CSE2005 – OPERATING SYSTEMS

Project component

An Improved Scheduling Algorithm

Tanishk Aggarwal 18BCE0578

Aditya Ruhatiya 18BCE0582

Aditya Agrawal 18BCE0586

Vansh Goel 18BCE0594

School of Computer Science and Engineering

VIT University

Submitted to: Prof. Meenakshi S.P.

ABSTRACT

Currently prevalent round robin CPU scheduling algorithms are not applicable in real time operating systems because of their large waiting time, large turnaround time, large response time, high context switch rates and less throughput and also because we can't calculate the Burst time of a process beforehand. The primary aim of this project is to come up with a new method for CPU scheduling which enhances the CPU performance in an operating system by combining both Round robin algorithm and priority algorithm. The proposed algorithm for real time systems is a hybrid of priority and round-robin scheduling algorithms. It combines the merits of round robin, like elimination of starvation, and priority scheduling. The proposed algorithm also deals with aging by assigning new ranks to the processes. Thus the algorithm corrects all the limitations of RR CPU scheduling algorithm. We will also present a comparative analysis of our new algorithm with the existing algorithms like priority and round robin on the basis of average turnaround time, average waiting time, varying time quantum and number of context switches.

I. Introduction:

In computer science, scheduling is the process by which processes are given access to system resources like processor cycles, communications bandwidth. Hence, there arises the need for a scheduling algorithm for the computer systems to perform multitasking and multiplexing.

Scheduling is a fundamental operating system function that determines which process run, when there are multiple run able processes. CPU scheduling is important because it impacts resource utilization and other performance parameters. There exists a number of CPU scheduling algorithms like First Come First Serve, Shortest Job First Scheduling, Round Robin scheduling, Priority Scheduling etc., but due to a number of disadvantages these are rarely used in real time operating systems except Round Robin scheduling.

A number of assumptions are considered in CPU scheduling which are as follows:

1. Job pool consists of run able processes waiting for the CPU.

2. All processes are independent and compete for resources.

3. The job of the scheduler is to distribute the limited resources of CPU to the different processes fairly and in a way that optimizes some performance criteria.

The scheduler is the component of the kernel that selects which process to run next. Operating systems may feature up to three distinct types of schedulers, a long term scheduler, a mid-term or medium term scheduler and a short-term scheduler. The long term scheduler or job scheduler selects processes from the job pool and loads them into memory for execution. The short term scheduler, or CPU scheduler selects from among the processes that are ready to execute, and allocates CPU to one of them. The medium term scheduler removes processes from memory and reduces the degree of multiprogramming results in the scheme of swapping. Swapping is the scheme which is performed by dispatcher which is the module that gives control of the CPU to the process selected by the short-term scheduler.

The CPU scheduling also plays an important role in the real time operating system which always has a time constraint on computations. A real time system is the one whose applications are mission-critical, where real-time tasks should be scheduled to be completed before their deadlines. Most real-time systems control unpredictable environments and may need operating systems that can handle unknown and changing tasks. So, not only a dynamic task scheduling is required, but both system hardware and software must adapt to unforeseen configurations.

There are two main types of real-time systems: Hard Real-Time System, Firm or Soft Real-Time System. In Hard Real-Time System, it requires that fixed deadlines must be met otherwise disastrous situation may arise whereas in Soft Real-Time System, missing an occasional deadline is undesirable, but nevertheless tolerable. System in which performance is

degraded but not destroyed by failure to meet response time constraints is called soft real time systems.

In real time systems each task should be invoked after the ready time and must be completed before its deadline, an attempt has been made to satisfy these constraints. Simple round robin architecture is not suitable to implement in Soft real time due to more number of context switches, longer waiting and response times. This in turn leads to low throughput in the system. If a real-time process having relatively larger CPU burst it will lead to the problem of starvation. Priority scheduling may be a better option for real-time scheduling but it will face the similar problem i.e. low priority processes will always starve.

II. SCHEDULING OBJECTIVES

A system architect must consider various elements in constructing a scheduling algorithm, for example, kind of frameworks utilized and client necessities. Relying upon the system, the client and designer may anticipate that the schedulers will:

- Maximize throughput: A scheduling algorithm ought to be equipped for adjusting the greatest number of jobs per unit of time.
- Avoid inconclusive blocking or starvation: A job ought not sit tight for unbounded time before or while process service.
- Minimize overhead: Overhead causes wastage of assets. Be that as it may, when we utilize system resources viably, then general system execution enhances extraordinarily.
- Enforcement of priorities: if system allocates priorities to forms, the booking system ought to support the higher-need forms.
- Attain balance between response, utilization: the scheduling algorithm must keep resources occupied
- Support jobs which show desirable behaviour.
- Corrupt gracefully under huge load.

A system might finish these objectives in many ways. The scheduler can stall indefinite blocking of jobs via aging. Scheduler increments throughput by favouring processes whose requests can be satisfied quickly, or whose completion cause other processes to run.

III. SCHEDULING CRITERIA

CPU scheduling algorithms maybe such which have different properties, and the choice of one particular algorithm might favour one category of processes over other. Choosing an algorithm for a particular situation, we must cater to properties of various algorithms. The scheduling criteria include the following:

- Context Switch: A context switch is process of storing and restoring context (state) of a pre-empted process, so that execution may be resumed from same point at a later stage. Context switching involves a lot of computation, lead to wastage of time and memory, which in turn increases the overhead of scheduler, hence operating system's design, is to optimize only these switches.
- Throughput: it is the number of processes completed per unit time. It is slow in round robin scheduling implementation. Context switching and throughput are inversely proportional.
- CPU Utilization: Tells the business of the CPU. One needs to maximize CPU utilization.
- Turnaround Time: Total time spent to complete the job. The time interval from the time of submission of a job to its completion is the turnaround time. Total turnaround time is the sum of the times spent waiting to get into memory, waiting time in the ready queue, execution time on the CPU and doing I/O.
- Waiting Time: Time a job has been stalled in ready queue. The CPU scheduling algorithm does not affect the amount of time during which a process executes or does input-output; it affects only the amount of time that a process spends waiting in ready queue.
- Response Time: in real time systems, turnaround time may not be best measure. Often, a job can provide with some output fairly early and continue computing new results while previous results are being produced to the user. Thus, it is the time from the submission of a request until the first response is produced that means time when the task is submitted until the first response is received. Thus, the response time should be less.

