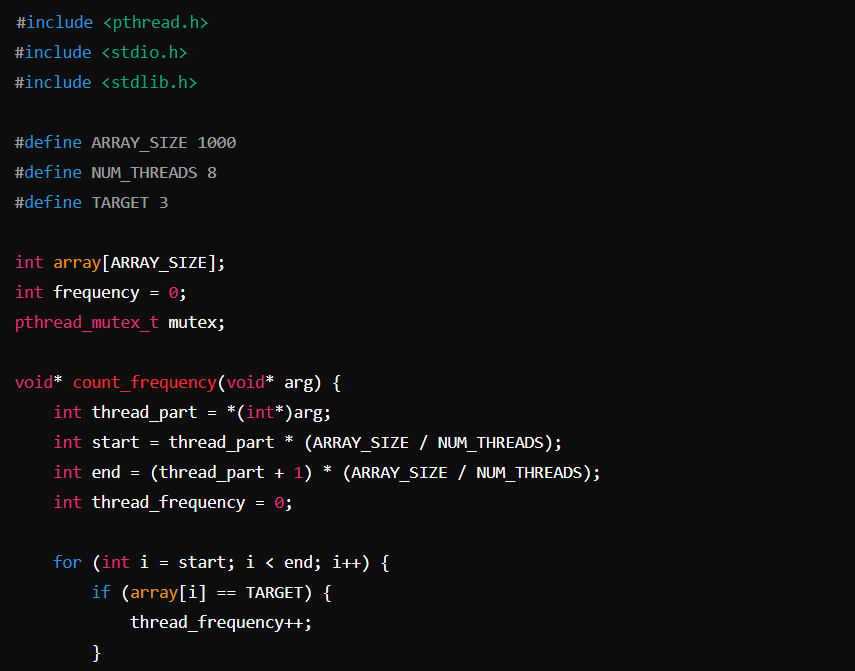
**PTHREAD IMPLEMENTATION:**

**REASONING:**

* Given that the system has 8 cores, we can create 8 threads to distribute the task of counting the frequency of the specific number across the entire array.

