OS LAB ASSIGNMENT - 3

Done By	Roll Number
N Sree Dhyuti	CED19I027

Preemptive Priority Scheduling:

CODE:

```
// N Sree Dhyuti
// CED19I027
// Lab 3 : Q1

// Inclusion of required libraries
#include <stdio.h>

// Function to sort the elements in 5 arrays consecutively : Bubble Sort
// Here arr1 is the array with respect to which the sorting is happening
// Eg : sort (AT, BT, PID, P, BT1, 1, num -1) means we are sorting the 5 arrays wrt Arrival time and in the array range 1 to num-1
void sort(float* arr1, float* arr2, float* arr3, float* arr4, float* arr5, int start, int end)
{
	int a, b;
	for(a = start; a < end - 1; a++)
	{
		for(b = a; b < end; b++)
		{
			if(arr1[a] > arr1[b])
			{
				float temp;
				// Swap
				temp = arr1[b]; arr1[b] = arr1[a]; arr1[a] = temp;
				temp = arr2[b]; arr2[b] = arr2[a]; arr2[a] = temp;
				temp = arr3[b]; arr3[b] = arr3[a]; arr3[a] = temp;
				temp = arr4[b]; arr4[b] = arr4[a]; arr4[a] = temp;
				temp = arr5[b]; arr5[b] = arr5[a]; arr5[a] = temp;
			}
		}
	}
}
```

```
// Function to check if an array has all values as zero or not
int check_array(float* arr, int num)
{
	for(int i = 0; i < num; i++)
	{
		if(arr[i] != 0)
			return 1;
	}
	return 0;
}

// Main
int main()
{
	// Define required variables
	int num, j;
	float current_time=0,avg_wt = 0, avg_tat = 0;

	printf("Number of Processes : ");
	scanf("%d", &num);

	// Incase the user types a negative value for num
	if(num < 0)
	{
		printf("Invalid Number of processes. Try again\n");
		main();
	}

	// Create arrays for storing Process ID, Arrival time and Burst time
	// CT (Completion time, WT (Waiting time), TAT (Turn Around time), P (Priority Value)
	float PID[num], AT[num], BT[num], P[num], CT[num], WT[num], TAT[num];

	// Check Array : To update the times left for each processes to reach completion
	float BT1[num];

	// Take all necessary inputs from user
	for(int i = 0; i < num; i++)
	{
		printf("Enter PID :");
		scanf("%f", &PID[i]);

		printf("Enter Arrival time of Process %f :", PID[i]);
		scanf("%f", &AT[i]);
```

```
		printf("Enter Burst time of Process %f :", PID[i]);
		scanf("%f", &BT[i]);

		BT1[i] = BT[i];

		printf("Enter Priority Value of Process %f :", PID[i]);
		scanf("%f", &P[i]);
	}
	printf("OUTPUTS : \n\n");
	// While we have atleast one process which is not completed...
	while (check_array(BT1, num))
	{
		// Sort the data based on Arrival time
		sort(AT,BT,PID,P,BT1,0,num);

		// Find all the processes whose arrival time < current time
		int j = 0;
		for(j = 0; j < num; j++)
		{
			if(AT[j] > current_time)
				break;
		}

		//Sort those processes being considered according to their Priority Value
		// Highest priority element will be in the j-1 th position
		sort(P, AT, BT, PID, BT1, 0, j);
		j = j - 1;

		// Find the process which has the highest priority among incomplete processes
		while (BT1[j] == 0)
		{
			j = j - 1;
		}

		// Check if two or more processes have the same priority values
		// Incase they do, choose the process with Least Arrival time to reduce the
Overall Average Waiting time
		int k = j;
		int min_at = k;
		while (P[k] == P[k - 1])
		{
			if(AT[k] <= AT[min_at] && BT1[k] != 0)
			{
				min_at = k;
```

```
                }
                k--;
            }
            if(AT[k] <= AT[min_at] && BT1[k] != 0)
            {
                min_at = k;
            }

            BT1[min_at] = BT1[min_at] - 1;

            current_time = current_time + 1;

            printf("From time = %f to time = %f, \n Undergoing Process %f....\n",
current_time - 1, current_time, PID[min_at]);
        printf("-------------------------------------------------------------\n \n");

            if (BT1[min_at] == 0)
            {
                // Other Calculations
                CT[min_at] = current_time;
                TAT[min_at] = CT[min_at] - AT[min_at];
                WT[min_at] = TAT[min_at] - BT[min_at];

                avg_wt = avg_wt + WT[min_at]/num;
                avg_tat = avg_tat + TAT[min_at]/num;
            }
        }

        // Print all details

        printf("-------------------------------------------------------------\n \n");
        printf("PID        AT        BT        CT        TAT        WT\n");
        for (int i = 0; i < num; i++)
        {
            printf("%f     %f     %f     %f     %f     %f\n",PID[i], AT[i], BT[i], CT[i], TAT[i],
WT[i]);
        }
        printf("-------------------------------------------------------------\n \n");

        printf("Average Waiting time : %f\n", avg_wt);
        printf("Average Turn Around time : %f\n", avg_tat);

        return 0;
}
```

