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 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

Shamanth G 4NI19CS102

Shubham M Sastekar 4NI19CS105

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, Belgavi)**



***CERTIFICATE***

This is to certify the work carried out by Shamanth G (4NI19CS102), Shubham M Sastekar (4NI19CS105), 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)

Contents

[2 SHORTEST JOB FIRST – CPU SCHEDULING ALGORITHM 4](#_Toc91517105)

[2.1 CPU Scheduling Algorithm: 4](#_Toc91517106)

[2.2 NON-PREEMPTIVE PRIORITY: 4](#_Toc91517107)

[2.3 Algorithm: 4](#_Toc91517108)

[2.4 Advantages and Disadvantages: 4](#_Toc91517109)

[2.5 Example: 5](#_Toc91517110)

[2.6 Code 6](#_Toc91517111)

[2.7 Output 8](#_Toc91517112)

[3 FIFO - PAGE REPLACEMENT ALGORITHM 8](#_Toc91517113)

[3.1 Page Replacement Algorithm: 8](#_Toc91517114)

[3.2 FIFO 8](#_Toc91517115)

[3.3 Advantages and Disadvantages 9](#_Toc91517116)

[3.4 Algorithm: 9](#_Toc91517117)

[3.5 Example 10](#_Toc91517118)

[3.6 Code: 10](#_Toc91517119)

[3.7 Output: 13](#_Toc91517120)

[4 GITHUB Links 13](#_Toc91517121)

# NON PRIEMPTIVE PRIORITY– CPU SCHEDULING ALGORITHM

## 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-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the pre-emptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

## SJF:

Shortest JOB FIRST (SJF/SJN) is the non-preemptive algorithm, where the processor is allocated to the job closest to completion.

## Algorithm:

1- Traverse until all process gets completely executed.

a) Find process with minimum burst time at every single time lap.

b)Add that burst time to the gantt chart completely.(non-pre-emptive)

c) Check the next minimum burst time and repeat the process.

d) Finally ,after all the process gets allocated calculate the following.

e) Completion time of current process = current\_time +1;

f) Calculate waiting time for each completed process. wt[i]= Completion time - arrival\_time-burst\_time

g) Increment time lap by one.

2- Find turnaround time,waiting time and response time.

3-LOGIC: lower the burst time higher the priority,when in case of same BT,FCFS is considered.

completed = 0

current\_time = 0

while(completed != n) {

find process with minimum b urst time among process that are in ready queue at current\_time

if(process found) {

start\_time = current\_time

completion\_time = start\_time + burst\_time

turnaround\_time = completion\_time - arrival\_time

waiting\_time = turnaround\_time - burst\_time

response\_time = start\_time - arrival\_time

mark process as completed

completed++

current\_time = completion\_time

}

else {

current\_time++

}

}

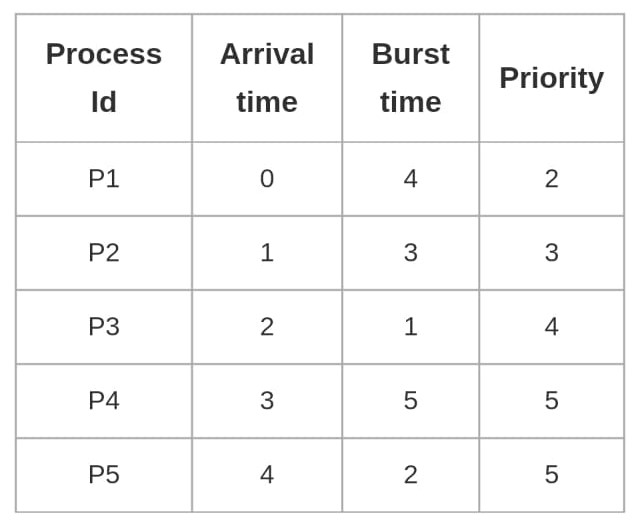
## Advantages and Disadvantages:

**Advantages**: Shortest jobs are favoured. It is provably optimal, in that it gives the minimum average waiting time for a given set of processes.

**Disadvantages**: SJF may cause starvation, if shorter processes keep coming. This problem is solved by aging.

## Example:

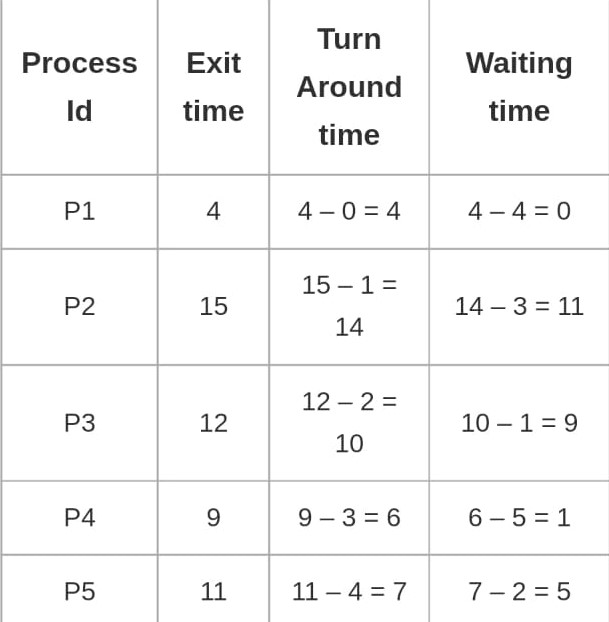
Suppose we have the following 4 processes with process ID's P0,P1, P2,P3 and they arrive into the CPU in the following manner:



Gant Chart:

Diagram

Description automatically generated



**Explanation**: At time 0 process P1 arrives and starts execution.At time 1 P3 arrives but P1 needs still time to complete the process.Also at time 4,5 process P2,P0 arrives nut since its nonpre-emptive all process waits in ready queue.Now after P1 execution at time 5 ,between P2,P0,P3 ,P3 has lowest burst time hence the process is scheduled,followed by P2 as p0 has not arrived at BT 7.Finally P2 gets executed.

Total Turn Around Time = 16+5+12+6 = 39 milliseconds

Average Turn Around Time= Total Turn Around Time / Total No. of Processes = 39 / 4

= 9.75 millisecond

Total Waiting Time = 8 + 0 + 3 +4 = 15 milliseconds

Average Waiting Time = Total Waiting Time / Total No. of Processes = 15 / 4

= 3.75 milliseconds

CODE:

#include<bits/stdc++.h>

|  |  |
| --- | --- |
| using namespace std;  struct process {  int pid;  int arrival\_time;  int burst\_time;  int start\_time;  int completion\_time;  int turnaround\_time;  int waiting\_time;  int response\_time;  };  int main(){  int n;  struct process p[100];  float avg\_turnaround\_time;  float avg\_waiting\_time;  float avg\_response\_time;  int total\_turnaround\_time = 0;  int total\_waiting\_time = 0;  int total\_response\_time = 0;  int is\_completed[100];  memset(is\_completed,0,sizeof(is\_completed));  cout<<"Enter the number of processes: ";  cin>>n;  for(int i = 0; i < n; i++) {  cout<<"Enter arrival time of process "<<i+1<<": ";  cin>>p[i].arrival\_time;  cout<<"Enter burst time of process "<<i+1<<": ";  cin>>p[i].burst\_time;  p[i].pid = i+1;  cout<<endl;  }  int current\_time = 0;  int completed = 0;  while(completed != n) {  int idx = -1;  int mn = 10000000;  for(int i = 0; i < n; i++) {  if(p[i].arrival\_time <= current\_time && is\_completed[i] == 0) {  if(p[i].burst\_time < mn) {  mn = p[i].burst\_time;  idx = i;  }  if(p[i].burst\_time == mn) {  if(p[i].arrival\_time < p[idx].arrival\_time) {  mn = p[i].burst\_time;  idx = i;  }  }  }  }  if(idx != -1) {  p[idx].start\_time = current\_time;  p[idx].completion\_time = p[idx].start\_time + p[idx].burst\_time;  p[idx].turnaround\_time = p[idx].completion\_time - p[idx].arrival\_time;  p[idx].waiting\_time = p[idx].turnaround\_time - p[idx].burst\_time;  p[idx].response\_time = p[idx].start\_time - p[idx].arrival\_time;    total\_turnaround\_time += p[idx].turnaround\_time;  total\_waiting\_time += p[idx].waiting\_time;  total\_response\_time += p[idx].response\_time;  is\_completed[idx] = 1;  completed++;  current\_time = p[idx].completion\_time;  }  else {  current\_time++;  }  }  avg\_turnaround\_time = (float) total\_turnaround\_time / n;  avg\_waiting\_time = (float) total\_waiting\_time / n;  avg\_response\_time = (float) total\_response\_time / n;  cout<<endl<<endl;  cout<<"Process ID\t"<<"AT\t"<<"BT\t"<<"ST\t"<<"CT\t"<<"TAT\t"<<"WT\t"<<"RT\t"<<"\n"<<end;  for(int i = 0; i < n; i++) {  cout<<"P"<<p[i].pid<<"\t\t"<<p[i].arrival\_time<<"\t"<<p[i].burst\_time<<"\t"<<p[i].start\_time<<"\t"<<p[i].completion\_time<<"\t"<<p[i].turnaround\_time<<"\t"<<p[i].waiti ng\_time<<"\t"<<p[i].response\_time<<"\t"<<"\n"<<endl;  }  cout<<"Average Turnaround Time = "<<avg\_turnaround\_time<<endl;  cout<<"Average Waiting Time = "<<avg\_waiting\_time<<endl;  cout<<"Average Response Time = "<<avg\_response\_time<<endl;    return 0;  } |  |
| 2.7 Output |  |
|  |  |
|  |  |

# FIFO- PAGE REPLACEMENT ALGORITHM

## 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.

## FIFO

FIFO which is also called First In First Out is one of the types of Replacement Algorithms. This algorithm is used in a situation where an Operating system replaces an existing page with the help of memory by bringing a new page from the secondary memory.

