CS306
Operating System
Project Report 2

Discussion on CPU Scheduling

Presented by:
Anuraag Singh
2015043
Paras Rastogi
2015175

Submitted to:
Dr. M. K Bajpai

TABLE OF CONTENTS

Implementation of Paper 1

(IJACSA) International Journal of Advanced Computer Science and Applications,
Vol.7, No.1, 2016

Link:
https://thesai.org/Downloads/Volume7No1/Paper_30-Performance_Analysis_of_CPU_Scheduling_Algorithms.pdf

Performance Analysis of CPU Scheduling Algorithms with Novel OMDRRS Algorithm

By-Dr. R. B. Garg, Ex-Professor Delhi University, India
Neetu Goel, Research Scholar Department of Computer Science,TMU,India

The Code implementation and the input and output file has been enclosed with the attachment. Here are some details about it.

To implement the OMDRRS Algorithm we have written the following code using C++ Standard Library Function and have properly commented it for better understanding of the implementation

Pseudo Code as give in Paper

First the factor analysis of each process was calculated using the Formulae
F= Arrival time * 0.3 + Priority of the process * 0.5 + Burst time * 0.2

Next, the process was shuffled according to the factor of each process in ascending order in the ready queue such that head of the ready queue contains the lowest factor process burst time, arrival time and priority of process. The pseudo-code of the algorithm is discussed below.

BEGIN:
While(process in ready queue)
LOW=burst value of the first process in the Ready Queue
HIGH=burst value of the last process in the Ready Queue
TQ=(low + high) / 2
k=TQ. *//This time Quantum was applied for each process*

IF(burst time of the process < k)
That process was assigned to the CPU till it terminates.

ELSE IF(Remaining burst time of the process < k/2)
That process was assigned to the CPU again till it terminates.

ELSE
The process will occupy the CPU till the time quantum and it is added to the

ready queue in ascending order according to the remaining burst time for the next round of execution.
*TQ= TQ *2*
K=TQ
END IF ELSE
END WHILE
END

