THE NATIONAL INSTITUTE OF ENGINEERING, MYSURU

(An Autonomous Institute under VTU, Belagavi)

Implementation of a Scheduling Algorithm and Page Replacement Algorithm

In partial fulfillment of the requirements for the completion of tutorial in the course

Operating Systems (CS5C02)

Semester V

Computer Science and Engineering

Submitted by

M N R SAI MAMATHA - 4NI19CS062
MAHALAKSHMI S - 4NI19CS065

To the course instructor

Dr. JAYASRI B S
(Professor & Dean – EAB)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

THE NATIONAL INSTITUTE OF ENGINEERING

Mysore-570 008

2021-2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

THE NATIONAL INSTITUTE OF ENGINEERING

(An Autonomous Institute under VTU, Belagavi)

CERTIFICATE

This is to certify that the work carried out by M N R SAI MAMATHA (4NI19CS062) , MAHALAKSHMI S (4NI19CS065) in partial fulfilment of the requirements for the completion of the tutorial in the course "OPERATING SYSTEMS" in the SEMESTER V, Department of Computer Science and Engineering as per the academic regulations of The National Institute of Engineering, Mysuru , during the academic year 2021-22.

Signature of the course instructor

_____

(Dr. JAYASRI B S)

SCHEDULING

When a computer is multi-programmed, it frequently has multiple processes or threads competing for the CPU at the same time. This situation occurs whenever two or more of them are simultaneously in the ready state. If only one CPU is available, a choice has to be made which process to run next. The part of the operating system that makes the choice is called the **scheduler**, and the algorithm it uses is called the **scheduling algorithm**.

Few scheduling algorithms are :

- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-First (SJF) Scheduling
- Priority Scheduling
- Shortest Remaining Time First
- Round Robin(RR) Scheduling

These algorithms are either **non-preemptive or pre-emptive**. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be pre-empted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may pre-empt a low priority running process anytime when a high priority process enters into a ready state.

SJF SCHEDULING ALGORITHM

Shortest job first (**SJF**) also known as **Shortest job next** (**SJN**), is a scheduling algorithm that selects for execution the waiting process with the smallest execution time. SJF is a non-pre-emptive algorithm. Shortest Remaining Time First is a pre-emptive variant of SJF.

ADVANTAGES

- Maximum throughput.

  Throughput means total number of processes executed per unit time. Shortest job first scheduling algorithm selects the waiting process with the smallest execution time. Thus, in SJF, shortest jobs are executed first making the CPU utilization maximum. So, maximum number of tasks are completed.

- Minimum waiting and turn around time as compared with other scheduling algorithms.

DISADVANTAGES

- It may cause **starvation** if shorter process keeps coming.

ALGORITHM

1. Sort all the processes according to their arrival time.
2. Select the process with minimum arrival time as well as minimum burst time.
3. After completion of the process, select from the ready queue the process which has the minimum burst time.
4. If processes have same burst time length then FCFS (First come First Serve) scheduling algorithm used.
5. Repeat above processes until all processes have finished their execution.

Completion Time: Time at which process completes its execution.

Turn Around Time: Time Difference between completion time and arrival time.

Turn Around Time = Completion Time – Arrival Time

Waiting Time(W.T): Time Difference between turn around time and burst time.

Waiting Time = Turn Around Time – Burst Time

CODE

```
#include<stdio.h>
int main()
{
	int at[10],bt[10],pr[10];
	int n,i,j,temp,time=0,count,over=0,sum_wait=0,sum_turnaround=0,start;
	float avgwait,avgturn;
	printf("Enter the number of processes\n");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		printf("Enter the arrival time and execution time for process %d\n",i+1);
		scanf("%d%d",&at[i],&bt[i]);
		pr[i]=i+1;
	}

//sorting processes according to their arrival time.
	for(i=0;i<n-1;i++)
	{
```

```
		for(j=i+1;j<n;j++)
		{
			if(at[i]>at[j])
			{
				temp=at[i];
				at[i]=at[j];
				at[j]=temp;
				temp=bt[i];
				bt[i]=bt[j];
				bt[j]=temp;
				temp=pr[i];
				pr[i]=pr[j];
				pr[j]=temp;
			}

		}
	}
	printf("\n\nProcess\t|Arrival time\t|Execution time\t|Start time\t|End
time\t|waiting 			time\t|Turnaround time\n\n");
	while(over<n)
	{
		count=0;
		for(i=over;i<n;i++)
		{
			if(at[i]<=time)
			count++;
			else
			break;
		}
		if(count>1){
			for(i=over;i<over+count-1;i++)
			{
				for(j=i+1;j<over+count;j++)
				{
			//sorting processes according to their burst times
					if(bt[i]>bt[j])
					{
						temp=at[i];
						at[i]=at[j];
						at[j]=temp;
						temp=bt[i];
						bt[i]=bt[j];
						bt[j]=temp;
						temp=pr[i];
						pr[i]=pr[j];
						pr[j]=temp;
					}
				}
			}
		}
		start=time;
```

```
                time+=bt[over];
                 printf("p[%d]\t|\t%d\t|\t%d\t|\t%d\t|\t%d\t|\t%d\t|\t%d\n",pr[over],
                        at[over],bt[over],start,time,time-at[over]-bt[over],time-at[over]);
                sum_wait+=time-at[over]-bt[over];


                sum_turnaround+=time-at[over];
                over++;
        }
        avgwait=(float)sum_wait/(float)n;
        avgturn=(float)sum_turnaround/(float)n;
        printf("Average waiting time is %f\n",avgwait);
        printf("Average turnaround time is %f\n",avgturn);
        return 0;
}
```