**Number of Threads:**

* 8 threads



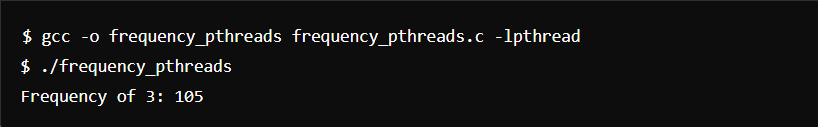
A computer screen shot of text

Description automatically generated

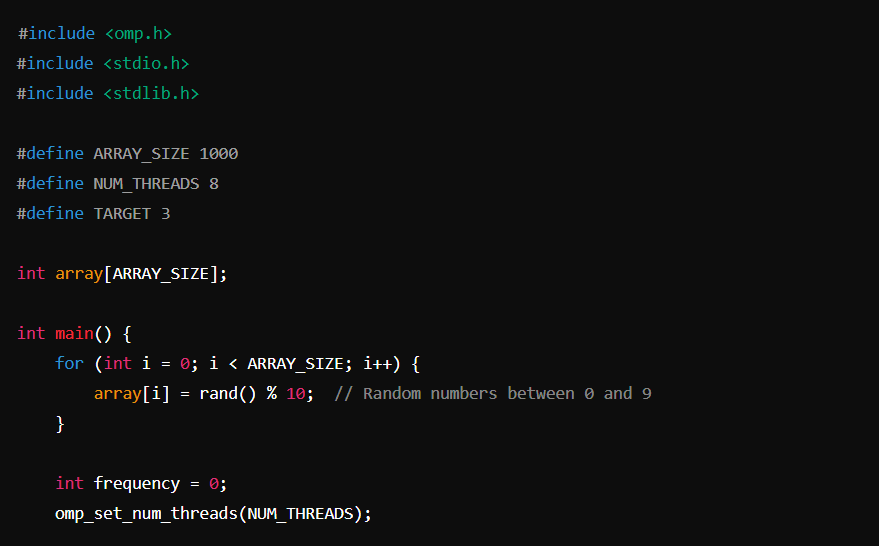
A computer screen with white text

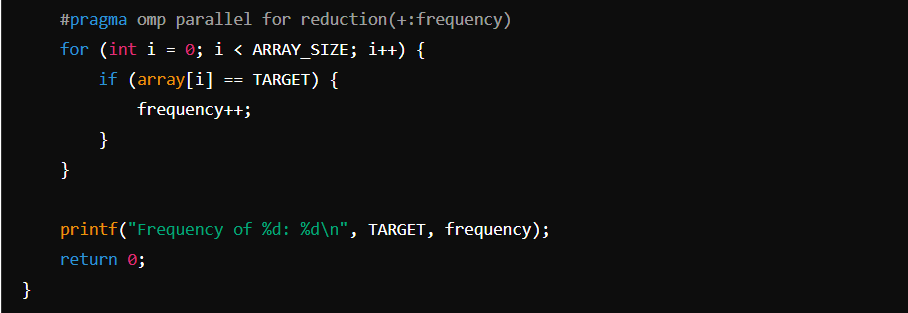
Description automatically generated

**Output:**

****

**OPEN-MP IMPLEMENTATION:**





**Output:**

**A black screen with white text

Description automatically generated**

**Question No.** 3:

Write a multithreading program to perform the following analyses on a very

large array:

* Calculate the sum of all elements
* Calculate the product of all elements
* Find the minimum value in the dataset
* Find the maximum value in the dataset

Considering the following system specifications:

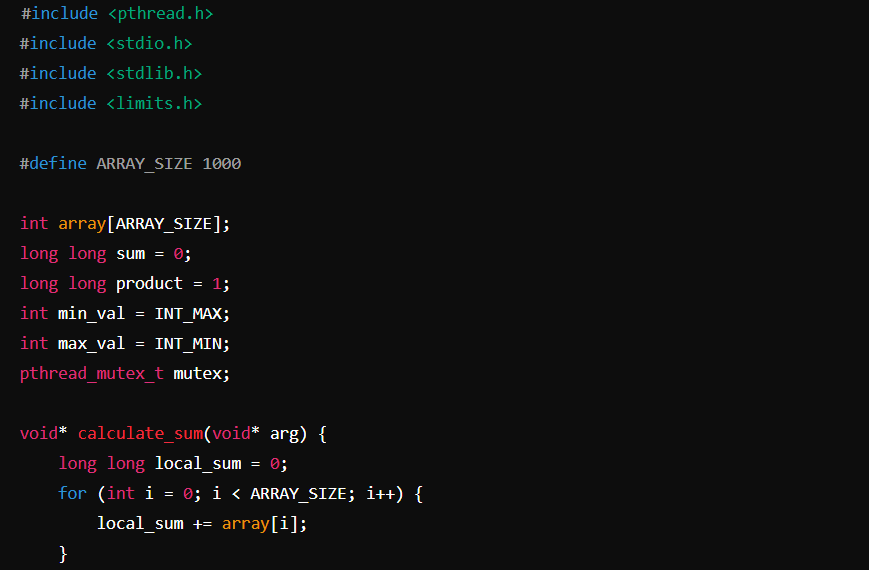
* The system has 2 cores
* it uses a 1-to-1 threading model
* No other process requires CPU resources
* How many threads would you use for this task? Explain your reasoning.

**Reasoning:**

Given that the system has 2 cores, we can create 2 threads to perform these tasks. We need to decide how to distribute the tasks among the threads to optimize performance. One approach is to have one thread perform the sum and product calculations, and the other thread find the minimum and maximum values. However, since these tasks are independent, we can also create 4 threads, one for each task, and let them run in parallel. This approach takes advantage of the fact that while we have only 2 cores, the operating system can still manage context switching efficiently.

**Number of Threads:** 4 threads.

**PTHREAD IMPLEMENTATION:**

****

**A screen shot of a computer code

Description automatically generated**A screen shot of a computer program

Description automatically generated

A computer screen shot of code

Description automatically generated

A computer screen shot of a program code

Description automatically generated

**OUTPUT:**

**A screen shot of a computer

Description automatically generated**

**OPEN-MP IMPLEMENTATION:**

**A computer screen shot of a black screen

Description automatically generated**

**A screen shot of a computer program

Description automatically generated**

A computer screen with many colorful text

Description automatically generated with medium confidence

A computer screen with white text and green text

Description automatically generated

**OUTPUT:**

A black screen with white text

Description automatically generated

**Question No. 4** “

Consider the following set of processes, with the length of the CPU burst given

in milliseconds/timeunits:

A screenshot of a computer screen

Description automatically generated

**Answer:**

**PART -A**

**FIRST-COME, FIRST-SERVED (FCFS):**

Processes are executed in the order they arrive: P1, P2, P3, P4, P5.

