OS INLAB EXAM
20th OCTOBER 2020

By

Devdatt Golwala

19BCE0551

Q1

AIM: To implement best fit strategy with fragmentation calculations

ALGORITHM

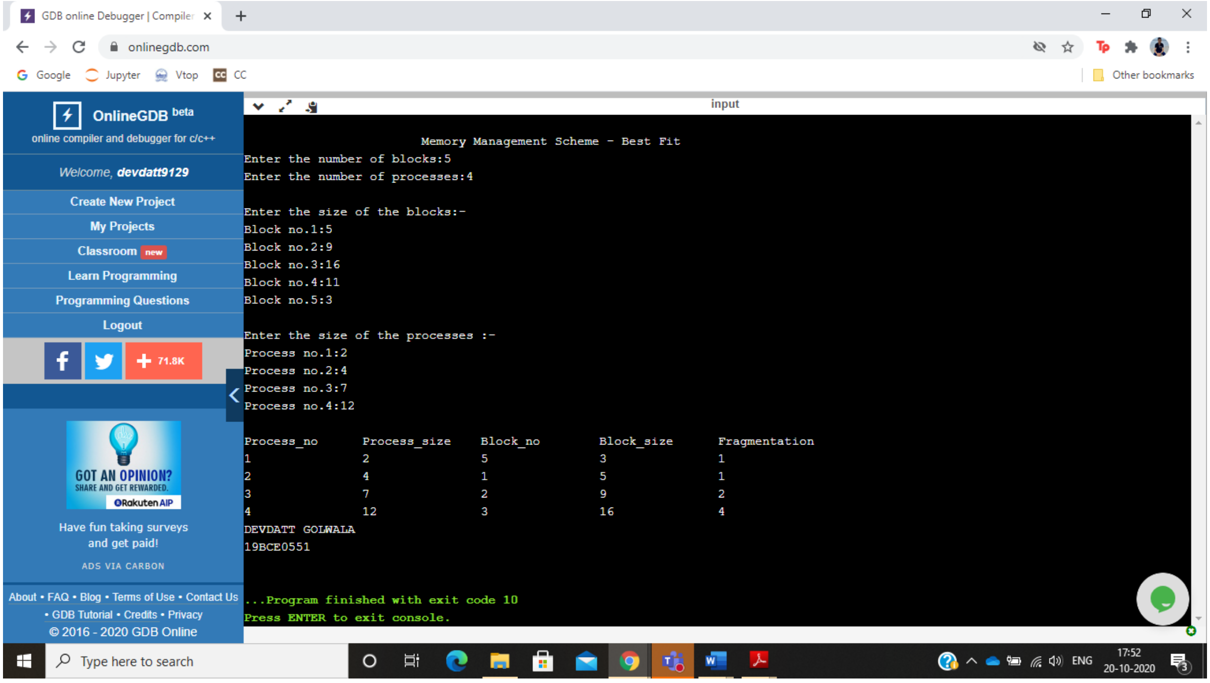
1) Enter the memory blocks with size.
2) Enter the process blocks with size.
3) Set all the memory blocks as free.
4) Start by picking up each process.
5) Find the minimum block size that is best to assign to the current process.
6) If the best fit memory size is found, it is allocated to the process.
7) If the memory block and memory demand do not match, leave the process and search for another process.

CODE

```
#include<stdio.h>

void main()
{
        int fragment[20],b[20],p[20],i,j,nb,np,temp,lowest=9999;
        static int barray[20],parray[20];

        printf("\n\t\t\tMemory Management Scheme - Best Fit");
        printf("\nEnter the number of blocks:");
        scanf("%d",&nb);
        printf("Enter the number of processes:");
        scanf("%d",&np);

        printf("\nEnter the size of the blocks:-\n");
        for(i=1;i<=nb;i++)
  {
                printf("Block no.%d:",i);
     scanf("%d",&b[i]);
  }

        printf("\nEnter the size of the processes :-\n");
        for(i=1;i<=np;i++)
  {
     printf("Process no.%d:",i);
     scanf("%d",&p[i]);
  }

        for(i=1;i<=np;i++)
        {
```

```
        for(j=1;j<=nb;j++)
        {
            if(barray[j]!=1)
            {
                temp=b[j]-p[i];
                if(temp>=0)
                    if(lowest>temp)
                    {
                        parray[i]=j;
                        lowest=temp;
                    }
            }
        }

        fragment[i]=lowest;
        barray[parray[i]]=1;
        lowest=10000;
    }

    printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
    for(i=1;i<=np && parray[i]!=0;i++)

    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],parray[i],b[parray[i]],fragment[i]);
}
```

