ASSIGNMENT - 1

Course Code	19CSC304A
Course Name	Operating Systems
Programme	B. Tech
Department	CSE
Faculty	FET

Name of the Student	K Srikanth
Reg. No	17ETCS002124
Semester/Year	5th / 3rd Year
Course Leader/s	Ms. Jishmi Jos Choondal

Declaration Sheet			
Student Name	K Srikanth		
Reg. No	17ETCS002124		
Programme	B.Tech	Semester/Year	5^{th} / 3^{rd} Year
Course Code	19CSC304A		
Course Title	Operating Systems		
Course Date	14/09/2020	to	16/02/2021
Course Leader	Ms. Jishmi Jos Choondal		

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

Faculty of Engineering and Technology			
Ramaiah University of Applied Sciences			
Department	Computer Science and Engineering	Programme	B. Tech. in CSE
Semester/Batch	5^{th} / 2020		
Course Code	19CSC304A	Course Title	Operating Systems
Course Leader(s)	Ms. Jishmi Jos Choondal/Ms. Naveeta		

Assignment					
Register No.		Name of Student			
Sections		Marking Scheme	Max Marks	First Examiner Marks	Moderator Marks
Question 1	Q1.1	Introduction to multi-programming	01		
	Q1.2	Effect of multi-programming on CPU utilisation	04		
		Question 1 Max Marks	**05**		
Question 2	Q2.1	Design and implementation of the application using sequential approach with functions	04		
	Q2.2	Design and implementation of the application using multithreaded approach	04		
	Q2.3	Comparison of the execution time of the above two versions of the program and its analysis	02		
		Question 2 Max Marks	**10**		
Q	Q3.1	Schedule of the processes using a Gantt chart	04		

RAMAIAH UNIVERSITY OF APPLIED SCIENCES

	Q3.2	Average waiting time and average turnaround time experienced	04		
	Q3.3	Scheduling algorithm with better performance and its justification	02		
		Question 3 Max Marks	**10**		
		Total Assignment Marks	**25**		

Course Marks Tabulation				
Component- 1(B) Assignment	**First Examiner**	**Remarks**	**Moderator**	**Remarks**
Q1				
Q2				
Q3				
Marks (out of 25)				
Signature of First Examiner				Signature of Second Examiner

Instructions to students:

1. Maximum marks is **25**
2. The assignment has to be neatly word processed as per the prescribed format.
3. The maximum number of pages should be restricted to **8**
4. The printed assignment must be submitted to the course leader.
5. **Submission Date: 28/11/2020**

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

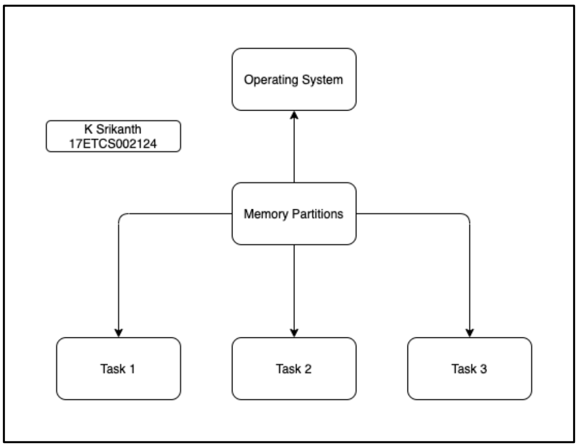

Figure 1 Diagram of how multiprocessing works

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

$$\mathbf{Clock\ Speed\ Time} = \frac{\mathbf{1}}{\boldsymbol{Frequency}}$$

$$\mathbf{Clock\ Speed\ Time} = \frac{\mathbf{1}}{\mathbf{1.8}\ \boldsymbol{X}\ \mathbf{10\hat{\ }9}} = \mathbf{5.55}\ \boldsymbol{nano\ seconds}$$

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

$$\mathbf{Clock\ Speed\ Time} = \frac{\mathbf{1}}{\boldsymbol{Frequency}}$$

$$\mathbf{Clock\ Speed\ Time} = \frac{\mathbf{1}}{\mathbf{1.8}\ \boldsymbol{X}\ \mathbf{10\hat{\ }9}} = \mathbf{5.55}\ \boldsymbol{nano\ seconds}$$

