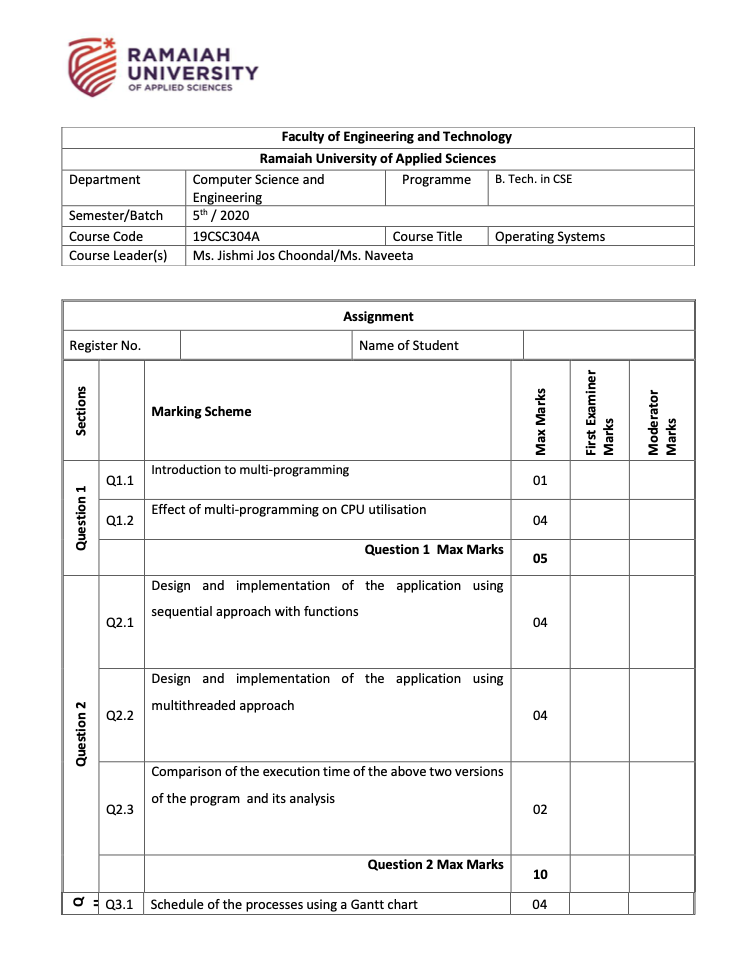


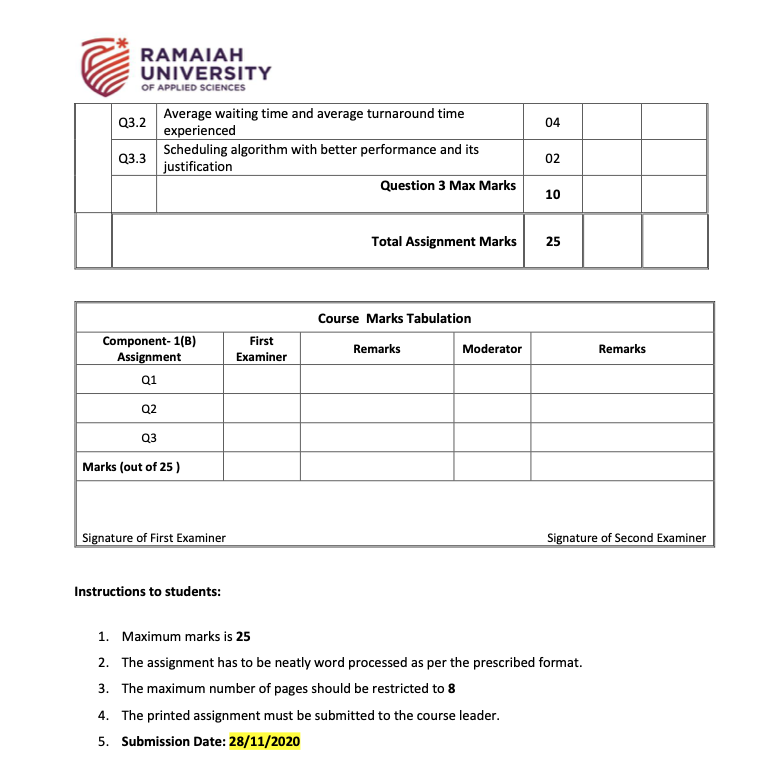
|  |  |
| --- | --- |
| **ASSIGNMENT** | |
| **Course Code** |  |
| **Course Name** |  |
| **Programme** |  |
| **Department** |  |
| **Faculty** |  |

