THE NATIONAL INSTITUTE OF ENGINEERING, MYSURU

(AN AUTONOMOUS INSTITUTE UNDER VTU, BELAGAVI)

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

Operating System

Semester 5

Computer Science and Engineering

Submitted by

BHAVANI B - 4NI19CS031

NIHARIKA - 4NI19CS075

PRIYANKA S M - 4NI19CS087

To the course instructor

Dr JAYASRI B S

(Associate Professor)

DEPARTMENT OF COMPTER SCIENCE AND ENGINEERING

THE NATIONAL INSTITUTE OF ENGINEERING

Mysuru-570008

2021-2022

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

THE NATIONAL INSTITUTE OF ENGINEERING

(An Autonomous Institute under VTU, Belagavi)

CERTIFICATE

This is to certify the work carried out by BHAVANI B (4NI19CS031), NIHARIKA (4NI19CS075), PRIYANKA S M (4NI19CS087) in partial fulfilment of the requirements for the completion of tutorial in the course Operating System in the V semester, Department of Computer Science and Engineering as per the academic regulations of The National Institute of Engineering, Mysuru, during the academic year 2021-2022.

Signature of the Couse Instructor

_____

(Dr JAYASRI B S)

Associate Professor

Contents

1 SJF – CPU SCHEDULING ALGORITHM

1.1 CPU SCHEDULING ALGORITHM:

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. These algorithms are either non-pre-emptive or pre-emptive. Non-pre-emptive 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 pre-emptive 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.

1.2 SJF:

Shortest Job First (SJF) is an algorithm in which the process having the smallest execution time is chosen for the next execution. This scheduling method can be pre-emptive or non-pre-emptive. The pre-emptive version of SJF is called Shortest Remaining Time First (SRTF).

1.3 ADVANTAGES AND DISADVANTAGES:

Advantages:

1. Maximum throughput.
2. Minimum average waiting and turnaround time.

Disadvantages:

1. May suffer with the problem of starvation.
2. It is not implementable because the exact burst time for a process can't be known in advance.

1.4 ALGORITHM:

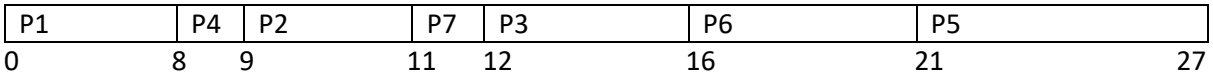
1. Sort all the processes according to their arrival time.
2. Select the process with minimum arrival time as well as minimum burst time.
3. After completion of the process, select from the ready queue the process which has the minimum burst time.
4. Repeat above processes until all processes have finished their execution.

1.5 EXAMPLE:

Suppose we have the following 7 processes with process ID's P1, P2, P3, P4, P5, P6 and P7, they arrive into the CPU in the following manner:

Process ID	Arrival Time (milliseconds)	Burst Time (milliseconds)
P1	0	8
P2	1	2
P3	3	4
P4	4	1
P5	5	6
P6	6	5
P7	10	1

Gantt Chart:



Explanation:

All the processes will be arranged in increasing order of their arrival time in ready queue. At the 0th unit of the CPU, we have only process P1, so it gets executed for 8th time unit. Among all the process which has arrived, the process with minimum burst time is process P4, so it gets executed for 1 time unit. The next minimum burst time is for process P2 so CPU will be allotted to P2. By this time all process will be arrived so remaining process will be executed in the order: P7, P3, P6 and P5.

Turn Around Time = Completion Time – Arrival Time

Waiting Time = Turn Around Time – Burst Time

Process ID	Arrival Time	Burst Time	Completion Time (milliseconds)	Turn Around Time (milliseconds)	Waiting Time (milliseconds)
P1	0	8	8	8	0
P2	1	2	11	10	8
P3	3	4	16	13	9
P4	4	1	9	5	4
P5	5	6	27	22	16
P6	6	5	21	15	10
P7	10	1	12	2	1

Total Turn Around Time = 8+10+13+5+22+15+2 = 75 milliseconds

Average Turn Around Time = Total Turn Around Time / Number of processes = 75/7
= 10.7143 milliseconds

Total Waiting Time = 0+8+9+4+16+10+1 = 48 milliseconds

Average Waiting Time = Total Waiting Time / Number of processes = 48/7
= 6.85714 milliseconds

1.6 CODE:

```
#include <iostream>

using namespace std;

int mat[10][6];

float wt=0,tat=0;

void swap(int* a, int* b)

{

int temp = *a;

*a = *b;

*b = temp;

}

void arrangeArrival(int num, int mat[][6])

{

for (int i = 0; i < num; i++) {
```