Hence a good scheduling algorithm for real time and time sharing system must have: -

- less context switches.
- high CPU utilization.
- high throughput.
- Less turnaround time.
- Less waiting time.

ROUND ROBIN SCHEDULING ALGORITHM

The round robin algorithm was designed mainly for time-shared systems not for real time systems. Time slice or time quantum is defined in case of RR algorithm, which refers to duration for which the process is allocated to the CPU and executed.

The processes which have to be executed are kept in a circular queue which has a head and a tail. The CPU scheduler will go around the queue, allocating the CPU to each process for a time interval of one quantum but the problem is that all the processes are arranged in FCFS (First Come First Serve) manner.

Arriving processes are then added to the tail of the queue.

The CPU scheduler will then select the Process Control Block from the head of the ready queue. This is a disadvantage in RR algorithm since all processes are basically given the same priority. Round robin also favours the process with short CPU burst and penalizes long ones.

Disadvantages:

The disadvantages of round robin CPU scheduling algorithm which affects execution process time shared system are as follows-

1. Larger waiting, response time and high rates of context switching.
   Since Real-time programs must guarantee response within specified time constraints therefore larger waiting, response time and high rates of context switching affect the system's performance and delay the results.
2. Low throughput
   Throughput is defined as number of process completed per time unit. If round robin is implemented in soft real time systems throughput will be low which leads to severe degradation of system performance because of high context switching. If the number of context switches is low, then the throughput will be high. Context switch and throughput are inversely proportional to each other.

With these observations it is found that the existing simple round robin architecture is not suitable for real time

PRIORITY SCHEDULING ALGORITHM

The operating system assigns a fixed priority number to every process, and the scheduler then arranges the processes in the ready queue in order of their priority. These processes are then allocated to the CPU one by one. Lower priority processes get interrupted by incoming higher priority processes.

Waiting time and response time in this algorithm depend on the priority of the processes. Higher priority processes have smaller waiting and response times. Deadlines can be easily met by giving higher priority to the earlier deadline processes.

Disadvantage:

Starvation of lower priority processes is possible if large number of higher priority processes keep arriving continuously.

Therefore a combination of the above two algorithms will be needed to make an algorithm fit for real time systems.

OUR CONTRIBUTION

We came up with a modified round robin/priority multilevel scheduling algorithm which works on a calculated time quantum that takes into account the priority and the burst time of the process queue. This will help in reducing the average waiting time, average turnaround time and reduce the number of context switch.

OUR PROPOSED ALGORITHM

We implement our algorithm using two queues

Algorithm:

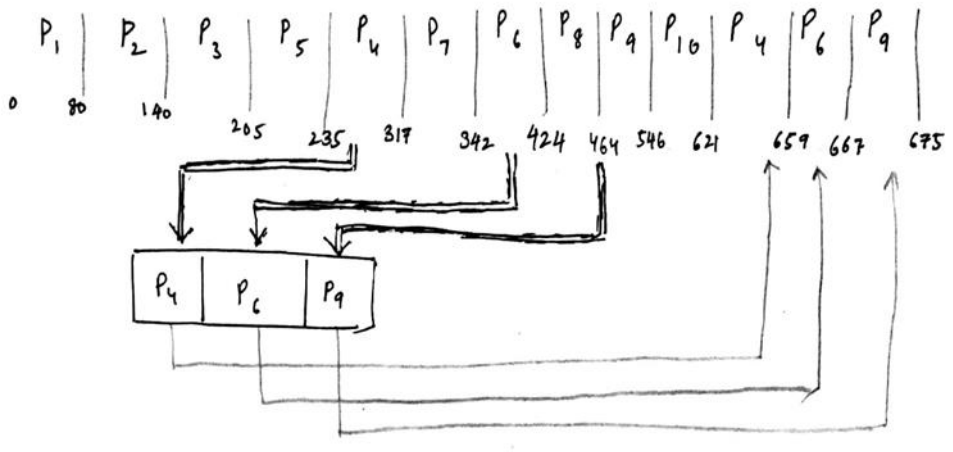
1. Take the burst time and priority of the processes and calculate the new priority rank by using 50% of priority and 50% of rank according to burst time.
2. Arrange the processes according to the calculated rank.
3. Now we calculate the smart time quanta(STQ) by the formula:
4. Mean of all burst time + 0.5*SD of burst time
5. Now start executing the processes by round robin algorithm with STQ time quanta.
6. If a process is not finished in one go, we add the process to the other wait queue.
7. Now after all the processes are over in our executing queue, we execute all the processes in the waiting queue.

	B_t	Priority	(B_t) rank	$f(P, B_t r) = \frac{3P + B_t r}{4}$	f rank
P_1	80	1	7	2.5	1
P_2	60	2	4	2.5	2
P_3	65	3	5	3.5	3
P_4	120	4	10	5.5	5
P_5	30	5	2	4.25	4
P_6	90	6	8	6.5	7
P_7	25	7	1	5.5	6
P_8	40	8	3	6.75	8
P_9	90	9	9	9	9
P_{10}	75	10	6	9	10

mean = 67.5 (average of burst time $\Rightarrow \frac{\Sigma B_t}{n}$)

Time Quantum = mean + 0.5 × Standard deviation

$$= 67.5 + 0.5 \times \left\{ \sqrt{\frac{\Sigma (x_i - \bar{x})^2}{n}} \right\} = 82.265$$



Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int i,j;
struct process{
   int process_id,burst_time,priority,shortness_rank,remaining_time;
   float new_rank;
   int waitting_time,turnaround_time;
};
int sort(struct process *p,int n,int type){
   struct process temp;
   if(type==1){
      for(i=0;i<n-1;i++)
         for(j=0;j<n-i-1;j++)
            if(p[j].new_rank-p[j+1].new_rank>0){/// our new algo
               temp=p[j];
               p[j]=p[j+1];
               p[j+1]=temp;
            }
   }
   else if(type==2){
      for(i=0;i<n-1;i++)
         for(j=0;j<n-1-i;j++)
            if(p[j].burst_time>p[j+1].burst_time){///  sjf
               temp=p[j];
               p[j]=p[j+1];
               p[j+1]=temp;
            }
   }
   else if(type==3){
      for(i=0;i<n-1;i++)
         for(j=0;j<n-i-1;j++)
            if(p[j].priority>p[j+1].priority){///  priority
               temp=p[j];
               p[j]=p[j+1];
               p[j+1]=temp;
            }
   }
   else if(type==4){
      for(i=0;i<n-1;i++)
         for(j=0;j<n-i-1;j++)
            if(p[j].process_id>p[j+1].process_id){/// printing
```

```
                temp=p[j];
                p[j]=p[j+1];
                p[j+1]=temp;
            }
    }
    return(1);
}
int print_pretty(struct process *p,int n,int t){
    float avg_wt=0,avg_tt=0;
    sort(p,n,4);
    printf("\nProcess_id\tWaitting Time\tTurnaround Time\n");
    for(i=0;i<n;i++){
        avg_wt+=p[i].waitting_time;
        avg_tt+=p[i].turnaround_time;
        printf("\tP%d\t\t%d\t\t%d\n",p[i].process_id,p[i].waitting_time,p[i].turnaround_time);
    }
    avg_wt=(float)avg_wt/n;
    avg_tt=(float)avg_tt/n;
    printf("\tAverage Waitting Time:- %.2f\n",avg_wt);
    printf("\tAverage Turnaround Time:- %.2f\n",avg_tt);
    printf("\tThroughput:- %f\n",(float)t/n);
}
int returnIndex(struct process *p,int n,int a){
    for(i=0;i<n;i++){
        if(p[i].process_id==a)
        return(i);
    }
    return(-1);
}
int fcfs(struct process *que1,int n){
    int total_time=0;
    for(i=0;i<n;i++){
        que1[i].waitting_time=total_time;
        total_time+=que1[i].burst_time;
        que1[i].turnaround_time=total_time;
        printf("-P%d-",que1[i].process_id);
    }
    print_pretty(que1,n,total_time);
    return(1);
}
int sjf(struct process *que1,int n){
    sort(que1,n,2);
    int total_time=0;
    for(i=0;i<n;i++){
```

```
        que1[i].waitting_time=total_time;
        total_time+=que1[i].burst_time;
        que1[i].turnaround_time=total_time;
        printf("-P%d-",que1[i].process_id);
    }
    print_pretty(que1,n,total_time);
    return(1);
}
int rr(struct process *que1,int n){
    int time_quanta;
    printf("Enter the time quanta:\t");
    scanf("%d",&time_quanta);
    int total_time=0;
    int n1=n,count=0;
    while(n1){
        if(que1[count].remaining_time<=time_quanta && que1[count].remaining_time>0){
            total_time+=que1[count].remaining_time;
            que1[count].remaining_time=0;
            que1[count].turnaround_time=total_time;
            que1[count].waitting_time=total_time-que1[count].burst_time;
            printf("-P%d-",que1[count++].process_id);
            n1--;
        }
        else if(que1[count].remaining_time>0){
            que1[count].remaining_time-=time_quanta;
            total_time+=time_quanta;
            printf("-P%d-",que1[count++].process_id);
        }
        else
            count++;
        if(count==n)
            count=count%n;
    }
    print_pretty(que1,n,total_time);
    return(1);
}
int priority(struct process *que1,int n){
    sort(que1,n,3);
    int total_time=0;
    for(i=0;i<n;i++){
        que1[i].waitting_time=total_time;
        total_time+=que1[i].burst_time;
        que1[i].turnaround_time=total_time;
        printf("-P%d-",que1[i].process_id);
```

```
    }
    print_pretty(que1,n,total_time);
    return(1);
}
int multiLevelScheduling(struct process *que1,int n){
    int i,j;
    int sum=0;
    float mean,standard_deviation=0.0;
 for(i=0;i<n;i++){
        sum+=que1[i].burst_time;
     int r=1;
     for(j=0;j<n;j++)
            if(j!=i && que1[j].burst_time<que1[i].burst_time)
                r++;
        que1[i].shortness_rank=r;
        que1[i].new_rank=(float)(que1[i].priority+que1[i].shortness_rank)/2;
    }
    mean=(float)sum/n;
    for(i=0;i<n;i++)
        standard_deviation+=(que1[i].burst_time-mean)*(que1[i].burst_time-mean);
    standard_deviation=sqrt(standard_deviation/n);
    int time_quanta=(int)(mean+0.5*standard_deviation);
    int t=n,n2,n1=n;
    int total_time=0;
    struct process *que2;
    struct process *p=que1;
    sort(que1,n,1);
    while (t){
        que2=(struct process*)malloc(sizeof(struct process)*n1);
        n2=0;
        for(i=0;i<n1;i++){
            if(que1[i].remaining_time<time_quanta){
                total_time+=que1[i].remaining_time;
                int x=returnIndex(p,n,que1[i].process_id);
                p[x].turnaround_time=total_time;
                p[x].waitting_time=total_time-p[x].burst_time;
                que1[i].remaining_time=0;
                t--;
            }
            else{
                total_time+=time_quanta;
                que1[i].remaining_time-=time_quanta;
                que2[n2++]=que1[i];
            }
```

