Assignment 5

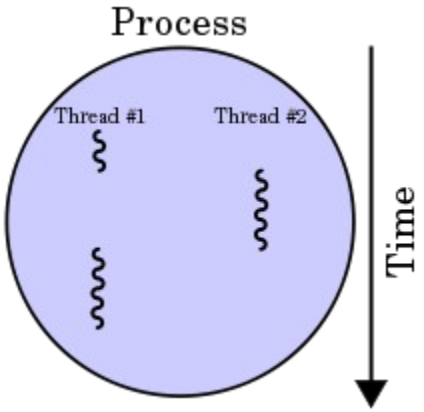
221071

1) AIM: Thread management using pthread library. Implement matrix multiplication using multithreading. Application should have pthread_create, pthread_join, pthread_exit. In the program, every thread must return the value and must be collected in pthread_join in the main function. Final sum of row column multiplication must be done by main thread (main function).

2) THEORY:

Thread

- it is an execution unit which consists of its own program counter, a stack, and a set of registers. Threads are also known as Lightweight processes.
- Threads are popular way to improve application through parallelism. The CPU switches rapidly back and forth among the threads giving illusion that the threads are running in parallel.
- As each thread has its own independent resource for process execution, multpile processes can be executed parallely by increasing number of threads.



Difference between Process and Thread

S. N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight, taking lesser resources than a process.
2	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
3	In multiple processing environments, each process	All threads can share same set of open

	executes the same code but has its own memory and file resources.	files, child processes.
4	If one process is blocked, then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, a second thread in the same task can run.
5	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
6	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Types of Thread

1. **User threads**, are above the kernel and without kernel support. These are the threads that application programmers use in their programs.
2. **Kernel threads** are supported within the kernel of the OS itself. All modern OSs support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

Advantages of Thread over Process

1. Responsiveness: If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.
2. Faster context switch: Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.
3. Effective utilization of multiprocessor system: If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.
4. Resource sharing: Resources like code, data, and files can be shared among all threads within a process.

   Note: stack and registers can’t be shared among the threads. Each thread has its own stack and registers.

5. Communication: Communication between multiple threads is easier, as the threads shares common address space. while in process we have to follow some specific communication technique for communication between two process.
6. Enhanced throughput of the system: If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system.

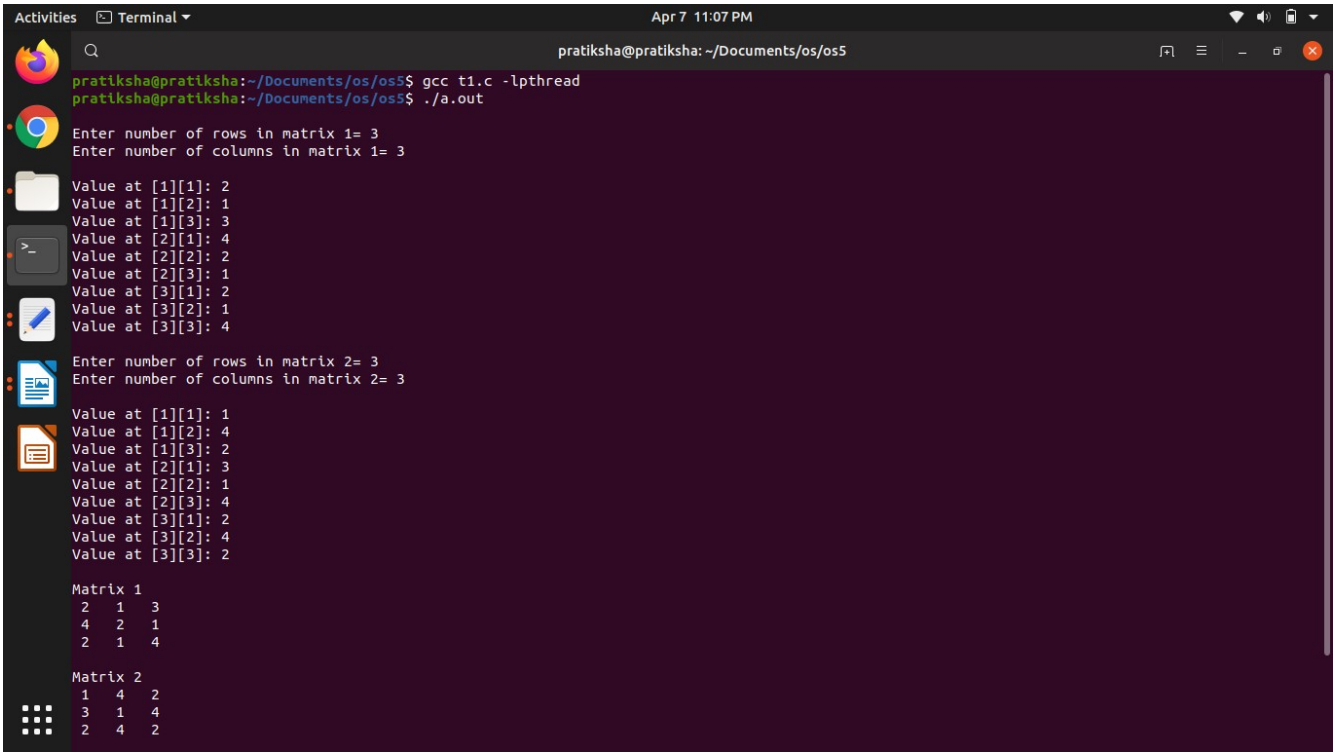
3) code:

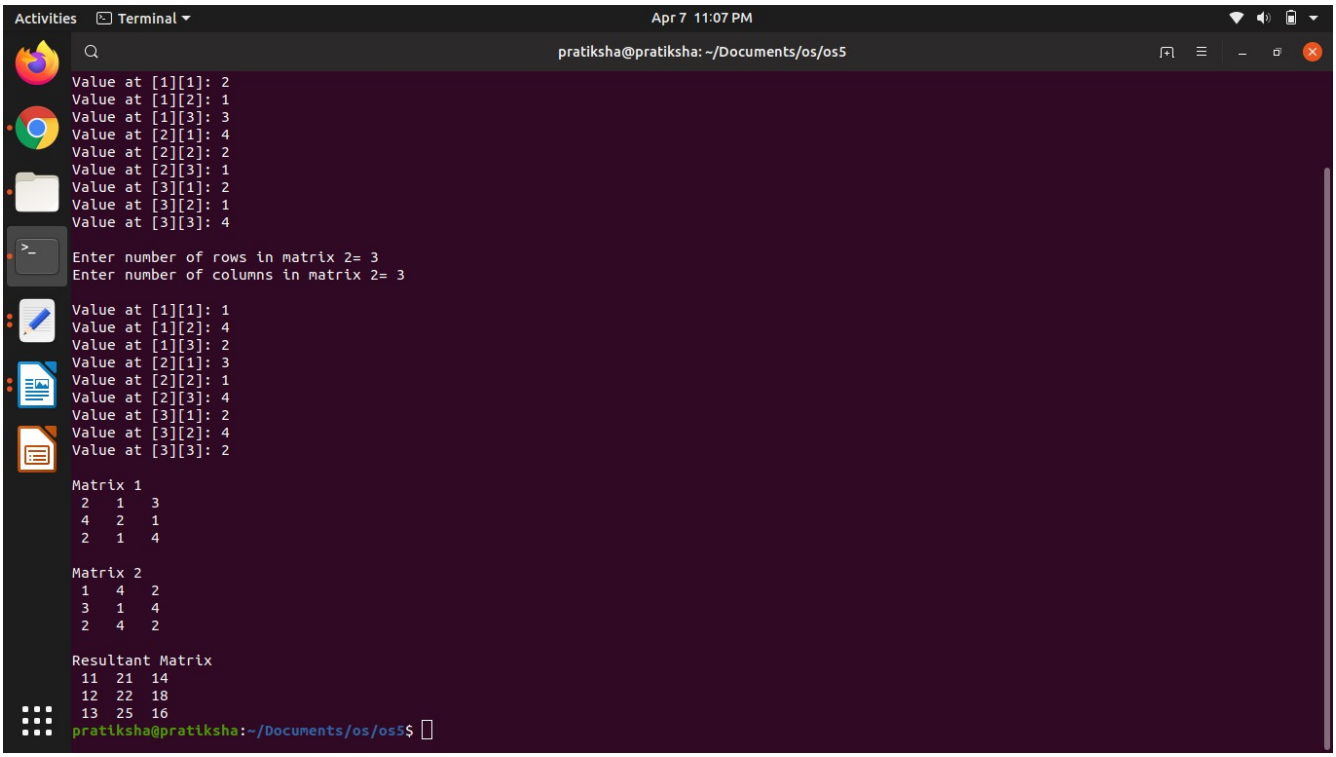
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#define SIZE 20
int A[SIZE][SIZE],B[SIZE][SIZE];
long int C[SIZE][SIZE];

void *mul_thread(void *arg)
{
	int i, row, col, *rcArgs;
	long int return_val; //int cannot be type casted to void
	rcArgs = (int *) arg;
	row = rcArgs[0];
	col = rcArgs[1];
	i= rcArgs[2];
	return_val = A[row][i] * B[i][col];
	//return ((void *) return_val);
	pthread_exit((void *) return_val);	//Thread exit
}

void accept_matrix(int M[SIZE][SIZE], int rows, int cols) //accepting matrix
{
	int i, j;
	printf("\n");
	for(i=0;i<rows;i++)
	{
		for(j=0;j<cols;j++)
		{
			printf("Value at [%d][%d]: ",i+1,j+1);
			scanf("%d",&M[i][j]);
		}
	}
}
```