#### 

|  |  |
| --- | --- |
| **Name of the Student** |  |
| **Reg. No** |  |
| **Semester/Year** |  |
| **Course Leader/s** |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Declaration Sheet** | | | | | | | | |
| Student Name |  | | | | | | | |
| Reg. No |  | | | | | | | |
| Programme |  | | | | | Semester/Year |  | |
| Course Code |  | | | | | | | |
| Course Title |  | | | | | | | |
| Course Date |  | | to | |  | | | |
| Course Leader |  | | | | | | | |
| **Declaration**  The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly. | | | | | | | | |
| Signature of the Student | |  | | | | | Date |  |
| Submission date stamp  (by Examination & Assessment Section) | |  | | | | | | |
| Signature of the Course Leader and date | | | | Signature of the Reviewer and date | | | | |
|  | | | |  | | | | |



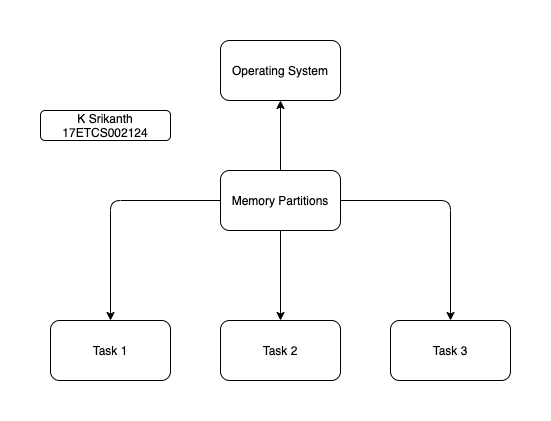


**Question 1)**

**1.1)**

**Introduction**

Multiprogramming is the ability of an operating system to execute more than one program on a single processor machine and one or task can run in the main memory at the same time



*Figure 1 Diagram of how multiprocessing works*

**Example:** A computer running MS Word along with PDF Viewer

**1.2)**

Before the concept of Multiprogramming CPU’s used to run a single process at a time and people noticed that if CPU isn’t doing any work it sits idle this happens like 80% of the time where CPU used to be idle so as the CPU’s and OS evolved, they started to handle multiprogramming concept so that CPU’s doesn’t sit idly. The idea of multiprogramming reduces the idle time of the CPU as multiprogramming accelerates the throughput of the system by efficiently using the CPU time.

**On a single core machine**

A computer with a single CPU core, only one task runs at any point in time, meaning that the CPU is actively executing instructions for that task. Multitasking solves this problem by scheduling which task may run at any given time and when another waiting task gets a turn.

**Example**: Using one application at a time and wait for that application to be closed and then running the next application if you try to run another application there is a chance that the system would crash.

**On a multi core machine**

A computer with a multicore system can do multitasking to execute multiple tasks concurrently. The multiple computing processes work independently on different tasks.

**Example**: Using one application at a time and running another application at the same time this helps CPU to compute more tasks compared on a single core machine

**Let’s take a scenario for a single core processer and see how much time does it take to execute**

Assume that a single core processer has a CPU frequency 1.8GHz lets say that it has to execute 8 instructions

**So, Clock speed is given by**

So once clock executes one instruction it means that 5.55 nano seconds are required to run a single instruction

To execute 8 instructions on a single core machine would take

**Execution time = 8 \* 5.55 ns. = 4 nano seconds**

**Now let’s assume the same clock speed 1.8GHz with quad core processer**

**So, Clock speed is given by**

So once clock executes one instruction it means that 5.55 nano seconds are required to run a single instruction

To execute 8 instructions on a quad core machine would take

**Execution time = = 1 nano seconds**

We can clearly see that multi core CPU utilization is less compared to single core.