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 = $\frac{\mathbf{8*5.55}}{\mathbf{4}}$ **= 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*
    a. *Declaring int m =100 and n =100*
    b. *Declaration of matrix1, matrix 2 and matrixadd with [m][n]*
    c. *Int i,j,k*
    d. *Srand (1) <- Seed Value*
    e. *For loop begin*
        i. *For i to rows[matrix1]*
        ii. *For j to columns[matrix1]*
        iii. *Matrix1[i][j] = rand () % 5*
        iv. *Matrix2[i][j] = rand () % 5*
        v. *exit*
    f. *for loop begin*
        i. *For i to rows[matrix1]*
        ii. *For j to columns[matrix1]*
        iii. *Display matrix1[i][j]*
        iv. *exit*

    g. *for loop begin*
        i. *For i to rows[matrix2]*
        ii. *For j to columns[matrix2]*
        iii. *Display matrix2[i][j]*
        iv. *exit*
    h. *for loop begin*
        i. *For i to rows[matrixadd]*
        ii. *For j to columns[matrixadd]*
        iii. *Display matrixadd[i][j] = matrix1[i][j] + matrix2[i][j]*
        iv. *Exit*
5. *return 0*
5. *Stop*

Code in C++

```cpp
#include <iostream>
#include <thread>
#include <time.h>
using namespace std;
// 17ETCS002124 K Srikanth
int main()
{
    cout << "**************** K Srikanth 17ETCS002124 ****************** " <<endl;
    cout <<endl;
    int m = 100, n = 100;
    int matrix1[m][n];
    int matrix2[m][n];
    int matrixadd[m][n];
    int i, j, k;
    srand(1);

    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            matrix1[i][j] = rand() % 5;
            matrix2[i][j] = rand() % 5;
        }
    }
```

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

```cpp
    cout << "This is Matrix 1" << endl;
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            printf("%3d", matrix1[i][j]);
        }
        cout << endl;
    }
    cout << "This is Matrix 2" << endl;
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {

            printf("%3d", matrix2[i][j]);
        }
        cout << endl;
    }
    cout << "This is Addition of Matrix 1 and Matrix 2" << endl;
    for (i = 0; i < m; i++)
    {
        for (j = 0; j < n; j++)
        {
            matrixadd[i][j] = matrix1[i][j] + matrix2[i][j];
            printf("%4d", matrixadd[i][j]);
        }
        cout << endl;
    }
    return 0;
}
```

