THE NATIONAL INSTITUTE OF ENGINEERING, MYSURU

(AN AUTONOMOUS INSTITUTE UNDER VTU, BELAGAVI)

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

Operating System

Semester 5

Computer Science and Engineering

Submitted by

SHALINI SG -- 4NI19CS101

SAUMYA NIGAM – 4NI19CS100

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 SHALINI SG (4NI19CS101),

SAUMYA NIGAM (4NI19CS100) 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)

Associate Professor

1   CONTENTS

2 ROUND ROBIN – CPU SCHEDULING ALGORITHM

2.1 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-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 ROUND ROBIN:

A round-robin scheduling algorithm is used to schedule the process fairly for each job a time slot or quantum and the interrupting the job if it is not completed by then the job come after the other job which is arrived in the quantum time that makes these scheduling fairly. Round-robin is cyclic in nature, so starvation doesn't occur. Round-robin is a variant of first come, first served scheduling. No priority, special importance is given to any process or task. RR scheduling is also known as Time slicing scheduling

2.3 IMPLEMENTATION:

1. Declare arrival[], burst[], wait[], turn[] arrays and initialize them. Also declare a timer variable and initialize it to zero. To sustain the original burst array create another array (temp_burst[]) and copy all the values of burst array in it.

2. To keep a check we create another array of bool type which keeps the record of whether process is completed or not. we also need to maintain a queue array which contains the process indicies (initially the array is filled with 0).

3. Now we increment the timer variable until the first process arrives and when it does, we add the process index to the queue array

4. Now we execute the first process until the time quanta and during that time quanta, we check whether any other process has arrived or not and if it has then we add the index in the queue (by calling the fxn. queueUpdation()).

5. Now, after doing the above steps if a process has finished, we store its exit time and execute the next process in the queue array. Else, we move the currently executed process at

the end of the queue (by calling another fxn. queueMaintainence()) when the time slice expires.

6. The above steps are then repeated until all the processes have been completely executed. If a scenario arises where there are some processes left but they have not arrived yet, then we shall wait and the CPU will remain idle during this interval.

2.4 ADVANTAGES AND DISADVANTAGES:

Advantages: Every process gets an equal share of the CPU. RR is cyclic in nature, so there is no starvation.

Disadvantages: Setting the quantum too short, increases the overhead and lowers the CPU efficiency, but setting it too long may cause poor response to short processes. Average waiting time under the RR policy is often long.

2.5 EXAMPLE:

Suppose we have the following 4 processes with process ID's P1, P2, P3 and P4 and they arrive into the CPU in the following manner:

PROCESSES	ARRIVAL TIME	BURST TIME
P1	0	5
P2	1	4
P3	2	2
P4	3	1

Gant Chart:

P1	P2	P3	P1	P4	P2	P1

0　　　2　　　4　　　6　　　8　　9　　　11　　12

Ready Queue :

P1, P2, P3, P1, P4, P2, P1

Table:

PROCESSES	ARRIVAL TIME	BURST TIME	COMPLETION TIME	TURN AROUND TIME	WAIT TIME
P1	0	5	12	12	7
P2	1	4	11	10	6
P3	2	2	6	4	2
P4	3	1	9	6	5

Total Turn Around Time = 12 + 10 + 4 + 6

        = 32 milliseconds

Average Turn Around Time= Total Turn Around Time / Total No. of Processes = 32 / 4