**Question 2)**

**2.1)**

**Note**

Srand function is a seed function which is used to place a seed value for random number generation

**Example:** Srand (1) means that the seed value for random number generation is 1

Rand function is used to generate random numbers with mod you can specify the range you want random numbers to be generated

**Example:** rand() % 5 mean that generate random numbers from 0 to 5 range.

**Using Sequential Approach**

**Algorithm (C++)**

*1. Start*

*2. Include headers <iostream>, <time.h>*

*3. using namespace as std*

*4. main*

* 1. *Declaring int m =100 and n =100*
  2. *Declaration of matrix1, matrix 2 and matrixadd with [m][n]*
  3. *Int i,j,k*
  4. *Srand (1) <- Seed Value*
  5. *For loop begin*
     1. *For i to rows[matrix1]*
     2. *For j to columns[matrix1]*
     3. *Matrix1[i][j] = rand () % 5*
     4. *Matrix2[i][j] = rand () % 5*
     5. *exit*

*f. for loop begin*

1. *For i to rows[matrix1]*
2. *For j to columns[matrix1]*
3. *Display matrix1[i][j]*
4. *exit*

*g. for loop begin*

1. *For i to rows[matrix2]*
2. *For j to columns[matrix2]*
3. *Display matrix2[i][j]*
4. *exit*

*h. for loop begin*

1. *For i to rows[matrixadd]*
2. *For j to columns[matrixadd]*
3. *Display matrixadd[i][j] = matrix1[i][j] + matrix2[i][j]*
4. *Exit*

*5. return 0*

*5. Stop*

**Code in C++**

****

Figure 2 C++ Code for addition of two matrixes using sequential approach

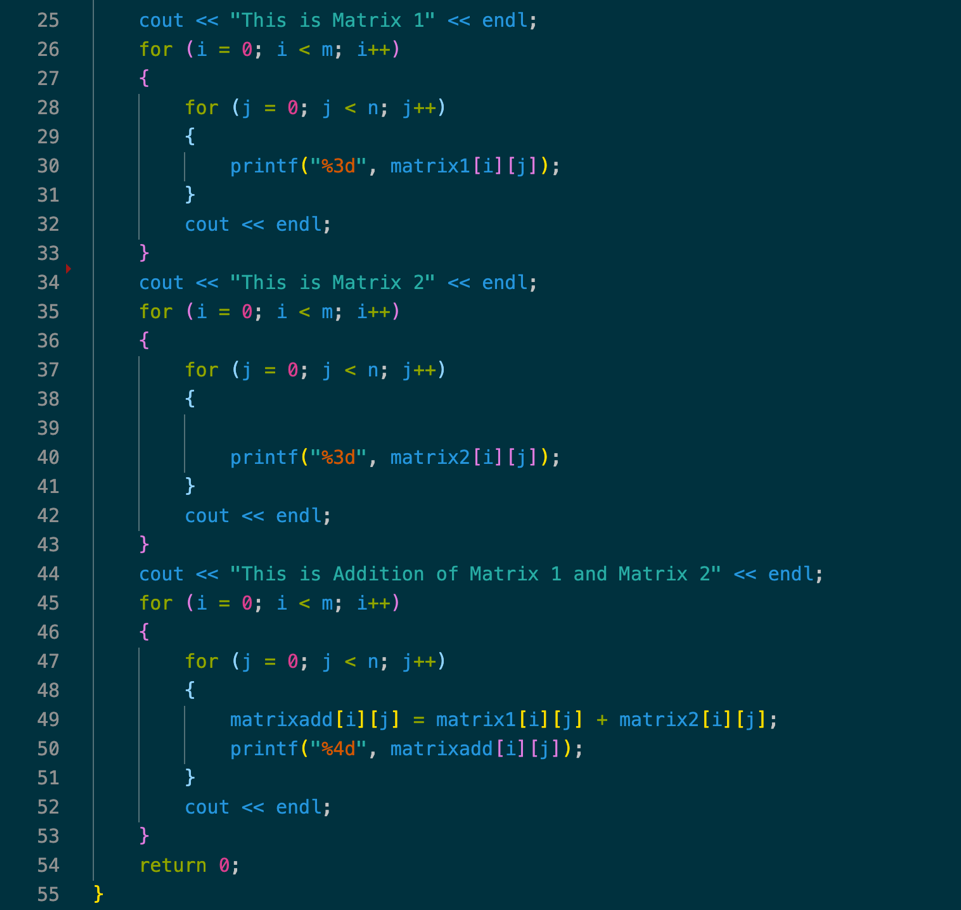
****

Figure 3 C++ Code for addition of two matrixes using sequential approach continued