OUTPUT



```
                         Memory Management Scheme - Best Fit
Enter the number of blocks:5
Enter the number of processes:4

Enter the size of the blocks:-
Block no.1:5
Block no.2:9
Block no.3:16
Block no.4:11
Block no.5:3

Enter the size of the processes :-
Process no.1:2
Process no.2:4
Process no.3:7
Process no.4:12

Process_no      Process_size    Block_no        Block_size      Fragmentation
1               2               5               3               1
2               4               1               5               1
3               7               2               9               2
4               12              3               16              4
DEVDATT GOLWALA
19BCE0551


...Program finished with exit code 10
Press ENTER to exit console.
```

Q2

AIM

Consider three processes, all arriving at 2, 6 and 9 time units with total execution time of 5, 10 and 3 units, respectively with the priority of 1, 2 and 0. Develop an algorithm and write code to find the average turnaround time, average waiting time and number of context switches.

I will be showing both preemptive and non preemptive approaches to this problem

First the preemptive approach

PREEMPTIVE

ALGORITHM

Basically we are implementing priority scheduling is this question

```
Completed=0
Current_time=0
While (completed ! = n){
    Find the process with maximum priority time among process that are in ready queue at current_time
    If(process found){
        If (process is getting cpu for the first time) {
            Start_time = current_time
        }
        Burst_time=burst_time-1
        Current_time=current_time+1
        If(burst_time==0) {
            Completion_time= current_time
            Turnaround_time=completion_time-arrival_time
            Waiting_time= turnaround_time – burst_time
            Response_time=start_time – arrival_time
            Mark process as completes
            Completed+
        }
    }
    Else {
        Current_time ++
    }
}
```