        = 8 milliseconds

Total Waiting Time = 7 + 6 + 2 + 5

        = 20 milliseconds

Average Waiting Time = Total Waiting Time / Total No. of Processes = 20 / 4

        = 5 milliseconds

2.6 CODE

```cpp
#include <iostream>
using namespace std;
void queueUpdation(int queue[],int timer,int arrival[],int n, int maxProccessIndex){
    int zeroIndex;
    for(int i = 0; i < n; i++){
        if(queue[i] == 0){
            zeroIndex = i;
            break;
        }
    }
    queue[zeroIndex] = maxProccessIndex + 1;
}
void queueMaintainence(int queue[], int n){
    for(int i = 0; (i < n-1) && (queue[i+1] != 0) ; i++){
        int temp = queue[i];
```

```cpp
            queue[i] = queue[i+1];

            queue[i+1] = temp;

        }

    }

    void checkNewArrival(int timer, int arrival[], int n, int maxProccessIndex,int queue[]){

        if(timer <= arrival[n-1]){

            bool newArrival = false;

            for(int j = (maxProccessIndex+1); j < n; j++){

                if(arrival[j] <= timer){

                 if(maxProccessIndex < j){

                    maxProccessIndex = j;

                    newArrival = true;

                 }

              }

           }

           //adds the incoming process to the ready queue (if any arrives)

           if(newArrival)

              queueUpdation(queue,timer,arrival,n, maxProccessIndex);

        }

    }

    //Driver Code

    int main(){

        int n,tq, timer = 0, maxProccessIndex = 0;

        float avgWait = 0, avgTT = 0;

        cout << "\nEnter the time quanta : ";

        cin>>tq;

        cout << "\nEnter the number of processess : ";

        cin>>n;

        int arrival[n], burst[n], wait[n], turn[n], queue[n], temp_burst[n];

        bool complete[n];

       cout << "\nEnter the arrival time of the processess : ";
```

```cpp
    for(int i = 0; i < n; i++)

        cin>>arrival[i];

    cout << "\nEnter the burst time of the processess : ";

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

        cin>>burst[i];

        temp_burst[i] = burst[i];

    }

for(int i = 0; i < n; i++){    //Initializing the queue and complete array

        complete[i] = false;

        queue[i] = 0;

    }

    while(timer < arrival[0])    //Incrementing Timer until the first process arrives

        timer++;

    queue[0] = 1;

    while(true){

        bool flag = true;

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

            if(temp_burst[i] != 0){

                flag = false;

                break;

            }

        }

        if(flag)

            break;

            for(int i = 0; (i < n) && (queue[i] != 0); i++){

            int ctr = 0;

            while((ctr < tq) && (temp_burst[queue[0]-1] > 0)){

                temp_burst[queue[0]-1] -= 1;

                timer += 1;

                ctr++;

                //Checking and Updating the ready queue until all the processes arrive
```

```cpp
            checkNewArrival(timer, arrival, n, maxProccessIndex, queue);
        }
        //If a process is completed then store its exit time and mark it as completed
        if((temp_burst[queue[0]-1] == 0) && (complete[queue[0]-1] == false)){
            //turn array currently stores the completion time
            turn[queue[0]-1] = timer;
            complete[queue[0]-1] = true;
        }
         //checks whether or not CPU is idle
bool idle = true;
        if(queue[n-1] == 0){
            for(int i = 0; i < n && queue[i] != 0; i++){
                if(complete[queue[i]-1] == false){
                    idle = false;
                }
            }
        }
        else
            idle = false;
        if(idle){
            timer++;
            checkNewArrival(timer, arrival, n, maxProccessIndex, queue);
        }
        //Maintaining the entries of processes after each premption in the ready Queue
        queueMaintainence(queue,n);
    }
  }
 for(int i = 0; i < n; i++){
     turn[i] = turn[i] - arrival[i];
     wait[i] = turn[i] - burst[i];
  }
```

```cpp
    cout << "\nProgram No.\tArrival Time\tBurst Time\tWait Time\tTurnAround Time"
        << endl;
    for(int i = 0; i < n; i++){
        cout<<i+1<<"\t\t"<<arrival[i]<<"\t\t"
            <<burst[i]<<"\t\t"<<wait[i]<<"\t\t"<<turn[i]<<endl;
    }
    for(int i =0; i< n; i++){
        avgWait += wait[i];
        avgTT += turn[i];
    }
    cout<<"\nAverage wait time : "<<(avgWait/n)
        <<"\nAverage Turn Around Time : "<<(avgTT/n);
    return 0;
}
```

2.7 OUTPUT

3   FIFO - PAGE REPLACEMENT ALGORITHM

3.1   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.

3.2   FIFO

First In First Out (FIFO) page replacement algorithm is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pages in the memory in a queue, 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.

3.3   ADVANTAGES AND DISADVANTAGES

Advantages :

It is easy to understand and implement. The number of operations is limited in a queue makes the implementation simple.

Disadvantage :

When the number of incoming pages is large, it might not provide excellent performance.

