Name :- Amit.Y.Zope
Roll No. :- 223067
Gr No. :- 21810714
Division :- C
Batch :- C3
Subject :- Operating System

Assignment No 5

Aim: 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).

Objective: To Study Thread management using pthread library in Operating System.

Theory:

What are Threads in Operating System ?

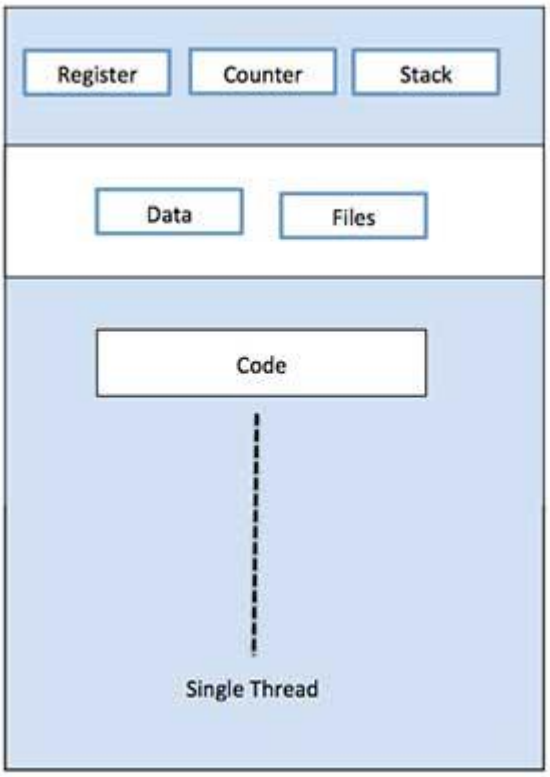
1. A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history.

2. A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that.

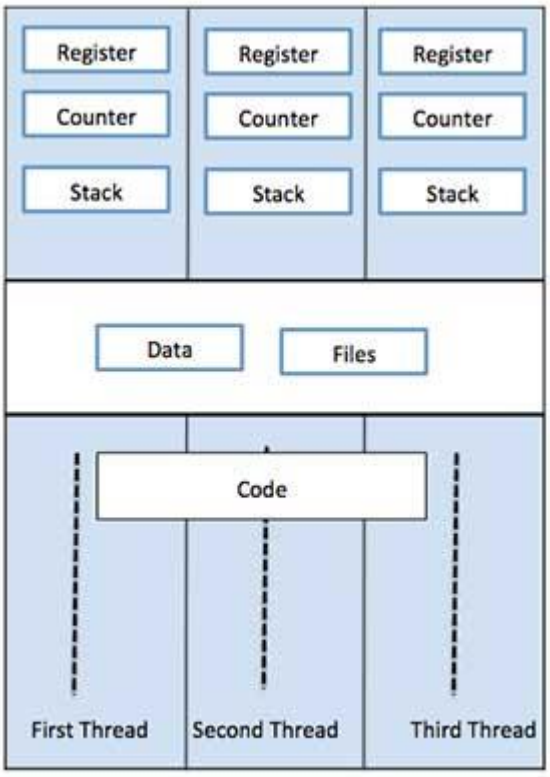
3. A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

4. Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors.

The following figure Shows working of single and multi-threaded process :-



Single Process P with single thread



Single Process P with three 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 executes the same code but has its own memory and file resources.	All threads can share same set of open 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.

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

pthreads Library in C Language

- Historically, hardware vendors have implemented their own proprietary versions of threads. These implementations differed substantially from each other making it difficult for programmers to develop portable threaded applications.
- In order to take full advantage of the capabilities provided by threads, a standardized programming interface was required.
    - For UNIX systems, this interface has been specified by the IEEE POSIX 1003.1c standard (1995).
    - Implementations adhering to this standard are referred to as POSIX threads, or Pthreads.
    - Most hardware vendors now offer Pthreads in addition to their proprietary API's.
    -
- The POSIX standard has continued to evolve and undergo revisions, including the Pthreads specification.

- **Pthreads are defined as a set of C language programming types and procedure calls, implemented with a `pthread.h` header/include file and a thread library - though this library may be part of another library, such as `libc`, in some implementations.**

Pthreads functions Used in program :-

1. ***pthread_create***(pthread_t *thread, const pthread_attr_t *attr , void *(*start_routine)(void *),void *arg);

- **pthread_create** creates a new thread and makes it executable. This routine can be called any number of times from anywhere within your code.

- **pthread_create** arguments:
  - **thread**: An opaque, unique identifier for the new thread returned by the subroutine.
  - **attr**: An opaque attribute object that may be used to set thread attributes. You can specify a thread attributes object, or NULL for the default values.
  - **start_routine**: the C routine that the thread will execute once it is created.
  - **arg**: A single argument that may be passed to *start_routine*. It must be passed by reference as a pointer cast of type void. NULL may be used if no argument is to be passed.