OS Lab Assignment-3

N. Sree Dhyuti
CED19I027

22/8/21

(1) Preemptive Priority Scheduling Algorithm

Code Explanation:

Step1: Take all inputs from user
(No of processes, Arrival Time, PID, Burst Time, Priority Value)

Step2: Create an additional check array BT1 [] to keep a check of the time left by each process to finish.

Step3: Sort the array w.r.t Arrival Time.
Find all processes that are already arrived.

Step4: Of those, choose the process with highest priority value.
Incase of multiple processes with same priority value, choose the one with least Arrival Time (To reduce avg waiting time).
If they have same arrival time too, choose the process with least ~~arrival time~~ PID.
Process it for 1 second.

Step5: update the BT1 check array with remaining times.

Step6: Continue step4 & 5 until all the processes are completed.

Req. Calculations:

CT = ~~Current~~ time when process finishes

$TAT = CT - AT$

$WT = CT - BT$

$avg. TAT = \Sigma TAT/n$

$avg. WT = \Sigma WT/n$

N. Sree Dhyuti
CED19I027

Example: Schedule the following processes using preemptive priority scheduling.

PID	AT	BT	Priority
1	0	4	4
2	0	5	5
3	0	1	7
4	0	2	2
5	0	3	1
6	0	6	6

Sol) ~~A~~ Here all processes have arrived at beginning.
~~A~~ So, we simply ~~A~~ keep choosing procedure with highest to least priorities

At time $t=0$,
Processes: $P_1, P_2, P_3, P_4, P_5, P_6$

Gantt chart:

P_3

0 1

At $t=1$,
Processes: P_1, P_2, P_4, P_5, P_6

Gantt Chart:

P_3	P_6

0 1 7

At $t=7$,
Processes: P_1, P_2, P_4, P_5

Gantt Chart:

P_3	P_6	P_2

0 1 7 12

At $t=12$,
Processes: P_1, P_4, P_5

Gantt Chart:

P_3	P_6	P_2	P_1

0 1 7 12 16

N. Sree Dhyuti
CED19I027

At $t = 16$,
Processes: P4, P5

Gantt Chart:

P3	P6	P2	P1	P4

0 1 7 12 16 18

At $t = 18$,
processes: P5

Final Gantt chart:

P3	P6	P2	P1	P4	P5

0 1 7 12 16 18 21

PID	AT	BT	Priority	CT	TAT	WT
1	0	4	4	16	16	12
2	0	5	5	12	12	7
3	0	1	7	1	1	0
4	0	2	2	18	18	16
5	0	3	1	21	21	18
6	0	6	6	7	7	1

$$\text{Avg TAT} = \frac{\Sigma TAT}{n} = \frac{16+12+1+18+21+7}{6} = \frac{75}{6} = 12.5$$

$$\text{Avg. WT} = \frac{\Sigma WT}{n}$$

$$= \frac{12+7+0+16+18+1}{6}$$

$$= \frac{54}{6}$$

$$= 9$$

$\therefore$ Avg TAT = 12.5
Avg WT = 9

CODE OUTPUT FOR THE SAME EXAMPLE:

D:\SEM 5\OS\LAB\LAB3\CED19I027_Lab3_Q1.exe

```
From time = 15.000000 to time = 16.000000,
 Undergoing Process 1.000000....
-----------------------------------------------------------------

From time = 16.000000 to time = 17.000000,
 Undergoing Process 4.000000....
-----------------------------------------------------------------

From time = 17.000000 to time = 18.000000,
 Undergoing Process 4.000000....
-----------------------------------------------------------------

From time = 18.000000 to time = 19.000000,
 Undergoing Process 5.000000....
-----------------------------------------------------------------

From time = 19.000000 to time = 20.000000,
 Undergoing Process 5.000000....
-----------------------------------------------------------------

From time = 20.000000 to time = 21.000000,
 Undergoing Process 5.000000....
-----------------------------------------------------------------

-----------------------------------------------------------------

PID             AT              BT              CT              TAT             WT
5.000000        0.000000        3.000000        21.000000       21.000000       18.000000
4.000000        0.000000        2.000000        18.000000       18.000000       16.000000
1.000000        0.000000        4.000000        16.000000       16.000000       12.000000
2.000000        0.000000        5.000000        12.000000       12.000000       7.000000
6.000000        0.000000        6.000000        7.000000        7.000000        1.000000
3.000000        0.000000        1.000000        1.000000        1.000000        0.000000
-----------------------------------------------------------------

Average Waiting time : 9.000000
Average Turn Around time : 12.500000

--------------------------------
Process exited after 20.2 seconds with return value 0
Press any key to continue . . .
```

N. Sree Dhyuti
CED19I027

Example: Schedule the following processes using priority scheduling (preemptive)

PID	AT	BT	Priority
1	0	4	4
2	1	5	5
3	2	1	7
4	3	2	2
5	4	3	1
6	5	6	6

Sol

At time t = 0,
Processes: P_1
Gantt chart:

P_1
0 – 1

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	3	5	1	2	3	6

At time t = 1,
Process: P_1, P_2
Gantt chart:

P_1	P_2
0 – 1	1 – 2

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	3	4	1	2	3	6

At t = 2,
Processes: P_1, P_2, P_3
Gantt chart:

P_1	P_2	P_3
0 – 1	1 – 2	2 – 3

	P_1	P_2	~~P_3~~	P_4	P_5	P_6
BT left	3	4	~~0~~	2	3	6

At t = 3,
processes: P_1, P_2, P_4
Gantt chart

P_1	P_2	P_3	P_2
0 – 1	1 – 2	2 – 3	3 – 5

	P_1	P_2	~~P_3~~	P_4	P_5	P_6
BT left	3	2	~~0~~	2	3	6

At t = 5,
Processes: P_1, P_2, P_4, P_5, P_6

P_1	P_2	P_3	P_2	P_6
0 – 1	1 – 2	2 – 3	3 – 5	5 – 11

	P_1	P_2	~~P_3~~	P_4	P_5	~~P_6~~
BT left	3	2	~~0~~	2	3	~~0~~

N· Sree Dhyuti
CED19I027

At $t=11$,
processes: P_1, P_2, P_4, P_5
Gantt chart:

P_1	P_2	P_3	P_2	P_6	P_2

0 1 2 3 5 11 13

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	3	0	0	2	3	0

At $t=13$,
Processes: P_1, P_4, P_5
Gantt chart:

P_1	P_2	P_3	P_2	P_6	P_2	P_1

0 1 2 3 5 11 13 16

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	0	0	0	2	3	0

At time $t=16$,
Processes: P_4, P_5

P_1	P_2	P_3	P_2	P_6	P_2	P_1	P_4

0 1 2 3 5 11 13 16 18

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	0	0	0	0	3	0

At time $t=18$,
Processes: P_5

FINAL GANTT CHART

P_1	P_2	P_3	P_2	P_6	P_2	P_1	P_4	P_5

0 1 2 3 5 11 13 16 18 21

PID	AT	BT	PR	CT	TAT	WT
1	0	4	4	16	16	12
2	1	5	5	13	12	7
3	2	1	7	3	1	0
4	3	2	2	18	15	13
5	4	3	1	21	17	14
6	5	6	6	11	6	0

~~ΣTAT = 16+12~~

$$\text{avg TAT} = \frac{\Sigma TAT}{n} = \frac{16+12+1+15+17+6}{6} = \frac{67}{6} = 11.16666\bar{6}$$

$$\text{avg WT} = \frac{\Sigma WT}{n} = \frac{12+7+0+13+14+0}{6} = \frac{46}{6} = 7.666\bar{6}$$

$$\boxed{\begin{aligned} \therefore\ &\text{Avg TAT} = 11.1\bar{6} \\ &\text{Avg WT} = 7.\bar{6} \end{aligned}}$$