**Gantt Chart:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P1** | **P2** | **P3** | **P4** | **P5** |

0 5 8 9 16 20

**SHORTEST JOB FIRST (SJF):**

Processes are executed in order of burst time: P3, P2, P5, P1, P4.

**Gantt Chart:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P3** | **P2** | **P5** | **P1** | **P4** |

0 1 4 8 13 20

**SHORTEST REMAINING TIME (SRT):**

This is a pre-emptive version of SJF. The process with the shortest remaining burst time is selected at each time step.

**Gantt Chart:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P3** | **P2** | **P5** | **P1** | **P4** |

0 1 4 8 13 20

**NON-PREEMPTIVE PRIORITY:**

A larger priority number implies a higher priority. The order of execution by priority: P1, P5, P3, P4, P2.

**Gantt Chart:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P1** | **P5** | **P3** | **P4** | **P2** |

0 5 9 10 17 20

**PREEMPTIVE PRIORITY:**

A larger priority number implies a higher priority. The process with the highest priority is selected at each time step.

**Gantt Chart:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P1** | **P5** | **P3** | **P4** | **P2** |

0 5 9 10 17 20

**ROUND ROBIN (RR) WITH QUANTUM = 2:**

Processes are executed in a round-robin manner with a quantum of 2 units.

**Gantt Chart:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **P1** | **P2** | **P3** | **P4** | **P5** | **P1** | **P2** | **P4** | **P5** | **P1** | **P4** | **P4** |

0 2 4 5 7 9 11 12 14 16 17 19 20

**PART-B:**

**TURNAROUND TIME FOR EACH PROCESS:**

Turnaround time = Finish time - Arrival time

**FCFS:**

* P1: 5 - 0 = 5
* P2: 8 - 0 = 8
* P3: 9 - 0 = 9
* P4: 16 - 0 = 16
* P5: 20 - 0 = 20

**SJF:**

* P1: 13 - 0 = 13
* P2: 4 - 0 = 4
* P3: 1 - 0 = 1
* P4: 20 - 0 = 20
* P5: 8 - 0 = 8

**SRT:**

* Same as SJF.

**Non-pre-emptive Priority:**

* P1: 5 - 0 = 5
* P2: 20 - 0 = 20
* P3: 10 - 0 = 10
* P4: 17 - 0 = 17
* P5: 9 - 0 = 9

**Pre-emptive Priority:**

* Same as Non Pre-emptive Priority.

**Round Robin (Quantum = 2):**

* P1: 15 - 0 = 15
* P2: 7 - 0 = 7
* P3: 5 - 0 = 5
* P4: 18 - 0 = 18
* P5: 13 - 0 = 13

**PART-C:**

**CPU UTILIZATION RATE:**

20/20\*100% => 100%

For all scheduling algorithms, CPU utilization is 100% as all time units are utilized without any idle time.

**PART-D:**

**WAITING TIME FOR EACH PROCESS:**

Waiting time = Turnaround time - Burst time

**FCFS:**

* P1: 5 - 5 = 0
* P2: 8 - 3 = 5
* P3: 9 - 1 = 8
* P4: 16 - 7 = 9
* P5: 20 - 4 = 16

**SJF:**

* P1: 13 - 5 = 8
* P2: 4 - 3 = 1
* P3: 1 - 1 = 0
* P4: 20 - 7 = 13
* P5: 8 - 4 = 4

**SRT:**

* Same as SJF.

**Non-Preemptive Priority:**

* P1: 5 - 5 = 0
* P2: 20 - 3 = 17
* P3: 10 - 1 = 9
* P4: 17 - 7 = 10
* P5: 9 - 4 = 5

**Preemptive Priority:**

* Same as Non-Preemptive Priority.

