Assignment 3: **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 the main thread (main function).**

Name: **Chaitanya Suresh Uge**
Roll Number: **333064**
GR Number: **21910718**
Division: **C** **Batch:** **C3**

Theory:

- Multiplication of matrices does take time surely. Time complexity: O(n^3)
- Strassen algorithm improves time complexity to O(n^(2.8074)).
- Multi-threading can be used to reduce this time complexity by creating separate threads for each element in the resultant matrix.

pthread_create():

- Syntax:

```
int pthread_create(pthread_t *restrict thread,
                   const pthread_attr_t *restrict attr,
                   void *(*start_routine)(void *),
                   void *restrict arg);
```

- Arguments:
  1st argument : pointer to thread variable
  2nd argument : attributes
  3rd argument : name of function / routine of thread
  4th argument : data passed to thread function
- The pthread_create() function starts a new thread in the calling process. The new thread starts execution by invoking start_routine(); arg is passed as the sole argument of start_routine().
- On success, pthread_create() returns 0; on error, it returns an error number, and the contents of *thread are undefined.

pthread_join():

- Syntax:
  int pthread_join(pthread_t thread, void **retval);
- The pthread_join() function waits for the thread specified by thread to terminate. If that thread has already terminated, then pthread_join() returns immediately. The thread specified by thread must be joinable.
- On success, pthread_join() returns 0; on error, it returns an error number.

pthread_exit():

- Syntax:
  noreturn void pthread_exit(void *retval);
- The pthread_exit() function terminates the calling thread and returns a value via retval that (if the thread is joinable) is available to another thread in the same process that calls pthread_join(3).
- This function does not return to the caller.

clock_t begin = clock();

- time.h library contains different predefined data types like time_t, clock_t for adding clock in our program.
- Here in the below source code I used the clock() function for calculating execution time of the program.

Source Code:

```
1   #include <stdio.h>
2   #include <unistd.h>
3   #include <pthread.h>
4   #include <time.h>
5   #include <stdlib.h>
6
7   #define MAX_THREADS 50
8   #define SIZE 10
9
10  int i=0, j=0, k=0;
11  int a[SIZE][SIZE];
12  int b[SIZE][SIZE];
13  int res[SIZE][SIZE];
14
15  // Type defining for passing function argumnents
16  typedef struct parameters
17  {
18    int x,y;
19  }args;
20
21
22  // Multiplication function
23  void* mult(void* arg)
24  {
25    args* p = arg;
26
27    // Calculating each element in res matrix using passed args
28    for(int n=0;n<j;n++)
29    {
30      res[p->x][p->y] += a[p->x][n]*b[n][p->y];
31    }
32
33    sleep(2);
34
35    // End of thread
36    pthread_exit(0);
37  }
38
39
40
41
42  int main()
43  {
44    // to store the execution time of code
45      double time_spent = 0.0;
46
47
48    // Initializing all matrices to 0
49    for(int x=0; x<SIZE; x++)
```

```
50      {
51       for(int y=0; y<SIZE; y++)
52       {
53        a[x][y] = 0;
54        b[x][y] = 0;
55        res[x][y] = 0;
56       }
57      }
58
59      printf("\n ---------- Matrix A ----------\n\n");
60      printf("Enter number of rows for matrix A: ");
61      scanf("%d",&i);
62      printf("Enter number of columns for matrix A: ");
63      scanf("%d",&j);
64
65      for(int x=0;x<i;x++)
66      {
67       for(int y=0;y<j;y++)
68       {
69        printf("Enter element (%d,%d) : ",x,y);
70        scanf("%d",&a[x][y]);
71       }
72      }
73
74
75
76      printf("\n ---------- Matrix B ----------\n\n");
77      printf("Number of rows for matrix B : %d\n",j);
78      printf("Enter number of columns for matrix B: ");
79      scanf("%d",&k);
80
81      for(int x=0;x<j;x++)
82      {
83       for(int y=0;y<k;y++)
84       {
85        printf("Enter element (%d,%d): ",x,y);
86        scanf("%d",&b[x][y]);
87       }
88      }
89
90
91
92      // Print matrices.....
93
94      printf("\n <<< Matrix A >>>\n\n");
95       for(int x=0;x<i;x++)
96       {
97        for(int y=0;y<j;y++)
98        {
99         printf("%5d",a[x][y]);
100       }
101       printf("\n\n");
```

```
102     }
103
104   printf(" <<< Matrix B >>>\n\n");
105     for(int x=0;x<j;x++)
106     {
107       for(int y=0;y<k;y++)
108       {
109         printf("%5d",b[x][y]);
110       }
111       printf("\n\n");
112     }
113
114
115   // Multiplication: using Threads
116
117   // Defining Threads
118   pthread_t thread[MAX_THREADS];
119   int thread_number = 0;
120
121   // Defining p for passing parameters to function as struct
122   args p[i*k];
123
124   // Timer start
125   // time_t start = time(NULL);
126   clock_t begin = clock();
127
128   for(int x=0;x<i;x++)
129   {
130     for(int y=0;y<k;y++)
131     {
132       // Parameter initialization
133       p[thread_number].x=x;
134       p[thread_number].y=y;
135
136       // Create thread for each element
137       int status;
138       status = pthread_create(&thread[thread_number], NULL, mult, (void *)
139 &p[thread_number]);
140
141       // ERROR check
142       if(status!=0)
143       {
144         printf("ERROR in threads");
145         exit(0);
146       }
147
148       thread_number++;
149     }
150   }
151
152
153
```

