THE NATIONAL INSTITUTE OF ENGINEERING, MYSURU

(An Autonomous Institute under VTU, Belagavi)

Bachelor of Engineering

in

Computer Science and Engineering

Operating Systems (CS5C02)

Semester: 5

Submitted by

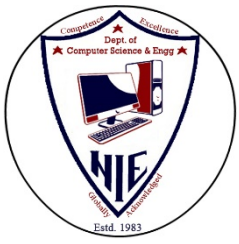
Sathkeerthi Y Agnihotri 4NI19CS099

Shyam Prasad M 4NI19CS106

Course Instructor:

Dr. Jayasri B S

(Professor and Dean – EAB)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

2021-2022

Table of Contents

FCFS:

LRU:

FCFS Scheduling Algorithm

What is FCFS?

First Come First Serve (FCFS) is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival. It is the easiest and simplest CPU scheduling algorithm. In this type of algorithm, processes which requests the CPU first get the CPU allocation first. This is managed with a FIFO queue.

Why is FCFS Scheduling Algorithm Useful?

1. Very simple, and they operate on the order of arrival time, no switching and complications.
2. Easy to implement into any system.
3. As calculating the algorithm for orders is so simple, it is easy for team members of any experience or skill level to manage order scheduling and write and understand code with ease, thus easy to implement.

Some disadvantages of FCFS scheduling:

1. Any process that has low execution time suffers, since waiting time is long.
2. Lower CPU and resource utilization, i.e not efficient.
3. Process with large burst time will keep the resources for long time, thus other processes till have to wait for long time unnecessarily (Convey Effect).
4. For time-sharing systems, it is essential that each user get a share of the CPU at regular intervals. Because FCFS is a non-preemptive system, this is not always feasible.

Algorithm:

Step1: Take the set of processes as the input with the corresponding arrival time and burst time.

Step2: Sort these processes based on their arrival time in ascending order**.**

Step3: The process with least arrival time is executed completely and corresponding completion time, turn-around time and waiting time are updated.

Turn-around Time = Completion Time - Arrival Time

Waiting Time = Turn Around Time - Burst Time

Step4: The Process that comes first will be executed first and next process starts only after the previous process gets fully executed.

Step5: Repeat Step4 until all the processes are executed completely.

Step6: Find the average turn-around time, average waiting time.

Average Turn-Around Time = Sum of all turn-around time/no. of processes

Average Waiting Time = Sum of all waiting time/no. of processes

Implementation:

```
#include <iostream>
#include <algorithm>
#include <iomanip>

using namespace std;

struct Schedule
{
    string pro_id;
    int artime,bt,ct,tat,wt;
    // artime = Arrival time,
    // bt = Burst time,
    // ct = Completion time,
    // tat = Turn around time,
    // wt = Waiting time
};

bool compare(Schedule p1,Schedule p2)
{
    return p1.artime < p2.artime;
    /* To sort the processes according to the order of their Arrival Times */
}
```

Implementation:

```
bool revind(Schedule p1,Schedule p2)
{
    return p1.pro_id < p2.pro_id;
    /* To revert back the processes in the order of process id */
}

int main()
{

    Schedule process[100];
    //An array of Processes

    int cpunon=0;

    int n,i;
    //n = number of processes, i= iteration variable

    float percentage;
    cout << "------\nINPUT\n------" << flush;
    cout << "\nEnter the number of processes: " << flush;
    cin >> n;

    cout << "Enter the Process id, Arrival time and Burst time of the " << n << " processes :" << endl;
    for(i=0;i<n;i++)
    {
        cin>>process[i].pro_id;
        cin >> process[i].artime;
        cin >> process[i].bt;
    }

    sort(process,process+n,compare);
    /* Sort is a predefined function defined in algorithm.h header file,
    it will sort the processes according to their arrival times */
    cpunon=process[0].artime-0;

    // initial values

    process[0].ct=process[0].artime+process[0].bt;
    process[0].tat=process[0].ct-process[0].artime;
    process[0].wt=0;

    for(i=1;i<n;i++)
    {
        if(process[i].artime<=process[i-1].ct)
        {
            process[i].ct=process[i-1].ct+process[i].bt;
            process[i].tat=process[i].ct-process[i].artime;
            process[i].wt=process[i].tat-process[i].bt;
        }
        else
        {
            process[i].ct=process[i].bt+process[i].artime;
            process[i].tat=process[i].ct-process[i].artime;
            process[i].wt=process[i].tat-process[i].bt;
            cpunon+=process[i].artime-process[i-1].ct;
        }
    }
```

Implementation:

```
    percentage=(float)((process[n-1].ct-cpunon)/process[n-1].ct)*100;

    cout<< endl << "-------\nOUTPUT\n-------" << endl;
    sort(process, process+n, revind);
    cout << "\n\tP#\t AT\t BT\t CT\t TAT\t WT\t\n\n" ;
    double total_tat = 0.0, total_wt = 0.0;
    for(i=0;i<n;i++)
    {
        //keep all statements in one line
        cout<<"\t"<<process[i].pro_id<<"\t "<<process[i].artime<<"\t "
        <<process[i].bt<<"\t "<<process[i].ct<<"\t "
        <<process[i].tat<<"\t "<<process[i].wt;
        total_tat += process[i].tat;
        total_wt += process[i].wt;

        cout<<endl;
    }
    cout << fixed << setprecision(2)<< endl;
    cout<<"Average Turn Around Time: "<<(total_tat/n)<<"\n";
    cout<<"Average Waiting Time: "<<(total_wt/n)<<"\n";
    return 0;
}
```

Output:

```
------
INPUT
------
Enter the number of processes: 5
Enter the Process id, Arrival time and Burst time of the 5 processes :
1 3 4
2 5 3
3 0 2
4 5 1
5 4 3

-------
OUTPUT
-------

       P#       AT       BT       CT       TAT      WT

       1        3        4        7        4        0
       2        5        3        13       8        5
       3        0        2        2        2        0
       4        5        1        14       9        8
       5        4        3        10       6        3

Average Turn Around Time: 5.80
Average Waiting Time: 3.20
```

Least Recently Used Page Replacement Algorithm

What is LRU Page Replacement Algorithm?