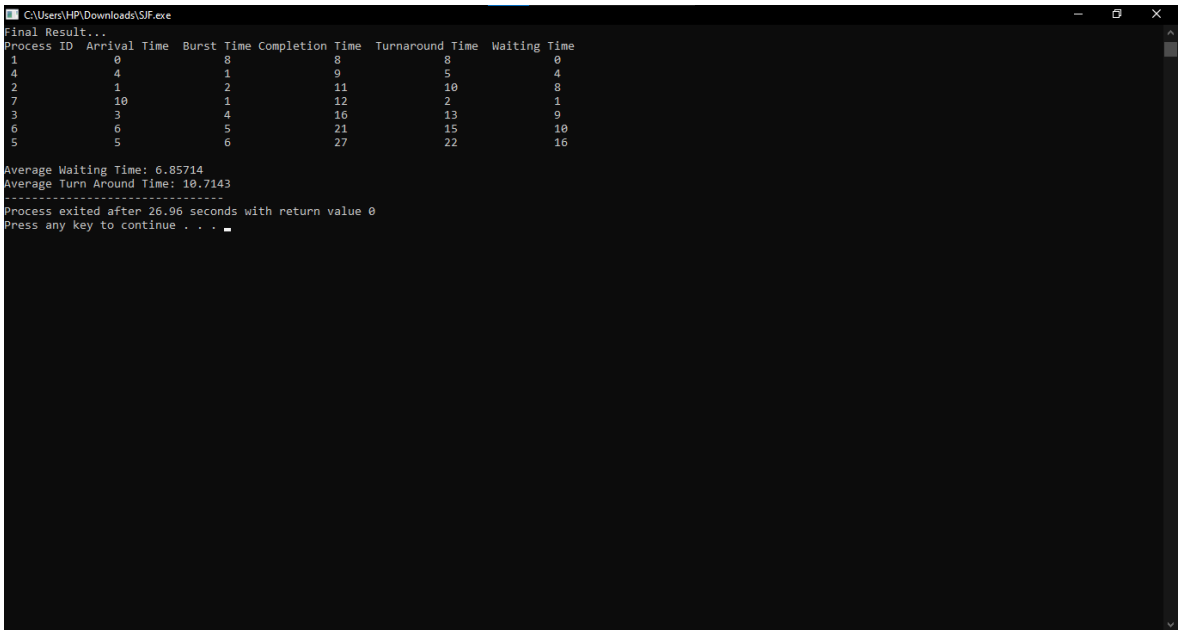
```
for (int j = 0; j < num - i - 1; j++) {
if (mat[j][1] > mat[j + 1][1]) {
for (int k = 0; k < 5; k++) {
swap(mat[j][k], mat[j + 1][k]);
}
}
}
}
}

void completionTime(int num, int mat[][6])
{
int temp, val, m=0;
int va = mat [0][2];
for(int i=1; i<num; i++){
if( mat[0][1] == mat[i][1] && va > mat[i][2] ){
va = mat[i][2];
m = i;

}
}
if (m!=0){
for(int k=0;k<3;k++){
 swap( mat[m][k], mat[0][k] );
}
}
mat[0][3] = mat[0][1] + mat[0][2];
mat[0][5] = mat[0][3] - mat[0][1];
mat[0][4] = mat[0][5] - mat[0][2];
```

```
for (int i = 1; i < num; i++) {
temp = mat[i - 1][3];
int low = mat[i][2];
for (int j = i; j < num; j++) {
if (temp >= mat[j][1] && low >= mat[j][2]) {
low = mat[j][2];
val = j;
}
}
mat[val][3] = temp + mat[val][2];
mat[val][5] = mat[val][3] - mat[val][1];
mat[val][4] = mat[val][5] - mat[val][2];
for (int k = 0; k < 6; k++) {
swap(mat[val][k], mat[i][k]);
}
}
}

int main()
{
int num, temp;

cout << "Enter number of Process: ";
cin >> num;
cout << "\n";
for (int i = 0; i < num; i++) {
cout << "Enter Process Id: ";
cin >> mat[i][0];
cout << "Enter Arrival Time: ";
cin >> mat[i][1];
```

```
cout << "Enter Burst Time: ";
cin >> mat[i][2];
cout << "\n";
}
cout << "\n";
cout << "Before Arrange...\n";
cout << "Process ID\tArrival Time\tBurst Time\n";
for (int i = 0; i < num; i++) {
cout << mat[i][0] << "\t\t" << mat[i][1] << "\t\t"
<< mat[i][2] << "\n";
}

arrangeArrival(num, mat);
completionTime(num, mat);
cout << "\n";
cout << "Final Result...\n";
cout << "Process ID " << " Arrival Time " << " Burst Time"
<< " Completion Time "<< " Turnaround Time "
<< " Waiting Time \n";
for (int i = 0; i < num; i++) {
cout << " " << mat[i][0] << "\t\t" << mat[i][1] << "\t\t"
<< mat[i][2] << "\t\t" << mat[i][3] << "\t\t"
<< mat[i][5] << "\t\t" << mat[i][4] << endl7;
}
for (int i = 0; i < num; i++){
wt=wt+mat[i][4];
}
cout << "\nAverage Waiting Time: ";
cout << wt/num;
for (int i = 0; i < num; i++){
```

```
tat=tat+mat[i][5];
}
cout << "\nAverage Turn Around Time: ";
cout << tat/num;
}
```