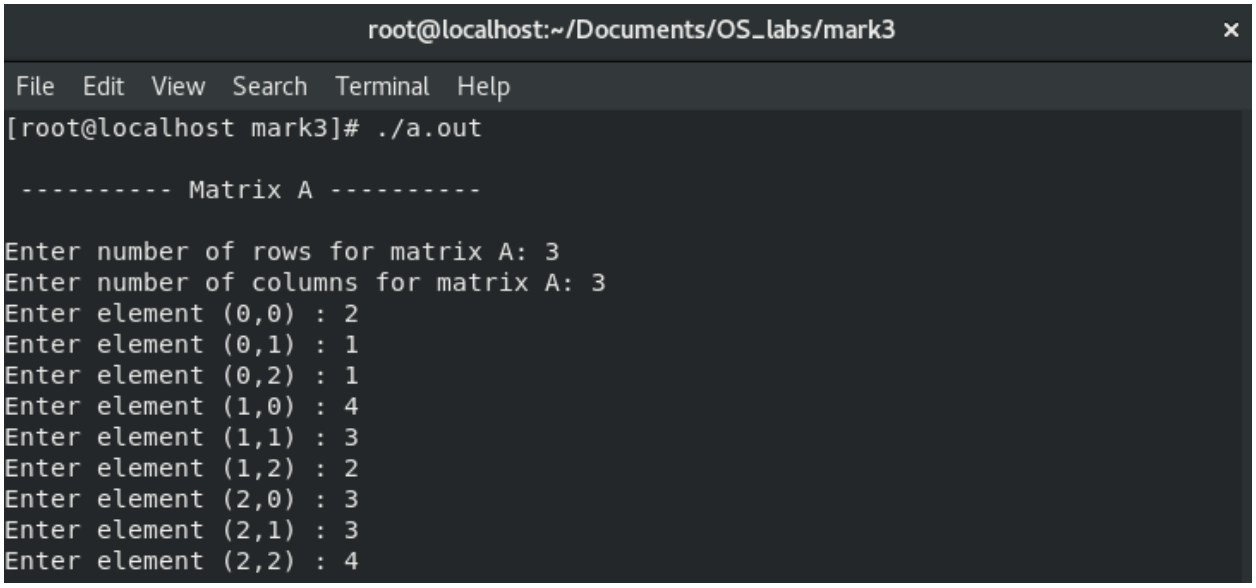
```
154
155   for(int z=0;z<(i*k);z++)
156     pthread_join(thread[z], NULL);
157
158
159   //Print Result
160   printf(" <<< Multiplied Matrix >>>\n\n");
161   for(int x=0;x<i;x++)
162   {
163     for(int y=0;y<k;y++)
164     {
165       printf("%5d",res[x][y]);
166     }
167     printf("\n\n");
168   }
169
170
171   clock_t end = clock();
172
173
174   // calculating elapsed time
175   time_spent += (double)(end - begin) / CLOCKS_PER_SEC;
176
177   printf("\n\n execution time :  %f s", time_spent);
178
179
180   printf("\n\n [ used threads : %d ]\n\n",thread_number);
181   printf("------------------------------\n\n");
182
183   for(int i=0;i<thread_number;i++)
184     printf(" * Thread %d ID : %d\n",i+1,(int)thread[i]);
185   printf("\n");
186
187   return 0;
189 }
```