C++ CODE:

```
//OMDRRS ALGORITHM IMPLEMENTATION USING C++ and its Standard Library Functions
//Anuraag Singh 2015043
//Paras Rastogi 2015175
#include<iostream>
#include<set>
#include<map>
#include<queue>

using namespace std;

// Swap Number used in Selection Sort
void swap(int * a1,int * a2)
{
	*a1=*a1+*a2;
	*a2=*a1-*a2;
	*a1=*a1-*a2;
}
void sort(int factoranalysis[],int processid[],int arrivaltime[],int bursttime[],int priority[],int n)
{
	//SELECTION SORT to sort the process according to Factor analysis time
	for(int i=0;i<n;i++)
	{
	int tempind=i;
	for(int j=i+1;j<n;j++)
	{
```

```
        // Picking the smallest element each time
        if(factoranalysis[j]<factoranalysis[tempind])
        {
                tempind=j;
        }
        }
        if(tempind!=i)
        {
        // Swapping Index
        swap(&factoranalysis[tempind],&factoranalysis[i]);
        swap(&processid[tempind],&processid[i]);
        swap(&arrivaltime[tempind],&arrivaltime[i]);
        swap(&bursttime[tempind],&bursttime[i]);
        swap(&priority[tempind],&priority[i]);
        }
        }
}

void sort2(int factoranalysis[],int processid[],int arrivaltime[],int bursttime[],int priority[],int
turnaroundtime[],int waitingtime[],int responsetime[],int n,int completiontime[])
{
        //SELECTION SORT to print answer
        for(int i=0;i<n;i++)
        {
        int tempind=i;
        for(int j=i+1;j<n;j++)
        {
        // Picking the smallest element each time
        if(processid[j]<processid[tempind])
        {
                tempind=j;
        }
        }
        if(tempind!=i)
        {
        // Swapping Index
        swap(&factoranalysis[tempind],&factoranalysis[i]);
        swap(&completiontime[tempind],&completiontime[i]);
        swap(&processid[tempind],&processid[i]);
```

```
        swap(&arrivaltime[tempind],&arrivaltime[i]);
        swap(&bursttime[tempind],&bursttime[i]);
        swap(&priority[tempind],&priority[i]);
        swap(&responsetime[tempind],&responsetime[i]);
        swap(&turnaroundtime[tempind],&turnaroundtime[i]);
        swap(&waitingtime[tempind],&waitingtime[i]);
        }
        }
}
int main()
{
        // Number of Process
        int n;
        cout<<"Enter Number of Process"<<endl;
        cin>>n;
        cout<<"Enter Details of each process in this order Process ID -> Arrival Time ->
Burst Time -> Priority -> "<<endl;
        int processid[n],arrivaltime[n],bursttime[n],bursttime2[n],priority[n];
        int turnaroundtime[n],waitingtime[n],responsetime[n],completiontime[n];
        int factoranalysis[n];


        //Deails of each Process
        for(int i=0;i<n;i++)
        {

        cin>>processid[i];
        cin>>bursttime[i];
        cin>>arrivaltime[i];
        cin>>priority[i];
        completiontime[i]=0;
        }

        //Calculating factoranalysis according to bursttime, arrivaltime, priority time
        for(int i=0;i<n;i++)
        {
        factoranalysis[i]=bursttime[i]*0.2+arrivaltime[i]*0.3+priority[i]*0.5;
        responsetime[i]=0;
        }
```

```
        //Sorting the processes according to factoranalysis
        sort(factoranalysis,processid,arrivaltime,bursttime,priority,n);

        map<int,int> mp;      //Hashing the process with its id, as it will change after
operation

        for(int i=0;i<n;i++)
        {
        bursttime2[i]=bursttime[i];
        }

        int low=bursttime[0];
        int high=bursttime[n-1];

        int timequantum=(high+low)/2;

        priority_queue<pair<int,int> > qq;
        // Min heap for gettint the process with minimum burst time after it is in the ready
queue

        int timer=arrivaltime[0];
   // int temp=factoranalysis[0];
        // Pushing the process in the ready queue as before executinn the first process
they should already be in the queue

        int toggle=0;   // To toggle Time Quantum
// Filling ready queue according to factoranalysis

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

        responsetime[i]=timer;
        if(timequantum>=bursttime[i])
        {
                timer+=bursttime[i];
                bursttime[i]=0;
                completiontime[i]=timer;
        }
```

```
else
{
        timer+=timequantum;
        // Premptive implementation
        bursttime[i]-=timequantum;
        if(bursttime[i]==0)
        {
        completiontime[i]=timer;
        }
        else
        {
        qq.push(make_pair(bursttime[i],i));
        }
        // Toggling Time Quantum everytime
        if(toggle==0)
        {
        timequantum=timequantum/2;
        toggle=1;
        }
        else
        {
        timequantum=timequantum*2;
        toggle=0;
        }
}
}

while(qq.size()>0)  // Till ready queue is not empty
{

pair<int,int> p=qq.top();
int id=p.second;//=mp[processid[p.second]];
qq.pop();
if(timequantum>=bursttime[id])
{
        // Process Completed
        timer+=bursttime[id];
        bursttime[id]=0;
        //Completion Time Updated
```