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

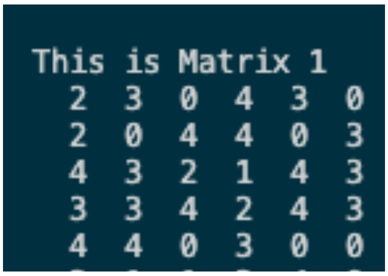

Figure 4 C++ output for matrix 1 using sequential approach

Matrix 2

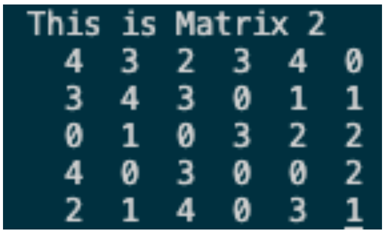

Figure 5 C++ output for matrix 2 using sequential approach

Matrix Addition

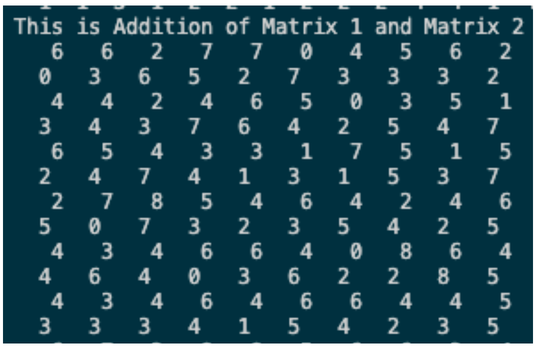

Figure 6 C++ output for matrix add using sequential approach

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)*
   i. *for loop begin*
   ii. *For i to columns[matrixadd]*
   iii. *Matrixadd[rownumber][i] = matrix1[rownumber][i]+ matrix2[rownumber][i]*
   iv. *exit*
7. *main*
   i. ***thread** threads[thread_count] <-declaration of threads*
   ii. *int a, b, i, j*
   iii. *for loop begin*
      a. *For a to rows[rows]*
      b. *For b to columns[columns]*
      c. *Matrix1[a][b] = rand () % 5*
      d. *Matrix2[a][b] = rand () % 5*
      e. *Exit*
   iv. *for loop begin*
      a. *For i to thread_count*
      b. *Display "creating a thread in main"*
      c. *For i to thread_count*
      d. *Thread[i] = thread{&addition,i}*
      e. *Exit "c" loop*
      f. *For i to thread_count*
      g. *Thread[i]. join ()*
      h. *Exit" f" loop*
      i. *Exit" a" loop*
   v. *for loop begin*
      a. *For i to rows[matrix1]*
      b. *For j to columns[matrix1]*
      c. *Display matrix1[i][j]*
      d. *exit*
   vi. *for loop begin*
      a. *For i to rows[matrix2]*
      b. *For j to columns[matrix2]*
      c. *Display matrix2[i][j]*
      d. *exit*
   vii. *for loop begin*
      a. *For i to rows[matrixadd]*

b. For j to columns[matrixadd]
c. Display matrixadd[i][j] = matrix1[i][j] + matrix2[i][j]
d. Exit

viii. return 0
ix. stop

Code in C++

```cpp
#include <iostream>
#include <thread>
#include <time.h>
#define rows 100
#define columns 100
#define threads_count 100
int matrixt1[rows][columns];
int matrixt2[rows][columns];
int matrixtadd[rows][columns];
// 17ETCS002124 K Srikanth
using namespace std;
void *addition(int rownumber)
{
    for (int i = 0; i < columns; i++)
    {
        matrixtadd[rownumber][i] = matrixt1[rownumber][i] + matrixt2[rownumber][i];
    }
}
```

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

```cpp
int main()
{
    cout << "**************** K Srikanth 17ETCS002124 ****************** " <<endl;
    cout <<endl;
    thread threads[threads_count];
    int  i, j;
    srand(1);
    for (int a = 0; a < rows; a++)
    {
        for (int b = 0; b < columns; b++)
        {
            matrixt1[a][b] = rand() % 5;
            matrixt2[a][b] = rand() % 5;
        }
    }

    for (int i = 0; i < threads_count; i++)
    {
        printf("Creating a thread in main %d \n", i + 1);
        for (int i = 0; i < threads_count; i++)
        {
            threads[i] = thread{&addition, i};
        }
        for (int i = 0; i < threads_count; i++)
        {
            threads[i].join();
        }
    }
```

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

```cpp
    cout << "This is Matrix 1" << endl;
    for (i = 0; i < rows; i++)
    {
        for (j = 0; j < columns; j++)
        {
            printf("%3d", matrixt1[i][j]);
        }
        cout
            << endl;
    }
    cout
        << "This is Matrix 2" << endl;
    for (i = 0; i < rows; i++)
    {
        for (j = 0; j < columns; j++)
        {
            printf("%3d", matrixt2[i][j]);
        }
        cout << endl;
    }
    cout
        << "This is Addition of Matrix 1 and Matrix 2" << endl;
    for (i = 0; i < rows; i++)
    {
        for (j = 0; j < columns; j++)
        {
            matrixtadd[i][j] = matrixt1[i][j] + matrixt2[i][j];
            printf("%4d", matrixtadd[i][j]);
        }
        cout << endl;
    }
    return 0;
}
```

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

```
**************** K Srikanth 17ETCS002124 ******************

Creating a thread in main 1
Creating a thread in main 2
Creating a thread in main 3
Creating a thread in main 4
Creating a thread in main 5
```

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

```
Creating a thread in main 95
Creating a thread in main 96
Creating a thread in main 97
Creating a thread in main 98
Creating a thread in main 99
Creating a thread in main 100
```

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

Matrix 1

```
This is Matrix 1
  2  3  0  4  3  0
  2  0  4  4  0  3
  4  3  2  1  4  3
  3  3  4  2  4  3
  4  4  0  3  0  0
```

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

Matrix 2

```
This is Matrix 2
  4  3  2  3  4  0
  3  4  3  0  1  1
  0  1  0  3  2  2
  4  0  3  0  0  2
  2  1  4  0  3  1
```

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

Matrix Addition

```
This is Addition of Matrix 1 and Matrix 2
   6    6    2    7    7    0    4    5    6    2
  0    3    6    5    2    7    3    3    3    2
   4    4    2    4    6    5    0    3    5    1
  3    4    3    7    6    4    2    5    4    7
   6    5    4    3    3    1    7    5    1    5
```

