THE NATIONAL INSTITUTE OF ENGINEERING, MYSURU

(AN AUTONOMOUS INSTITUTE UNDER VTU, BELAGAVI)

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

Operating System

Semester 5

Computer Science and Engineering

Submitted by

T PRASANNA -- 4NI19CS111

YELLALA MANEESH REDDY -- 4NI19CS124

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 T PRASANNA (4NI19CS111), YELLALA MANEESH REDDY (4NI19CS124) in partial fulfillment 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 Course Instructor

_____

(Dr JAYASRI B S)

1 CONTENTS

2 PRIORITY – CPU SCHEDULING ALGORITHM (NON-PREEMPTIVE)

2.1 CPU SCHEDULING ALGORITHM:

CPU Scheduling is a process of determining which process will own CPU for execution while another process is on hold. The main task of CPU scheduling is to make sure that whenever the CPU remains idle, the OS at least select one of the processes available in the ready queue for execution. The selection process will be carried out by the CPU scheduler. It selects one of the processes in memory that are ready for execution. These algorithms are either non-preemptive or preemptive. 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 preemptive 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.

2.2 PRIORITY:

In Priority Scheduling, processes are assigned Priorities and the process with the highest is executed first. Each process is assigned a priority and processes are given CPU time according to their priority.

- If 2 processes have equal priorities the those process are scheduled in First Come First Serve order.
- Priority can be defined internally or externally.
- It can be either preemptive or non preemptive algorithm.

2.3 ALGORITHM:

- First input the processes with their arrival time, burst time and priority.
- First process will schedule, which have the lowest arrival time, if two or more processes will have lowest arrival time, then whoever has higher priority will schedule first.
- Now further processes will be schedule according to the arrival time and priority of the process. (Here we are assuming that lower the priority number

having higher priority). If two process has arrived and priority are same then sort according to burst time .

- Once all the processes have been arrived, we can schedule them based on their priority.

2.4 ADVANTAGES AND DISADVANTAGES:

Advantages: Priority Scheduling algorithm provides a good mechanism where the relative importance of each process maybe precisely defined.

Disadvantages: If high priority processes use up a lot of CPU time, lower priority processes may starve and be postponed indefinitely. The situation where a process never gets scheduled to run is called **starvation**. To prevent this, we do aging where the priority of the process is increased as it waits in the queue. So, a process which has been in a queue for a long time will reach high priority, hence it won't be starved.

2.5 EXAMPLE

Note – Consider lower number to have higher priority.

Process	Arrival Time	Burst Time	Priority
P1	0	11	2
P2	5	28	0
P3	12	2	3
P4	2	10	1
P5	9	16	4

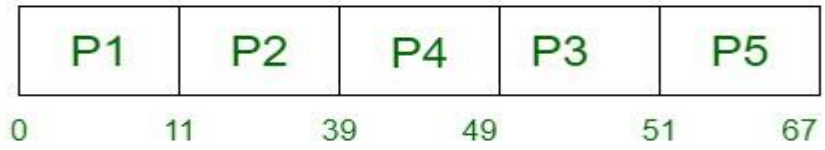
STEPS-

1. At time t = 0, there is only process that has arrived i.e., P1 so it will execute first for next 11 ms.

2. At time t = 11, P1 is completed and there are three more processes that have arrived, P2 ,P4,P5
    1. P2 has priority 0 and P4 has 1 and P5 has 4
    2. Thus, P2 will execute for next 28 ms, i.e. till t = 39 ms.
    3. At time t = 39ms, P2 has just completed and P3 has arrived

3. At time t = 39ms all the processes have arrived.
4. Thus, P4 with priority 1 will be next
5. Similarly, P3 → P5
6. P1→P2→P4→P3→P5

The Gantt Chart will look like –



TIME →

Turn Around Time = Completion Time – Arrival Time

Waiting Time = Turn Around Time – Burst Time

- Waiting Time for P1 = 11-0-11 =0
- Waiting Time for P2= 39-5-28 =6
- Waiting Time for P3= 51-12-2 =37
- Waiting Time for P4= 49-2-10 =37
- Waiting Time for P5= 67-9-16 =42

Average Wait Time= (0+6+37+37+42)/5 =24.4

Average Turn Around Time= (11+34+39+47+59)/5 =38

Average Response Time= (0+6+37+37+42)/5 =24.