This algorithm stands for "Least Recently Used" and this algorithm helps the operating system to search those pages that are used over a short duration of time frame. It works on the concept that pages that have been highly used in the past are likely to be significantly used again in the future. It removes the page that has not been utilized in the memory for the longest time.

Advantages of LRU:

1.LRU doesn't suffer from Belady's Anomaly.
2.The page in the main memory that hasn't been used in the longest will be chosen for replacement.
3.It gives fewer page faults than any other page replacement algorithm. So, LRU is the most commonly utilized method.
4.It is a very effective algorithm.
5.It helps in the full analysis.

Disadvantages of LRU:

1.LRU's weakness is that its performance tends to degenerate under many quite common reference patterns.
2.It isn't easy to implement because it requires hardware assistance.
3.It is expensive and more complex.
4.It needs an additional data structure.

Algorithm:

We set ‘nopages’ to be the number of pages that memory can hold. Let frame array be the current set of pages in memory and let count hold number of page faults.

Traversing the pages:

If {frame array holds less pages than nopages},

1) Insert page into the frame array one by one until the size of frame array reaches capacity or all page requests are processed.
2) Simultaneously we maintain the recent occurred index of each page in an array called fcount.
3)We then increment count.

Else If {current page is present in frame array },we do nothing.

Else

1) We find the page in the frame array that was least recently used. We find it using fcount array. We basically need to replace the page with minimum index.
2) Then we replace the found page with current page.
3) We then increment count.
4) Then update index of current page in fcount.

Calculating number of hits, Hit ratio, Miss ratio:

Number of hits = nopages - count

Hit ratio = (nopages - count)*100/nopages

Miss ratio= (Count) *100/nopages

Implementation:

```
#include<iostream>
using namespace std;
int main()
{
    int nopages, nofaults, page[20],i ,count=0;
    cout<<"Enter the Number of pages :\t";
    cin>> nopages;  //it will store the number of Pages
    cout<<"\nEnter the Reference String :\t";
    for(i=0;i< nopages;i++)
    {
        cout<<"";
        cin>>page[i];
    }
    cout<<"\nEnter the Number of frames:\t";
    cin>> nofaults;
    int frame[nofaults],fcount[nofaults];
    for(i=0;i< nofaults;i++)
    {
        frame[i]=-1;
        fcount[i]=0; //it will keep the track of when the page was last used
    }
    i=0;
    while(i< nopages)
    {
        int j=0,flag=0;
        while(j< nofaults)
        {
            if(page[i]==frame[j])  //it will check whether the page already exist in frames or not
            {
                flag=1;
                fcount[j]=i+1;
            }
            j++;
        }
        j=0;
        cout<<"\n\t\n";
        cout<<"\t"<<page[i]<<"  ->";
        if(flag==0)
        {
            int min=0,k=0;
            while(k<nofaults-1)
            {
                if(fcount[min]>fcount[k+1]) //It will calculate the page which is least recently used
                min=k+1;
                k++;
            }
            frame[min]=page[i];
            fcount[min]=i+1;  //Increasing the time
            count++;          //it will count the total Page Fault
            while(j< nofaults)
            {
                cout<<"\t|"<<frame[j]<<"|";
                j++;
            }
        }
        i++;
    }
```

Implementation:

```
    cout<<"\n\nNumber of Page Faults :"<<count;
    cout<<"\n\nNumber of Hits :"<<nopages - count;
    cout<<"\n\nHit Ratio: "<<(nopages-count)*100/nopages<< "%" << endl;
    cout<<"\n\nMiss Ratio: "<<(count)*100/nopages<< "%" << endl;
    cout<<"\n";
    return 0;
}
```

Output:

i) Output of LRU for 3 frames

```
Enter the Number of pages :     12

Enter the Reference String :    1 3 2 4 2 3 1 4 2 4 1 3

Enter the Number of frames:     3


       1  ->   |1|     |-1|    |-1|
       3  ->   |1|     |3|     |-1|
       2  ->   |1|     |3|     |2|
       4  ->   |4|     |3|     |2|
       2  ->
       3  ->
       1  ->   |1|     |3|     |2|
       4  ->   |1|     |3|     |4|
       2  ->   |1|     |2|     |4|
       4  ->
       1  ->
       3  ->   |1|     |3|     |4|
Number of Page Faults :8

Number of Hits :4

Hit Ratio: 33%

Miss Ratio: 66%
```

Output:

ii) Output of LRU for 4 frames

```
Enter the Number of pages :     12

Enter the Reference String :    2 3 2 1 5 2 4 5 3 2 5 2

Enter the Number of frames:     4

       2  ->   |2|     |-1|    |-1|    |-1|

       3  ->   |2|     |3|     |-1|    |-1|

       2  ->

       1  ->   |2|     |3|     |1|     |-1|

       5  ->   |2|     |3|     |1|     |5|

       2  ->

       4  ->   |2|     |4|     |1|     |5|

       5  ->

       3  ->   |2|     |4|     |3|     |5|

       2  ->

       5  ->

       2  ->

Number of Page Faults :6

Number of Hits :6

Hit Ratio: 50%

Miss Ratio: 50%
```

GitHub Link:

https://github.com/5th-sem-OS-tut/FCFS-LRU