PRACTICAL NO : 6

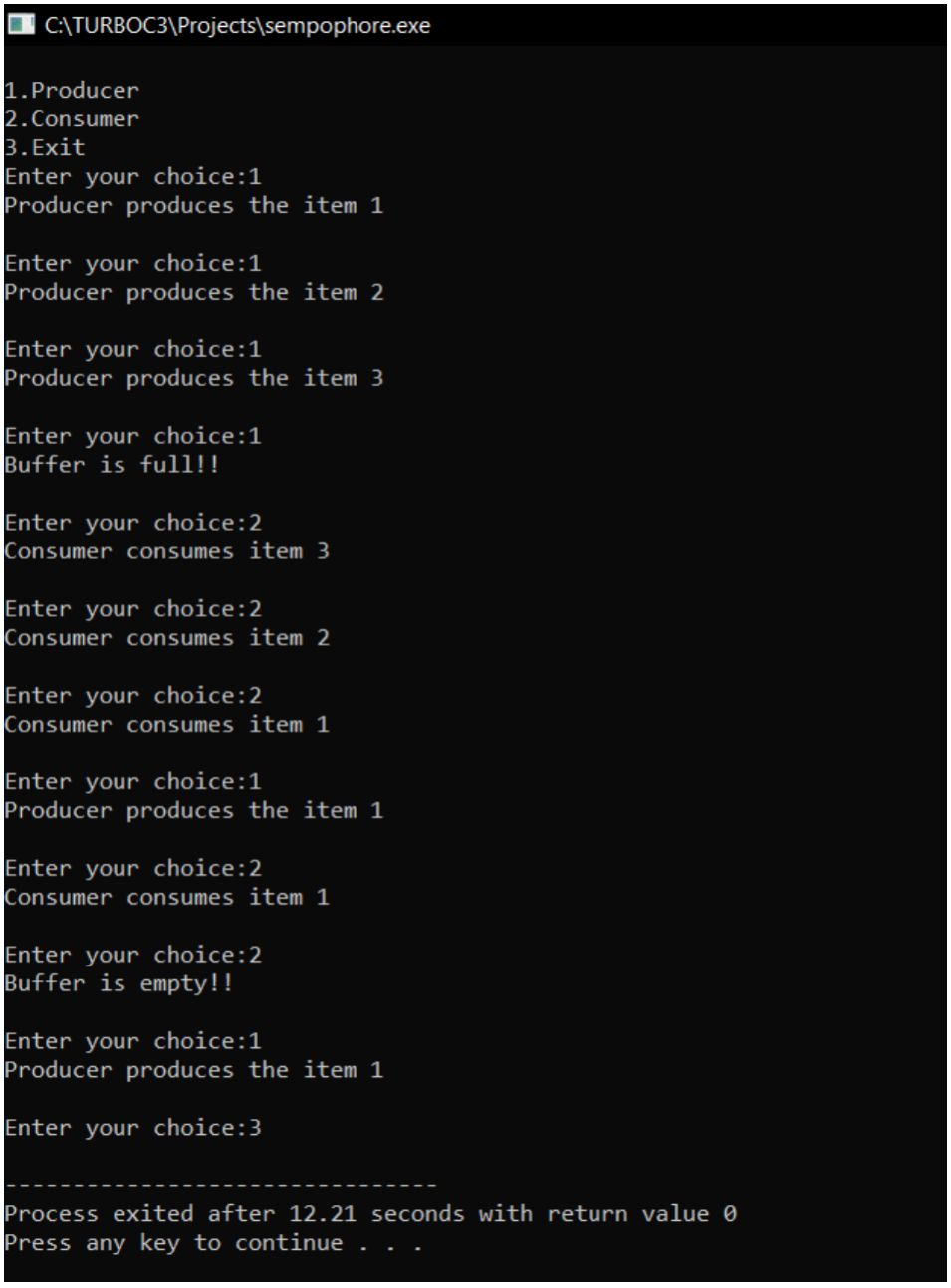
Program:

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n1.Producer\n2.Consumer\n3.Exit"
);
while(1)
{
printf("\nEnter your choice:");
scanf("%d",&n);
switch(n)

{
case 1: if((mutex==1)&&(empty!=0))
producer();
else
printf("Buffer is full!!\n");
break;
case 2: if((mutex==1)&&(full!=0))
consumer();
else
printf("Buffer is empty!!\n");
break;
case 3:
exit(0);
break;
}
}
return 0;
}
int wait(int s)
{
return (--s);
}
int signal(int s)
{
return(++s);
}
void producer()
{
mutex=wait(mutex);
full=signal(full);
empty=wait(empty);
x++;
printf("Producer produces the item
%d\n",x);
mutex=signal(mutex);
}
void consumer()
{
mutex=wait(mutex);
full=wait(full);
empty=signal(empty);

printf("Consumer consumes item
%d\n",x);
x--;
mutex=signal(mutex);
}
```

Output:


C:\TURBOC3\Projects\sempophore.exe
1.Producer
2.Consumer
3.Exit
Enter your choice:1
Producer produces the item 1
Enter your choice:1
Producer produces the item 2
Enter your choice:1
Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:2
Consumer consumes item 3
Enter your choice:2
Consumer consumes item 2
Enter your choice:2
Consumer consumes item 1
Enter your choice:1
Producer produces the item 1
Enter your choice:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1
Producer produces the item 1
Enter your choice:3

Process exited after 12.21 seconds with return value 0
Press any key to continue . . .

PRACTICAL NO : 5

Demonstrate the concept of non-preemptive scheduling algorithms.

1. FCFS

Program:

```
#include <stdio.h>

// Function to Calculate waiting time,
// average waiting time, and average turnaround time
void CalculateTimes(int at[], int bt[], int N)
{
    // Declare the arrays for waiting time,
    // turnaround time, and completion time
    int wt[N], tat[N], ct[N];

    // Calculate completion time for each process
    ct[0] = bt[0];
    for (int i = 1; i < N; i++)
    {
        ct[i] = ct[i - 1] + bt[i];
    }

    // Calculate turnaround time for each process
    for (int i = 0; i < N; i++)
    {
        tat[i] = ct[i] - at[i];
    }

    // Calculate waiting time for each process
    for (int i = 0; i < N; i++)
    {
        wt[i] = tat[i] - bt[i];
    }

    // Print process details
    printf("PN\tAT\tBT\tWT\tTAT\tCT\n\n");
    for (int i = 0; i < N; i++)
    {
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], wt[i], tat[i], ct[i]);
    }

    // Calculate average waiting time and average turnaround time
    float avg_wt = 0.0, avg_tat = 0.0; // Define as float
    for (int i = 0; i < N; i++)
    {
        avg_wt += wt[i];
        avg_tat += tat[i];
    }
    avg_wt /= N;
    avg_tat /= N;
```

```
    // Print average waiting time and average turnaround time
    printf("\nAverage waiting time = %.2f\n", avg_wt);
    printf("Average turnaround time = %.2f\n", avg_tat);
}

// Function to print Gantt Chart
void PrintGanttChart(int at[], int bt[], int N)
{
    printf("\n\nGantt Chart:\n\n");

    // Printing process numbers
    for (int i = 0; i < N; i++)
    {
        printf("| P%d ", i + 1);
    }
    printf("|\n");

    // Printing bars representing processes
    int total_time = 0;
    for (int i = 0; i < N; i++)
    {
        printf("%d   ", total_time);
        total_time += bt[i];
    }
    printf("%d\n\n", total_time);
}

// Driver code
int main()
{
    // Number of process
    int N = 5;

    // Array for Arrival time
    int at[] = {0, 1, 2, 3, 4};

    // Array for Burst Time
    int bt[] = {4, 3, 1, 2, 5};

    // Function call to calculate times
    CalculateTimes(at, bt, N);

    // Function call to print Gantt Chart
    PrintGanttChart(at, bt, N);

    return 0;
}
```

Output:

```
Shaikhs-MacBook-Air:os aveis$ cd "/Users/aveis/Desktop/sem 4/
ac6.c -o os_prac6 && "/Users/aveis/Desktop/sem 4/practicals/o
PN      AT      BT      WT      TAT     CT

1       0       4       0       4       4
2       1       3       3       6       7
3       2       1       5       6       8
4       3       2       5       7       10
5       4       5       6       11      15

Average waiting time = 3.80
Average turnaround time = 6.80


Gantt Chart:

| P1 | P2 | P3 | P4 | P5 |
0   4   7   8   10   15
```

2. SJF

Program:

```
#include <stdio.h>

// Function to Calculate waiting time,

// average waiting time, and average turnaround time

void CalculateTimes(int at[], int bt[], int N)

{

   // Declare the arrays for waiting time,

   // turnaround time, and completion time

   int wt[N], tat[N], ct[N], bt_copy[N], process[N];


   // Copy burst times to maintain the original array

   for (int i = 0; i < N; i++){

      bt_copy[i] = bt[i];

      process[i] = i + 1;

   }

   // Sort processes based on burst time

   for (int i = 0; i < N - 1; i++){

      for (int j = 0; j < N - i - 1; j++){

         if (bt_copy[j] > bt_copy[j + 1])

         {

            int temp = bt_copy[j];

            bt_copy[j] = bt_copy[j + 1];

            bt_copy[j + 1] = temp;


            temp = process[j];

            process[j] = process[j + 1];

            process[j + 1] = temp;

         }

      }

   }

}


// Calculate completion time for each process

ct[0] = bt_copy[0];

for (int i = 1; i < N; i++){

   ct[i] = ct[i - 1] + bt_copy[i];

}
```

```
    // Calculate turnaround time for each process
    for (int i = 0; i < N; i++){
        tat[i] = ct[i] - at[process[i] - 1];
    }

    // Calculate waiting time for each process
    for (int i = 0; i < N; i++){
        wt[i] = tat[i] - bt[process[i] - 1];
    }
    // Print process details
    printf("PN\tAT\tBT\tWT\tTAT\tCT\n\n");
    for (int i = 0; i < N; i++){
        printf("%d\t%d\t%d\t%d\t%d\t%d\n",
process[i], at[process[i] - 1], bt[process[i] - 1],
wt[i], tat[i], ct[i]);
    }
    // Calculate average waiting time and average
turnaround time
    float avg_wt = 0.0, avg_tat = 0.0;
    for (int i = 0; i < N; i++){
        avg_wt += wt[i];
        avg_tat += tat[i];
    }
    avg_wt /= N;
    avg_tat /= N;

    // Print average waiting time and average
turnaround time
    printf("\nAverage waiting time = %.2f\n",
avg_wt);
    printf("Average turnaround time = %.2f\n",
avg_tat);
}

// Function to print Gantt Chart
void PrintGanttChart(int at[], int bt[], int N){
    printf("\n\nGantt Chart:\n\n");
    // Printing process numbers
    for (int i = 0; i < N; i++)
    {
        printf("| P%d ", i + 1);
    }
    printf("|\n");

    // Printing bars representing processes
    int total_time = 0;
    for (int i = 0; i < N; i++)
    {
        printf("%d   ", total_time);
        total_time += bt[i];
    }
    printf("%d\n\n", total_time);
}
int main(){
    // Number of process
    int N = 5;

    // Array for Arrival time
    int at[] = {0, 1, 2, 3, 4};

    // Array for Burst Time
    int bt[] = {4, 3, 1, 2, 5};

    // Function call to calculate times
    CalculateTimes(at, bt, N);

    // Function call to print Gantt Chart
    PrintGanttChart(at, bt, N);
    return 0;
}
```

Output:

```
Shaikhs-MacBook-Air:os aveis$ cd "/Users/aveis/Desktop/sem 4/
 -o sjf && "/Users/aveis/Desktop/sem 4/practicals/os/"sjf
PN      AT      BT      WT      TAT     CT

