THE NATIONAL INSTITUTE OF ENGINEERING, MYSURU

(AN AUTONOMOUS INSTITUTE UNDER VTU, BELAGAVI)

Logo, company name

Description automatically generated

Implementation of a cpu scheduling algorithm and a page replacement algorithm

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

**Operating System (CS5C02)**

**Semester 5**

**Computer Science and Engineering**

Submitted by

ABDUL SHUKUR -- 4NI20CS400

BHUVANA S – 4NI20CS402

To the course instructor

Dr. JAYASRI B S

(Professor & Dean--EAB)

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 ABDUL SHUKUR (4NI20CS400),

BHUVANA S (4NI20CS 402) 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 -- Professor & Dean (EAB)

1. Contents

[2 SRTF – CPU SCHEDULING ALGORITHM **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc411790766)

[2.1 CPU Scheduling Algorithm: **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc1470375410)

[2.2 SRTF: **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc475131657)

[2.3 Algorithm: **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc1000231199)

[2.4 Advantages and Disadvantages: **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc1860448008)

[2.5 Example: **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc1882990805)

[2.6 Code **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc706950164)

[2.7 Output **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc946277615)

[3 LRU - PAGE REPLACEMENT ALGORITHM **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc870093762)

[3.1 Page Replacement Algorithm: **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc938494964)

[3.2 LRU **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc362230729)

[3.3 Advantages and Disadvantages **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc538271992)

[3.4 Algorithm: **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc1474530391)

[3.5 Example **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc1147430380)

[3.6 Code: **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc333824803)

[3.7 Output: **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc953737745)

[4 GITHUB Links **Error! Bookmark not defined.**](file:///C:\Users\spars\Downloads\OS%20Tutorial.docx#_Toc1206579241)

**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 preemptive or non-preemptive. It significantly reduces the average waiting time for other processes awaiting execution. The full form of SJF is Shortest Job First.

**There are basically two types of SJF methods:**

* Non-Preemptive SJF
* Preemptive SJF

## Preemptive SJF

In Preemptive SJF Scheduling, jobs are put into the ready queue as they come. A process with shortest burst time begins execution. If a process with even a shorter burst time arrives, the current process is removed or preempted from execution, and the shorter job is allocated CPU cycle.

## Non-Preemptive SJF

In non-preemptive scheduling, once the CPU cycle is allocated to process, the process holds it till it reaches a waiting state or terminated.

**Characteristics of SJF Scheduling**

* It is associated with each job as a unit of time to complete.
* This algorithm method is helpful for batch-type processing, where waiting for jobs to complete is not critical.
* It can improve process throughput by making sure that shorter jobs are executed first, hence possibly have a short turnaround time.
* It improves job output by offering shorter jobs, which should be executed first, which mostly have a shorter turnaround time.

**Advantages of SJF**

Here are the benefits/pros of using SJF method:

* SJF is frequently used for long term scheduling.
* It reduces the average waiting time over FIFO (First in First Out) algorithm.
* SJF method gives the lowest average waiting time for a specific set of processes.
* It is appropriate for the jobs running in batch, where run times are known in advance.
* For the batch system of long-term scheduling, a burst time estimate can be obtained from the job description.
* For Short-Term Scheduling, we need to predict the value of the next burst time.
* Probably optimal with regard to average turnaround time.

**Disadvantages/Cons of SJF**

Here are some drawbacks/cons of SJF algorithm:

* Job completion time must be known earlier, but it is hard to predict.
* It is often used in a batch system for long term scheduling.
* SJF can’t be implemented for CPU scheduling for the short term. It is because there is no specific method to predict the length of the upcoming CPU burst.
* This algorithm may cause very long turnaround times or starvation.
* Requires knowledge of how long a process or job will run.
* It leads to the starvation that does not reduce average turnaround time.
* It is hard to know the length of the upcoming CPU request.
* Elapsed time should be recorded, that results in more overhead on the processor.

ALGORITHM:

1.Sort all the process according to the arrival time.

2.Then select that process which has minimum arrival time and minimum Burst time.