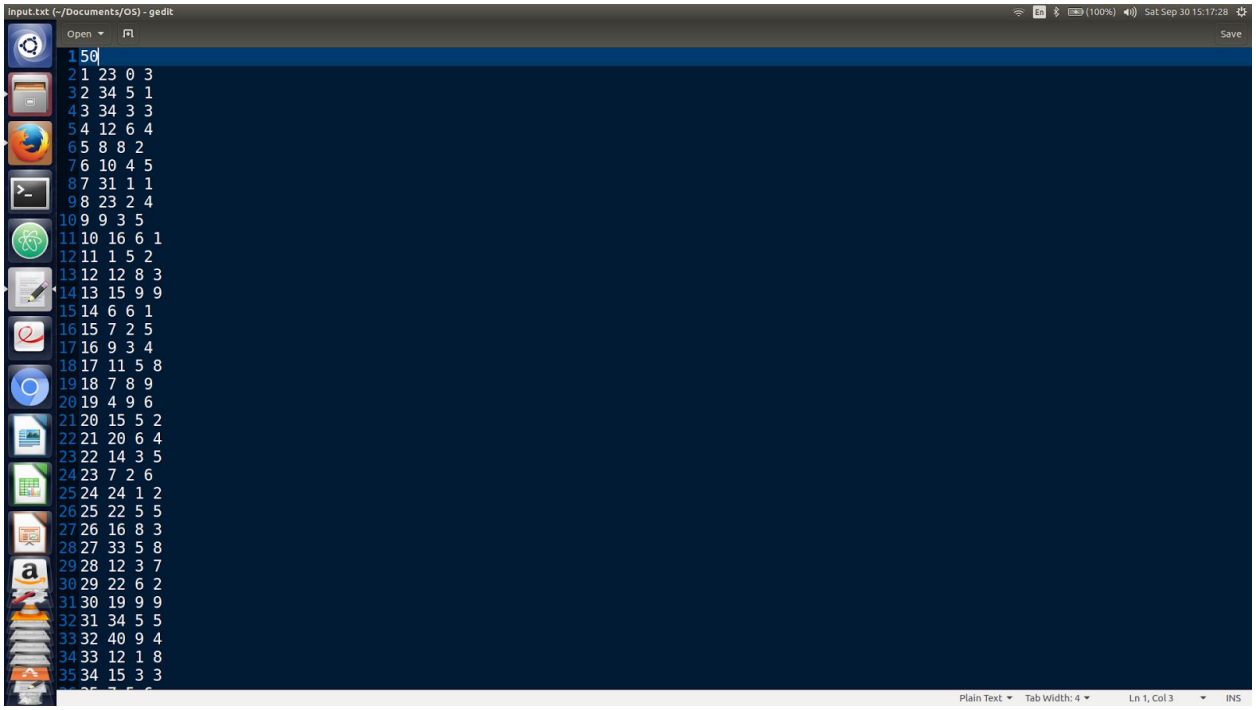
```
        completiontime[id]=timer;
}
else
{
        timer+=timequantum;
        // Premptive implementation
        bursttime[id]-=timequantum;
        if(bursttime[id]==0)
        {
        completiontime[id]=timer;
        }
        else
        {
        qq.push(make_pair(bursttime[id],id));
        }
        // Toggling Time Quantum everytime
        if(toggle==0)
        {
        timequantum=timequantum/2;
        toggle=1;
        }
        else
        {
        timequantum=timequantum*2;
        toggle=0;
        }
}

}

// Calculaiting Different Parameters
int totalturnaroundtime=0;
int totalwaitingtime=0;
int totalresponsetime=0;
for(int i=0;i<n;i++)
{

// TurnAroundTime= Completion Time - Arrival Time
turnaroundtime[i]=completiontime[i]-arrivaltime[i];
```

```
        // WaitingTime= TurnAround Time - Burst Time
        waitingtime[i]=turnaroundtime[i]-bursttime2[i];

        totalturnaroundtime+=turnaroundtime[i];
        totalwaitingtime+=waitingtime[i];
        totalresponsetime+=responsetime[i];
        }

sort2(factoranalysis,processid,arrivaltime,bursttime2,priority,turnaroundtime,waitingtime,
responsetime,n,completiontime);

        cout<<"\nProcess ID |"<<"Arrival Time |"<<"Burst Time |"<<"Completion Time
|"<<"Turnaround Time |"<<"Waiting Time  |"<<"Response Time |"<<endl;
        for(int i=0;i<n;i++)
        {

cout<<processid[i]<<"\t|\t"<<arrivaltime[i]<<"\t|\t"<<bursttime2[i]<<"\t|\t"<<completiontime
[i]<<"\t|\t"<<turnaroundtime[i]<<"\t|\t"<<waitingtime[i]<<"\t|\t"<<responsetime[i]<<endl;
        }

        float avgwaitingtime=totalwaitingtime/(1.0*n);
        float avgturnaroundtime=totalturnaroundtime/(1.0*n);
        float avgresponsetime=totalresponsetime/(1.0*n);

        cout<<endl<<"Average Waiting Time ="<<avgwaitingtime<<endl;
        cout<<endl<<"Average Response Time ="<<avgresponsetime<<endl;
        cout<<endl<<"Average TurnAroundTime ="<<avgturnaroundtime<<endl;

        return 0;
}
```

