Operating System tutorials (CS5C02) 2021-22

THE NATIONAL INSTITUTE OF ENGINEERING, MYSURU

(An Autonomous Institute under VTU, Belagavi)

ESTD : 1946

Bachelor of Engineering
In
Computer Science and Engineering

Operation system assignment

Submitted by

Narayani H (4NI19CS073)
Thanushree G (4NI18CS102)

Topic:

SJF (Shortest Job First CPU Scheduling Algorithm)
FIFO (First In First Out Page Replacement)

To the course instructor

Dr. Jayashri B S
Professor

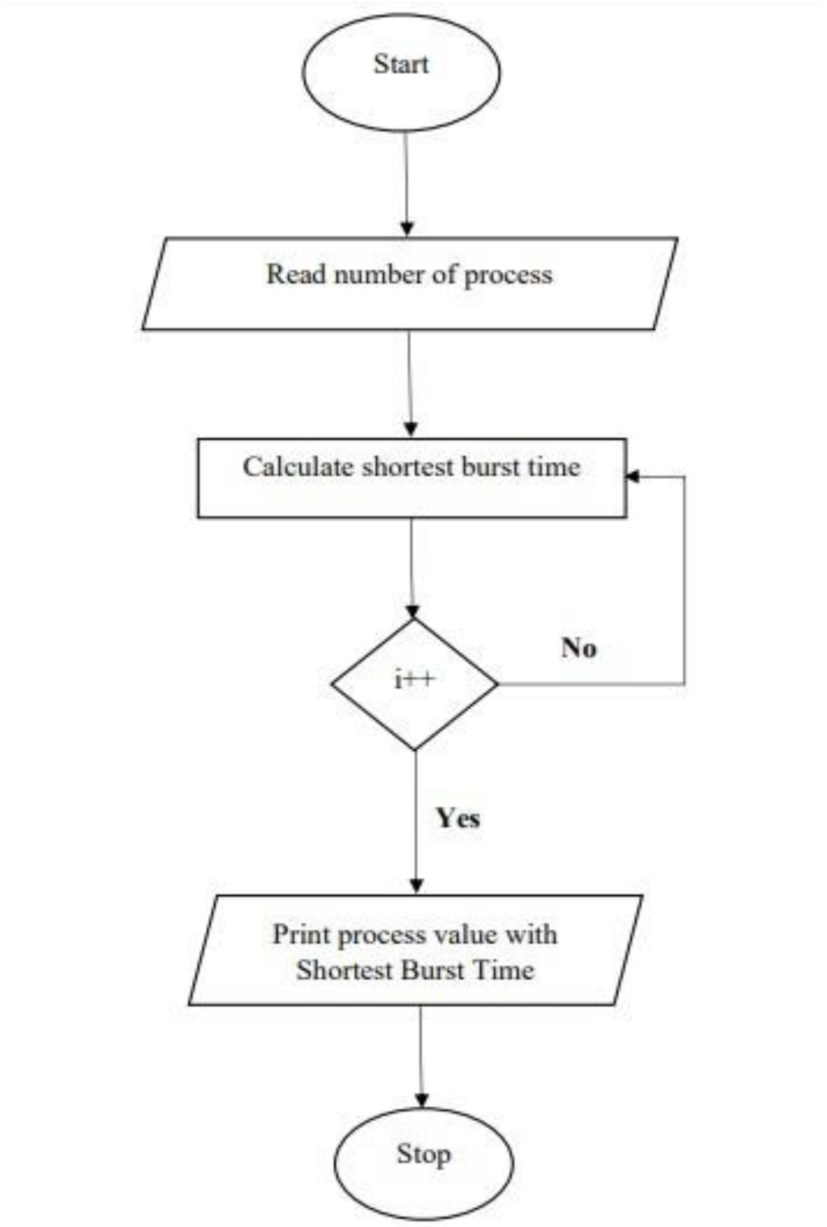
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
THE NATIONAL INSTITUTE OF ENGINEERING
Mysore-570008

2021-2022

SJF (Shortest job first) scheduling algorithm

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. This is an approach which considers the CPU burst. When CPU is available, the process having the smallest CPU burst is allocated with CPU. Now it may happen that two or more processes have the same CPU burst then, which process to allocate will be decided as per FCFS scheduling.