2. ***pthread_join***(pthread_t thread,void **value_ptr);

- The **pthread_join()** subroutine blocks the calling thread until the specified **threadid** thread terminates.
- The programmer is able to obtain the target thread's termination return **status** if it was specified in the target thread's call to **pthread_exit()**.
- A joining thread can match one **pthread_join()** call. It is a logical error to attempt multiple joins on the same thread.

3. ***pthread_exit***(void * value_ptr);

- The **pthread_exit()** routine allows the programmer to specify an optional termination *status* parameter. This optional parameter is typically returned to threads "joining" the terminated thread (covered later).
- In subroutines that execute to completion normally, you can often dispense with calling **pthread_exit() -** unless, of course, you want to pass the optional status code back.

CODE :

```
/* Matrix Multiplication using Threading.
   This program creates the theard for each number.  */
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>

#define MAX 20

int M1[MAX][MAX];
int M2[MAX][MAX];
int R[MAX][MAX];

int M,K1,N,K2;

//structure for row and columns
struct v {
   int i;
   int j;
}*c_data,*rout_data;

int n,c_sum=0,sum2=0;
int i,j,k=0;
void *runner(void * param);//Declaring Prototype of runner function

int main() {
        printf("\n|-------------------------------------------------------------
|\n");
        printf("|               Multithreading Assignment
|\n");
        printf("|-------------------------------------------------------------
|\n");

        int ch;


       while(1){
               printf("Choose The Process Execution Manner :- \n 1. Sequential
Manner\n 2. Parallel Manner \n 3. Exit \n  Enter Your Choice : ");
        scanf("%d",&ch);
        switch(ch){

        case 1: {
        printf("\n|-------------------------------------------------------------
|\n");
        printf("|                  Sequential Execution
|\n");
        printf("|-------------------------------------------------------------
|\n");
                float time_spent1 = 0.0;
        clock_t begin = clock();
```

```
        printf("Enter the Rows and Columns of first Matrix 1 :- \n"); //Input
rows and columns of matrix 1
        scanf("%d %d",&M,&K1);
        printf("Enter the Rows and Columns of second Matrix 2 :- \n");//Input
rows and columns of matrix 2
        scanf("%d %d",&K2,&N);

        if(K1 != K2){
                printf("\n Multiplication is not Possible As \n ");
                printf("\n The number of Columns of First Matrix is not Equal to
Number of Rows of Second Matrix \n ");
                exit(0);
        }else{

                printf("Enter the Elements of Matrix 1 :- \n "); //Input elements
of Matrix 1
                  for( i=0;i<M;i++ ){
                        for(j =0;j<K1;j++){
                                scanf("%d",&M1[i][j]);
                        }
                }
                printf("Enter the Elements of Matix 2 :- \n");//Input elements of
Matrix 2
                  for( i=0;i<K2;i++ ){
                        for(j =0;j<N;j++){
                                scanf("%d",&M2[i][j]);
                        }
                }

                for( i =0 ;i<M;i++){
                        for(j = 0;j<N;j++){
                                for(int k = 0;k<K1;k++){
                                        sum2 += M1[i][k]*M2[k][j];
// Performing Multiplication of Matrix 1 and 2
                                }
                                R[i][j] = sum2;
                                sum2 = 0;
                        }
                }
                  printf("\nResulting Matrix is : -\n");
                  for( i=0; i<M ;i++ ){
                        for(j =0;j<N;j++){
                                printf("%d",R[i][j]);//Displaying Resulting
Matrix
                                printf("\t");
                        }
                        printf("\n");
                }

                 clock_t end = clock();

        time_spent1 += (double)(end - begin)/CLOCKS_PER_SEC;

        printf("\n\nSequential Time calculated is :- %f miliseconds
\n\n",time_spent1*1000); // Displaying time req for sequential exection
```

```
                }
        }
                break;
        case 2: {

        printf("\n|-------------------------------------------------------------
|\n");
        printf("|                  Parallel Execution
|\n");
        printf("|-------------------------------------------------------------
|\n");
                        float time_spent = 0.0;
                        clock_t begin = clock();

                        pthread_t tid;
                void *arg;


                        printf("Enter the Number of Rows and Columns of First
Matrix : - \n");
                        scanf("%d %d",&M,&K1);

                        printf("Enter the Number of Rows and Columns of First
Matrix : - \n");
                        scanf("%d %d",&K2,&N);



                        if(K1 != K2){
                                printf("\n Multiplication is not Possible As \n
");
                                printf("\n The number of Columns of First Matrix
is not Equal to Number of Rows of Second Matrix \n ");
                                exit(0);
                        }else{

                                printf("Enter the Elements of Matrix 1 :- \n ");
                                  for( i=0;i<M;i++ ){
                                        for(j =0;j<K1;j++){
                                                scanf("%d",&M1[i][j]);
                                        }
                                }
                                printf("Enter the Elements of Matix 2 :- \n");
                                  for( i=0;i<K2;i++ ){
                                        for(j =0;j<N;j++){
                                                scanf("%d",&M2[i][j]);
                                        }
                                }

                                printf("\n\n");

                           for(i = 0; i < M; i++) {
                                        for(j = 0; j < N; j++) {
                                 c_data = (struct v *) malloc(sizeof(struct v));
//allocate dynamic memory to struct variable data
                                        c_data->i = i;  //i represents Rows and
```

```
                                        c_data->j = j;  //j represents Columns of
th e matrix

                                        pthread_create(&tid,NULL,runner,c_data);
//create thread and pass struct variable as argument
                                    pthread_join(tid,&arg); // wait for thread
completion and collect the result in variable "arg", returned by each thread

                                    k = *((int *)arg); //typecast of arg to int
                                    c_sum = c_sum + k;  //add the result in variable
"sum" return by thread
                                    R[c_data->i][c_data->j] = k;      //add result in
Matric C
                                    //printf("k is %d\n ",k);   //print each value of
K
                                        //count++;
                                        }
                                }



                            //Print the result - Matrix C
                            printf("\n");
                            printf("Resulting Matrix is : - \n ");
                               for(i = 0; i < M; i++) {
                                        for(j = 0; j < N; j++) {
                                             printf("%d ", R[i][j]);
                                             printf("\t");
                                             }
                                        printf("\n");
                               }
                               printf("Sum  is %d\n ",c_sum );
                            }

                            clock_t end = clock();

                            time_spent += (double)(end - begin)/CLOCKS_PER_SEC;

                            printf("\n\nParallel Time calculated is :- %f miliseconds
\n\n",time_spent*1000);
                 }

                        break;
                    case 3 :
                            exit(0);
                                    default:
                                            printf("Wrong Choice Selected!! Try Again ");
                    }

                }
}

//Threading Function
void *runner(void *param) {
     rout_data =param; //typecasting of param variable as it is void *,  whereas
data2 is struct *
        sum2=0;
```

```
      for(n = 0; n< K1; n++){
                  sum2 += M1[rout_data->i][n] * M2[n][rout_data->j];
      }
      printf("Sum2 is %d\n ", sum2);
      pthread_exit(&sum2); //this thread return the value calculated by it to main
thread
}
```