**Result**

**To compile the code run**

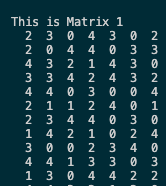
g++ filename.cpp

**Then you I will get an exe file or a.out depending on your OS**

./a.out

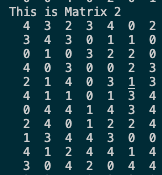
**Output**

**Matrix 1**

****

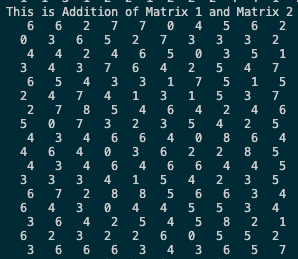
*Figure 4 C++ output for matrix 1 using sequential approach*

**Matrix 2**

****

*Figure 5 C++ output for matrix 2 using sequential approach*

**Matrix Addition**

****

*Figure 6 C++ output for matrix add using sequential approach*

**2.2)**

**Using Multi-Threading Approach**

**Algorithm (C++)**

*1. Start*

*2. Include headers <iostream>, <threads> and <time.h>*

*3. using namespace as std*

*4. define rows 100, columns 100, thread\_count100*

*5. Declaration of matrix1, matrix 2 and matrixadd with [rows][columns]*

*6. Void \*addition (args int rownumber)*

* + 1. *for loop begin*
    2. *For i to columns[matrixadd]*
    3. *Matrixadd[rownumber][i] =*

*matrix1[rownumber][i]+ matrix2[rownumber][i]*

* + 1. *exit*

*7. main*

1. ***thread*** *threads[thread\_count] <-declaration of threads*
2. *int a, b, i, j*

*iii. for loop begin*

1. *For a to rows[rows]*
2. *For b to columns[columns]*
3. *Matrix1[a][b] = rand () % 5*
4. *Matrix2[a][b] = rand () % 5*
5. *Exit*

*iv. for loop begin*

1. *For i to thread\_count*
2. *Display “creating a thread in main”*
3. *For i to thread\_count*
4. *Thread[i] = thread{&addition,i}*
5. *Exit “c” loop*
6. *For i to thread\_count*
7. *Thread[i]. join ()*
8. *Exit” f” loop*
9. *Exit” a” loop*

*v. for loop begin*

1. *For i to rows[matrix1]*
2. *For j to columns[matrix1]*
3. *Display matrix1[i][j]*
4. *exit*

*vi. for loop begin*

1. *For i to rows[matrix2]*
2. *For j to columns[matrix2]*
3. *Display matrix2[i][j]*
4. *exit*

*vii. for loop begin*

1. *For i to rows[matrixadd]*
2. *For j to columns[matrixadd]*
3. *Display matrixadd[i][j] = matrix1[i][j] + matrix2[i][j]*
4. *Exit*