```
void display_matrix(int M[SIZE][SIZE], int rows, int cols) //displaying matrix
{
	int i, j;
	printf("\n");
	for(i=0;i<rows;i++)
	{
		for(j=0;j<cols;j++)
		{
			printf("%2d  ",M[i][j]);
		}
		printf("\n");
	}
}

int main()
{
	int row1, col1, row2, col2;
	int rcArgs[3];
	int i, j, k, *status;
	pthread_t P[SIZE][SIZE][SIZE];

	printf("\nEnter number of rows in matrix 1= ");
	scanf("%d",&row1);
	printf("Enter number of columns in matrix 1= ");
	scanf("%d",&col1);
	accept_matrix(A, row1, col1);

	printf("\nEnter number of rows in matrix 2= ");
	scanf("%d",&row2);
	printf("Enter number of columns in matrix 2= ");
	scanf("%d",&col2);
	accept_matrix(B, row2, col2);

	if(col1==row2)
	{
		for(i=0;i<row1;i++)
		{
			for(j=0;j<col2;j++)
			{
				for(k=0;k<col1;k++)
				{
					rcArgs[0] = i;
					rcArgs[1] = j;
					rcArgs[2] = k;
					//Creating a new thread
					if(pthread_create(&P[i][j][k], NULL, mul_thread, rcArgs) != 0)
						printf("\nNo thread creation\n");
					else
						sleep(1);
```

```
                        }
                    }
                }
            }
            else
            {
                printf("\nMatrix multiplication not possible");
                exit(1);
            }

            printf("\nMatrix 1");
            display_matrix(A, row1, col1);
            printf("\nMatrix 2");
            display_matrix(B, row2, col2);

            for(i=0;i<row1;i++)
            {
                for(j=0;j<col2;j++)
                {
                    for(k=0;k<col1;k++)
                    {
                        //joining all the threads
                        if(pthread_join(P[i][j][k],(void **) &status) != 0)
                            perror("\nThread join failed\n");
                        C[i][j]+=(long int)status;
                    }
                }
            }

            printf("\nResultant Matrix\n");          //printing result
            for(i=0;i<row1;i++)
            {
                for(j=0;j<col2;j++)
                {
                    printf(" %2ld ",C[i][j]);
                }
                printf("\n");
            }

            exit(EXIT_SUCCESS);
}
```

4) output


Activities
Terminal
Apr 7 11:07 PM
pratiksha@pratiksha: ~/Documents/os/os5
pratiksha@pratiksha:~/Documents/os/os5$ gcc t1.c -lpthread
pratiksha@pratiksha:~/Documents/os/os5$ ./a.out
Enter number of rows in matrix 1= 3
Enter number of columns in matrix 1= 3
Value at [1][1]: 2
Value at [1][2]: 1
Value at [1][3]: 3
Value at [2][1]: 4
Value at [2][2]: 2
Value at [2][3]: 1
Value at [3][1]: 2
Value at [3][2]: 1
Value at [3][3]: 4
Enter number of rows in matrix 2= 3
Enter number of columns in matrix 2= 3
Value at [1][1]: 1
Value at [1][2]: 4
Value at [1][3]: 2
Value at [2][1]: 3
Value at [2][2]: 1
Value at [2][3]: 4
Value at [3][1]: 2
Value at [3][2]: 4
Value at [3][3]: 2
Matrix 1
2 1 3
4 2 1
2 1 4
Matrix 2
1 4 2
3 1 4
2 4 2


Activities
Terminal
Apr 7 11:07 PM
pratiksha@pratiksha: ~/Documents/os/os5
Value at [1][1]: 2
Value at [1][2]: 1
Value at [1][3]: 3
Value at [2][1]: 4
Value at [2][2]: 2
Value at [2][3]: 1
Value at [3][1]: 2
Value at [3][2]: 1
Value at [3][3]: 4
Enter number of rows in matrix 2= 3
Enter number of columns in matrix 2= 3
Value at [1][1]: 1
Value at [1][2]: 4
Value at [1][3]: 2
Value at [2][1]: 3
Value at [2][2]: 1
Value at [2][3]: 4
Value at [3][1]: 2
Value at [3][2]: 4
Value at [3][3]: 2
Matrix 1
2 1 3
4 2 1
2 1 4
Matrix 2
1 4 2
3 1 4
2 4 2
Resultant Matrix
11 21 14
12 22 18
13 25 16
pratiksha@pratiksha:~/Documents/os/os5$