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

Links for my codes

Sequential Approach

Multithreading Approach

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

```
./a.out  0.01s user 0.00s system 1% cpu 0.620 total
```

Figure 15 Execution Time case 1 for sequential approach

Total Time = 0.620 seconds

Case 2

```
./a.out  0.01s user 0.00s system 6% cpu 0.140 total
```

Figure 16 Execution Time case 2 for sequential approach

Total Time = 0.140 seconds

Case 3

```
./a.out  0.01s user 0.00s system 6% cpu 0.130 total
```

Figure 17 Execution Time case 3 for sequential approach

Total Time = 0.130 seconds

Case 4

```
./a.out  0.01s user 0.00s system 7% cpu 0.123 total
```

Figure 18 Execution Time case 4 for sequential approach

Total Time = 0.123 seconds

Case 5

```
./a.out  0.01s user 0.00s system 7% cpu 0.133 total
```

Figure 19 Execution Time case 5 for sequential approach

Total Time = 0.133 seconds

Average

$$\frac{0.620 + 0.140 + 0.130 + 0.123 + 0.133}{5} = 0.2292\ Seconds$$

Multithreading Approach

Case 1

```
./Multithreaded  0.08s user 0.22s system 44% cpu 0.676 total
```

Figure 20 Execution Time case 1 for Multithreading approach

Total Time = 0.676 seconds

Case 2

```
./Multithreaded  0.08s user 0.25s system 122% cpu 0.274 total
```

Figure 21 Execution Time case 2 for Multithreading approach

Total Time = 0.274 seconds

Case 3

```
./Multithreaded  0.08s user 0.24s system 119% cpu 0.264 total
```

Figure 22 Execution Time case 3 for Multithreading approach

Total Time = 0.264 seconds

Case 4

```
./Multithreaded  0.08s user 0.24s system 118% cpu 0.268 total
```

Figure 23 Execution Time case 4 for Multithreading approach

Total Time = 0.268 seconds

Case 5

```
./Multithreaded  0.08s user 0.24s system 119% cpu 0.269 total
```

Figure 24 Execution Time case 5 for Multithreading approach

Total Time = 0.269 seconds

Average

$$\frac{0.676 + 0.274 + 0.264 + 0.268 + 0.269}{5} = 0.3502\ Seconds$$

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.

<u>Question 3)</u>

Processes	Burst Time (ns)	Arrival Time (ns)	Priority
1	10	15	6
2	15	20	8
3	5	25	2
4	12	10	4