*viii. return 0*

*ix. stop*

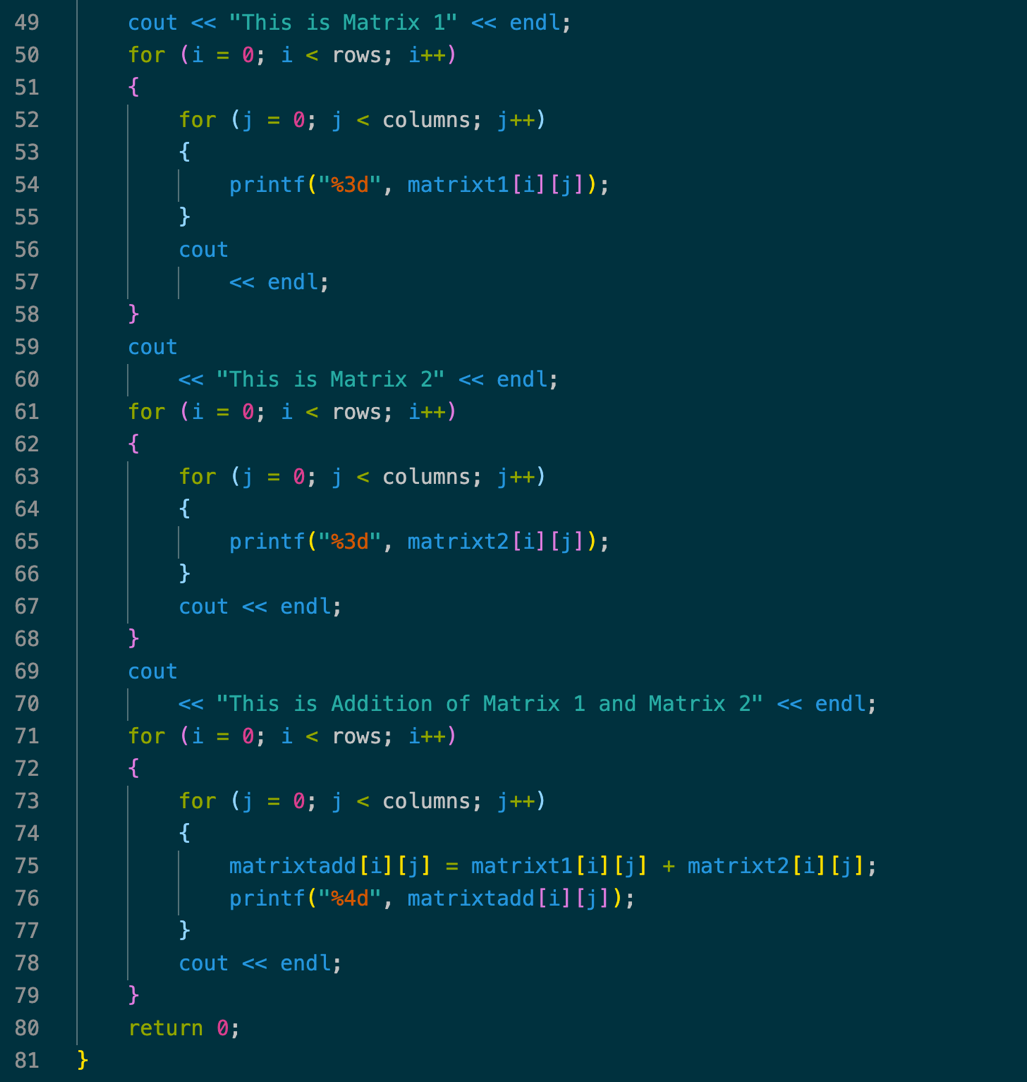
**Code in C++**



Figure 7 C++ Code for addition of two matrixes using multithreading approach



*Figure 8 C++ Code for addition of two matrixes using multithreading approach continued*



*Figure 9 C++ Code for addition of two matrixes using multithreading approach continued*

**Result**

**To compile the code run**

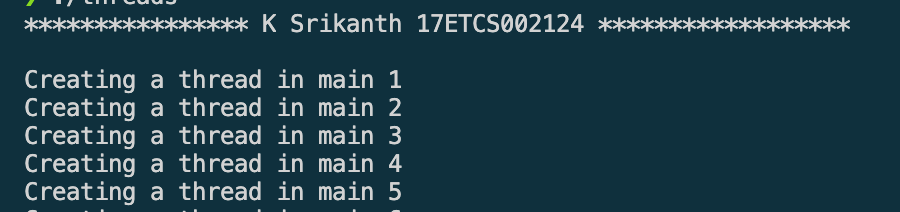
g++ -std=c++11 -o filename filename.cpp

**Then you I will get an exe file depending on your OS**

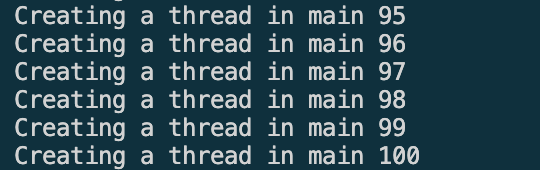
./filename

**Output**

**Threads Creation**

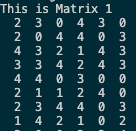


*Figure 10 C++ output for thread creation in main function from 1 to 5 and so*



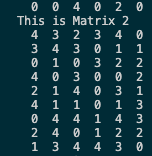
*Figure 11 C++ output for thread creation in main function continued from 95 to 100*