CODE

```
#include <iostream>

#include <algorithm>

#include <iomanip>

#include <string.h>
```

```
using namespace std;
struct process {
int pid;
int arrival_time;
int burst_time;
int priority;
int start_time;
int completion_time;
int turnaround_time;
int waiting_time;
int response_time;
};
int main() {
int n;
struct process p[100];
float avg_turnaround_time;
float avg_waiting_time;
float avg_response_time;
float cpu_utilisation;
int total_turnaround_time = 0;
int total_waiting_time = 0;
int total_response_time = 0;
int total_idle_time = 0;
float throughput;
int burst_remaining[100];
int is_completed[100];
memset(is_completed,0,sizeof(is_completed));
cout << setprecision(2) << fixed;
cout<<"Enter the number of processes: ";
```

```
cin>>n;
for(int i = 0; i < n; i++) {
cout<<"Enter arrival time of process "<<i+1<<": ";
cin>>p[i].arrival_time;
cout<<"Enter burst time of process "<<i+1<<": ";
cin>>p[i].burst_time;
cout<<"Enter priority of the process "<<i+1<<": ";
cin>>p[i].priority;
p[i].pid = i+1;
burst_remaining[i] = p[i].burst_time;
cout<<endl;
}
int current_time = 0;
int completed = 0;
int prev = 0;
while(completed != n) {
int idx = -1;
int mx = -1;
for(int i = 0; i < n; i++) {
if(p[i].arrival_time <= current_time && is_completed[i] == 0) {
if(p[i].priority > mx) {
mx = p[i].priority;
idx = i;
}
if(p[i].priority == mx) {
if(p[i].arrival_time < p[idx].arrival_time) {
mx = p[i].priority;
idx = i;
}
}
}
}
if(idx != -1) {
if(burst_remaining[idx] == p[idx].burst_time) {
p[idx].start_time = current_time;
total_idle_time += p[idx].start_time - prev;
}
burst_remaining[idx] -= 1;
current_time++;
prev = current_time;
if(burst_remaining[idx] == 0) {
p[idx].completion_time = current_time;
p[idx].turnaround_time = p[idx].completion_time - p[idx].arrival_time;
p[idx].waiting_time = p[idx].turnaround_time - p[idx].burst_time;
p[idx].response_time = p[idx].start_time - p[idx].arrival_time;
total_turnaround_time += p[idx].turnaround_time;
total_waiting_time += p[idx].waiting_time;
total_response_time += p[idx].response_time;
is_completed[idx] = 1;
completed++;

}
}
else {
current_time++;
}
}
int min_arrival_time = 10000000;
int max_completion_time = -1;
for(int i = 0; i < n; i++) {
min_arrival_time = min(min_arrival_time,p[i].arrival_time);
max_completion_time = max(max_completion_time,p[i].completion_time);
}
avg_turnaround_time = (float) total_turnaround_time / n;
avg_waiting_time = (float) total_waiting_time / n;
avg_response_time = (float) total_response_time / n;
cpu_utilisation = ((max_completion_time - total_idle_time) / (float) max_completion_time )*100;
throughput = float(n) / (max_completion_time - min_arrival_time);
cout<<endl<<endl;
cout<<"#P\t"<<"AT\t"<<"BT\t"<<"PRI\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n" <<endl;
for(int i = 0; i < n; i++) {
cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<p[i].priority<<"\t"<<p[i].start_time<<"\t"<<p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time<<"\t"<<p[i].response_time<<"\t"<<"\n"<<endl;
}
cout<<"Average Turnaround Time = "<<avg_turnaround_time<<endl;
cout<<"Average Waiting Time = "<<avg_waiting_time<<endl;
cout<<"Average Response Time = "<<avg_response_time<<endl;
cout<<"CPU Utilization = "<<cpu_utilisation<<"%"<<endl;
cout<<"Throughput = "<<throughput<<" process/unit time"<<endl;
cout<<"DEVDATT GOLWALA"<<endl; cout<<"19BCE0551"<<endl;
}
```

OUTPUT

Here I have taken highest priority as 1 and lowest priority as 3(instead of 0 and 2)