3.After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

**CODE:**

#include<stdio.h>

int main()

{

int i,n,p[10]={1,2,3,4,5,6,7,8,9,10},min,k=1,btime=0;

int bt[10],temp,j,at[10],wt[10],tt[10],ta=0,sum=0;

float wavg=0,tavg=0,tsum=0,wsum=0;

printf(" -------Shortest Job First Scheduling \n");

printf("\nEnter the No. of processes :"); scanf("%d",&n);

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

{

printf("\tEnter the burst time of %d process :",i+1);

scanf(" %d",&bt[i]);

printf("\tEnter the arrival time of %d process :",i+1);

scanf(" %d",&at[i]);

}

/\*Sorting According to Arrival Time\*/ for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

if(at[i]<at[j])

{

temp=p[j];

p[j]=p[i];

p[i]=temp;

temp=at[j];

at[j]=at[i];

at[i]=temp;

temp=bt[j];

bt[j]=bt[i];

bt[i]=temp;

}

}

}

/\*Arranging the table according to Burst time, Execution time and Arrival Time

Arrival time <= Execution time

\*/

for(j=0;j<n;j++)

{

btime=btime+bt[j];

min=bt[k];

for(i=k;i<n;i++)

{

if (btime>=at[i] && bt[i]<min)

{

temp=p[k];

p[k]=p[i];

p[i]=temp;

temp=at[k];

at[k]=at[i];

at[i]=temp;

temp=bt[k];

bt[k]=bt[i];

bt[i]=temp;

}

}

k++;

}

//to calculate waiting time wt[0]=0;

for(i=1;i<n;i++)

{

sum=sum+bt[i-1];

wt[i]=sum-at[i];

wsum=wsum+wt[i];

}

wavg=(wsum/n);

//to calculate Turn-around time for(i=0;i<n;i++)

{

ta=ta+bt[i];

tt[i]=ta-at[i];

tsum=tsum+tt[i];

}

tavg=(tsum/n);

printf("\nProcess\t Burst\t Arrival\t Waiting\t Turn-around\n" );

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

{

printf("\n p%d\t %d\t %d\t\t %d\t\t\t%d\n",p[i],bt[i],at[i],wt[i],tt[i]);

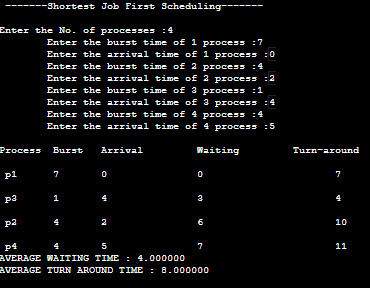
}

printf("AVERAGE WAITING TIME : %f\n",wavg); printf("AVERAGE TURN AROUND TIME : %f\n",tavg);

return 0;

}

**Output:**



**Fifo:**

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the 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.

### **FIFO Page Replacement Algorithm**

It is a very simple way of Page replacement and is referred to as First in First Out. This algorithm mainly replaces the oldest page that has been present in the main memory for the longest time.

* This algorithm is implemented by keeping the track of all the pages in the queue.
* As new pages are requested and are swapped in, they are added to the tail of a queue and the page which is at the head becomes the victim.
* This is not an effective way of page replacement but it can be used for small systems.

### **Advantages**

* This algorithm is simple and easy to use.
* FIFO does not cause more overhead.

### **Disadvantages**

* This algorithm does not make the use of the frequency of **last used time rather**it just replaces the Oldest Page.
* There is an increase in **page faults** as page frames increases.
* The performance of this algorithm is the worst.

**Code:**

#include<stdio.h>

int

main ()

{

int i, j, n, a[50], frame[10], no,avail, k;

float fault, hit, count = 0;

printf ("\n ENTER THE NUMBER OF PAGES:\n");

scanf ("%d", &n);

printf ("\n ENTER THE PAGE NUMBER :\n");

for (i = 1; i <= n; i++)

scanf ("%d", &a[i]);

printf ("\n ENTER THE NUMBER OF FRAMES :");

scanf ("%d", &no);

for (i = 0; i < no; i++)

frame[i] = -1;

j = 0;

printf ("\tref string\t page frames\n");

for (i = 1; i <= n; i++)

{

printf ("%d\t\t", a[i]);

avail = 0;

for (k = 0; k < no; k++)

if (frame[k] == a[i])

avail = 1;

if (avail == 0)

{

frame[j] = a[i];

j = (j + 1) % no;

count++;

for (k = 0; k < no; k++)

printf ("%d\t", frame[k]);

}

printf ("\n");

}

printf ("Page Fault Is %f\n", count);

fault = count;

printf ("miss ratio=%f\n",(fault / n)/100);

hit=(n-count)/n;

printf("hit ratio=%f",hit/100);

return 0;

}

**Output:**