Flow chart


Start
Read number of process
Calculate shortest burst time
i++
No
Yes
Print process value with Shortest Burst Time
Stop

Advantages

- Reduces average waiting time.
- It provides good CPU utilization than FCFS (First Come First Search).

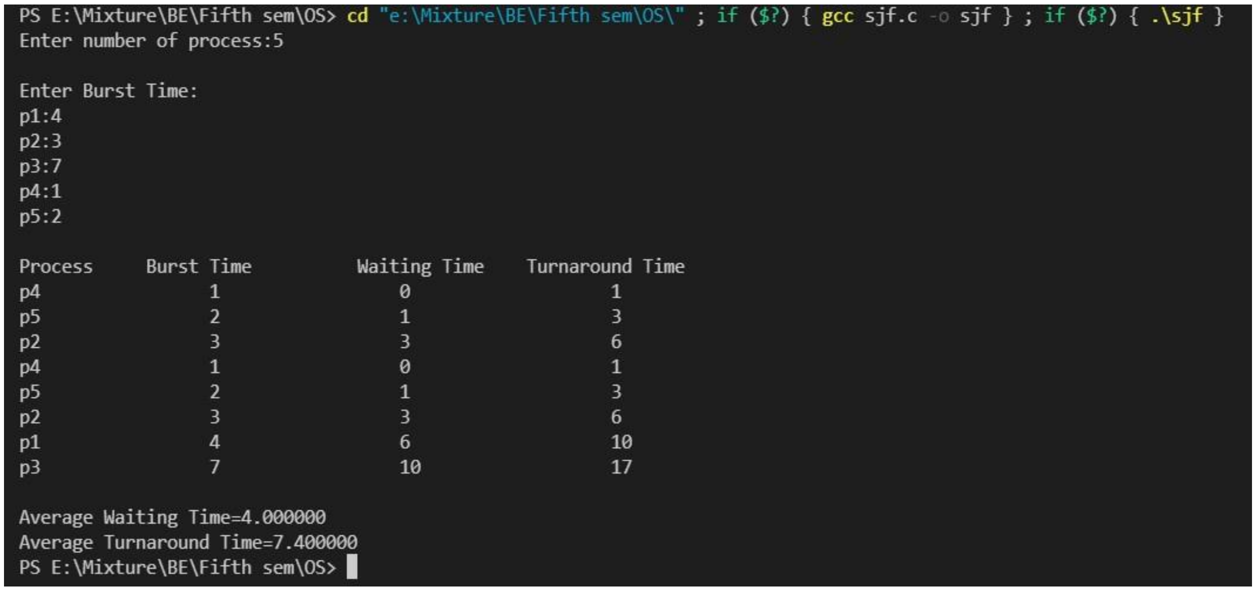
Disadvantages

- **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 time.
- This algorithm may cause very long turn-around time or starvation.

Code

```
#include<stdio.h>
 int main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");
    scanf("%d",&n);
    printf("\nEnter Burst Time:\n");
    for(i=0;i<n;i++)
    {
        printf("p%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;
    }
   //sorting of burst times
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }
        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }
    wt[0]=0;
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }
    avg_wt=(float)total/n;
    total=0;
    printf("\nProcess\t    Burst Time    \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("\np%d\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }
    avg_tat=(float)total/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\nAverage Turnaround Time=%f\n",avg_tat);
}
```

Output



```
PS E:\Mixture\BE\Fifth sem\OS> cd "e:\Mixture\BE\Fifth sem\OS\" ; if ($?) { gcc sjf.c -o sjf } ; if ($?) { .\sjf }
Enter number of process:5

Enter Burst Time:
p1:4
p2:3
p3:7
p4:1
p5:2

Process     Burst Time          Waiting Time    Turnaround Time
p4                1                 0               1
p5                2                 1               3
p2                3                 3               6
p4                1                 0               1
p5                2                 1               3
p2                3                 3               6
p1                4                 6               10
p3                7                 10              17

Average Waiting Time=4.000000
Average Turnaround Time=7.400000
PS E:\Mixture\BE\Fifth sem\OS> 
```

