**Operating System Assignments**

Student Name:- Atul Shukla

Reg No:- 11804538

Section:- K18SP

Roll No:- 44

Email Address:- at887439@gmail.com

Github Link:- <https://github.com/at887439/Os-assignment>

**Problem:-**

Question No 21

**Explain the problem in terms of operating system concept?**

The process of loading the page into memory on demand is known as demand paging. If CPU try to refer a page that is currently not available in the main memory, it generates an interrupt indicating memory access fault. The OS puts the interrupted process in a blocking state.In this we have to find Effective Access Time (EAT) for a given page-fault rate(p).

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= 100ms

Effective Access time= 200ms

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 = 6.09756097561e-6 ~ 6.01e-6

# Algorithm:-

Step 1) The execution begins with process P1, which has burst time 4. Here, every process executes for 2 seconds. P2 and P3 are still in the waiting queue.

Step 2) At time =2, P1 is added to the end of the Queue and P2 starts executing.

Step 3) At time=4 , P2 is preempted and add at the end of the queue. P3 starts executing.

Step 4) At time=6 , P3 is preempted and add at the end of the queue. P1 starts executing.

Step 5) At time=8 , P1 has a burst time of 4. It has completed execution. P2 starts execution

Step 6) P2 has a burst time of 3. It has already executed for 2 interval. At time=9, P2 completes execution. Then, P3 starts execution till it completes.

# Functions:-

1. Start traversing the pages
   1. If set holds less pages than capacity.
   2. Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.
   3. Simultaneously maintain the pages in the queue to perform FIFO
   4. Increment page fault
   5. Else

If current page is present in set, do nothing.

Else

* 1. Remove the first page from the queue as it was the first to be entered in the memory
  2. Replace the first page in the queue with current page in the string
  3. Store current page in the queue
  4. Increment page faults

1. Return page faults

# Coding :-

#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\_a ccess\_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

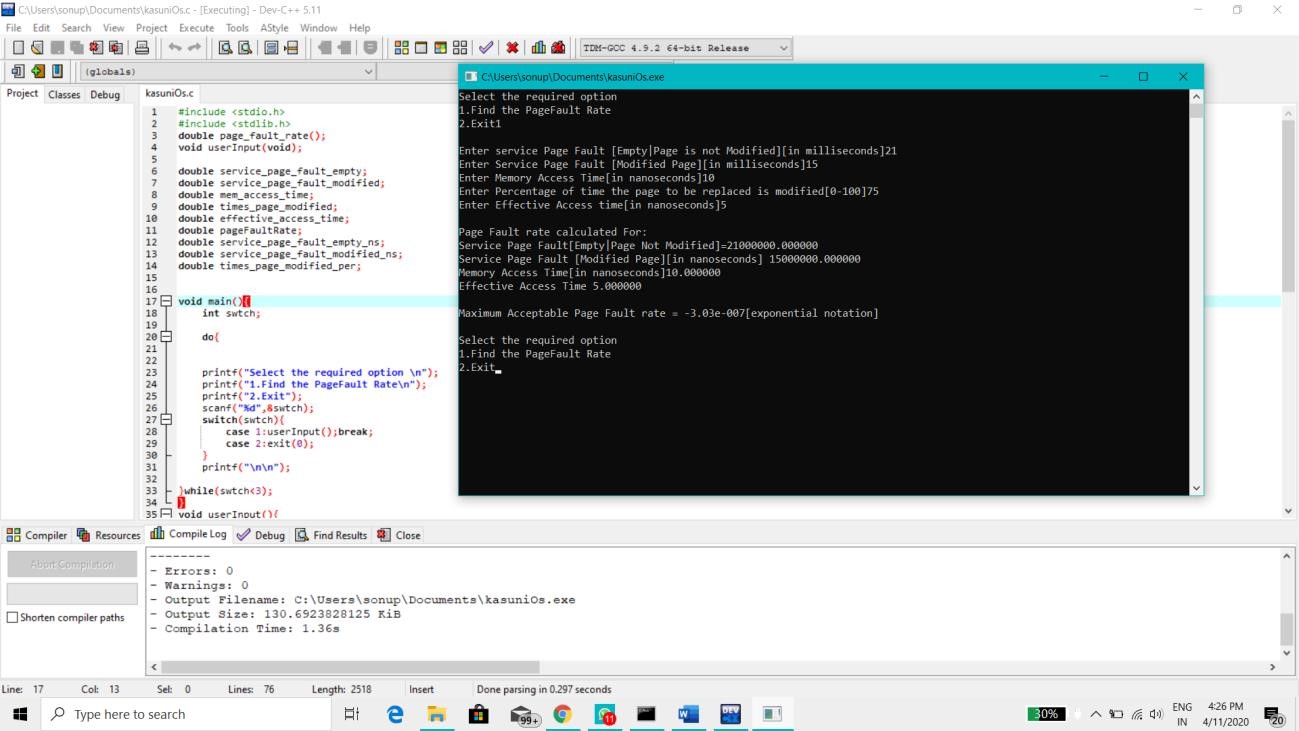
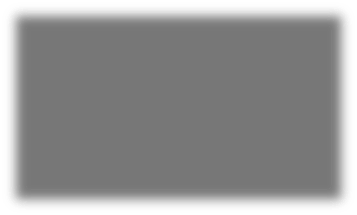
servicePageFaultMod,double memAccess,double timesPages,double effAccess){ double assume,serve; double numErator,denOminator; double pageFault;

assume = (1- timesPages)\*servicePageFaultEmpty; serve = timesPages\*servicePageFaultMod; numErator = effAccess - memAccess; denOminator = (assume+serve);

pageFault = numErator/denOminator; return pageFault;

}

# Output:-



**References:-**

[www.javatpoint.com](http://www.javatpoint.com/) [www.tutorialspoint.com](http://www.tutorialspoint.com/) [www.geeksforgeeks.com](http://www.geeksforgeeks.com/)