OPERATING SYSTEM CA PROJECT

STUDENT NAME:Damam SakethNath
STUDENT ID:
SECTION:K18ZV
ROLLNO:04
EMAIL ADDRESS:damamsaketh19@gmail.com
GITHUB LINK:

PROBLEM:
Question no:4

The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, ... Formally, it can be expressed as: fib0=0 fib1=1 fibn=fibn−1+fibn−2 Write a multithreaded program that generates the Fibonacci sequence. This program should work as follows: On the command line, the user will enter the number of Fibonacci numbers that the program is to generate. The program will then create a separate thread that will generate the Fibonacci numbers, placing the sequence in data that can be shared by the threads (an array is probably the most convenient data structure). When the thread finishes execution, the parent thread will output the sequence generated by the child thread .Because the parent thread cannot begin outputting the Fibonacci sequence until the child thread finishes, the parent thread will have to wait for the child thread to finish.

TOPICS USED:

1.Multithreading
2.Fibonacci series
3.Inheritance
4.Arrays

ALGORITM:

Computing the Fibonacci numbers can be done with the following small algoritm
Fibonacci(n)
If n<2 then return n;
X=Fibonacci(n-1);

```
Y=Fibonacci(n-2);
return x+y;
```

PURPOSE OF USE:

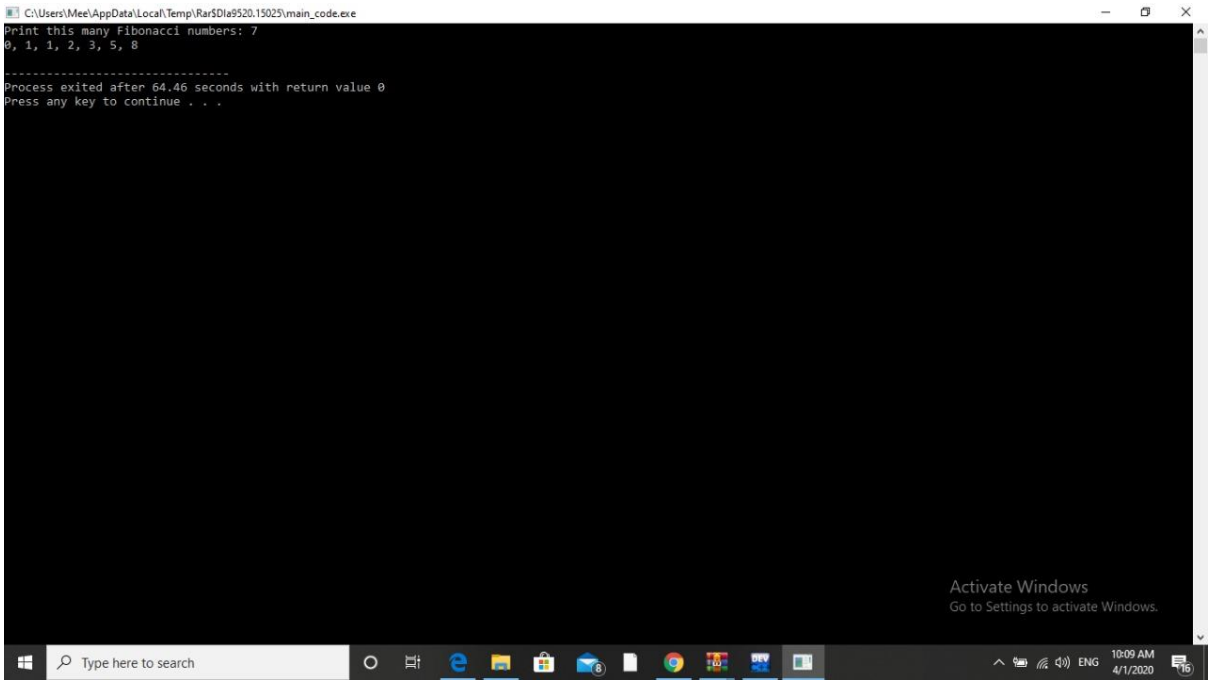
Multithreading is the ability of an **operating system** process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the **programming** running in the **computer**

Code:

```
 #include <stdio.h>
#include <pthread.h>

int j = 0;
 long fib[500];
void *runner(void *param)
{
if (j <= 0)
pthread_exit(0);
fib[0] = 0;
 if (j > 1)
{
fib[1] = 1;
 for (int i = 2; i < j; i++)
 fib[i] = fib[i-1] + fib[i-2];
 }
pthread_exit(0);
 }

int main(int argc, char *argv[])
{
 pthread_t tid;
 pthread_attr_t attr;
 pthread_attr_init(&attr);
 printf("Print this many Fibonacci numbers: ");
 scanf("%d", &j);
 if (j > 500)
 {
 printf("Printing as many as possible: 500\n");
```

```
j = 500;
 }
 pthread_create(&tid, &attr, runner, argv[1]);
pthread_join(tid, NULL);
 if (j > 0)
 printf("%ld", fib[0]);
 for (int i = 1; i < j; i++)
printf(", %ld", fib[i]);
 printf("\n");
 return 0;
 }
```

OUTPUT:



QUESTION :23

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 20milliseconds if the replaced page is modified. Memory access time is 100nanoseconds. Assume that the page to be replaced is modified 70percent of the time. Generate a solution to find maximum acceptable page-fault rate for access time that is not morethan 200 nanoseconds.