AT - Arrival Time of the process

BT - Burst time of the process

ST - Start time of the process

CT - Completion time of the process

TAT - Turnaround time of the process

WT - Waiting time of the process

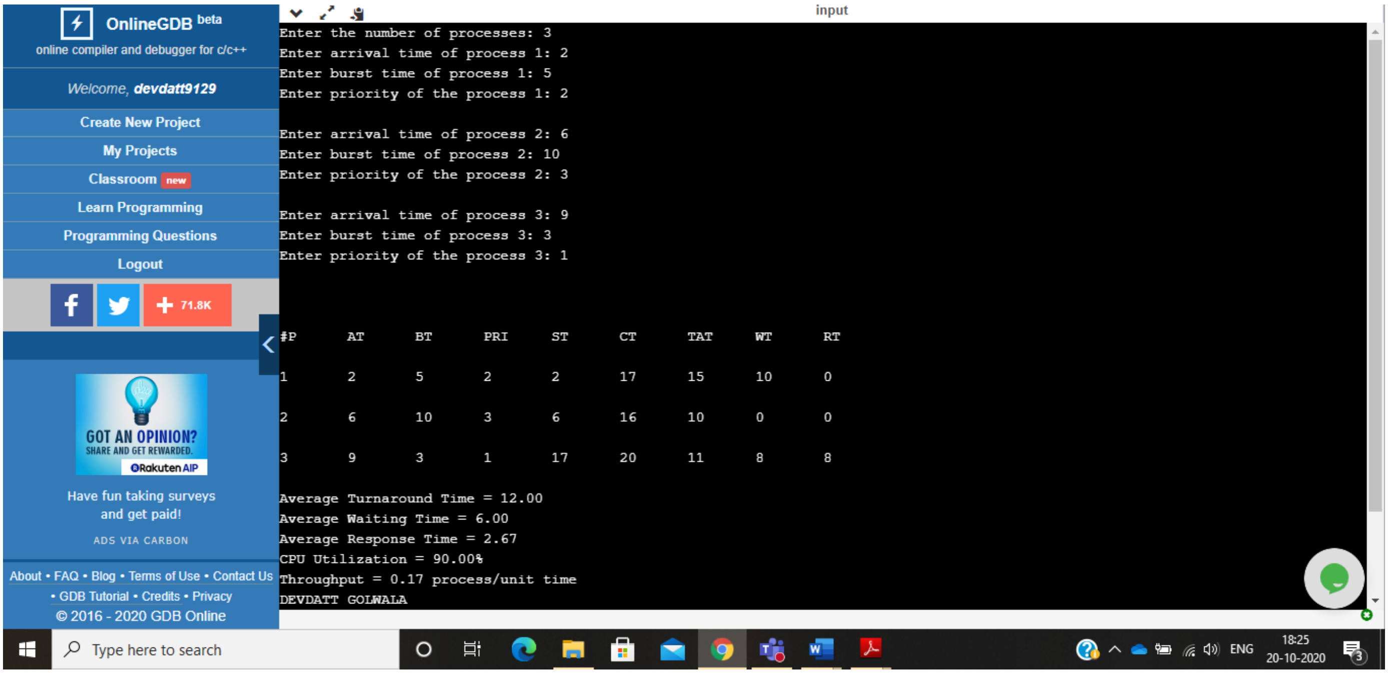
RT - Response time of the process

Formulas used: TAT = CT – AT

WT = TAT – BT

RT = ST - AT

`



```
Enter the number of processes: 3
Enter arrival time of process 1: 2
Enter burst time of process 1: 5
Enter priority of the process 1: 2

Enter arrival time of process 2: 6
Enter burst time of process 2: 10
Enter priority of the process 2: 3

Enter arrival time of process 3: 9
Enter burst time of process 3: 3
Enter priority of the process 3: 1



#P      AT      BT      PRI     ST      CT      TAT     WT      RT

1       2       5       2       2       17      15      10      0

2       6       10      3       6       16      10      0       0

3       9       3       1       17      20      11      8       8

Average Turnaround Time = 12.00
Average Waiting Time = 6.00
Average Response Time = 2.67
CPU Utilization = 90.00%
Throughput = 0.17 process/unit time
DEVDATT GOLWALA
```

NON PREEMPTIVE

ALGORITHM

```
/*1. sort the processes according to arrival time
2. if arrival time is same the acc to priority
3. apply fcfs
*/
```

CODE

```
#include <bits/stdc++.h>

using namespace std;

#define totalprocess 5

// Making a struct to hold the given input

struct process
{
int at,bt,pr,pno;
};

process proc[50];

/*
Writing comparator function to sort according to priority if
arrival time is same
*/

bool comp(process a,process b)
{
if(a.at == b.at)
{
```