**Matrix 1**

****

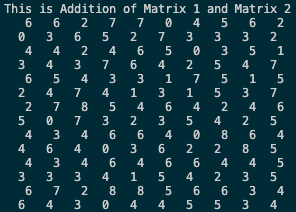
*Figure 12 C++ output for matrix 1 using multithreading approach*

**Matrix 2**

****

*Figure 13 C++ output for matrix 2 using multithreading approach*

**Matrix Addition**

**

*Figure 14 C++ output for matrix addition using multithreading approach*

**Links for my codes**

[Sequential Approach](https://github.com/Srikanth-Kandarp/Labs-Code-5th-Sem/blob/main/OS%20/Codes/Assignment%20Code/Sequential.cpp)

[Multithreading Approach](https://github.com/Srikanth-Kandarp/Labs-Code-5th-Sem/blob/main/OS%20/Codes/Assignment%20Code/Multithreaded.cpp)

**2.3)**

Execution times for Sequential Approach and Multithreading Approach

**Note: We will be calculating average of 5 executions for each approach**

**To find the execution time of your program run**

time ./executable file name

**Sequential Approach**

**Case 1**

****

Figure 15 Execution Time case 1 for sequential approach

**Total Time = 0.620 seconds**

**Case 2**

****

Figure 16 Execution Time case 2 for sequential approach

**Total Time = 0.140 seconds**

**Case 3**

****

Figure 17 Execution Time case 3 for sequential approach

**Total Time = 0.130 seconds**

**Case 4**

****

Figure 18 Execution Time case 4 for sequential approach

**Total Time = 0.123 seconds**

**Case 5**

****

Figure 19 Execution Time case 5 for sequential approach

**Total Time = 0.133 seconds**

**Average**

**Multithreading Approach**

**Case 1**

****

Figure 20 Execution Time case 1 for Multithreading approach

**Total Time = 0.676 seconds**

**Case 2**

****

Figure 21 Execution Time case 2 for Multithreading approach

**Total Time = 0.274 seconds**

**Case 3**

****

Figure 22 Execution Time case 3 for Multithreading approach

**Total Time = 0.264 seconds**

**Case 4**

****

Figure 23 Execution Time case 4 for Multithreading approach

**Total Time = 0.268 seconds**

**Case 5**

****

Figure 24 Execution Time case 5 for Multithreading approach

**Total Time = 0.269 seconds**

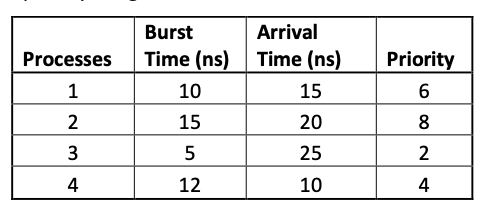
**Average**

**Conclusion**

So, looking at the average of two approach’s we can clearly see that sequential is 0.121 seconds faster than multithreaded approach but in theory multithreaded is way faster than a sequential approach because it is going to share the data while running the threads.

Why did this happen you ask? This can be caused by many reasons few of them would be creating 100 threads took a lot of time If the code were to be using ongoing threads from the CPU while compiling then it would have been a significant difference and secondly if there are any large applications running and using your CPU threads then also you find multithreaded approach a bit slower.