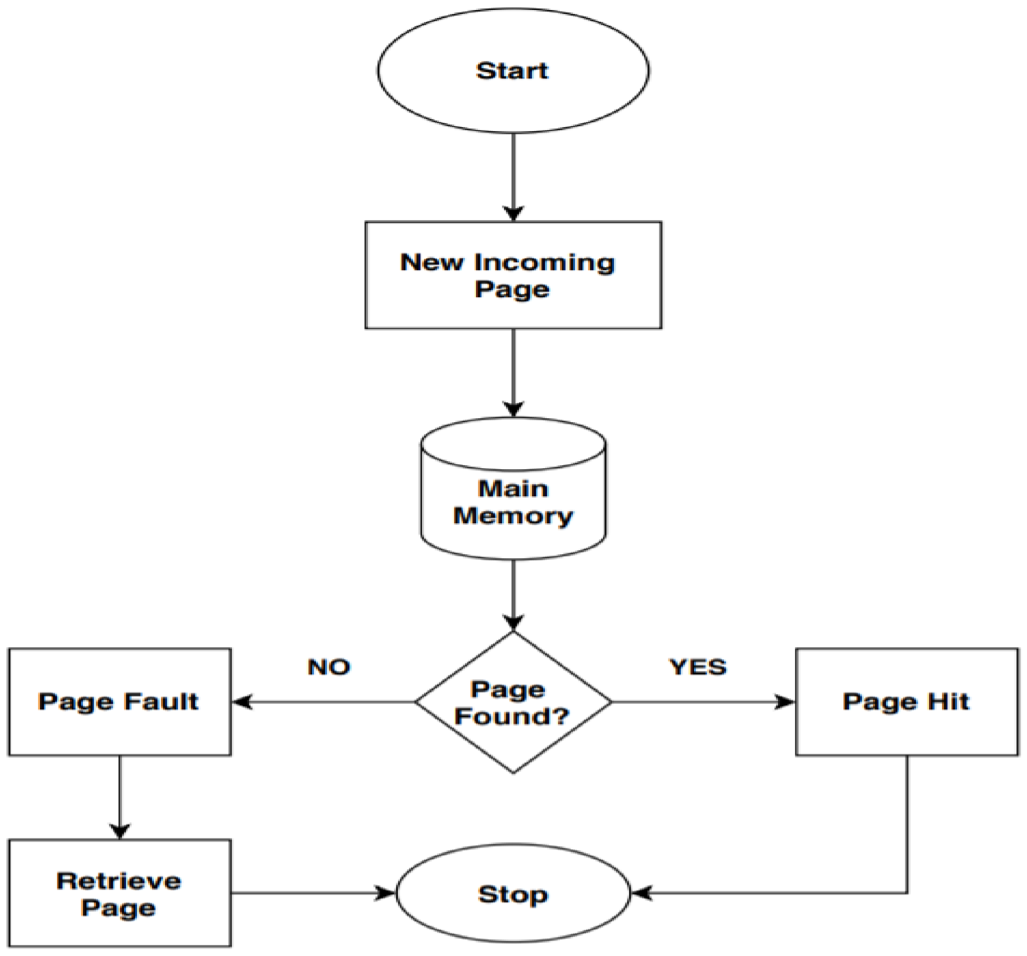
FIFO (First come first out) Page Replacement Algorithm

Introduction

In operating system that uses paging for memory management, page replacement algorithm are needed to decide which page needed to be replaced when new page comes in, which is not present in memory. Page fault occurs and operating system replaces one of the existing pages with new page. Different page replacement algorithms suggest different ways to decide which page to replace. The target of all algorithms is to reduce number of page faults.

This is the simplest page replacement algorithm. In this, 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 the queue is selected for removal.

Flowchart



Advantages

- It is easy to understand and implement.