When we increase the number of frames or capacity to store pages in the queue, it should give us less number of page faults.

Sometimes FIFO may behave abnormally, and it may increase the number of page faults. This behavior of FIFO is called Belady's anomaly.

In FIFO, the system should keep track of all the frames. Sometimes it results in slow process execution.

3.4 IMPLEMENTATION:

1- Start traversing the pages.

 i) If set holds less pages than capacity.

   a) Insert page into the set one by one until

     the size of set reaches capacity or all

     page requests are processed.

   b) Simultaneously maintain the pages in the

     queue to perform FIFO.

   c) Increment page fault

 ii) Else

   If current page is present in set, do nothing.

   Else

     a) Remove the first page from the queue

       as it was the first to be entered in

       the memory

     b) Replace the first page in the queue with

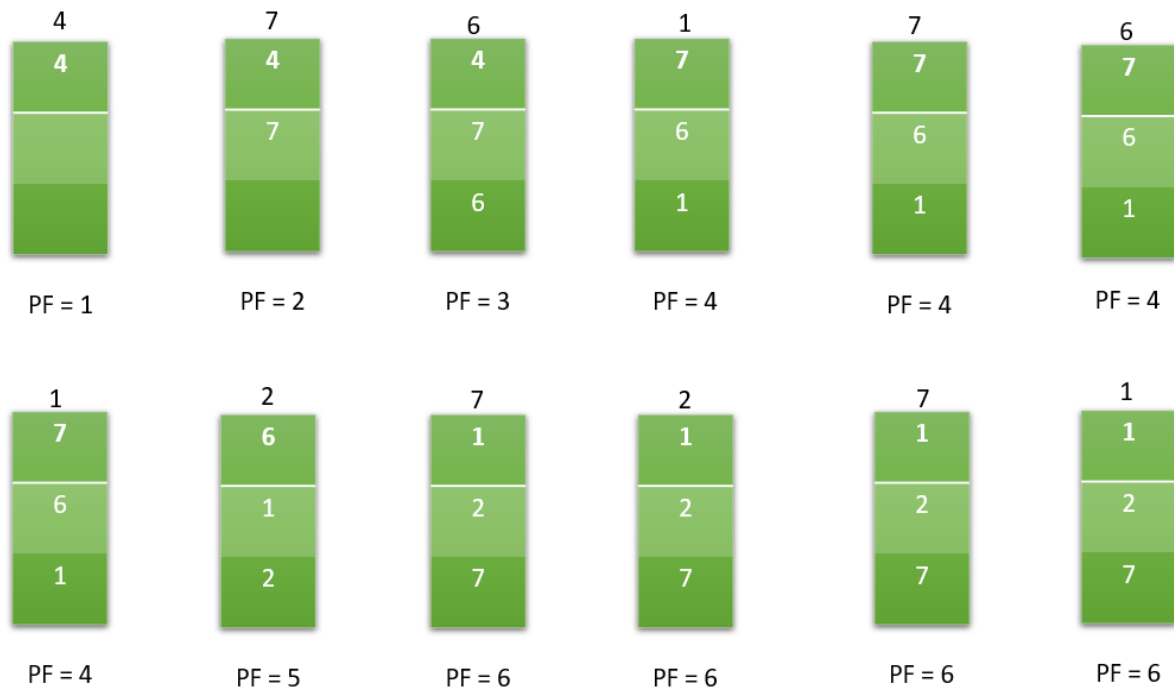   the current page in the string.

     c) Store current page in the queue.

     d) Increment page faults.

