|  |
| --- |
| Student Name: **Vijay Guttula**  Student ID: **11804064 (A27)**  Email Address: **ssvijayg@gmail.com**  GitHub Link: **https://github.com/vijay-guttula/OS\_CA** |

Problem:

**Ques2**. Considering the arrival time and burst time requirement of the process the scheduler schedules the processes by interrupting the processor after every 6 units of time and does consider the completion of the process in this iteration. The scheduler than checks for the number of process waiting for the processor and allots the processor to the process but interrupting the processor every 10 unit of time and considers the completion of the processes in this iteration. The scheduler checks the number of processes waiting in the queue for the processor after the second iteration and gives the processor to the process which needs more time to complete than the other processes to go in the terminated state. The inputs for the number of requirements, arrival time and burst time should be provided by the user.

Consider the following units for reference.

|  |  |  |
| --- | --- | --- |
| Process | Arrival time | Burst time |
| P1 | **0** | **20** |
| P2 | **5** | **36** |
| P3 | **13** | **19** |
| P4 | **26** | **42** |

Develop a scheduler which submits the processes to the processor in the defined scenario and compute the scheduler performance by providing the waiting time for process, turnaround time for process and average waiting time and turnaround time.

**Ans:** We can solve this question by using algorithm of Operating System:

1. Round Robin Scheduling Algorithm: for fixed time slice i.e for X unit and Y unit of time.

Algorithm:

time\_req = 0;

// Add time for process on left of p

// (Scheduled before p in a round of

// 1 unit time slice)

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

{

if (arr[i] < arr[p])

time\_req += arr[i];

else

time\_req += arr[p];

}

// step 2 : Add time of process p

time\_req += arr[p];

// Add time for process on right

// of p (Scheduled after p in

// a round of 1 unit time slice)

for (int i=p+1; i<n; i++)

{

if (arr[i] < arr[p])

time\_req += arr[i];

else

time\_req += arr[p]-1;

}

.

Description:

To implement the above problem, we must make three iterations in which first iteration with a time slice of X units reduce the burst time of each process by X. In second iteration ------- --- --- ------ ----- ---- waiting time of each process.

Code:

#include<stdio.h>

#include<conio.h>

void rr(int no,int remt[10],int Cur\_t,int arT[10], int bsT[10]);

int main()

{

    int P\_no, j, no, CurT, RemProc, indicator, time\_quan, wait, tut, arT[10], bsT[10], remt[10], x=1 ;

 indicator = 0;

    wait = 0;

    tut = 0;

    printf("Enter number of processes ");

    scanf("%d",&no);

    RemProc = no;

    printf("\nEnter the arrival time and burst time of the processes\n");

    for(P\_no = 0; P\_no < no; P\_no++)

    {

        printf("\nProcess P%d\n", P\_no + 1);

        printf("Arrival time = ");

        scanf("%d", &arT[P\_no]);

        printf("Burst time = ");

        scanf("%d",&bsT[P\_no]);

        remt[P\_no] = bsT[P\_no];

    }

    printf("The details of time quantum are as follows:\n");

    printf("The time quantum for first round is 6.\n");

    time\_quan = 6;

    CurT = 0;

    for(P\_no = 0; RemProc != 0 ;)

    {

        if(remt[P\_no] <= time\_quan && remt[P\_no] >0 )

        {

            CurT += remt[P\_no];

            remt[P\_no] = 0;

            indicator = 1;

        }

        else if(remt[P\_no] > 0)

        {

            remt[P\_no] -= time\_quan;

            CurT += time\_quan;

}

if(remt[P\_no] == 0 && indicator == 1)

 { printf("%d",P\_no);

            RemProc-- ;

            printf("P %d", P\_no + 1);

            printf("\t\t\t%d", CurT - arT[P\_no]);

            printf("\t\t\t%d\n", CurT - bsT[P\_no] - arT[P\_no]);

            wait += CurT - arT[P\_no] - bsT[P\_no];

            tut += CurT - arT[P\_no];

            indicator = 0;

        }

        if(P\_no == no - 1)

        {

            x++ ;

            if(x == 2)

  {

                P\_no = 0;

                time\_quan = 10;

                printf("The time quantum for second round is 10. \n");

            }

            else

            {

                break;

            }

        }

        else if(CurT >= arT[P\_no + 1])

        {

            P\_no++ ;

        }

        else

        {

            P\_no = 0 ;

        }

    }

    rr(no, remt, CurT, arT, bsT);

    return 0;

}

void rr(int no, int remt[10], int Cur\_t, int arT[10], int bsT[10])

{

    float avg\_wait, avg\_tut;

    int i , j, n=no, temp, btime[20], P\_no[20], w\_time[20], tut\_t[20], total=0,

loc;

printf("Third round with least burst time.\n");

for(i = 0; i < n; i++)

    {btime[i] = remt[i];

w\_time[i] = Cur\_t - arT[i] - btime[i];

        P\_no[i] = i + 1;

    }

    for(i=0;i<n;i++)

    {

        loc = i;

        for(j = i + 1; j < n; j++)

        {

            if(btime[j] < btime[loc])

 {

loc = j;

            }

        }

        temp = btime[i];

        btime[i] = btime[loc];

        btime[loc] = temp;

        temp = P\_no[i];

        P\_no[i] = P\_no[loc];

        P\_no[loc] = temp;

    }

    for(i = 1; i < n; i++)

    {

        for(j = 0; j < i ;j++){

            w\_time[i] += btime[j];

        }

        total += w\_time[i];