FIFO is the simplest among all algorithms which are responsible for maintaining all the pages in a queue for an operating system and also keeping track of all the pages in a queue.

The older pages are kept in the front and the newer ones are kept at the end of the queue. Pages that are in the front are removed first and the pages which are demanded are added.

## Advantages and Disadvantages

**Advantages :**

* It is simple and easy to understand & implement.

**Disadvantage :**

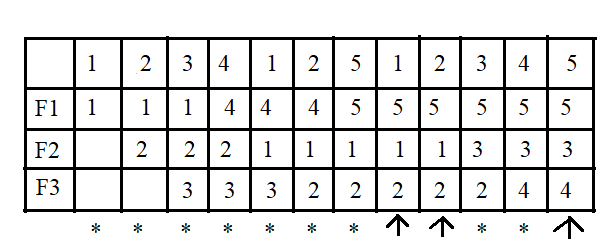
* The process effectiveness is low.
* When we increase the number of frames while using FIFO, we are giving more memory to processes. So, page fault should decrease, but here the page faults are increasing.
* Every frame needs to be taken account off.

## Algorithm:

1. Start traversing the pages.
2. Now declare the size w.r.t length of the Page.
3. Check need of the replacement from the page to memory.
4. Similarly, Check the need of the replacement from the old page to new page in memory.
5. Now form the queue to hold all pages.
6. Insert Require page memory into the queue.
7. Insert Require page memory into the queue.
8. Check bad replacement and page faults.
9. Check bad replacements and page faults.
10. Get no of processes to be inserted.
11. Get no of processes to be inserted.
12. Show the values.
13. Stop the process.

## Example

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



Hit Ratio = 3/12 

Miss Ratio = 9/12

No.of page faults=9

## Code:

#include<bits/stdc++.h>

using namespace std;

int main()

{

int i,j,n,ref\_str[50],frame[10],no,k,avail;

float fcount=0;

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

scanf("%d",&n);

printf("\n ENTER THE REFERENCE STRING :\n");

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

scanf("%d",&ref\_str[i]);

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

scanf("%d",&no);

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

frame[i]= -1;

j=0;

printf("\n ref string \t page frames \tHit/Fault\n");

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

{

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

avail=0;

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

{

if(frame[k]==ref\_str[i])

{

avail=1;

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

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

printf("H");

}

}

if (avail==0)

{

frame[j]=ref\_str[i];

j=(j+1)%no;

fcount++;

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

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

printf("F");

}

printf("\n");

}

printf("Page Fault Is %.1f\n",fcount);

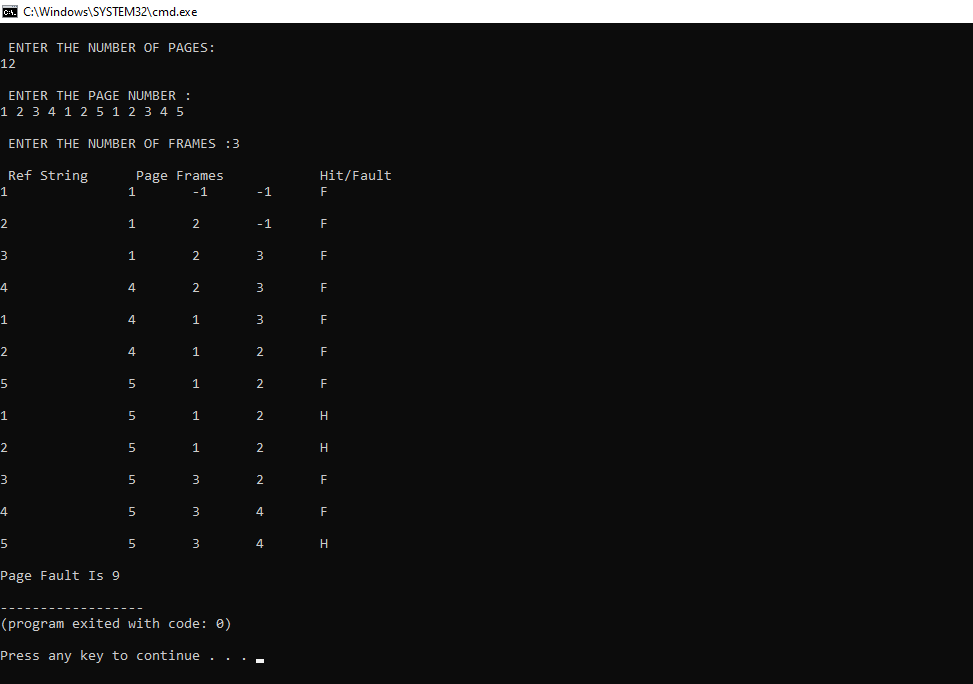
cout<<"Hit ratio: "<<((n-fcount)/n)<<"\n";

cout<<"Miss ratio: "<<(fcount/n);

return 0;

}

## Output



# GITHUB link

Shamanth G (4NI19CS102): <https://github.com/Shamanth-123/OS-tutorial>

Shubham M Sastekar (4NI19CS105): <https://github.com/Shubhamsastekar/OS_Tutorial>