**Question 3)**

****

*Figure 25 Given Problem Table for Question 3*

**3.1)**

**Preemptive scheduling**

**Step 1:**

Looking at **Table 24** the first Process is **Number 4 at arrival time 10ns with Priority being 4**

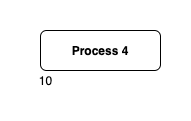
****

Figure 26 Gantt Chat for Step 1 for Preemptive Scheduling

**Step 2:**

Now we look at the next arrival time which is **15ns for Process 1** with **Priority being 6** which is higher than **Process 4 so we kill Process 4 and start doing Process 1**

**Note that time remaining for Process 4 is 7ns**

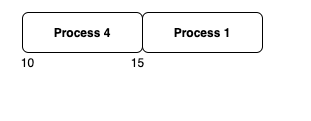


Figure 27 Gantt Chat for Step 2 for Preemptive Scheduling

**Step 3:**

Now we look at the next arrival time which is **20ns for Process 2** with **Priority being 8** which is higher than **Process 1 so we kill Process 1 and start doing Process 2**

**Note that time remaining for Process 4 is 7ns and Process 1 is 5ns**

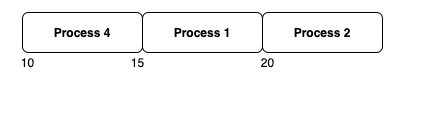
****

Figure 28 Gantt Chat for Step 3 for Preemptive Scheduling

**Step 4:**

Now we look at the next arrival time which is **25ns for Process 3** with **Priority being 2** which is least than **Process 1, Process 2 and Process 4 so we continue processing Process 2 cause it has the highest priority until its burst time.**

**Note that time remaining for Process 4 is still 7ns, Process 1 is still 5ns and Process 3 is 5ns**

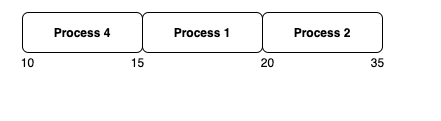
****

Figure 29 Gantt Chat for Step 4 for Preemptive Scheduling

**Step 5:**

Now we look at the next arrival time and we don’t have any arrival time for any new process so we start to execute the killed processes with highest priority and that would be **process 1 with priority being 6 with remaining process time 5ns**

**Note that time remaining for Process 4 is still 7ns and Process 3 is still 5ns**

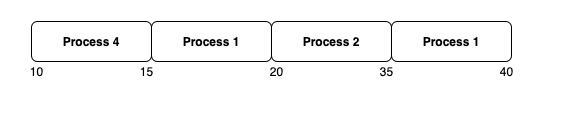
****

Figure 30 Gantt Chat for Step 5 for Preemptive Scheduling

**Step 6:**

Now we look at the next priority process which is **Process 4 with Priority being 4** and still it has **7ns to complete** its Process so we run **Process 4 as Process 3 has least Priority than all.**

**Note that time remaining for Process 3 is still 5ns**

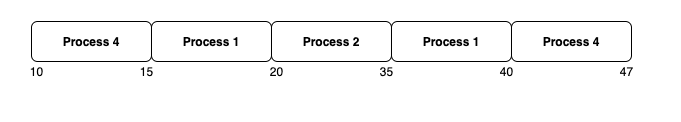
****

Figure 31 Gantt Chat for Step 6 for Preemptive Scheduling

**Step 7:**

Now we look at the next priority process which is **Process 3 with Priority being 2** and this is the final process that has to be executed

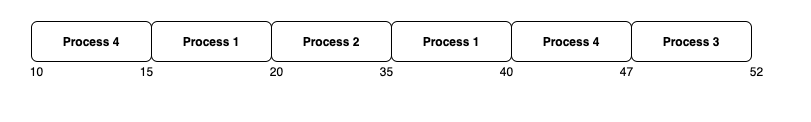
****

Figure 32 Gantt Chat for Step 7 for Preemptive Scheduling

**Final Gantt Chat for Preemptive** **Scheduling Algorithm**

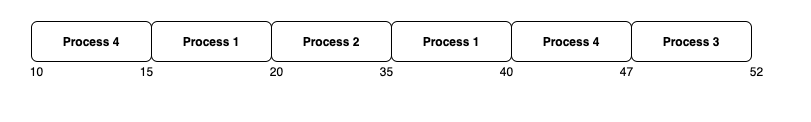
****

Figure 33 Gantt Chat for Preemptive Scheduling

**Non-Preemptive scheduling**

**Step 1:**

Looking at **Table 24** the first Process is **Number 4 at arrival time 10ns with Priority being 4 the difference between Preemptive and Non-Preemptive is that the process completes the execution and it is not being killed by a priority process.**