```
            printf("-P%d-",que1[i].process_id);
        }
        que1=que2;
        n1=n2;
    }
    print_pretty(p,n,total_time);
    return(1);
}

int main(){
    int c,n;
    while(1){
        printf("\n\t\tAvailable Scheduling Algorithms\n\t1. FCFS(first come first served)\n\t2.
SJF(shortest job first)\n\t3. RR(round robin)\n");
        printf("\t4. Priority based\n\t5. Modified multilevel(our proposed algorithm)\n\t6. Exit\nSelect
any one option:-\t");
        scanf("%d",&c);
        if(c<6){
            printf("Enter the number of processes:-\t");
            scanf("%d",&n);
            struct process queue1[n];
            if(c<4)
                for(i=0;i<n;i++){
                    printf("enter the burst time of process with id number p%d:-\t",i+1);
                    scanf("%d",&queue1[i].burst_time);
                    queue1[i].remaining_time=queue1[i].burst_time;
                    queue1[i].process_id=i+1;
                }
            else
                for(i=0;i<n;i++){
                    printf("enter the burst time and priority of process with id number p%d:-\t",i+1);
                    scanf("%d %d",&queue1[i].burst_time,&queue1[i].priority);
                    queue1[i].remaining_time=queue1[i].burst_time;
                    queue1[i].process_id=i+1;
                }
            switch (c){
                case 1: fcfs(queue1,n);
                        break;
                case 2: sjf(queue1,n);
                        break;
                case 3: rr(queue1,n);
                        break;
                case 4: priority(queue1,n);
                        break;
```