Output : -

Parallel Time :-

```
|------------------------------------------------------------|
|              Multithreading Assignment                     |
|------------------------------------------------------------|
Choose The Process Execution Manner :-
 1. Sequential Manner
 2. Parallel Manner
 3. Exit
  Enter Your Choice : 2

|------------------------------------------------------------|
|                 Parallel Execution                         |
|------------------------------------------------------------|
Enter the Number of Rows and Columns of First Matrix : -
4 4
Enter the Number of Rows and Columns of First Matrix : -
4 4
Enter the Elements of Matrix 1 :-
 5 2 6 1
0 6 2 0
3 8 1 4
1 8 5 6
Enter the Elements of Matix 2 :-
7 5 8 0
1 8 2 6
9 4 3 8
5 3 7 9


Sum2 is 96
 Sum2 is 68
 Sum2 is 69
 Sum2 is 69
 Sum2 is 24
 Sum2 is 56
 Sum2 is 18
 Sum2 is 52
 Sum2 is 58
 Sum2 is 95
 Sum2 is 71
 Sum2 is 92
 Sum2 is 90
 Sum2 is 107
 Sum2 is 81
 Sum2 is 142

Resulting Matrix is : -
 96      68      69      69
24      56      18      52
58      95      71      92
90      107     81      142
Sum  is 1188


Parallel Time calculated is :- 4.203000 miliseconds
```

Sequential Time :-

```
vedant@vedant:~/Downloads$ gcc -pthread THREAD_SEQ_PARALLEL.c
vedant@vedant:~/Downloads$ ./a.out

|-------------------------------------------------------------|
|               Multithreading Assignment                     |
|-------------------------------------------------------------|
Choose The Process Execution Manner :-
 1. Sequential Manner
 2. Parallel Manner
 3. Exit
  Enter Your Choice : 1

|-------------------------------------------------------------|
|                 Sequential Execution                        |
|-------------------------------------------------------------|
Enter the Rows and Columns of first Matrix 1 :-
4 4
Enter the Rows and Columns of second Matrix 2 :-
4 4
Enter the Elements of Matrix 1 :-
 5 2 6 1
0 6 2 0
3 8 1 4
1 8 5 6
Enter the Elements of Matix 2 :-
7 5 8 0
1 8 2 6
9 4 3 8
5 3 7 9

Resulting Matrix is : -
96      68      69      69
24      56      18      52
58      95      71      92
90      107     81      142


Sequential Time calculated is :- 0.893000 miliseconds

Choose The Process Execution Manner :-
 1. Sequential Manner
 2. Parallel Manner
 3. Exit
  Enter Your Choice :
```

→ Sequential Execution takes less time than Parallel