CODE OUTPUT FOR SAME EXAMPLE:

```
From time = 18.000000 to time = 19.000000,
 Undergoing Process 5.000000....
-----------------------------------------------------------------

From time = 19.000000 to time = 20.000000,
 Undergoing Process 5.000000....
-----------------------------------------------------------------

From time = 20.000000 to time = 21.000000,
 Undergoing Process 5.000000....
-----------------------------------------------------------------

-----------------------------------------------------------------

PID            AT            BT            CT            TAT            WT
5.000000      4.000000      3.000000      21.000000     17.000000      14.000000
4.000000      3.000000      2.000000      18.000000     15.000000      13.000000
1.000000      0.000000      4.000000      16.000000     16.000000      12.000000
2.000000      1.000000      5.000000      13.000000     12.000000      7.000000
6.000000      5.000000      6.000000      11.000000     6.000000      0.000000
3.000000      2.000000      1.000000      0.000000     0.000000     0.000000
-----------------------------------------------------------------

Average Waiting time : 7.666666
Average Turn Around time : 11.166666

--------------------------------
Process exited after 19.88 seconds with return value 0
Press any key to continue . . .
```

Preemptive Round Robin Scheduling:

CODE:

```
// N Sree Dhyuti
// CED19I027
// Lab 3 : Q2

// Inclusion of required libraries
#include <stdio.h>

//global variables for ready queue
int front = -1;
int rear = -1;

// Queue Structure definition
struct q
```

```
{
   //Function Prototypes
          void (*enqueue) (int* readyq, int x, struct q*, int n);
   int (*dequeue) (int* readyq, struct q*, int n);
   int (*isfilled) (int* readyq, struct q*, int n);
   int (*search) (int* readyq, struct q*, int n, int a);
};
//structure variables
void enqueue1(int* readyq, int x, struct q*, int n);
int dequeue1(int* readyq, struct q*, int n);
int isfilled1(int* readyq, struct q*, int n);
int search1(int* readyq, struct q*, int n, int a);

// Function to sort the elements in 4 arrays consecutively : Bubble Sort
// Here arr1 is the array with respect to which the sorting is happening
// Eg : sort (AT, BT, PID, BT1, 1, num -1) means we are sorting the 4 arrays wrt Arrival Time and
in the array range 1 to num-1
void sort(float* arr1, float* arr2, float* arr3, int* arr4, int start, int end)
{
        int a, b;
        for(a = start; a < end - 1; a++)
        {
                for(b = a; b < end; b++)
                {
                        if(arr1[a] > arr1[b])
                        {
                                float temp;
                                // Swap
                                temp = arr1[b]; arr1[b] = arr1[a]; arr1[a] = temp;
                                temp = arr2[b]; arr2[b] = arr2[a]; arr2[a] = temp;
                                temp = arr3[b]; arr3[b] = arr3[a]; arr3[a] = temp;
                                temp = arr4[b]; arr4[b] = arr4[a]; arr4[a] = temp;
                        }
                }
        }
}

// Function to check if an array has all values as zero or not
int check_array(float* arr, int num)
{
        for(int i = 0; i < num; i++)
        {
                if(arr[i] != 0)
                {
```

```
                        return 1;
                }
        }
        return 0;
}

// Main
int main()
{
   struct q q1;
   // STRUCTURE ENCAPSULATION
   // Assigning Functions to the queue structure
   q1.enqueue = enqueue1;
   q1.dequeue = dequeue1;
   q1.isfilled = isfilled1;
   q1.search = search1;

        // Define required variables
        int num, time_quantum, p_flag = 0;
        float current_time = 0, avg_wt = 0, avg_tat = 0;

        printf("Number of Processes : ");
        scanf("%d", &num);

        // Incase the user types a negative value for num
        if(num < 0)
        {
                printf("Invalid Number of processes. Try again\n");
                main();
        }

        // Create arrays for storing Process ID, Arrival Time and Burst Time
        // CT (Completion Time, WT (Waiting Time), TAT (Turn Around Time)
        int PID[num];
        float AT[num], BT[num], CT[num], WT[num], TAT[num];

        // Check Array : To update the Times left for each processes to reach completion
        float BT1[num];

        // Ready Queue - the queue which holds the names of all processes that are to be
readily processed
        int readyq[num];

        // Take all necessary inputs from user
```

```
	for(int i = 0; i < num; i++)