2       1       1       -1      0       1
4       3       2       -2      0       3
5       4       5       -1      4       8
1       0       6       8       14      14
3       2       8       12      20      22

Average waiting time = 3.20
Average turnaround time = 7.60


Gantt Chart:

| P1 | P2 | P3 | P4 | P5 |
0   6   7   15   17   22
```

3. Priority Scheduling

Program:

```
#include <stdio.h>

// Function to Calculate waiting time,

// average waiting time, and average turnaround time

void CalculateTimes(int at[], int bt[], int priority[], int N)

{

   // Declare the arrays for waiting time,

   // turnaround time, and completion time

   int wt[N], tat[N], ct[N], bt_copy[N], process[N];


   // Copy burst times to maintain the original array

for (int i = 0; i < N; i++)

   {

      bt_copy[i] = bt[i];

      process[i] = i + 1;

   }


   // Sort processes based on priority

   for (int i = 0; i < N - 1; i++)

   {

      for (int j = 0; j < N - i - 1; j++)

      {

         if (priority[j] > priority[j + 1] || (priority[j] == priority[j + 1] && at[j] > at[j + 1]))

         {

            int temp = bt_copy[j];
```

```
            bt_copy[j] = bt_copy[j + 1];
            bt_copy[j + 1] = temp;

            temp = process[j];
            process[j] = process[j + 1];
            process[j + 1] = temp;

            temp = at[j];
            at[j] = at[j + 1];
            at[j + 1] = temp;

            temp = priority[j];
            priority[j] = priority[j + 1];
            priority[j + 1] = temp;
        }
    }
}

    // Calculate completion time for each process
    ct[0] = bt_copy[0];
    for (int i = 1; i < N; i++)
    {
        ct[i] = ct[i - 1] + bt_copy[i];
    }

    // Calculate turnaround time for each process
    for (int i = 0; i < N; i++)
    {
        tat[i] = ct[i] - at[process[i] - 1];
    }

    // Calculate waiting time for each process
    for (int i = 0; i < N; i++)
    {
        wt[i] = tat[i] - bt[process[i] - 1];
    }

    // Print process details

printf("PN\tAT\tBT\tPriority\tWT\tTAT\tCT\n\n");
    for (int i = 0; i < N; i++)
    {

printf("%d\t%d\t%d\t%d\t\t%d\t%d\t%d\n", process[i], at[process[i] - 1], bt[process[i] - 1], priority[i], wt[i], tat[i], ct[i]);
    }

    // Calculate average waiting time and average turnaround time
    float avg_wt = 0.0, avg_tat = 0.0;
    for (int i = 0; i < N; i++)
    {
        avg_wt += wt[i];
        avg_tat += tat[i];
    }
    avg_wt /= N;
    avg_tat /= N;
```

```
    // Print average waiting time and average turnaround time
    printf("\nAverage waiting time = %.2f\n", avg_wt);
    printf("Average turnaround time = %.2f\n", avg_tat);
}

// Function to print Gantt Chart
void PrintGanttChart(int at[], int bt[], int N)
{
    printf("\n\nGantt Chart:\n\n");

    // Printing process numbers
    for (int i = 0; i < N; i++)
    {
        printf("| P%d ", i + 1);
    }
    printf("|\n");

    // Printing bars representing processes
    int total_time = 0;
    for (int i = 0; i < N; i++)
    {
        printf("%d   ", total_time);
        total_time += bt[i];
    }
    printf("%d\n\n", total_time);
}

// Driver code
int main()
{
    // Number of process
    int N = 5;

    // Array for Arrival time
    int at[] = {0, 1, 2, 3, 4};

    // Array for Burst Time
    int bt[] = {4, 3, 1, 2, 5};

    // Array for Priority
    int priority[] = {2, 1, 3, 4, 5}; // Lower number means higher priority

    // Function call to calculate times
    CalculateTimes(at, bt, priority, N);