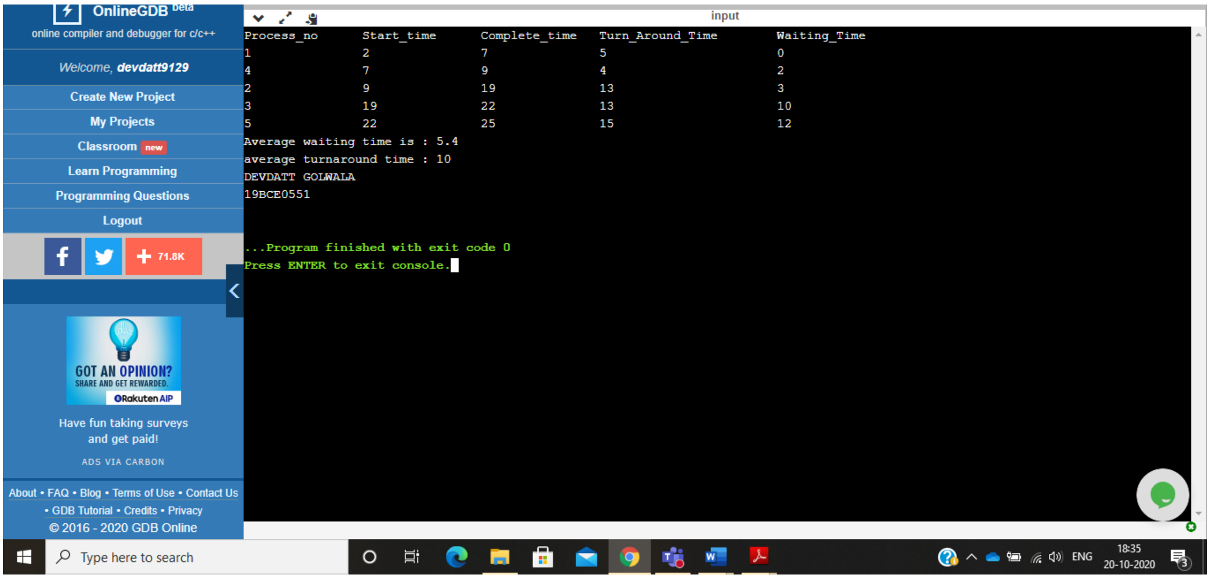
```
return a.pr<b.pr;
}
else
{
        return a.at<b.at;
}
}

// Using FCFS Algorithm to find Waiting time
void get_wt_time(int wt[])
{
// declaring service array that stores cumulative burst time
int service[50];

// Initilising initial elements of the arrays
service[0] = proc[0].at;
wt[0]=0;



for(int i=1;i<totalprocess;i++)
{
service[i]=proc[i-1].bt+service[i-1];

wt[i]=service[i]-proc[i].at;

// If waiting time is negative, change it into zero

        if(wt[i]<0)
        {
        wt[i]=0;
        }
```

```
}


}


void get_tat_time(int tat[],int wt[])
{
// Filling turnaroundtime array

for(int i=0;i<totalprocess;i++)
{
        tat[i]=proc[i].bt+wt[i];
}


}


void findgc()
{
//Declare waiting time and turnaround time array
int wt[50],tat[50];

double wavg=0,tavg=0;

// Function call to find waiting time array
get_wt_time(wt);
//Function call to find turnaround time
get_tat_time(tat,wt);

int stime[50],ctime[50];

stime[0] = proc[0].at;
ctime[0]=stime[0]+tat[0];
```

```
// calculating starting and ending time
for(int i=1;i<totalprocess;i++)
        {
                stime[i]=ctime[i-1];
                ctime[i]=stime[i]+tat[i]-wt[i];
        }

cout<<"Process_no\tStart_time\tComplete_time\tTurn_Around_Time\tWaiting_Time"<<endl;

        // display the process details

for(int i=0;i<totalprocess;i++)
        {
                wavg += wt[i];
                tavg += tat[i];

                cout<<proc[i].pno<<"\t\t"<<
                        stime[i]<<"\t\t"<<ctime[i]<<"\t\t"<<
                        tat[i]<<"\t\t\t"<<wt[i]<<endl;
        }

                // display the average waiting time
                //and average turn around time

        cout<<"Average waiting time is : ";
        cout<<wavg/(float)totalprocess<<endl;
        cout<<"average turnaround time : ";
        cout<<tavg/(float)totalprocess<<endl;
        cout<<"DEVDATT GOLWALA"<<endl;
        cout<<"19BCE0551"<< endl;
```

```
}

int main()
{
int arrivaltime[] = { 2,6,9 };
int bursttime[] = { 5,10,3 };
int priority[] = { 2,3,1 };

for(int i=0;i<totalprocess;i++)
{
        proc[i].at=arrivaltime[i];
        proc[i].bt=bursttime[i];
        proc[i].pr=priority[i];
        proc[i].pno=i+1;
        }

        //Using inbuilt sort function

        sort(proc,proc+totalprocess,comp);

        //Calling function findgc for finding Gantt Chart

        findgc();

        return 0;
}
```

OUTPUT



```
Process_no      Start_time      Complete_time   Turn_Around_Time        Waiting_Time
1               2               7               5                       0
4               7               9               4                       2
2               9               19              13                      3
3               19              22              13                      10
5               22              25              15                      12
Average waiting time is : 5.4
average turnaround time : 10
DEVDATT GOLWALA
19BCE0551


...Program finished with exit code 0
Press ENTER to exit console.
```