So here **Process No 4** arrives at 10ns and it completes it **execution of 12ns**

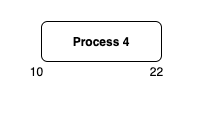
****

Figure 34 Gantt Chat for Step 1 of Non Preemptive Scheduling

**Step 2:**

Now we look at the next priority process which is **Process No 2** with **priority being 8** so it completes **execution of total 15ns**

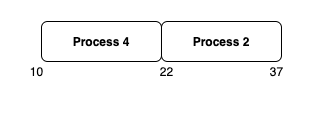
****

Figure 35 Gantt Chat for Step 2 of Non Preemptive Scheduling

**Step 3:**

Now we look at the next priority process which is **Process No 1** with **priority being 6** so it completes **execution of total 10ns**

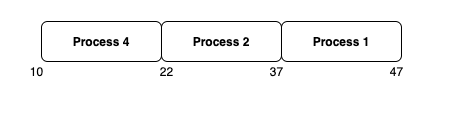
****

Figure 36 Gantt Chat for Step 3 of Non Preemptive Scheduling

**Step 4:**

Now we look at the next priority process which is **Process No 3** with **priority being 2 the least of all so** it completes **execution of total 5ns**

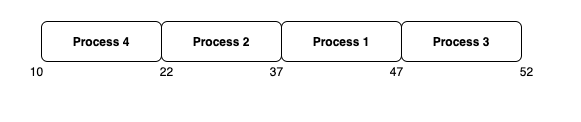
****

Figure 37 Gantt Chat for Step 4 of Non Preemptive Scheduling

**Final Gantt Chat for Non - Preemptive** **Scheduling Algorithm**

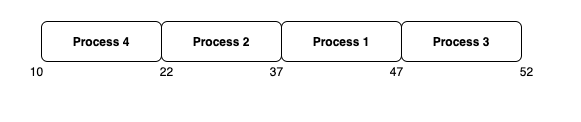
****

Figure 38 Gantt Chat for Non Preemptive Scheduling

**3.2)**

**Preemptive scheduling**

**Formulas**

**Turn Around Time = Completion Time - Arrival Time**

**Waiting Time = Turn Around Time - Burst Time**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Process Number** | **Arrival Time(ns))** | **Priority** | **Burst Time(ns)** | **Remaining Burst Time(ns)** | **Completion**  **Time(ns)** | **Turn Around**  **Time(ns)** | **Waiting Time(ns)** |
| 4 | 10 | 4 | 12 | 7 | 47 | 37 | 25 |
| 1 | 15 | 6 | 10 | 5 | 40 | 25 | 15 |
| 2 | 20 | 8 | 15 | 0 | 35 | 15 | 0 |
| 3 | 25 | 2 | 5 | 5 | 52 | 27 | 22 |

**Table 1**

**Average Waiting Time**

**Average Turn Around Time**

**Non-Preemptive scheduling**

Same Formulas are used to find **Waiting and Turnaround Time as for Preemptive Scheduling for Non-Preemptive scheduling**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Process Number** | **Burst Time(ns)** | **Arrival Time(ns)** | **Priority** | **Completion**  **Time(ns)** | **Turn Around**  **Time(ns)** | **Waiting Time(ns)** |
| 1 | 10 | 15 | 6 | 47 | 32 | 22 |
| 2 | 15 | 20 | 8 | 37 | 17 | 2 |
| 3 | 5 | 25 | 2 | 52 | 27 | 22 |
| 4 | 12 | 10 | 4 | 22 | 12 | 0 |

**Table 2**

**Average Waiting Time**

**Average Turn Around Time**

**3.3)**

Looking at the average waiting time of 11.5ns Non - Preemptive Algorithm performs better then Preemptive with 15.5ns

Reason being that Preemptive Algorithm follows a pattern of which is if a process with higher priority comes it kills the existing process and starts processing the higher priority process so killing process with for higher priority process makes it slow and it takes more time to execute when compared with Non- Preemptive Algorithm where process gets completed and process with next highest priority gets executed so here we don’t see process being killed as the system is consistent throughout the completion of all the process