TOPICS USED:
1.Page fault
2.Page fault rate
3.Memory
4.Access time
ALGORITM:

- For pages in the backing store, the present bit is cleared in the page table entries.
- If present is not set, then a reference to the page causes a trap to the operating system.
- These traps are called *page faults*.
- To handle a page fault, the operating system
  - Finds a free page frame in memory
  - Reads the page in from backing store to the page frame
  - Updates the page table entry, setting present
  - Resumes execution of the thread

PURPOSE OF USE:

In computer operating systems, **demand paging** (as opposed to anticipatory **paging**) is a **method** of **virtual memory** management.

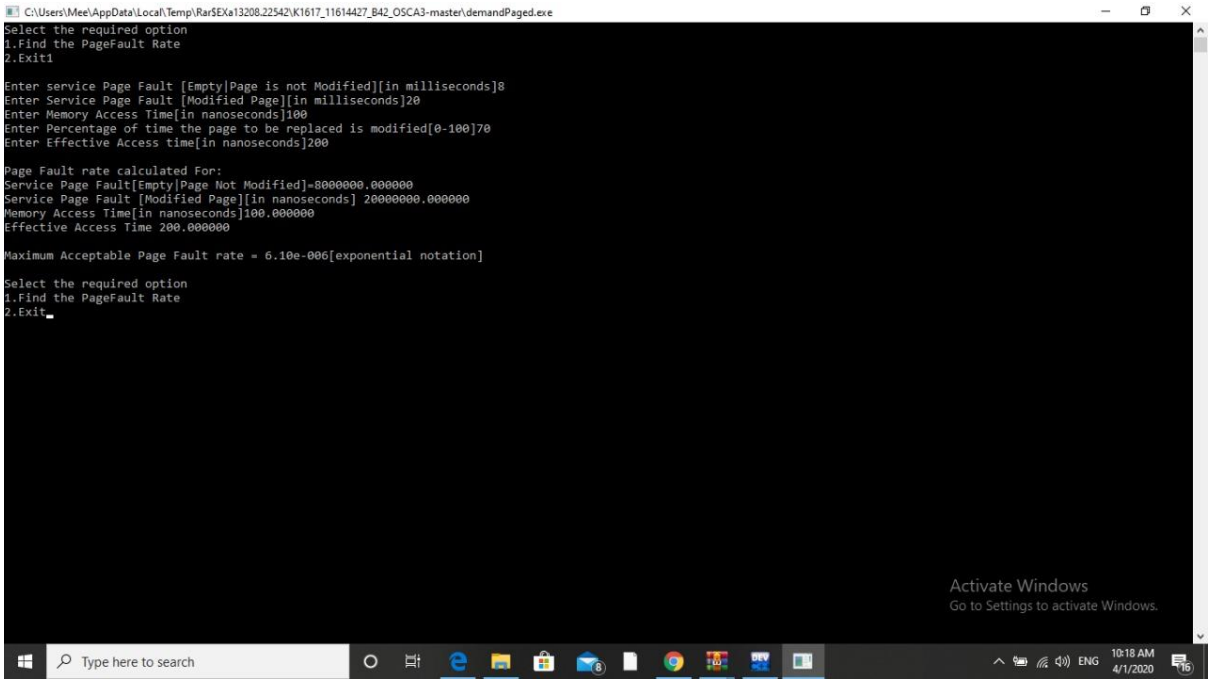
Code:

```
#include <stdio.h>
#include <stdlib.h>
double page_fault_rate();
void userInput(void);

double spfe;
double spfm;
double mat;
double tpm;
double eat;
double pfr;
```

```
double spfe_ns;
double spfm_ns;
double tpm_per;


void main(){
	int swtch;

	do{


	printf("Select one 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",&spfe);
	printf("Enter Service Page Fault [Modified Page][in milliseconds]");
	scanf("%lf",&spfm);
	printf("Enter Memory Access Time[in nanoseconds]");
	scanf("%lf",&mat);
	printf("Enter Percentage of time the page to be replaced is
modified[0-100]");
	scanf("%lf",&tpm);
	printf("Enter Effective Access time[in nanoseconds]");
	scanf("%lf",&eat);

	spfe_ns = (spfe*1000000);
	spfm_ns = (spfm*1000000);
	tpm_per = (tpm/100);
```

```
        printf("\nPage Fault rate calculated For:\n");
        printf("Service Page Fault[Empty|Page Not Modified]=%lf
\n",spfe_ns);
        printf("Service Page Fault [Modified Page][in nanoseconds] %lf
\n",spfm_ns);
        printf("Memory Access Time[in nanoseconds]%lf\n",mat);
        printf("Effective Access Time %lf\n",eat);
   pageFaultRate =  page_fault_rate(spfe_ns,spfm_ns,mat,tpm_per,eat);
        printf("\nMaximum Acceptable Page Fault rate = %.2e[exponential
notation]",pfr);


}

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:


C:\Users\Mee\AppData\Local\Temp\Rar$EXa13208.22542\K1617_11614427_B42_OSCA3-master\demandPaged.exe
Select the required option
1.Find the PageFault Rate
2.Exit1
Enter service Page Fault [Empty|Page is not Modified][in milliseconds]8
Enter Service Page Fault [Modified Page][in milliseconds]20
Enter Memory Access Time[in nanoseconds]100
Enter Percentage of time the page to be replaced is modified[0-100]70
Enter Effective Access time[in nanoseconds]200
Page Fault rate calculated For:
Service Page Fault[Empty|Page Not Modified]=8000000.000000
Service Page Fault [Modified Page][in nanoseconds] 20000000.000000
Memory Access Time[in nanoseconds]100.000000
Effective Access Time 200.000000
Maximum Acceptable Page Fault rate = 6.10e-006[exponential notation]
Select the required option
1.Find the PageFault Rate
2.Exit
Activate Windows
Go to Settings to activate Windows.
Type here to search
ENG
10:18 AM
4/1/2020