**Round Robin (Quantum = 2):**

* P1: 15 - 5 = 10
* P2: 7 - 3 = 4
* P3: 5 - 1 = 4
* P4: 18 - 7 = 11
* P5: 13 - 4 = 9

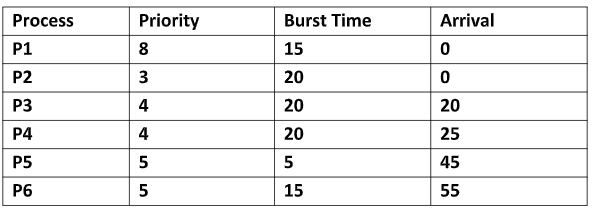
**PART-E:**

**BEST PERFORMING ALGORITHM:**

**Shortest Job First (SJF) and Shortest Remaining Time (SRT):**

***Reason:*** These algorithms minimize the average waiting time and turnaround time compared to other algorithms. Since they always select the process with the shortest burst time, they are efficient in terms of process completion time.

**Question No. 5**: The following processes are being scheduled using a preemptive, priority-based, round-robin scheduling algorithm.



Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. The scheduler will execute the highest priority process. For processes with the same priority, a round-robin scheduler will be used with a time quantum of 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

* 1. Show the scheduling order of the processes using a Gantt chart.
  2. What is the turnaround time for each process?
  3. What is the waiting time for each process?

**Answer:**

**PART-A:**

**SCHEDULING ORDER USING A GANTT CHART:**

* Given the scheduling algorithm and process details, we can construct the Gantt chart step by step:
* **P1** starts at time 0 (since P1 and P2 arrive at the same time, and P1 has a higher priority).
* At time 10, P1 is preempted by P2(next highest priority arriving at time 0).
* At time 20, **P3** arrives, but P2 continues until its time quantum is used up.
* At time 30, **P2** is preempted (time quantum) and **P1** resumes because it has the highest priority among ready processes.
* At time 35, **P1** completes, and **P2** continues.
* At time 40, **P4** arrives, but **P2** continues as it has higher priority.
* At time 45, **P2** finishes its current quantum.
* **P4** starts at time 50 (since **P3** and **P4** have the same priority and round-robin is used).
* **P5** arrives at time 45 but has lower priority, and waits until **P4** finishes its quantum.
* At time 60, **P4** continues (as **P3** and **P4** are round-robin and **P4** was started earlier).
* At time 70, **P4** finishes and **P3** starts.
* At time 75, **P5** starts (higher priority over **P3** and **P6**).
* At time 80, **P5** finishes, **P3** resumes.
* At time 90, **P3** continues (due to round-robin).
* At time 100, **P3** finishes, **P2** resumes.
* **P2** finishes, **P6** starts (round-robin).
* **P6** continues and finishes.

**THE GANTT CHART**:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **P1** | **P2** | **P2** | **P1** | **P1** | **P4** | **P4** | **P3** | **P3** | **P5** | **P5** | **P3** | **P3** | **P6** |

0 10 20 30 35 40 50 60 70 75 80 90 100 110

**PART-B**

**TURNAROUND TIME FOR EACH PROCESS:**

Turnaround Time = Completion Time - Arrival Time

* **P1**: Completion at 35, Arrival at 0: 35−0=35
* **P2**: Completion at 110, Arrival at 0: 110−0=110
* **P3**: Completion at 100, Arrival at 20: 100−20=80
* **P4**: Completion at 70, Arrival at 25: 70−25=45
* **P5**: Completion at 80, Arrival at 45: 80−45=35
* **P6**: Completion at 110, Arrival at 55: 110−55=55

**PART-C**

**WAITING TIME FOR EACH PROCESS:**

Waiting Time = Turnaround Time - Burst Time

* **P1**: Turnaround time 35, Burst time 15: 35−15=20
* **P2**: Turnaround time 110, Burst time 20: 110−20=90
* **P3**: Turnaround time 80, Burst time 20: 80−20=60
* **P4**: Turnaround time 45, Burst time 20: 45−20=25
* **P5**: Turnaround time 35, Burst time 5: 35−5=30
* **P6**: Turnaround time 55, Burst time 15: 55−15=40