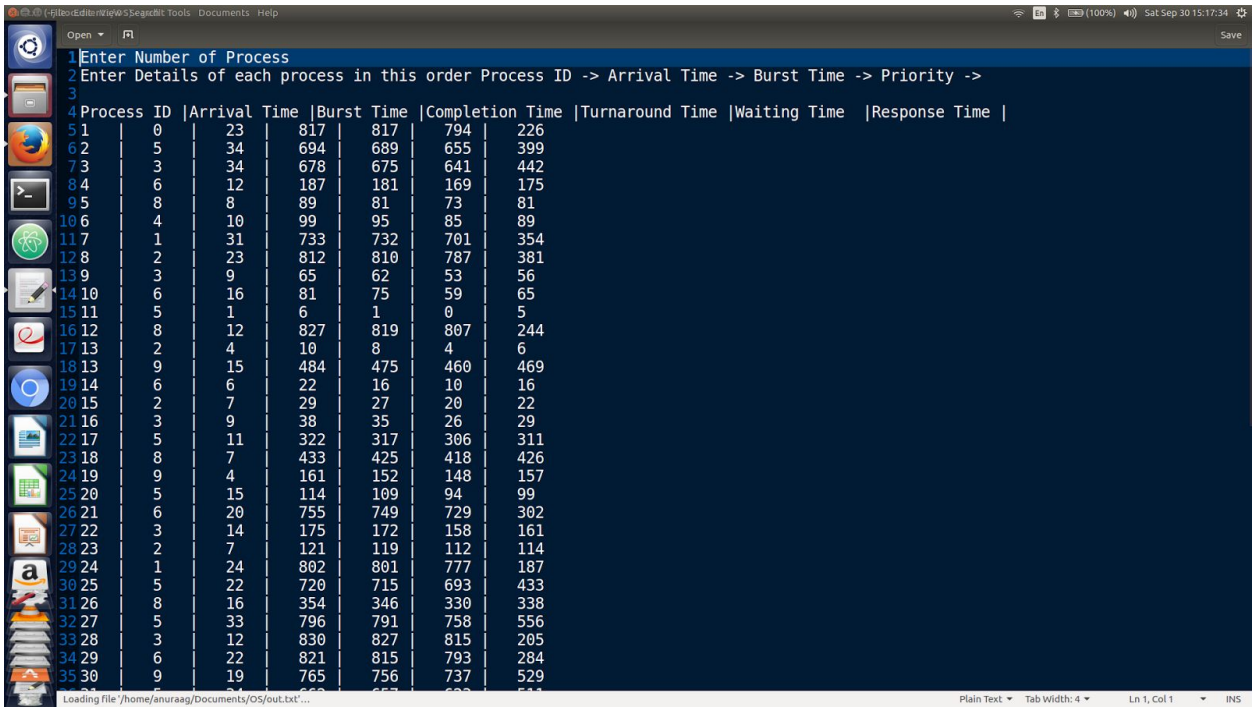
OUTPUT

Output file has been enclosed, here are the screenshots

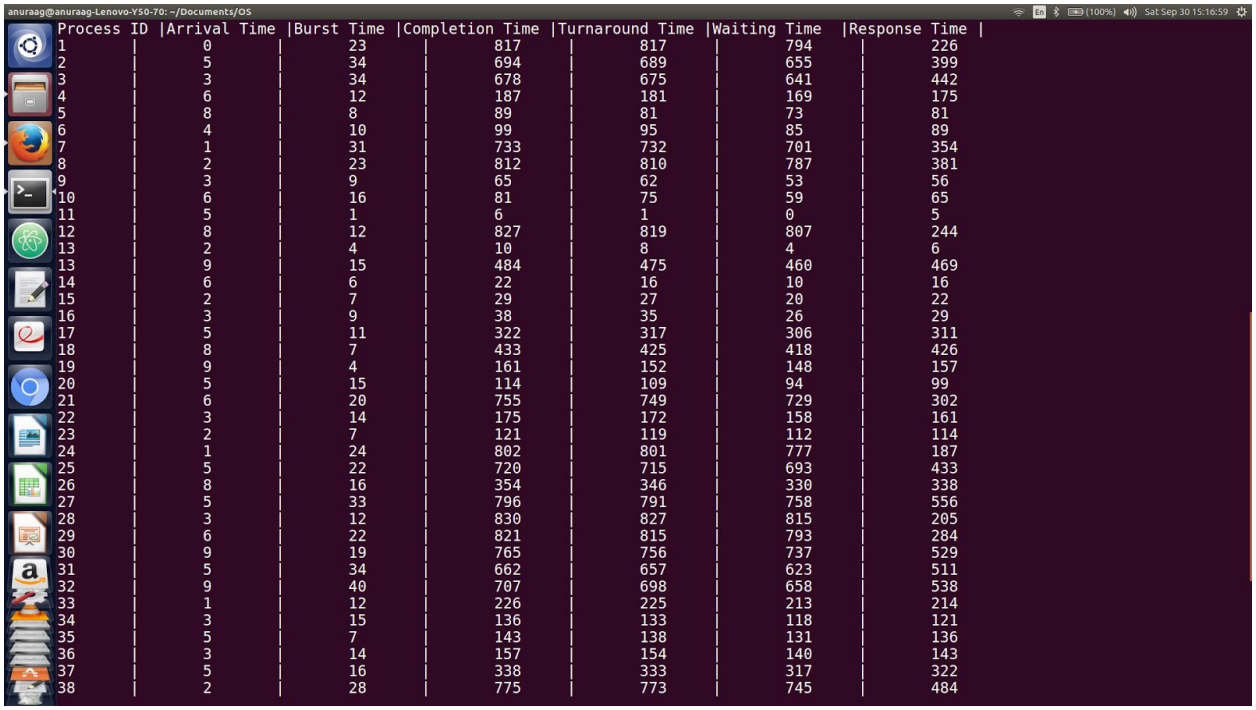
Input FIle



OUTPUT



```
Enter Number of Process
Enter Details of each process in this order Process ID -> Arrival Time -> Burst Time -> Priority ->

Process ID |Arrival Time |Burst Time |Completion Time |Turnaround Time |Waiting Time  |Response Time |
1    |    0    |    23   |    817   |    817   |    794   |    226
2    |    5    |    34   |    694   |    689   |    655   |    399
3    |    3    |    34   |    678   |    675   |    641   |    442
4    |    6    |    12   |    187   |    181   |    169   |    175
5    |    8    |    8    |    89    |    81    |    73    |    81
6    |    4    |    10   |    99    |    95    |    85    |    89
7    |    1    |    31   |    733   |    732   |    701   |    354
8    |    2    |    23   |    812   |    810   |    787   |    381
9    |    3    |    9    |    65    |    62    |    53    |    56
10   |    6    |    16   |    81    |    75    |    59    |    65
11   |    5    |    1    |    6     |    1     |    0     |    5
12   |    8    |    12   |    827   |    819   |    807   |    244
13   |    2    |    4    |    10    |    8     |    4     |    6
13   |    9    |    15   |    484   |    475   |    460   |    469
14   |    6    |    6    |    22    |    16    |    10    |    16
15   |    2    |    7    |    29    |    27    |    20    |    22
16   |    3    |    9    |    38    |    35    |    26    |    29
17   |    5    |    11   |    322   |    317   |    306   |    311
18   |    8    |    7    |    433   |    425   |    418   |    426
19   |    9    |    4    |    161   |    152   |    148   |    157