SAMPLE OUTPUT:

```
Enter the number of processes
7
Enter the arrival time and execution time for process 1
0 8
Enter the arrival time and execution time for process 2
1 2
Enter the arrival time and execution time for process 3
3 4
Enter the arrival time and execution time for process 4
4 1
Enter the arrival time and execution time for process 5
5 6
Enter the arrival time and execution time for process 6
6 5
Enter the arrival time and execution time for process 7
10 1

Process |Arrival time   |Execution time |Start time     |End time       |waiting time   |Turnaround time
p[1]    |       0       |       8       |       0       |       8       |       0       |       8
p[4]    |       4       |       1       |       8       |       9       |       4       |       5
p[2]    |       1       |       2       |       9       |       11      |       8       |       10
p[7]    |       10      |       1       |       11      |       12      |       1       |       2
p[3]    |       3       |       4       |       12      |       16      |       9       |       13
p[6]    |       6       |       5       |       16      |       21      |       10      |       15
p[5]    |       5       |       6       |       21      |       27      |       16      |       22
Average waiting time is 6.857143
Average turnaround time is 10.714286

--------------------------------
Process exited after 23.54 seconds with return value 0
Press any key to continue . . .
```

EXAMPLE

Consider the following five processes each having its own unique burst time and arrival time.

Process	Burst time	Arrival time
P1	3	0
P2	8	1
P3	6	2
P4	4	4
P5	2	5

At time t=0 , P1 is the only process in the ready queue and starts execution.

At time t=1 , P2 is added to the ready queue and P1 will continue for two more units of time.

At time t=2 , P3 is added to the ready queue and P1 will continue execution for one more unit of time.

At time t=3, P1 completes execution.The burst time of P2 and P3 is compared.P3 is allocated with the cpu since P3 has less burst time compared to P2.

At time t=4, P4 is added to the ready queue.P3 will continue execution.

At time t=5,P5 is added to the ready queue.P3 will continue execution.

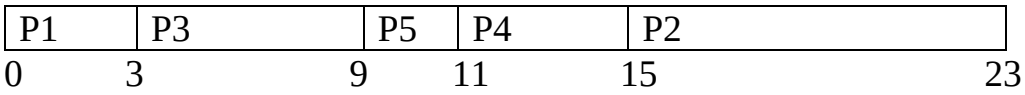
At time t=9,P3 completes execution.Burst time of P2,P4,P5 is compared . P5 is executed as it has the lowest burst time.

At time t=11,P5 completes execution.Burst time of P2 and P4 is compared.P4 is executed as it has less burst time.

At time t=15,P4 completes execution and P2 starts execution.

At time t=23,P2 completes execution.

GANTT CHART



Calculation of Average Waiting Time and Turn Around Time:

Process	BT	AT	CT	TAT	WT
P1	3	0	3	3-0=3	3-3=0
P2	8	1	23	23-1=22	22-8=14
P3	6	2	9	9-2=7	7-6=1
P4	4	4	15	15-4=11	11-4=7
P5	2	5	11	11-5=6	6-2=4

Avg.TAT=(3+22+7+11+6)/5=49/5=9.8ms

Avg.WT=(0+14+1+7+4)/5=26/5=5.2ms

PAGE REPLACEMENT ALGORITHM

The page replacement algorithm decides which memory page is to be replaced.