Disadvantages

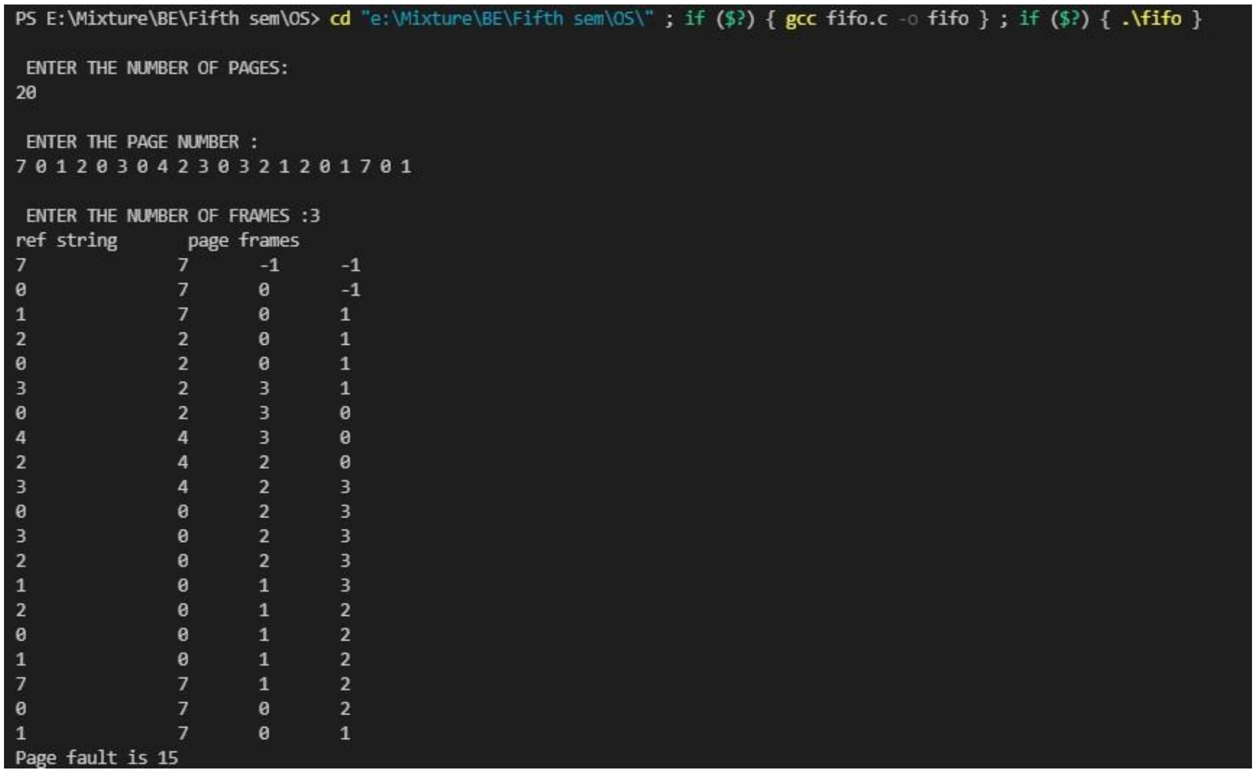
- The efficiency is low.
- When we increase the number of frames while using FIFO, we are giving more memory to processes. So, page faults should decrease, but here the page faults are increasing. This problem is called as **"Belady's Anomaly"**.

Code

```c
#include<stdio.h>

int main()
{
    int i,j,n,a[50],frame[10],no,k,avail,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("ref 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 %d",count);
    return 0;
}
```

Output



```
PS E:\Mixture\BE\Fifth sem\OS> cd "e:\Mixture\BE\Fifth sem\OS\" ; if ($?) { gcc fifo.c -o fifo } ; if ($?) { .\fifo }

 ENTER THE NUMBER OF PAGES:
20

 ENTER THE PAGE NUMBER :
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

 ENTER THE NUMBER OF FRAMES :3
ref string       page frames
7               7       -1      -1
0               7       0       -1
1               7       0       1
2               2       0       1
0               2       0       1
3               2       3       1
0               2       3       0
4               4       3       0
2               4       2       0
3               4       2       3
0               0       2       3
3               0       2       3
2               0       2       3
1               0       1       3
2               0       1       2
0               0       1       2
1               0       1       2
7               7       1       2
0               7       0       2
1               7       0       1
Page fault is 15
```