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**

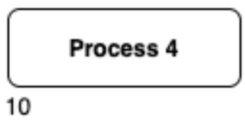

Figure 26 Gantt Chat for Step 1 for Preemptive Scheduling

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

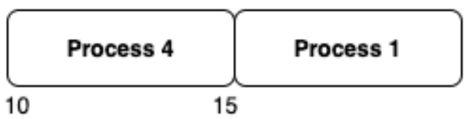

Figure 27 Gantt Chat for Step 2 for Preemptive Scheduling

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

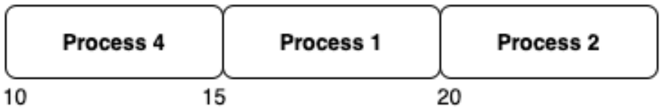

Figure 28 Gantt Chat for Step 3 for Preemptive Scheduling

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

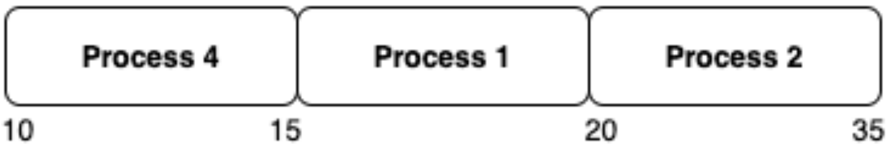

Figure 29 Gantt Chat for Step 4 for Preemptive Scheduling

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

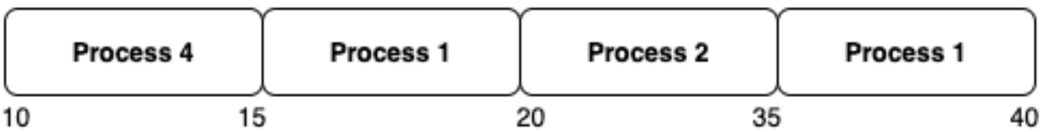

Figure 30 Gantt Chat for Step 5 for Preemptive Scheduling

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

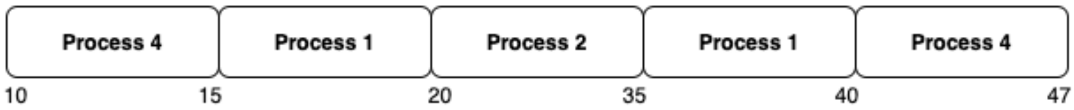

Figure 31 Gantt Chat for Step 6 for Preemptive Scheduling

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

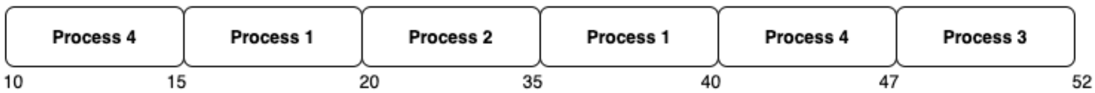

Figure 32 Gantt Chat for Step 7 for Preemptive Scheduling

Final Gantt Chat for Preemptive Scheduling Algorithm

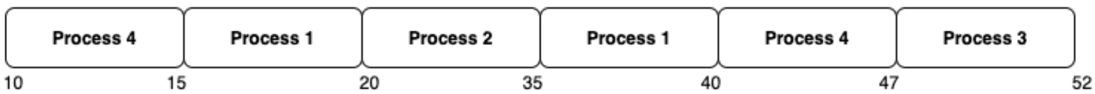

Figure 33 Gantt Chat for Preemptive Scheduling

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**

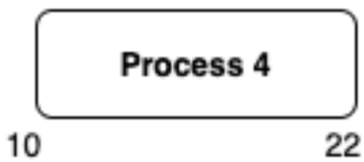

Figure 34 Gantt Chat for Step 1 of Non Preemptive Scheduling

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**

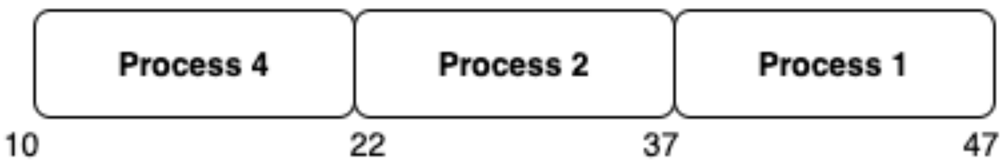

Figure 36 Gantt Chat for Step 3 of Non Preemptive Scheduling

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**

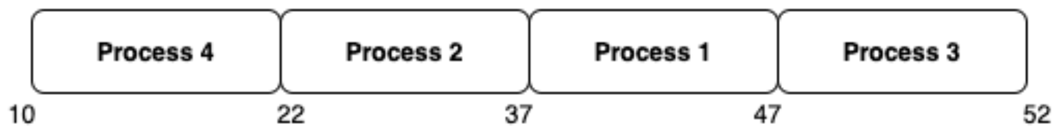

Figure 37 Gantt Chat for Step 4 of Non Preemptive Scheduling

Final Gantt Chat for Non - Preemptive Scheduling Algorithm

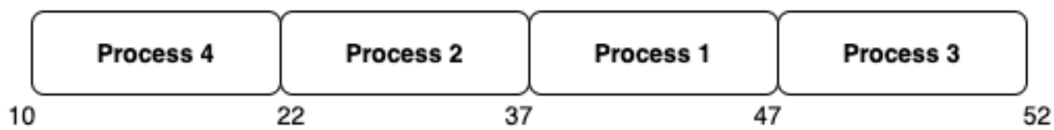

Figure 38 Gantt Chat for Non Preemptive Scheduling

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

$$\boldsymbol{Average\ Waiting\ Time} = \frac{25+15+0+22}{4} = 15.5\ ns$$

Average Turn Around Time

$$\boldsymbol{Average\ Turn\ Arround\ Time} = \frac{37+25+15+27}{4} = 26\ ns$$

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

$$\boldsymbol{Average\ Waiting\ Time} = \frac{22 + 2 + 22 + 0}{4} = 11.5\ ns$$

Average Turn Around Time

$$\boldsymbol{Average\ Turn\ Arround\ Time} = \frac{32 + 17 + 27 + 12}{4} = 22\ ns$$

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