    // Function call to print Gantt Chart
    PrintGanttChart(at, bt, N);

    return 0;
}
```

Output:

```
Shaikhs-MacBook-Air:os aveis$ cd "/Users/aveis/Desktop/sem 4/practicals/o
ity.c -o priority && "/Users/aveis/Desktop/sem 4/practicals/os/"priority
PN      AT      BT      Priority        WT      TAT     CT

2       0       3       1               0       3       3
1       1       10      2               2       12      13
3       2       8       3               11      19      21
5       3       5       4               18      23      26
4       4       2       5               22      24      28

Average waiting time = 10.60
Average turnaround time = 16.20


Gantt Chart:

| P1 | P2 | P3 | P4 | P5 |
0   10   13   21   23   28
```

4. Round Robin

Program:

```
#include <stdio.h>

// Function to Calculate waiting time,

// average waiting time, and average turnaround time

void CalculateTimes(int at[], int bt[], int quantum, int N)

{

   // Declare the arrays for waiting time,

   // turnaround time, and completion time

   int wt[N], tat[N], ct[N], remaining_bt[N];


   // Initialize remaining burst time

   for (int i = 0; i < N; i++)

   {

      remaining_bt[i] = bt[i];

   }

   // Initialize time and current completion time

   int time = 0, current_ct = 0;


   // Process until all processes are done

   while (1)

   {

      int done = 1; // Assume all processes are done initially


      // Traverse all processes

      for (int i = 0; i < N; i++)

      {

         // If burst time is remaining for this process
```

```
            if (remaining_bt[i] > 0)
            {
                done = 0; // There are still processes remaining

                // Execute this process for a quantum or remaining time, whichever is smaller
                if (remaining_bt[i] > quantum)
                {
                    // Update time and remaining burst time
                    time += quantum;
                    remaining_bt[i] -= quantum;
                }
                else
                {
                    // Update time and remaining burst time
                    time += remaining_bt[i];
                    remaining_bt[i] = 0;

                    // Calculate completion time for this process
                    ct[i] = time;
                }
            }
        }

        // If all processes are done, exit the loop
        if (done == 1)
            break;
    }

    // Calculate turnaround time for each process
    for (int i = 0; i < N; i++)
    {
        tat[i] = ct[i] - at[i];
    }

    // Calculate waiting time for each process
    for (int i = 0; i < N; i++)
    {
        wt[i] = tat[i] - bt[i];
    }

    // Print process details
    printf("PN\tAT\tBT\tWT\tTAT\tCT\n\n");
    for (int i = 0; i < N; i++)
    {
        printf("%d\t%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], wt[i], tat[i], ct[i]);
    }

    // Calculate average waiting time and average turnaround time
    float avg_wt = 0.0, avg_tat = 0.0;
    for (int i = 0; i < N; i++)
```

```
    {
        avg_wt += wt[i];
        avg_tat += tat[i];
    }
    avg_wt /= N;
    avg_tat /= N;

    // Print average waiting time and average turnaround time
    printf("\nAverage waiting time = %.2f\n", avg_wt);
    printf("Average turnaround time = %.2f\n", avg_tat);
}
int main()
{
    // Number of process
    int N = 5;

    // Array for Arrival time
    int at[] = {0, 1, 2, 3, 4};

    // Array for Burst Time
    int bt[] = {4, 3, 1, 2, 5};

    // Time quantum
    int quantum = 2;

    // Function call to calculate times
    CalculateTimes(at, bt, quantum, N);
    return 0;
}
```

Output:

```
PN      AT      BT      WT      TAT     CT

1       0       4       7       11      11
2       1       3       8       11      12
3       2       1       2       3       5
4       3       2       2       4       7
5       4       5       6       11      15


Gantt Chart:

______________
|P1P1P1P1P1P1P1P1P1P1    | P2P2P2P2P2P2P2P2P2P2   |  P3P3P3        |   P4P4P4P4       |    P5P5P5P5P5P5P5P5P5P5|
______________
```