```
            case 5: multiLevelScheduling(queue1,n);
                    break;
            default: printf("Wrong option selected");
          }
       }
       else{
          exit(0);
       }
    }
    return(0);
}
```

Comparison:

Algorithm	Parameter											
	Process Number	1	2	3	4	5	6	7	8	9	10	AVG
	Burst Time	120	60	90	75	65	80	40	90	30	25	
Round Robin	Waiting Time	535	375	575	435	470	495	240	585	320	350	438.0
	Turnaround Time	655	435	665	510	535	575	280	675	350	375	505.5
Priority	Waiting Time	205	80	510	600	140	0	470	355	325	445	313
	Turnaround Time	325	140	600	675	205	80	510	445	355	470	380.5
	PRIORITY	4	2	9	10	3	1	8	6	5	7	
Proposed Algorithm	Waiting Time	537	0	585	462	90	155	260	576	60	235	296
	Turnaround Time	657	60	675	537	155	235	300	666	90	260	363.5

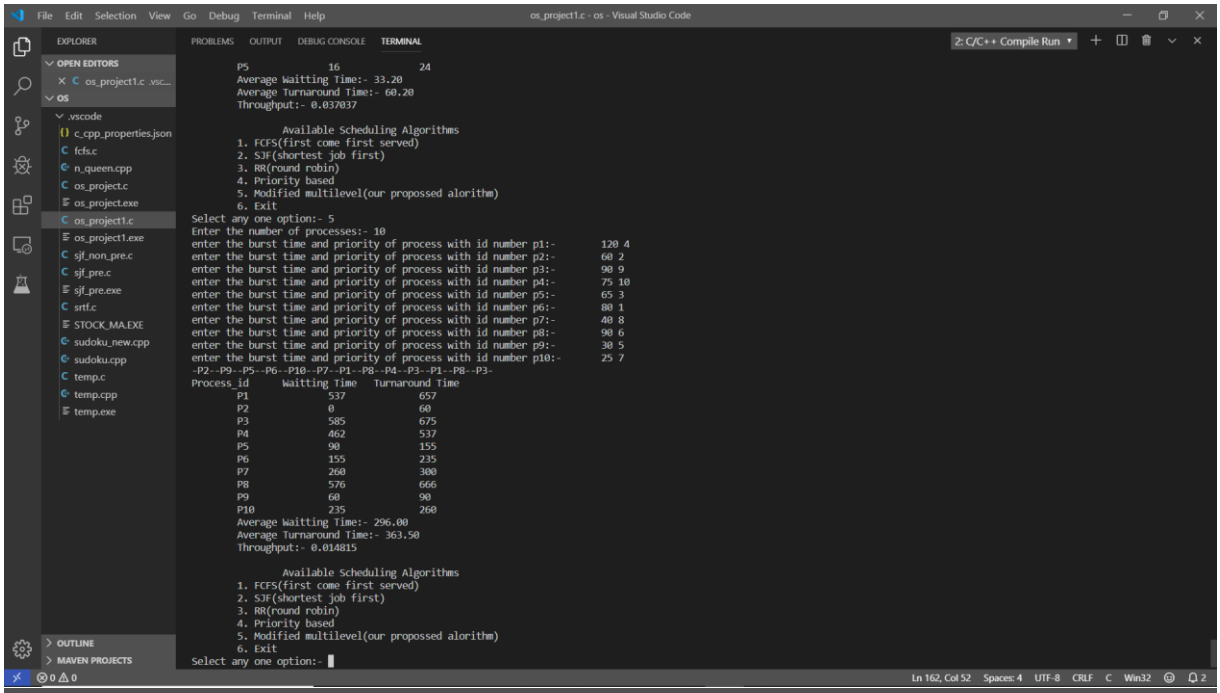
FCFS	**Waiting Time**	**0**	**120**	**180**	**270**	**345**	**410**	**490**	**530**	**620**	**650**	**361.5**
	Turnaround Time	**120**	**180**	**270**	**345**	**410**	**490**	**530**	**620**	**650**	**675**	**429.0**
SJF	**Waiting Time**	**555**	**95**	**375**	**220**	**155**	**295**	**55**	**465**	**25**	**0**	**224.0**
	Turnaround Time	**675**	**155**	**465**	**295**	**220**	**375**	**95**	**555**	**55**	**25**	**291.5**

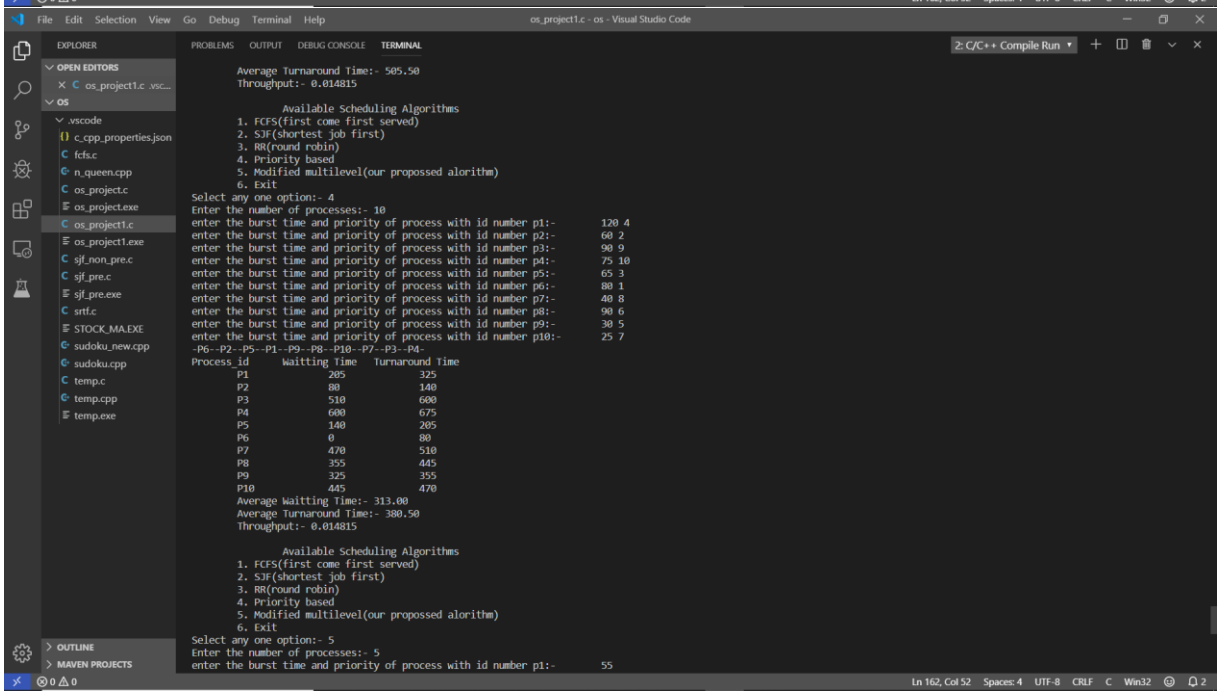
Throughput 0.014815

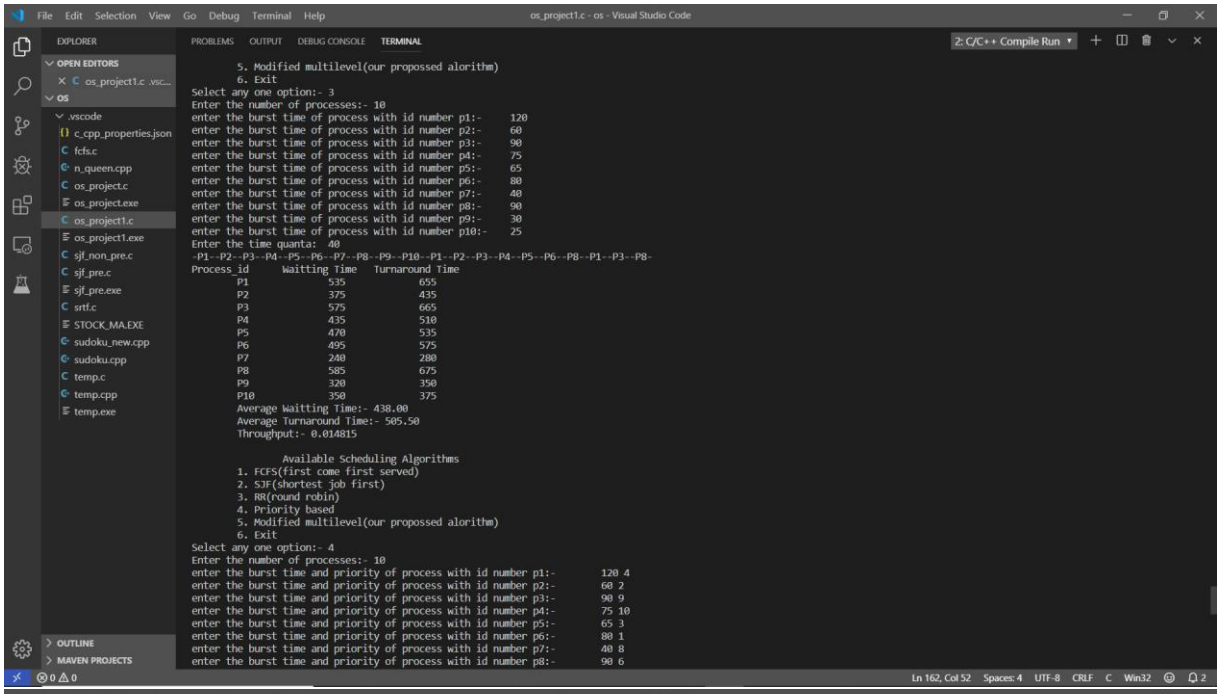
Round Robin time quanta=40

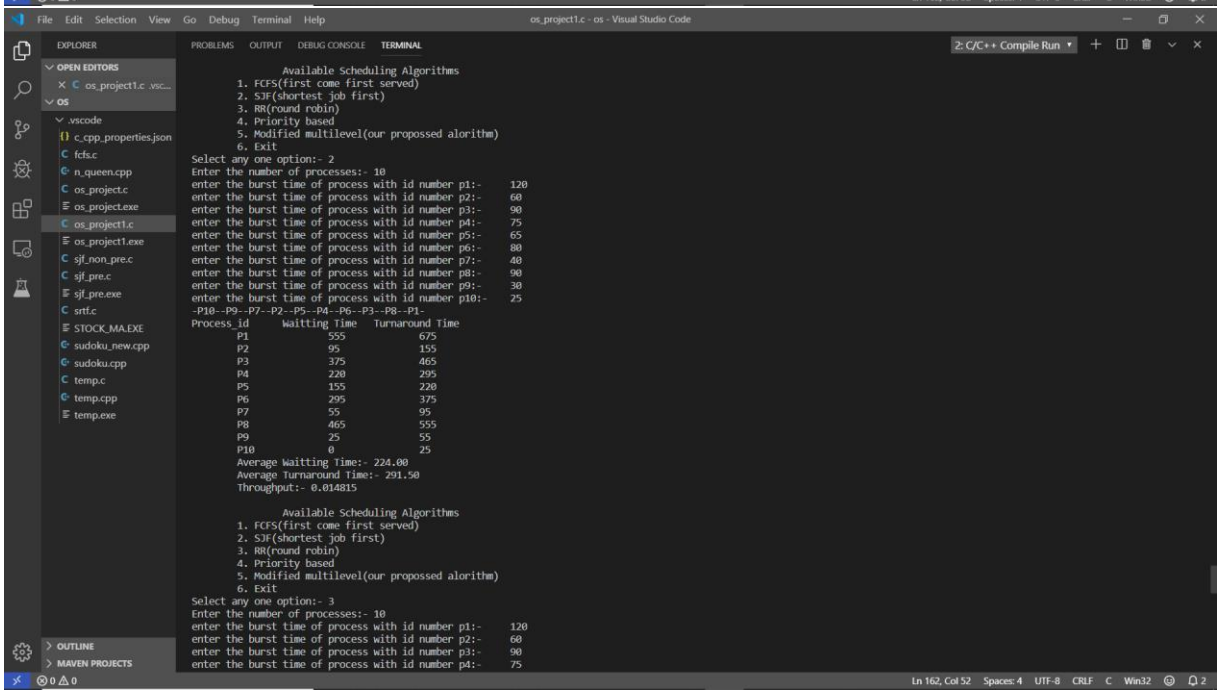
The proposed algorithm has less Avg. waiting time and Avg. turnaround time as compared to RR and Priority Algorithm.

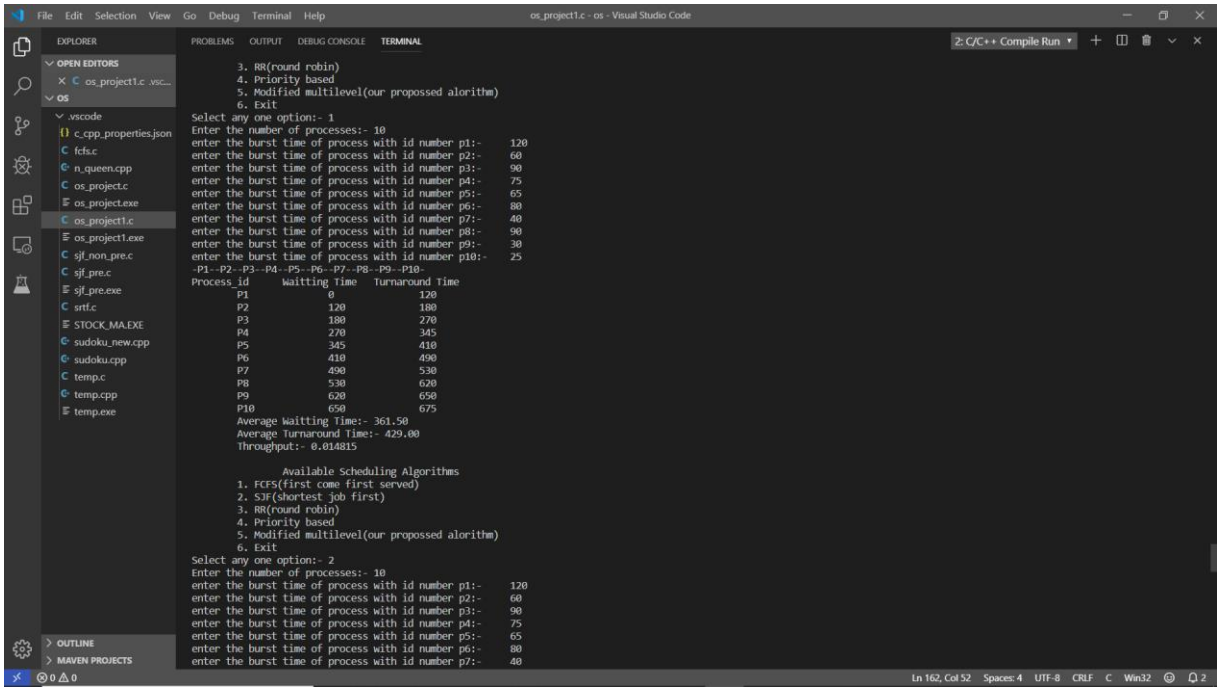
Output:


File Edit Selection View Go Debug Terminal Help
os_project1.c - os - Visual Studio Code
EXPLORER
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: C/C++ Compile Run
OPEN EDITORS
os_project1.c .vsc...
OS
.vscode
c_cpp_properties.json
fcfs.c
n_queen.cpp
os_project.c
os_project.exe
os_project1.c
os_project1.exe
sjf_non_pre.c
sjf_pre.c
sjf_pre.exe
srtf.c
STOCK_MA.EXE
sudoku_new.cpp
sudoku.cpp
temp.c
temp.cpp
temp.exe
OUTLINE
MAVEN PROJECTS
P5 16 24
Average Waitting Time:- 33.20
Average Turnaround Time:- 60.20
Throughput:- 0.037037
Available Scheduling Algorithms
1. FCFS(first come first served)
2. SJF(shortest job first)
3. RR(round robin)
4. Priority based
5. Modified multilevel(our proposed alorithm)
6. Exit
Select any one option:- 5
Enter the number of processes:- 10
enter the burst time and priority of process with id number p1:- 120 4
enter the burst time and priority of process with id number p2:- 60 2
enter the burst time and priority of process with id number p3:- 90 9
enter the burst time and priority of process with id number p4:- 75 10
enter the burst time and priority of process with id number p5:- 65 3
enter the burst time and priority of process with id number p6:- 80 1
enter the burst time and priority of process with id number p7:- 40 8
enter the burst time and priority of process with id number p8:- 90 6
enter the burst time and priority of process with id number p9:- 30 5
enter the burst time and priority of process with id number p10:- 25 7
-P2--P9--P5--P6--P10--P7--P1--P8--P4--P3--P1--P8--P3-
Process_id Waitting Time Turnaround Time
P1 537 657
P2 0 60
P3 585 675
P4 462 537
P5 90 155
P6 155 235
P7 260 300
P8 576 666
P9 60 90
P10 235 260
Average Waitting Time:- 296.00