Page Fault – A page fault occurs when a page requested by a program is not present in the main memory**.**

The effectiveness of a page replacement algorithm is measured with the number of page fault it generates. The more effective the page replacement algorithm is, the less the number of page faults generated by the algorithm.

FIFO PAGE REPLACEMENT ALGORITHM

First-in, first-out (FIFO) algorithm has a simple approach to this problem. We keep track of all the pages by using a queue in the main memory. As soon as a page comes in, we'll put it in the queue and continue. In this way, the oldest page would always be in the first place of the queue.

Now when a new page comes in and there is no space in the memory, we remove the first page in the queue, which is also the oldest one. It repeats this process until the operating system has page flow in the system.

ADVANATAGES

- Easy to understand and implement.

DISADVANTAGES

- The process effectiveness is low.
- When we increase the number of frames while using FIFO, we are giving more memory to processes. So, page fault should decrease, but here the page faults are increasing. This problem is called as Belady's Anomaly.
- Every frame needs to be taken account off.

ALGORITHM

- Step 1. Start to traverse the pages.
- Step 2. If the memory holds fewer pages, then the capacity else goes to step 5.
- Step 3. Push pages in the queue one at a time until the queue reaches its maximum capacity or all page requests are fulfilled.
- Step 4. If the current page is present in the memory, do nothing.
- Step 5. Else, pop the topmost page from the queue as it was inserted first.
- Step 6. Replace the topmost page with the current page from the string.
- Step 7. Increment the page faults.
- Step 8. Stop

CODE:

```
#include <stdio.h>
int main()
{
int referenceString[10], pageFaults = 0, m, n, s, pages, frames,hits,i;
float miss_percent,hit_percent;
printf("\nEnter the number of Pages:\t");
scanf("%d", &pages);
printf("\nEnter reference string values:\n");
for( m = 0; m < pages; m++)
{
   printf("Value No. [%d]:\t", m + 1);
   scanf("%d", &referenceString[m]);
}
printf("\n What are the total number of frames:\t");
{
   scanf("%d", &frames);
}
int temp[frames];
for(m = 0; m < frames; m++)
{
  temp[m] =-1;
}
printf("Ref.String Frame 1\t\tFrame 2\t\tFrame 3");
int k=0;
for(m= 0; m < pages; m++)
{
s=0;
  for(n = 0; n < frames; n++)
   {
      if(referenceString[m] == temp[n])
```

```
        {
          s++;
          pageFaults--;
        }
    }
    pageFaults++;
    if((pageFaults <= frames) && (s == 0))
      {
       temp[m] = referenceString[m];
      }
    else if(s == 0)
      {
       temp[(pageFaults - 1) % frames] = referenceString[m];
      }
      printf("\n");
      printf("%d\t\t",referenceString[k++]);
      for(n = 0;n<frames;n++)
       {
         printf("%d\t\t",temp[n]);
       }

}
miss_percent=((pageFaults*1.0)/pages)*100;
hits=pages-pageFaults;
hit_percent=((hits*1.0)/pages)*100;
printf("\nTotal Page Faults: %d\nNo.of hits: %d\nMiss Ratio: %.2f%%\nHit Ratio: %.2f%%",
pageFaults,pages-pageFaults,miss_percent,hit_percent);
return 0;
}
```

SAMPLE OUTPUT:

```
Enter the number of Pages:      12

Enter reference string values:
Value No. [1]:  1
Value No. [2]:  2
Value No. [3]:  3
Value No. [4]:  4
Value No. [5]:  1
Value No. [6]:  2
Value No. [7]:  5
Value No. [8]:  1
Value No. [9]:  2
Value No. [10]: 3
Value No. [11]: 4
Value No. [12]: 5

 What are the total number of frames:   3
Ref.String Frame 1              Frame 2         Frame 3
1               1               -1              -1
2               1               2               -1
3               1               2               3
4               4               2               3
1               4               1               3
2               4               1               2
5               5               1               2
1               5               1               2
2               5               1               2
3               5               3               2
4               5               3               4
5               5               3               4
Total Page Faults: 9
No.of hits: 3
Miss Ratio: 75.00%
Hit Ratio: 25.00%
```

EXAMPLE

Consider a reference string 1 3 5 1 4 2 6 1

No.of frames = 3

Ref.string	1	3	5	1	4	2	6	1
Frame1	1	1	1	1	4	4	4	1
Frame 2		3	3	3	3	2	2	2
Frame 3			5	5	5	5	6	6

Hit

Page Fault = 7

No.of Hits = 1

Hit ratio = 1/8

=12.5%

Miss ratio = 7/8

=87.5%