20   |    5    |    15   |    114   |    109   |    94    |    99
21   |    6    |    20   |    755   |    749   |    729   |    302
22   |    3    |    14   |    175   |    172   |    158   |    161
23   |    2    |    7    |    121   |    119   |    112   |    114
24   |    1    |    24   |    802   |    801   |    777   |    187
25   |    5    |    22   |    720   |    715   |    693   |    433
26   |    8    |    16   |    354   |    346   |    330   |    338
27   |    5    |    33   |    796   |    791   |    758   |    556
28   |    3    |    12   |    830   |    827   |    815   |    205
29   |    6    |    22   |    821   |    815   |    793   |    284
30   |    9    |    19   |    765   |    756   |    737   |    529
```



Process ID	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time	Response Time
1	0	23	817	817	794	226
2	5	34	694	689	655	399
3	3	34	678	675	641	442
4	6	12	187	181	169	175
5	8	8	89	81	73	81
6	4	10	99	95	85	89
7	1	31	733	732	701	354
8	2	23	812	810	787	381
9	3	9	65	62	53	56
10	6	16	81	75	59	65
11	5	1	6	1	0	5
12	8	12	827	819	807	244
13	2	4	10	8	4	6
13	9	15	484	475	460	469
14	6	6	22	16	10	16
15	2	7	29	27	20	22
16	3	9	38	35	26	29
17	5	11	322	317	306	311
18	8	7	433	425	418	426
19	9	4	161	152	148	157
20	5	15	114	109	94	99
21	6	20	755	749	729	302
22	3	14	175	172	158	161
23	2	7	121	119	112	114
24	1	24	802	801	777	187
25	5	22	720	715	693	433
26	8	16	354	346	330	338
27	5	33	796	791	758	556
28	3	12	830	827	815	205
29	6	22	821	815	793	284
30	9	19	765	756	737	529
31	5	34	662	657	623	511
32	9	40	707	698	658	538
33	1	12	226	225	213	214
34	3	15	136	133	118	121
35	5	7	143	138	131	136
36	3	14	157	154	140	143
37	5	16	338	333	317	322
38	2	28	775	773	745	484