

**Lovely Professional University**

**Academic Task No. 21**

**(Operating System)**

School of computer Sciences and Engineering

PROJECT ON DEMAND PAGING

Name of Faculty in-charge: - Mrs. Ashu

Course Code: - CSE 316

Course Title: - Operation System

Student Name: Satyam

Roll number: - 21

Student ID: 11607168

Email ID: satyammehta3@gmail.com

GitHub Link: <https://github.com/satyam2579/Operating-System>

* **INDEX OF CONTENTS:**

1. Question and its explanation…………………………………………………………....……

11. [Algorithm](https://github.com/mrjatinchauhan/LRTF-Scheduling/blob/master/README.md) used to solve and formulae……………………………………………………...

111. Code Snippet…………………………………………………………………………….

* Question and its explanation.

Consider a scenario of demand paged memory. Page table is held in registers. It takes 8 milliseconds to service a page fault if an empty page is available or the replaced page is not modified and 20 milliseconds if the replaced page is modified. Memory access time is 100 nanoseconds. Assume that the page to be replaced is modified 70 percent of the time. Generate a solution to find maximum acceptable page-fault rate for access time that is not more than 200 nanoseconds.

We know how to find an effective access time (EAT) for a given page-fault rate (p).

Here, we have to find 'p' for given 'EAT' so we set up the following equation and solve for 'p':

(Note: 1 millisecond = 1,000,000 nanoseconds = 1e6 nanoseconds)

Time taken to service page Fault for empty page or unmodified page = 8ms.

Time taken to service page Fault for modified page = 20ms

Memory access time = 100ns

Effective Access time = 200ns

EAT = (1-p)\*(100) + (p)\*(100 + (1-.7)\*(8msec) + (.7)\*(20msec))

= 100 - 100p + 100p + (2.4e6)\*p + (14e6)\*p

= 100 + (16.4e6)\*p

200 = 100 + (16.4e6)\*p

p = 100/16.4e6 = 6.0975609756097560975609756097561e-6 ~ 6.01e-6

p-->page Fault Rate

* Algorithms and Formulas used to solve the problem.
* Whenever a page fault occurs, the CPU takes considerable amount of time to perform the above mentioned steps. This time taken by the CPU is called as page fault service time.
* Access time for Main memory : m
* Service time for Page fault: s
* Rate for Page fault Page fault rate is : p
* Effective memory access time = (p\*s) + (1-p)\*m
* CONCEPT:

According to the concept of Virtual Memory, in order to execute some process, only a part of the process needs to be present in the main memory which means that only a few pages will only be present in the main memory at any time.

However, deciding, which pages need to be kept in the main memory and which need to be kept in the secondary memory, is going to be difficult because we cannot say in advance that a process will require a particular page at particular time.

Therefore, to overcome this problem, there is a concept called Demand Paging is introduced. It suggests keeping all pages of the frames in the secondary memory until they are required. In other words, it says that do not load any page in the main memory until it is required.

What is a Page Fault?

If the referred page is not present in the main memory then there will be a miss and the concept is called Page miss or page fault.

The CPU has to access the missed page from the secondary memory. If the number of page fault is very high then the effective access time of the system will become very high.

* Code Snippets: -

#include <stdio.h>

#include <stdlib.h>

double page\_fault\_rate();

void userInput(void);

double service\_page\_fault\_empty;

double service\_page\_fault\_modified;

double mem\_access\_time;

double times\_page\_modified;

double effective\_access\_time;

double pageFaultRate;

double service\_page\_fault\_empty\_ns;

double service\_page\_fault\_modified\_ns;

double times\_page\_modified\_per;

void main(){

int swtch;

do{

printf("Select the required option \n");

printf("1.Find the PageFault Rate\n");

printf("2.Exit");

scanf("%d",&swtch);

switch(swtch){

case 1:userInput();break;

case 2:exit(0);

}

printf("\n\n");

}

while(swtch<3);

}

void userInput()

{

printf("\nEnter service Page Fault [Empty|Page is not Modified][in milliseconds]");

scanf("%lf",&service\_page\_fault\_empty);

printf("Enter Service Page Fault [Modified Page][in milliseconds]");

scanf("%lf",&service\_page\_fault\_modified);

printf("Enter Memory Access Time[in nanoseconds]");

scanf("%lf",&mem\_access\_time);

printf("Enter Percentage of time the page to be replaced is modified[0-100]");

scanf("%lf",&times\_page\_modified);

printf("Enter Effective Access time[in nanoseconds]");

scanf("%lf",&effective\_access\_time);

service\_page\_fault\_empty\_ns = (service\_page\_fault\_empty\*1000000);

service\_page\_fault\_modified\_ns = (service\_page\_fault\_modified\*1000000);

times\_page\_modified\_per = (times\_page\_modified/100);

printf("\nPage Fault rate calculated For:\n");

printf("Service Page Fault[Empty|Page Not Modified]=%lf \n",service\_page\_fault\_empty\_ns);

printf("Service Page Fault [Modified Page][in nanoseconds] %lf \n",service\_page\_fault\_modified\_ns);

printf("Memory Access Time[in nanoseconds]%lf\n",mem\_access\_time);

printf("Effective Access Time %lf\n",effective\_access\_time);

pageFaultRate = page\_fault\_rate(service\_page\_fault\_empty\_ns,service\_page\_fault\_modified\_ns,mem\_access\_time,times\_page\_modified\_per,effective\_access\_time);

printf("\nMaximum Acceptable Page Fault rate = %.2e[exponential notation]",pageFaultRate);

}

double page\_fault\_rate(double servicePageFaultEmpty,double servicePageFaultMod,double memAccess,double timesPages,double effAccess){

double assume,serve;

double numErator,denOminator;

double pageFault;

assume = (1- timesPages)\*servicePageFaultEmpty;

serve = timesPages\*servicePageFaultMod;

numErator = effAccess - memAccess;

denOminator = (assume+serve);

pageFault = numErator/denOminator;

return pageFault;

}