Average Turnaround Time:- 363.50
Throughput:- 0.014815
Available Scheduling Algorithms
1. FCFS(first come first served)
2. SJF(shortest job first)
3. RR(round robin)
4. Priority based
5. Modified multilevel(our proposed alorithm)
6. Exit
Select any one option:-
Ln 162, Col 52 Spaces: 4 UTF-8 CRLF C Win32


File Edit Selection View Go Debug Terminal Help
os_project1.c - os - Visual Studio Code
EXPLORER
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: C/C++ Compile Run
OPEN EDITORS
os_project1.c .vsc...
OS
.vscode
c_cpp_properties.json
fcfs.c
n_queen.cpp
os_project.c
os_project.exe
os_project1.c
os_project1.exe
sjf_non_pre.c
sjf_pre.c
sjf_pre.exe
srtf.c
STOCK_MA.EXE
sudoku_new.cpp
sudoku.cpp
temp.c
temp.cpp
temp.exe
OUTLINE
MAVEN PROJECTS
Average Turnaround Time:- 505.50
Throughput:- 0.014815
Available Scheduling Algorithms
1. FCFS(first come first served)
2. SJF(shortest job first)
3. RR(round robin)
4. Priority based
5. Modified multilevel(our proposed alorithm)
6. Exit
Select any one option:- 4
Enter the number of processes:- 10
enter the burst time and priority of process with id number p1:- 120 4
enter the burst time and priority of process with id number p2:- 60 2
enter the burst time and priority of process with id number p3:- 90 9
enter the burst time and priority of process with id number p4:- 75 10
enter the burst time and priority of process with id number p5:- 65 3
enter the burst time and priority of process with id number p6:- 80 1
enter the burst time and priority of process with id number p7:- 40 8
enter the burst time and priority of process with id number p8:- 90 6
enter the burst time and priority of process with id number p9:- 30 5
enter the burst time and priority of process with id number p10:- 25 7
-P6--P2--P5--P1--P9--P8--P10--P7--P3--P4-
Process_id Waitting Time Turnaround Time
P1 205 325
P2 80 140
P3 510 600
P4 600 675
P5 140 205
P6 0 80
P7 470 510
P8 355 445
P9 325 355
P10 445 470
Average Waitting Time:- 313.00
Average Turnaround Time:- 380.50
Throughput:- 0.014815
Available Scheduling Algorithms
1. FCFS(first come first served)
2. SJF(shortest job first)
3. RR(round robin)
4. Priority based
5. Modified multilevel(our proposed alorithm)
6. Exit
Select any one option:- 5
Enter the number of processes:- 5
enter the burst time and priority of process with id number p1:- 55
Ln 162, Col 52 Spaces: 4 UTF-8 CRLF C Win32


File Edit Selection View Go Debug Terminal Help
os_project1.c - os - Visual Studio Code
EXPLORER
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: C/C++ Compile Run
OPEN EDITORS
os_project1.c .vsc...
OS
.vscode
c_cpp_properties.json
fcfs.c
n_queen.cpp
os_project.c
os_project.exe
os_project1.c
os_project1.exe
sjf_non_pre.c
sjf_pre.c
sjf_pre.exe
srtf.c
STOCK_MA.EXE
sudoku_new.cpp
sudoku.cpp
temp.c
temp.cpp
temp.exe
OUTLINE
MAVEN PROJECTS
5. Modified multilevel(our proppossed alorithm)
6. Exit
Select any one option:- 3
Enter the number of processes:- 10
enter the burst time of process with id number p1:- 120
enter the burst time of process with id number p2:- 60
enter the burst time of process with id number p3:- 90
enter the burst time of process with id number p4:- 75
enter the burst time of process with id number p5:- 65
enter the burst time of process with id number p6:- 80