In a non preemptive scheduling algorithm Wait Time (WT) and Response Time (RT) are always same.

Process	Execution Time (Burst)	Priority	Arrival Time	Completion Time	Turnaround Time	Waiting Time	Response Time
P1	11	2	0	11	11	0	0
P2	28	0	5	39	34	6	6
P3	2	3	12	51	39	37	37
P4	10	1	2	49	47	37	37
P5	16	4	9	67	58	42	42

2.6 CODE

```
#include <iostream>
using namespace std;
int main()
{
   int a[10], b[10], x[10], pr[10], rt[10] = {0};
   int waiting[10], turnaround[10], completion[10];
   int i, j, smallest, count = 0, time, n;
   double avg = 0, tt = 0, end;
   cout << "\nEnter the number of Processes: ";
   cin >> n;
   for (i = 0; i < n; i++)
   {
      cout << "\nEnter arrival time of process: ";
```

```
    cin >> a[i];
  }
  for (i = 0; i < n; i++)
  {
    cout << "\nEnter burst time of process: ";
    cin >> b[i];
  }
  for (i = 0; i < n; i++)
  {
    cout << "\nEnter priority of process: ";
    cin >> pr[i];
  }
  for (i = 0; i < n; i++)
    x[i] = b[i];

  pr[9] = 100;
  for (time = 0; count != n; time++)
  {
    smallest = 9;
    for (i = 0; i < n; i++)
    {
      if (a[i] <= time && pr[i] < pr[smallest] && b[i] > 0)
        smallest = i;
    }
    time += b[smallest];
    b[smallest] = -1;
    count++;
    end = time;
    completion[smallest] = end;
    waiting[smallest] = end - a[smallest] - x[smallest];
```

```
        turnaround[smallest] = end - a[smallest];


        time = time - 1;
    }
    cout << "Process"
         << "\t"
         << "burst-time"
         << "\t"
         << "arrival-time"
         << "\t"
         << "waiting-time"
         << "\t"
         << "turnaround-time"
         << "\t"
         << "completion-time"
         << "\t"
         << "Priority" << endl;
    for (i = 0; i < n; i++)
    {
        cout << "p" << i + 1 << "\t\t" << x[i] << "\t\t" << a[i] << "\t\t" << waiting[i] << "\t\t" <<
turnaround[i] << "\t\t" << completion[i] << "\t\t" << pr[i] << endl;
        avg = avg + waiting[i];
        tt = tt + turnaround[i];
    }
    cout << "\n\nAverage waiting time =" << avg / n;
    cout << "  Average Turnaround time =" << tt / n << endl;
}
```

2.7 OUTPUT

Enter the number of Processes: 5

Enter arrival time of process: 0

Enter arrival time of process: 5

Enter arrival time of process: 12

Enter arrival time of process: 2

Enter arrival time of process: 9

Enter burst time of process: 11

Enter burst time of process: 28

Enter burst time of process: 2

Enter burst time of process: 10

Enter burst time of process: 16

Enter priority of process: 2

Enter priority of process: 0

Enter priority of process: 3

Enter priority of process: 1

Enter priority of process: 4

Process	burst time	arrival-time	waiting time	turnaround-time	completion-time	Priority
P1	11	0	0	11	11	2
P2	28	5	6	34	39	0
P3	2	12	37	39	51	3
P4	10	2	37	47	49	1
P5	16	9	42	58	67	4

Average Wait Time=24.4

Average Turn Around Time = 37.8

3 FIFO - PAGE REPLACEMENT ALGORITHM

3.1 PAGE REPLACEMENT ALGORITHM:

The operating system uses the method of paging for memory management. This method involves page replacement algorithms to decide about which pages should be replaced when new pages are demanded. The demand occurs when the operating system needs a page for processing, and it is not present in the main memory. The situation is known as a page fault. In this situation, the operating system replaces an existing page from the main memory by bringing a new page from the secondary memory. 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.

3.2 FIFO

In such situations, the FIFO method is used, which is also refers to the First in First Out concept. This is the simplest page replacement method in which the operating system maintains all the pages in a queue. Oldest pages are kept in the front, while the newest is kept at the end. On a page fault, these pages from the front are removed first, and the pages in demand are added.

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.

3.3 ADVANTAGES AND DISADVANTAGES

Advantages :

- It is simple and easy to understand & implement.