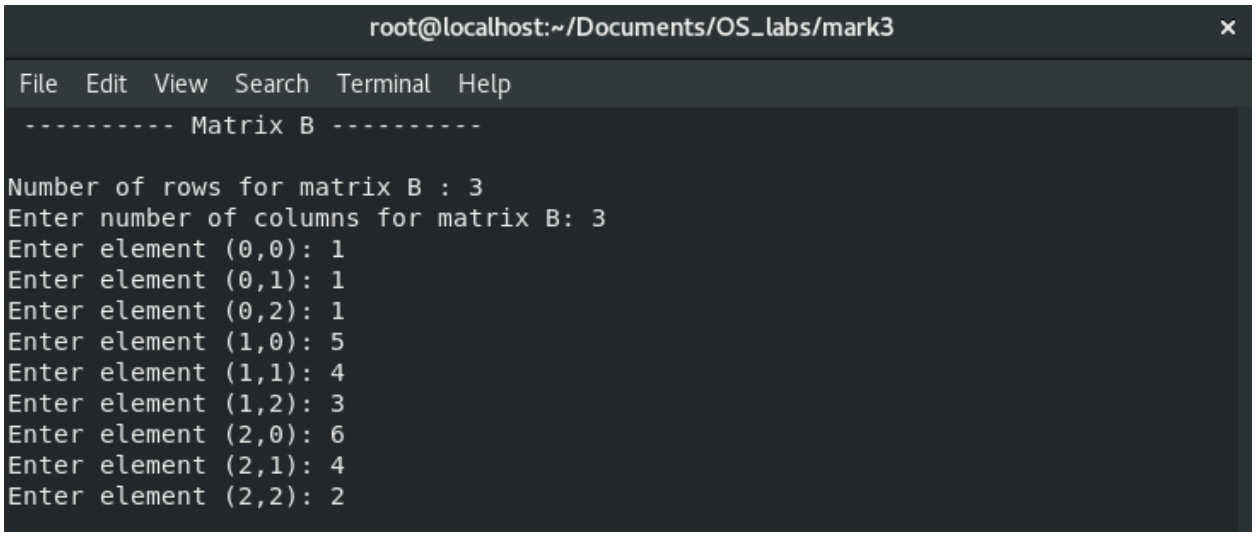
OUTPUT:

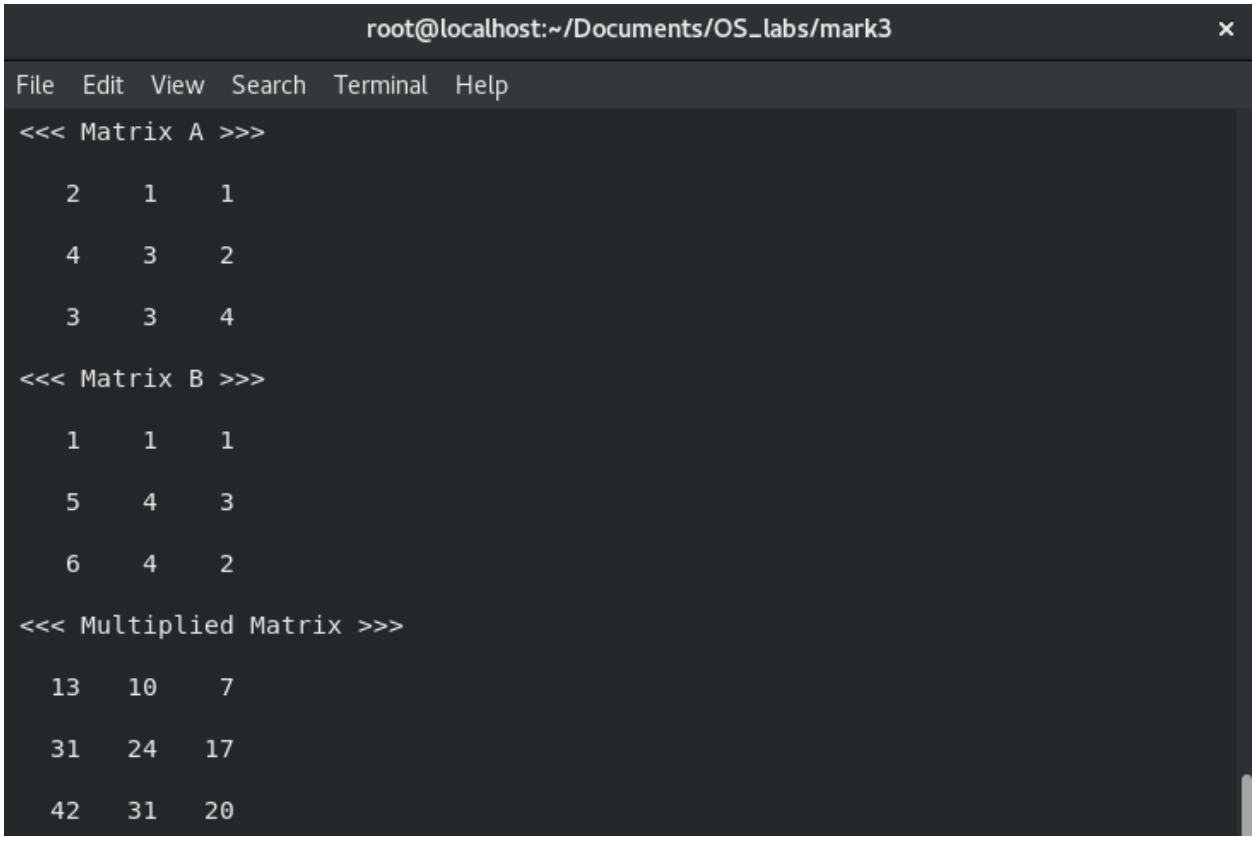
Compilation

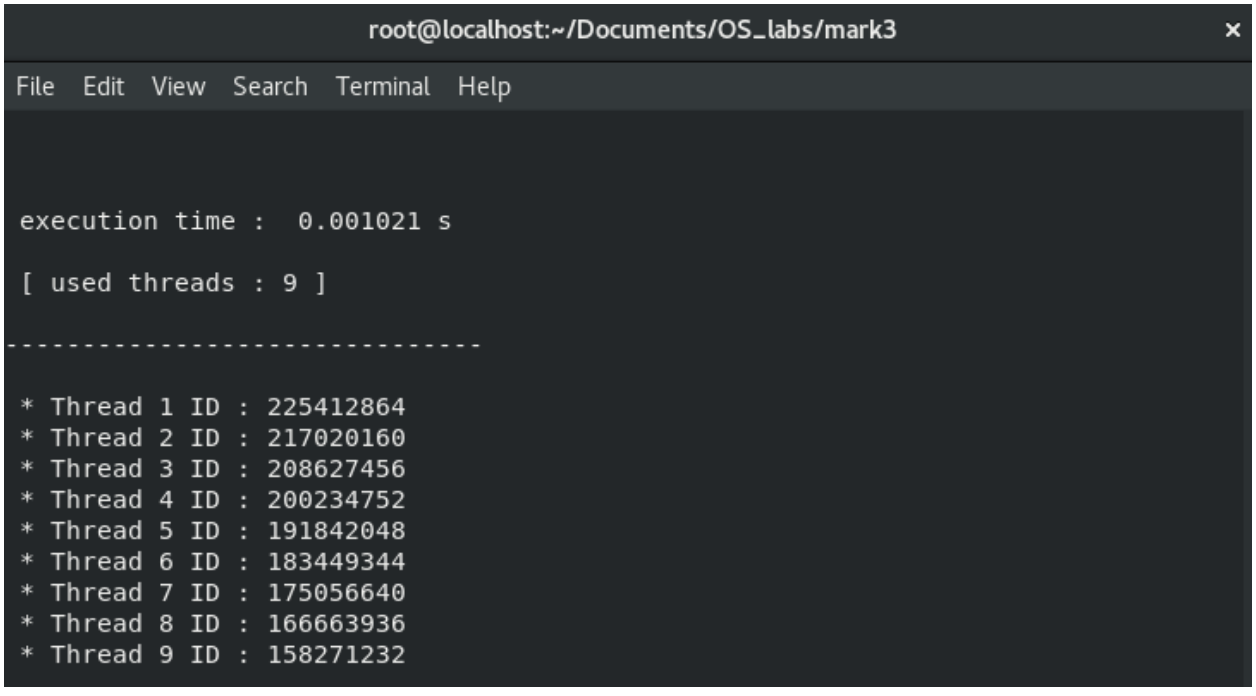
```
root@localhost:~/Documents/OS_labs/mark3
File  Edit  View  Search  Terminal  Help
[root@localhost mark3]# gcc matrix_multi.c -lpthread
[root@localhost mark3]#
```

Execution:


root@localhost:~/Documents/OS_labs/mark3
File Edit View Search Terminal Help
[root@localhost mark3]# ./a.out
---------- Matrix A ----------
Enter number of rows for matrix A: 3
Enter number of columns for matrix A: 3
Enter element (0,0) : 2
Enter element (0,1) : 1
Enter element (0,2) : 1
Enter element (1,0) : 4
Enter element (1,1) : 3
Enter element (1,2) : 2
Enter element (2,0) : 3
Enter element (2,1) : 3
Enter element (2,2) : 4


root@localhost:~/Documents/OS_labs/mark3
File Edit View Search Terminal Help
---------- Matrix B ----------
Number of rows for matrix B : 3
Enter number of columns for matrix B: 3
Enter element (0,0): 1
Enter element (0,1): 1
Enter element (0,2): 1
Enter element (1,0): 5
Enter element (1,1): 4
Enter element (1,2): 3
Enter element (2,0): 6
Enter element (2,1): 4
Enter element (2,2): 2


root@localhost:~/Documents/OS_labs/mark3
File Edit View Search Terminal Help
<<< Matrix A >>>
2 1 1
4 3 2
3 3 4
<<< Matrix B >>>
1 1 1
5 4 3
6 4 2
<<< Multiplied Matrix >>>
13 10 7
31 24 17
42 31 20


root@localhost:~/Documents/OS_labs/mark3
File Edit View Search Terminal Help
execution time : 0.001021 s
[used threads : 9]
* Thread 1 ID : 225412864
* Thread 2 ID : 217020160
* Thread 3 ID : 208627456
* Thread 4 ID : 200234752
* Thread 5 ID : 191842048
* Thread 6 ID : 183449344
* Thread 7 ID : 175056640
* Thread 8 ID : 166663936
* Thread 9 ID : 158271232