2. Return page faults.

3.5 EXAMPLE

Sample String : 4 7 6 1 7 6 1 2 7 2 7 1

4	7	6	1	7	6
4	4	4	7	7	7
	7	7	6	6	6
		6	1	1	1
PF = 1	PF = 2	PF = 3	PF = 4	PF = 4	PF = 4

1	2	7	2	7	1
7	6	1	1	1	1
6	1	2	2	2	2
1	2	7	7	7	7
PF = 4	PF = 5	PF = 6	PF = 6	PF = 6	PF = 6

Number of page faults = 6

Miss – ratio = (6/12)*100 = 50%

Hit – ratio = 100 – 50 = 50%

3.6 CODE:

```
#include<bits/stdc++.h>
using namespace std;

// Function to find page faults using FIFO
int pageFaults(int pages[], int n, int capacity)
{
    /* To represent set of current pages. We use an unordered_set so     that we quickly check if a page is present in set or not */
    unordered_set<int> s;

    // To store the pages in FIFO manner
    queue<int> indexes;
```

```cpp
    // Start from initial page
    int page_faults = 0;
    for (int i=0; i<n; i++)
    {
        // Check if the set can hold more pages
        if (s.size() < capacity)
        {
            /* Insert it into set if not present already which represents page fault */
            if (s.find(pages[i])==s.end())
            {
                // Insert the current page into the set
                s.insert(pages[i]);

                // increment page fault
                page_faults++;

                // Push the current page into the queue
                indexes.push(pages[i]);
            }
        }
        /* If the set is full then need to perform FIFO i.e. remove the first page of the queue from  set
               and queue both and insert the current page */
        else
        {
            // Check if current page is not already present in the set
            if (s.find(pages[i]) == s.end())
            {
                // Store the first page in the queue to be used to find and erase the page from the set
                int val = indexes.front();
```

```cpp
            // Pop the first page from the queue
            indexes.pop();

            // Remove the indexes page from the set
            s.erase(val);

            // insert the current page in the set
            s.insert(pages[i]);

            // push the current page into the queue
            indexes.push(pages[i]);

            // Increment page faults
            page_faults++;
        }
    }
}
return page_faults;
}
int main()
{
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4,
            2, 3, 0, 3, 2};
    int n = sizeof(pages)/sizeof(pages[0]);
    int capacity = 4;
    float miss_per, hit_per;
    cout << pageFaults(pages, n, capacity);
    miss_per = ((pageFaults(pages, n, capacity)*1.0)/n)*100;
    hit_per = 100 - miss_per;
    cout<<"\nMiss ratio = "<<miss_per<<"%";
    cout<<"\nHit ratio = "<<hit_per<<"%";
```
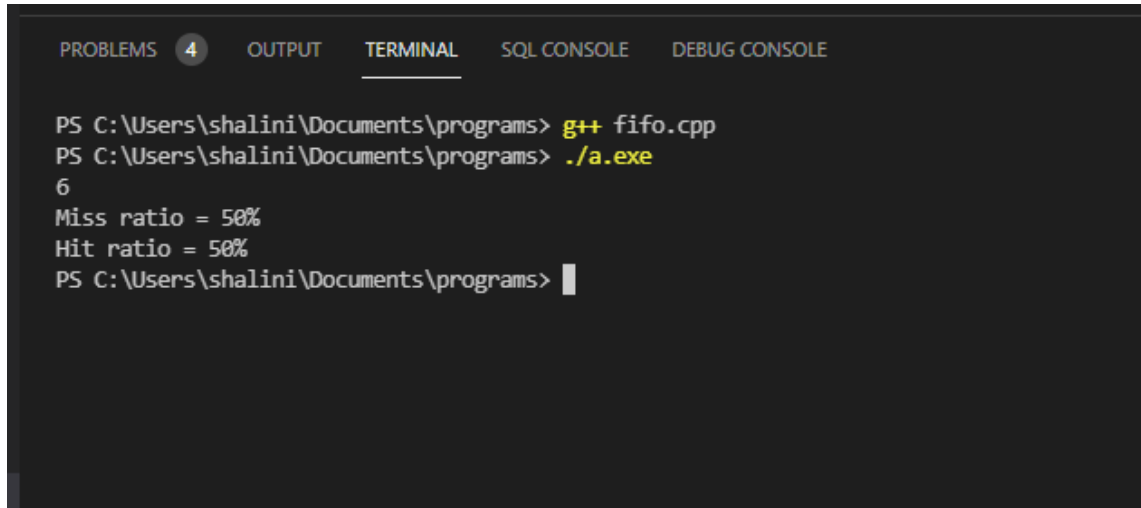
```
    return 0;

}
```

3.7   OUTPUT:

4   GITHUB LINKS

SHALINI SG (4NI19CS101) -- https://github.com/shalinishygan

SAUMYA NIGAM (4NI19CS100) -- https://github.com/SaumyaNigam1404