enter the burst time of process with id number p7:- 40
enter the burst time of process with id number p8:- 90
enter the burst time of process with id number p9:- 30
enter the burst time of process with id number p10:- 25
Enter the time quanta: 40
-P1--P2--P3--P4--P5--P6--P7--P8--P9--P10--P1--P2--P3--P4--P5--P6--P8--P1--P3--P8-
Process_id Waitting Time Turnaround Time
P1 535 655
P2 375 435
P3 575 665
P4 435 510
P5 470 535
P6 495 575
P7 240 280
P8 585 675
P9 320 350
P10 350 375
Average Waitting Time:- 438.00
Average Turnaround Time:- 505.50
Throughput:- 0.014815
Available Scheduling Algorithms
1. FCFS(first come first served)
2. SJF(shortest job first)
3. RR(round robin)
4. Priority based
5. Modified multilevel(our proppossed alorithm)
6. Exit
Select any one option:- 4
Enter the number of processes:- 10
enter the burst time and priority of process with id number p1:- 120 4
enter the burst time and priority of process with id number p2:- 60 2
enter the burst time and priority of process with id number p3:- 90 9
enter the burst time and priority of process with id number p4:- 75 10
enter the burst time and priority of process with id number p5:- 65 3
enter the burst time and priority of process with id number p6:- 80 1
enter the burst time and priority of process with id number p7:- 40 8
enter the burst time and priority of process with id number p8:- 90 6
Ln 162, Col 52 Spaces: 4 UTF-8 CRLF C Win32


File Edit Selection View Go Debug Terminal Help
os_project1.c - os - Visual Studio Code
EXPLORER
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: C/C++ Compile Run
OPEN EDITORS
os_project1.c .vsc...
OS
.vscode
c_cpp_properties.json
fcfs.c
n_queen.cpp
os_project.c
os_project.exe
os_project1.c
os_project1.exe
sjf_non_pre.c
sjf_pre.c
sjf_pre.exe
srtf.c
STOCK_MA.EXE
sudoku_new.cpp
sudoku.cpp
temp.c
temp.cpp
temp.exe
OUTLINE
MAVEN PROJECTS
Available Scheduling Algorithms
1. FCFS(first come first served)
2. SJF(shortest job first)
3. RR(round robin)
4. Priority based
5. Modified multilevel(our proppossed alorithm)
6. Exit
Select any one option:- 2
Enter the number of processes:- 10
enter the burst time of process with id number p1:- 120
enter the burst time of process with id number p2:- 60
enter the burst time of process with id number p3:- 90
enter the burst time of process with id number p4:- 75
enter the burst time of process with id number p5:- 65
enter the burst time of process with id number p6:- 80
enter the burst time of process with id number p7:- 40
enter the burst time of process with id number p8:- 90
enter the burst time of process with id number p9:- 30
enter the burst time of process with id number p10:- 25
-P10--P9--P7--P2--P5--P4--P6--P3--P8--P1-
Process_id Waitting Time Turnaround Time
P1 555 675
P2 95 155
P3 375 465
P4 220 295
P5 155 220
P6 295 375
P7 55 95
P8 465 555
P9 25 55
P10 0 25
Average Waitting Time:- 224.00
Average Turnaround Time:- 291.50
Throughput:- 0.014815
Available Scheduling Algorithms
1. FCFS(first come first served)
2. SJF(shortest job first)
3. RR(round robin)
4. Priority based
5. Modified multilevel(our proppossed alorithm)
6. Exit
Select any one option:- 3
Enter the number of processes:- 10
enter the burst time of process with id number p1:- 120
enter the burst time of process with id number p2:- 60