1.7 OUTPUT:



2 FIFO - PAGE REPLACEMENT ALGORITHM

2.1 PAGE REPLACEMENT ALGORITHM:

In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in. Page Fault – A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with the newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce the number of page faults.

2.2 FIFO:

First In First Out (FIFO) is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pages in the memory in a queue, oldest page is in the front of the queue. When a page needs to be replaced, page in the front of the queue is selected for removal.

2.3 ADVANTAGES AND DISADVANTAGES:

Advantages:

1. It is simple and easy to understand.

Disadvantages:

1. When the number of incoming pages is large, it might not provide excellent performance.
2. When we increase the number of frames or capacity to store pages in the queue, it should give us less number of page faults.
3. Sometimes FIFO may behave abnormally, and it may increase the number of page faults. This behaviour of FIFO is called Belady's anomaly.
4. In FIFO, the system should keep track of all the frames. Sometimes it results in slow process execution.

2.4 ALGORITHM:

Let capacity be the number of pages that memory can hold. Let set be the current set of pages in memory.

1. Start traversing the pages.

i) If set holds less pages than capacity.

a) Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.

b) Simultaneously maintain the pages in the queue to perform FIFO.

c) Increment page fault.

ii) Else If current page is present in set, do nothing. Else

a) Remove the first page from the queue as it was the first page to be entered in the memory.

b) Replace the first page in the queue with the current page in the string.

c) Store current page in the queue.

d) Increment page faults.

2. Return page faults.

2.5 EXAMPLE:

Reference String: 4, 1, 2, 4, 5

Total number of frames: 3

4	**1**	**2**	**4**	**5**
4	4	4	4	5
	1	1	1	1
		2	2	2
★	★	★	⬆	★

★ ➔ Number of Page Faults
⬆ ➔ Number of Hits

Number of Page Faults = 4 Miss Ratio = 4/5 = 0.8 = 80%

Number of Hits = 1 Hit Ratio = 1/5 = 0.2 = 20%

In the above String 4, 1 and 2 will be placed in frame 1, frame 2 and frame 3 respectively. Since 4 is already present in the table, no changes will occur in the column and it will be

considered as a HIT. 4 has entered first so, 5 will be stored in the place of 4. All remaining columns will be considered as MISS.

2.6 CODE:

```
#include<iostream>
using namespace std;
int main(){
int reference_string[10],page_faults=0,m,n,s,pages,frames;
cout<<"Enter the number of pages : ";
cin>>pages;
cout<<"\nEnter the reference string values : \n";
for(m=0;m<pages;m++){
cout<<"Value no.["<<m+1<<"]: \t";
cin>>reference_string[m];
}
cout<<"\nEnter the total number of frames : ";
cin>>frames;
int temp[frames];
for(m=0;m<frames;m++){
temp[m]=-1;
}
for(m=0;m<pages;m++){
s=0;
for(n=0;n<frames;n++){
if(reference_string[m]==temp[n]){
s++;
page_faults--;
}
}
page_faults++;
```

```
if((page_faults<=frames)&&(s==0)){
temp[m]=reference_string[m];
}
else if(s==0){
temp[(page_faults-1)%frames]=reference_string[m];
}
cout<<"\n";
for(n=0;n<frames;n++){
cout<<temp[n]<<"\t";
}
}
int hits=(pages-page_faults);
int mr=((page_faults*1.0)/pages)*100;
int hr=((hits*1.0)/pages)*100;
cout<<"\n\nTotal page faults : "<<page_faults<<"\n";
cout<<"Total Hits : "<<hits<<"\n";
cout<<"Miss Ratio : "<<mr<<"%\n";
cout<<"Hits Ratio : "<<hr<<"%\n";
return 0;
}
```

2.7 OUTPUT:

3 GITHUB LINKS:

BHAVANI B - https://github.com/BHAVANIB29

NIHARIKA - https://github.com/NiharikaAcharya/OS.git

PRIYANKA S M - https://github.com/priya-082001