Disadvantage :

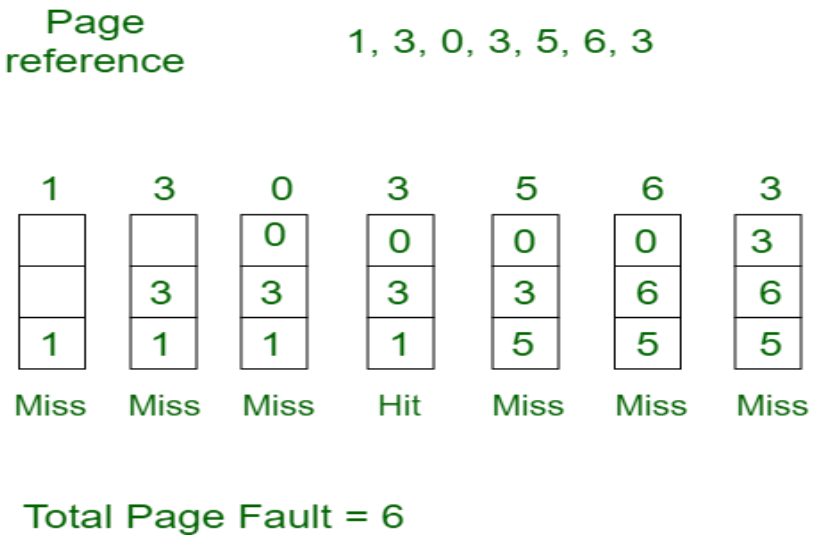
- The process effectiveness is low.
- When we increase the number of frames while using FIFO, we are giving more memory to processes. So, page fault should decrease, but here the page faults are increasing. This problem is called as **Belady's Anomaly**
- Every frame needs to be taken account off.

3.4 ALGORITHM:

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

- Step 1. Start to traverse the pages.
- Step 2. If the memory holds fewer pages, then the capacity else goes to step 5.
- Step 3. Push pages in the queue one at a time until the queue reaches its maximum capacity, or all page requests are fulfilled.
- Step 4. If the current page is present in the memory, do nothing.
- Step 5. Else, pop the topmost page from the queue as it was inserted first.
- Step 6. Replace the topmost page with the current page from the string.
- Step 7. Increment the page faults.
- Step 8. Stop

3.5 EXAMPLE



Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots → **3 Page Faults.**

when 3 comes, it is already in memory so → **0 Page Faults.**

Then 5 comes, it is not available in memory, so it replaces the oldest page slot i.e 1. →**1 Page Fault.**

6 comes, it is also not available in memory, so it replaces the oldest page slot i.e 3 →**1 Page**

Fault.

Finally, when 3 come it is not available, so it replaces 0→**1 page fault**

Belady's anomaly – Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10-page faults.

3.6 CODE:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int referenceString[10], pageFaults = 0, m, n, s, pages, frames;
    printf("\nEnter the number of Pages: ");
    scanf("%d", &pages);
    printf("\nEnter reference string values:\n");
    for(m = 0; m < pages; m++)
    {
        printf("Value No. [%d]:\t", m + 1);
        scanf("%d", &referenceString[m]);
    }
    printf("\nWhat are the total number of frames: ");
    {
        scanf("%d", &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(referenceString[m] == temp[n])
        {
            s++;
            pageFaults--;
        }
    }
    pageFaults++;
    if((pageFaults <= frames) && (s == 0))
    {
        temp[m] = referenceString[m];
    }
    else if(s == 0)
    {
        temp[(pageFaults - 1) % frames] = referenceString[m];
    }
    printf("\n");
    for(n = 0; n < frames; n++)
    {
        printf("%d\t", temp[n]);
```

```
 }
    }
    printf("\nTotal Page Faults: %d\n", pageFaults);
    return 0;
}
```

3.7 OUTPUT:

Enter the number of Pages: 5

Enter reference string values:

Value No. [1]: 4

Value No. [2]: 1

Value No. [3]: 2

Value No. [4]: 4

Value No. [5]: 5

What are the total number of frames: 3

4 -1 -1

4 1 -1

4 1 2

4 1 2

5 1 2

Total Page Faults: 4

4 GITHUB LINKS

T PRASANNA (4NI19CS111) -- https://github.com/Tprasanna47/ostutorial.git

YELLALA MANEESH REDDY (4NI19CS124) -- https://github.com/Maneeshreddy631/OS-Tutorial