    }

    avg\_wait = (float)total / n;

    total = 0;

    printf("\nProcess\t\tBurst time\t\twaiting time\t\tTurnaround Time");

    for(i = 0; i < n; i++)

    {

        tut\_t[i] = btime[i] + w\_time[i];

        total = total + tut\_t[i];

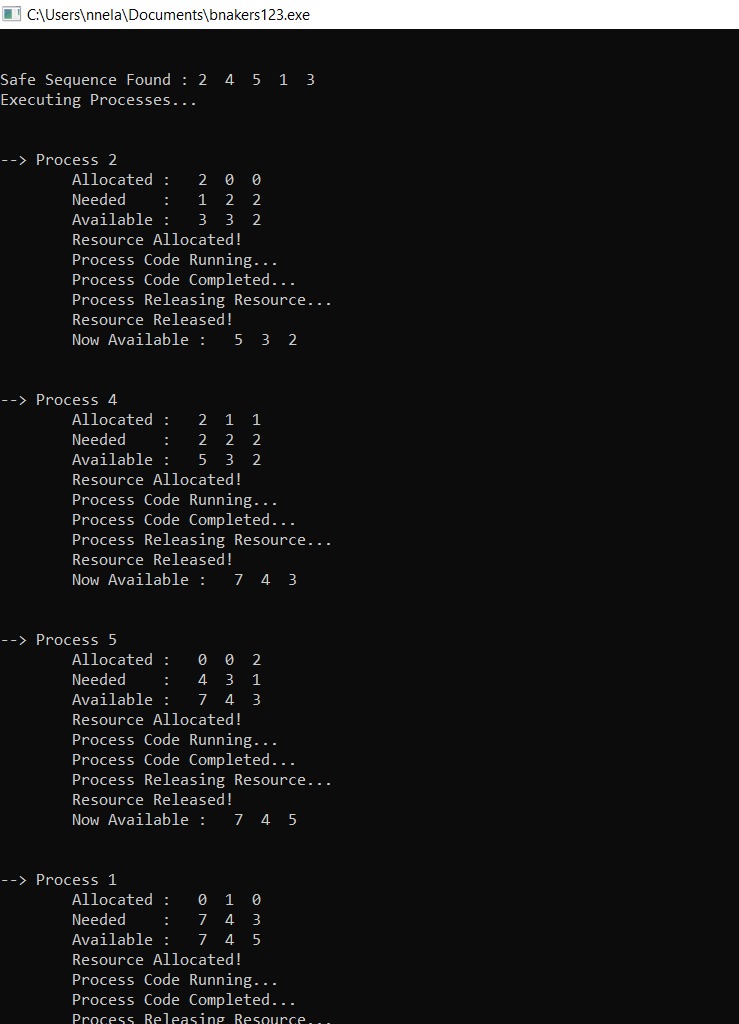
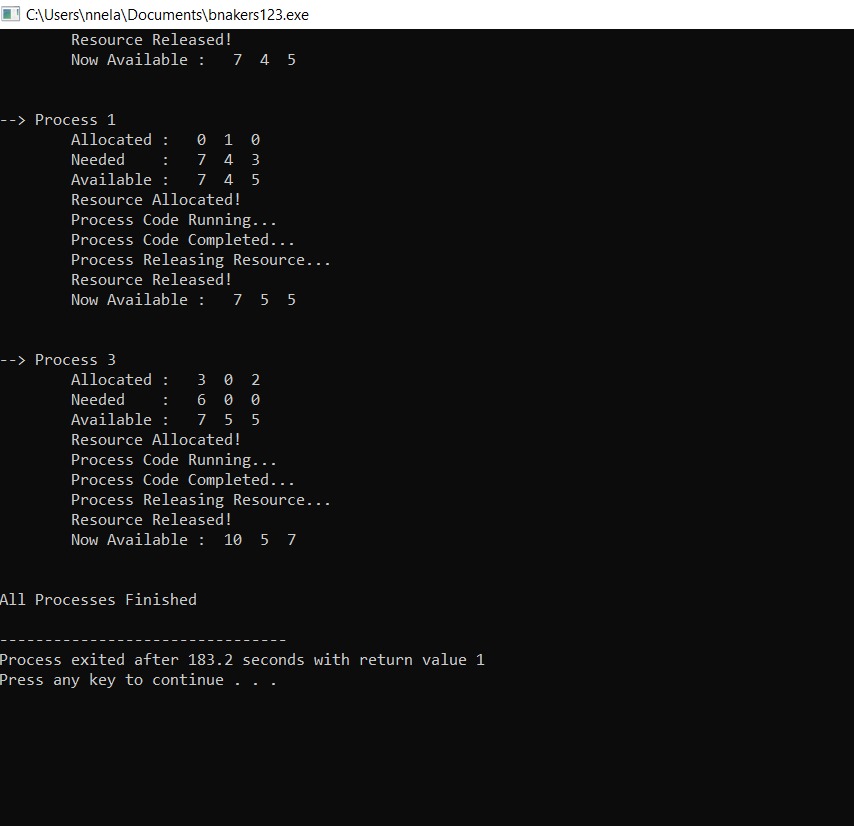
        printf("\nP%d\t\t\t%d\t\t\t%d\t\t\t%d", P\_no[i], btime[i], w\_time[i], tut\_t[i]);

    }

    avg\_tut = (float)total / n;

 printf("\n\nAverage waiting time = %f", avg\_wait);

 printf("\n Average turnaround time = %f\n", avg\_tut);

****