	{
		printf("Enter PID :");
		scanf("%d", &PID[i]);

		printf("Enter Arrival Time of Process %d :", PID[i]);
		scanf("%f", &AT[i]);

		printf("Enter Burst Time of Process %d :", PID[i]);
		scanf("%f", &BT[i]);

		BT1[i] = BT[i];
	}

	printf("Value of Time Quantum : ");
	scanf("%d",&time_quantum);

	// Sort all processes w.r.t Arrival Time
	sort(AT, BT, BT1, PID, 0, num);

	while(check_array(BT1, num))
	{
		// Find all the processes whose arrival time < current time and enqueue them in
ready queue
		int j = 0;
		for(j = 0; j < num; j++)
		{
			if(AT[j] > current_time)
			{
				break;
			}
			// If PID is not present in the queue, then enqueue
			if((!(q1.search(readyq, &q1, num, PID[j]))) && BT1[j] != 0 && PID[j] !=
p_flag)
			{
				q1.enqueue(readyq, PID[j], &q1, num);
			}
		}
		// Incase the previously processed process is still incomplete, enque it
		if(p_flag != 0)
		{
			if(!(q1.search(readyq, &q1,num, p_flag)))
			{
				q1.enqueue(readyq, p_flag, &q1, num);
```

```
        }
}
int a;
// While the ready queue is filled with atleast one process...
if(q1.isfilled(readyq, &q1, num))
{
        // Dequeue a process from ready queue
        a = q1.dequeue(readyq, &q1, num);

        // Search for that process in the processes arrays using PID
        int i = 0;
        for (i = 0; i < num; i++)
        {
                if (PID[i] == a)
                        break;
        }

        // Update current time, Time remaining for that process to complete
accordingly
        if(BT1[i] > time_quantum)
        {
                current_time = current_time + time_quantum;
                BT1[i] = BT1[i] - time_quantum;
                p_flag = PID[i];
        }
        else
        {
                current_time = current_time + BT1[i];
                BT1[i] = 0;
                p_flag = 0;

                // When a process is completed, do all calculations
                CT[i] = current_time;
                TAT[i] = CT[i] - AT[i];
                WT[i] = TAT[i] - BT[i];

                avg_wt = avg_wt + (WT[i] / num);
                avg_tat = avg_tat + (TAT[i] / num);
        }
}
else
{
        current_time = current_time + 1;
}
```

```
        }

        // Print all details

        printf("------------------------------------------------------------------\n \n");
        printf("PID          AT          BT          CT          TAT          WT\n");
        for (int i = 0; i < num; i++)
        {
                printf("%d     %f     %f     %f     %f     %f\n",PID[i], AT[i], BT[i], CT[i], TAT[i],
WT[i]);
        }
        printf("------------------------------------------------------------------\n \n");

        printf("Average Waiting Time : %f\n", avg_wt);
        printf("Average Turn Around Time : %f\n", avg_tat);

        return 0;
}

// Structure Encapsulated function to enqueue an element into a queue
void enqueue1(int* readyq, int x, struct q* stk, int n)
{
   // When queue is empty
        if(front == -1 && rear == -1)
   {
      front = 0;
      rear = 0;
      readyq[rear] = x;
   }
   // When queue is full
   else if((rear + 1) % n == front)
   {
      printf("Given circular queue is full.\n");
   }
   else
   {
      rear = (rear + 1) % n;
      readyq[rear] = x;
   }
}

// Structure Encapsulated function to dequeue an element from a queue
int dequeue1(int* readyq, struct q* stk, int n)
{
```

```
    //When queue is empty
        if(front == -1 && rear == -1)
    {
        printf("No data has been inputted by user.\n");
        return -1;
    }
    // When only one element is left in queue
    else if(front == rear)
    {
        int e = front;
        front = -1;
        rear = -1;
        return readyq[e];
    }
    else
    {
        int e = front;
        front = (front + 1) % n;
        return readyq[e];
    }
}

// Structure Encapsulated function to check if a queue is filled or empty
int isfilled1(int* readyq, struct q* stk, int n)
{
    //When queue is empty
        if(front == -1 && rear == -1)
        return 0;
    else
        return 1;
}

// Structure Encapsulated function to search for a particular element in the queue
int search1(int* readyq, struct q* stk, int n, int a)
{
        if(front == -1 && rear == -1)
                return 0;
        else
        {
                for(int i = front; i <= n; i++)
                {
                        if(readyq[i%n] == a)
                                return 1;
                        i = i % n;
```

```
                if(i == rear)
                    break;
            }
            return 0;
        }
}
```