enter the burst time of process with id number p3:- 90
enter the burst time of process with id number p4:- 75
Ln 162, Col 52 Spaces: 4 UTF-8 CRLF C Win32



Conclusion:

From the above comparisons, we observed that our new proposed algorithm is performing better than the RR, Priority algorithm. Modified Round Robin proposed in this project in terms of average waiting time and average turnaround time thereby reducing the overhead, saving of memory spaces and solving the problem of starvation which was caused by Priority Algorithm.

REFERENCES

[1] Silberschatz, A., Peterson, J. L., and Galvin, B.,Operating System Concepts, Addison Wesley, 7th Edition, 2006.

[2] E.O. Oyetunji, A. E. Oluleye," Performance Assessment of Some CPU Scheduling Algorithms", Research Journal of Information Technology, 1(1): pp 22-26, 2009.

[3] Ajit Singh, Priyanka Goyal, Sahil Batra," An Optimized Round Robin Scheduling Algorithm for CPU Scheduling", (IJCSE) International Journal on Computer Science and Engineering Vol. 02, No. 07, 2383-2385, 2010.

[4] Rakesh kumar yadav, Abhishek K Mishra, Navin Prakash, Himanshu Sharma," An Improved Round Robin Scheduling Algorithm for CPU Scheduling", (IJCSE) International Journal on Computer Science and Engineering Vol. 02, No. 04, 1064-1066, 2010.

[5] William Stallings, Operating Systems Internal and Design Principles, 5th Edition , 2006.

[6] Saroj Hiranwal, K. C. Roy," Adaptive Round Robin Scheduling using shortest burst approach based on smart time slice", International Journal of Computer Science and Communication Vol. 2, No. 2, pp. 319-323, July-December 2011.

[7] Abbas Noon, Ali Kalakech, Seifedine Kadry, "A New Round Robin Based Scheduling Algorithm for Operating Systems: Dynamic Quantum Using the Mean Average", IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 3, No. 1, May 2011.

[8] Blazewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., and Weglarz, J.: Scheduling Computer and Manufacturing Processes, Berlin, Springer, (2001).

[9] Stallings, W.: Operating Systems Internals and Design Principles, 5th edition, Prentice Hall, (2004).

[10] Swin, B. R., Tayli, M., and Benmaiza, M.: Prospects for Predictable Dynamic Scheduling in RTDOS, Journal King Saud University, Computer & Information Science, Vol. 9, pp. 57-93, (1997).

[11] Barbara Korousic –Seljak (1994) "Task scheduling policies for real-time systems" Journal on MICROPROCESSOR AND MICROSYSTEMS, VOL 18, NO. 9, pg 501-512.

[12] Enrico Bini and Giorgio C. Buttazzo (2004) "Schedulability Analysis of Periodic Fixed Priority Systems" journal on IEEE TRANSACTIONS ON COMPUTERS VOL 53 NO.11, pg 14621473.

[13] Zhi Quan and Jong-Moon Chang (2003) " A Statistical Framework for EDF Scheduling" journal on IEEE COMMUNICATION LETTERS VOL 7 NO.10, pg 493-495.

[14] Houssine Chetto and Maryline Chetto (1989) " Some Results of Earliest Deadline Scheduling Algorithm" journal on IEEE TRANSACTION ON SOFTWARE ENGINEERING. VOL 15. NO.10, pg 1261-1269.

[15] Harry Katzan, Jr., "Operating Systems / A Pragmatic Approach", pages 113, 157, 350353, Van Nostrand Reinhold Company, 1973.

[16] M.Kaladevi, M.C.A.,M.Phil., and Dr.S.Sathiyabama, M.Sc.,M.Phil.,Ph.D, "A Comparative Study of Scheduling Algorithms for Real Time Task", International Journal of Advances in Science and Technology, Vol. 1, No. 4, 2010