N. Sree Dhyuti
CED19I027

(2) Preemptive Round Robin Scheduling Algorithm.

Code explanation:

Step1: Take all inputs from the user.
(No of processes, Arrival Time, Burst time, Time Quantum)

Step2: Create an additional check array BT1[ ] to keep a check of the time left by each process to finish.

Step3: Sort the arrays w.r.t arrival Time. Find all the processes that are already arrived.

Step4: Enqueue those processes (if not enqueued) in the Ready Queue.

Step5: Dequeue one process & complete the process by the value of time Quantum.

Step6: If the process is to be still completed, check for all processes that are arrived at this new time & then enque this process too if required.

Step7: Update BT1 array to keep a check of times remaining for each process.

Step8: Repeat steps 3,4,5,6,7 untill all the processes are completed.

Req. Calculations:

CT = time at which a process is completed

$TAT = CT - AT$

$WT = TAT - BT$

$$\text{avg } TAT = \frac{\Sigma TAT}{n}$$

$$\text{avg } WT = \frac{\Sigma WT}{n}$$

N. Sree Dhyuti
CED19I027

Example: Schedule the following processes using preemptive round robin scheduling.

PID	AT	BT
1	0	4
2	0	5
3	0	1
4	0	2
5	0	3
6	0	6

Time Quantum = 2

Sol) At time t = 0
processes: P_1, P_2, P_3, P_4, P_5, P_6

Gantt chart:

P_1
0 – 2

~~(Complete fully)~~

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	2	5	1	2	3	6

At time t = 2

~~Processes~~ Ready Queue: P_2 P_3 P_4 P_5 P_6 P_1

Gantt Chart:

P_1	P_2
0 – 2	2 – 4

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	2	3	1	2	3	6

At t = 4,

Ready Queue: P_3 P_4 P_5 P_6 P_1 P_2

Gantt chart:

P_1	P_2	P_3
0 – 2	2 – 4	4 – 5

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	2	3	0	2	3	6

At t = 5,

Ready Queue: P_4 P_5 P_6 P_1 P_2

Gantt Chart:

P_1	P_2	P_3	P_4
0 – 2	2 – 4	4 – 5	5 – 7

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	2	3	0	0	3	6

N. Sree Dhyuti
CED19I027

At $t = 7$,

Ready Queue: P_5 P_6 P_1 P_2

Gantt Chart:

P_1	P_2	P_3	P_4	P_5
0 – 2	2 – 4	4 – 5	5 – 7	7 – 9

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	2	3	0	0	~~2~~ 1	6

At $t = 9$,

Ready Queue: P_6 P_1 P_2 P_5

Gantt chart:

P_1	P_2	P_3	P_4	P_5	P_6
0 – 2	2 – 4	4 – 5	5 – 7	7 – 9	9 – 11

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	2	3	0	0	1	4

At $t = 11$,

Ready Queue: P_1 P_2 P_5 P_6

Gantt chart

P_1	P_2	P_3	P_4	P_5	P_6	P_1
0 – 2	2 – 4	4 – 5	5 – 7	7 – 9	9 – 11	11 – 13

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	0	3	0	0	1	4

At $t = 13$

Ready Queue: P_2 P_5 P_6

Gantt chart:

P_1	P_2	P_3	P_4	P_5	P_6	P_1	P_2
0 – 2	2 – 4	4 – 5	5 – 7	7 – 9	9 – 11	11 – 13	13 – 15

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	0	1	0	0	1	4

At $t = 15$,

Ready Queue: P_5 P_6 P_2

P_1	P_2	P_3	P_4	P_5	P_6	P_1	P_2	P_5
0 – 2	2 – 4	4 – 5	5 – 7	7 – 9	9 – 11	11 – 13	13 – 15	15 – 16

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	0	1	0	0	0	4

At $t = 16$,

Ready Queue: P_6 P_2

P_1	P_2	P_3	P_4	P_5	P_6	P_1	P_2	P_5	P_6
0 – 2	2 – 4	4 – 5	5 – 7	7 – 9	9 – 11	11 – 13	13 – 15	15 – 16	16 – 18

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	0	1	0	0	0	2

N. Sree Dhyuti
CED19I027

At $t=18$,
Ready Queue : P_2 P_6

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	0	0	0	0	0	2

P_1	P_2	P_3	P_4	P_5	P_6	P_1	P_2	P_5	P_6	P_2

0 2 4 5 7 9 11 13 15 16 18 19

At $t=19$,
Ready Queue : P_6

FINAL GANTT CHART :

P_1	P_2	P_3	P_4	P_5	P_6	P_1	P_2	P_5	P_6	P_2	P_6

0 2 4 5 7 9 11 13 15 16 18 19 21

PID	AT	BT	CT	TAT	WT
1	0	4	13	13	9
2	0	5	19	19	14
3	0	1	5	5	4
4	0	2	7	7	5
5	0	3	16	16	13
6	0	6	21	21	15

$$\text{Avg. TAT} = \frac{13+19+5+7+16+21}{6} = 13.500001$$

$$\text{Avg. WT} = \frac{\Sigma WT}{n} = \frac{9+14+4+5+13+15}{6} = 10$$

$$\boxed{\begin{array}{l} \therefore \text{Avg. TAT} = 13.5 \\ \text{Avg. WT} = 10 \end{array}}$$

CODE OUTPUT FOR THE SAME EXAMPLE:

D:\SEM 5\OS\LAB\LAB3\CED19I027_Lab3_Q2.exe

```
Enter Burst Time of Process 2 :5
Enter PID :3
Enter Arrival Time of Process 3 :0
Enter Burst Time of Process 3 :1
Enter PID :4
Enter Arrival Time of Process 4 :0
Enter Burst Time of Process 4 :2
Enter PID :5
Enter Arrival Time of Process 5 :0
Enter Burst Time of Process 5 :3
Enter PID :6
Enter Arrival Time of Process 6 :0
Enter Burst Time of Process 6 :6
Value of Time Quantum : 2
-----------------------------------------------------------------

PID            AT             BT             CT             TAT            WT
1      0.000000      4.000000     13.000000     13.000000     9.000000
2      0.000000      5.000000     19.000000     19.000000     14.000000
3      0.000000      1.000000     5.000000     5.000000     4.000000
4      0.000000      2.000000     7.000000     7.000000     5.000000
5      0.000000      3.000000     16.000000     16.000000     13.000000
6      0.000000      6.000000     21.000000     21.000000     15.000000
-----------------------------------------------------------------

Average Waiting Time : 10.000000
Average Turn Around Time : 13.500001

--------------------------------
Process exited after 33.64 seconds with return value 0
Press any key to continue . . .
```

N. Sree Dhyuti
CED19I027

<u>Example</u>: Schedule the following process using premptive Round Robin Scheduling

PID	AT	BT
1	0	4
2	1	5
3	2	1
4	3	2
5	4	3
6	5	6

Time Quantum = 2

Sol) At time $t=0$

Ready Queue: P_1

Gantt Chart

P_1

0 2

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	2	5	1	2	3	6

At $t=2$,

Ready Queue: $P_2 P_3 P_1$

Gantt Chart:

P_1	P_2

0 2 4

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	2	3	1	2	3	6

At $t=4$,

Ready Queue: $P_3 P_1 P_4 P_5 P_2$

Gantt Chart:

P_1	P_2	P_3

0 2 4 5

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	2	3	~~0~~	2	3	6

At $t=5$,

Ready Queue: P_1 P_4 P_5 P_2 P_6

Gantt Chart:

P_1	P_2	P_3	P_1

0 2 4 5 7

	~~P_1~~	P_2	~~P_3~~	P_4	P_5	P_6
BT left	~~0~~	3	~~0~~	2	3	6

N. Sree Dhyuti
CED19I027

At $t = 7$,

Ready queue: P_4 P_5 P_2 P_6

Gantt Chart:

P_1	P_2	P_3	P_1	P_4

0 2 4 5 7 9

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	0	3	0	0	3	6

At $t = 9$,

Ready Queue: P_5 P_2 P_6

Gantt Chart:

P_1	P_2	P_3	P_1	P_4	P_5

0 2 4 5 7 9 11

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	0	3	0	0	1	6

At $t = 11$,

Ready Queue: P_2 P_6 P_5

Gantt chart:

P_1	P_2	P_3	P_1	P_4	P_5	P_2

0 2 4 5 7 9 11 13

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	0	1	0	0	1	6

At $t = 13$,

Ready Queue: P_6 P_5 P_2

Gantt chart:

P_1	P_2	P_3	P_1	P_4	P_5	P_2	P_6

0 2 4 5 7 9 11 13 15

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	0	1	0	0	1	4

At $t = 15$,

Ready Queue: P_5 P_2 P_6

Gantt Chart:

P_1	P_2	P_3	P_1	P_4	P_5	P_2	P_6	P_5

0 2 4 5 7 9 11 13 15 16

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	0	1	0	0	0	4

At $t = 16$,

Ready Queue: P_2 P_6

P_1	P_2	P_3	P_1	P_4	P_5	P_2	P_6	P_5	P_2

0 2 4 5 7 9 11 13 15 16 17

	P_1	P_2	P_3	P_4	P_5	P_6
BT left	0	0	0	0	0	4

N. Sree Dhyuti
CED19I027

At $t = 17$,

Ready Queue : P_6

FINAL GANTT CHART

P_1	P_2	P_3	P_1	P_4	P_5	P_2	P_6	P_5	P_2	P_6

0 2 4 5 7 9 11 13 15 16 17 21

PID	AT	BT	CT	TAT	WT
1	0	4	7	7	3
2	1	5	17	16	11
3	2	1	5	3	2
4	3	2	9	6	4
5	4	3	16	12	9
6	5	6	21	16	10

$$\text{Avg. TAT} = \frac{\Sigma TAT}{n}$$

$$= \frac{7+16+3+6+12+16}{6}$$

$$= 10$$

$$\text{Avg. WT} = \frac{\Sigma WT}{n}$$

$$= \frac{3+11+2+4+9+10}{6}$$

$$= 6.5$$

$$\boxed{\begin{array}{l} \therefore \text{Avg TAT} = 10 \\ \text{Avg WT} = 6.5 \end{array}}$$

CODE OUTPUT FOR THE SAME EXAMPLE:

D:\SEM 5\OS\LAB\LAB3\CED19I027_Lab3_Q2.exe

```
Enter PID :3
Enter Arrival Time of Process 3 :2
Enter Burst Time of Process 3 :1
Enter PID :4
Enter Arrival Time of Process 4 :3
Enter Burst Time of Process 4 :2
Enter PID :5
Enter Arrival Time of Process 5 :4
Enter Burst Time of Process 5 :3
Enter PID :6
Enter Arrival Time of Process 6 :5
Enter Burst Time of Process 6 :6
Value of Time Quantum : 2
-----------------------------------------------------------------

PID             AT             BT             CT             TAT             WT
1      0.000000      4.000000     7.000000     7.000000     3.000000
2      1.000000      5.000000     17.000000     16.000000     11.000000
3      2.000000      1.000000     5.000000     3.000000     2.000000
4      3.000000      2.000000     9.000000     6.000000     4.000000
5      4.000000      3.000000     16.000000     12.000000     9.000000
6      5.000000      6.000000     21.000000     16.000000     10.000000
-----------------------------------------------------------------

Average Waiting Time : 6.500000
Average Turn Around Time : 10.000000

--------------------------------
Process exited after 19.21 seconds with return value 0
Press any key to continue . . .
```

THE END