# Laboratory 3

## Title of the Laboratory Exercise: Programs based on multithreaded programming

### 1. Introduction and Purpose of Experiment

Multithreading is the ability of a processor or a single core in a multi-core processor to execute multiple threads concurrently, supported by the operating system. By solving students will be able to manipulate multiple threads in a program.

### 2. Aim and Objectives

#### Aim

- To develop programs using multiple threads.

#### Objectives

At the end of this lab, the student will be able to

- Identify multiple tasks
- Use threads constructs for creating threads
- Apply threads for different/multiple tasks

### 3. Experimental Procedure

i. Analyse the problem statement
ii. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
iii. Implement the algorithm in C language
iv. Compile the C program
v. Test the implemented program
vi. Document the Results
vii. Analyse and discuss the outcomes of your experiment

### 4. Questions

Create multithreaded programs to implement the following

a) Display “Hello World” message by 3 different threads
b) Create two threads;
   - Thread1 adds marks (out of 10) of student1 from subject1 to subject5, Thread2 adds marks of student2 from subject1 to subject 5. Main process takes the sum

returned from the Thread1 and Thread2, decides who scored more marks and displays student with its highest score.

## 5. Calculations/Computations/Algorithms

**Pseudocode for Hello Program**

```
void *thread(void*arg){
   Display("%s by Child thread\n" ,(char*)arg);
   pthread_exit("Bye Have a nice day");
}
int main(void) {
   int retcode1,retcode2,retcode3,i;
   pthread_t t1,t2,t3;
   void*exitstatus;
   retcode1=pthread_create(&t1,NULL,thread,"Hello World");
   retcode2=pthread_create(&t2,NULL,thread,"Hello World");
   retcode3=pthread_create(&t3,NULL,thread,"Hello World");
   pthread_join(t1,&exitstatus);
   pthread_join(t2,&exitstatus);
   pthread_join(t3,&exitstatus);
   Display("From Child Thread = %s\n",(char*)exitstatus);
   return (EXIT_SUCCESS);
}
```

**Pseudocode for Student Program**

```
input(int arr[5]){
  for( i=0;i<5;i++){
    Read and Display arr[i] values
  }
}
void *Compute_Sum(void *arg){ //Chid Thread execution Block
  int sum=0;
  int *arr=(int*)arg;
  for(int i=0;i<5;i++)
    sum+=arr[i];
  pthread_exit(sum);
}
int main() {//main process Block]
  pthread_t Thread1,Thread2;
  int m1[5],m2[5];
  void *total1,*total2;
  Display Student 1
  input(m1);//Inputing marks of Student 1
  pthread_create(&Thread1,NULL,Compute_Sum,(void*)m1);
  Display Student 2
  input(m2);//Inputing marks of Student 2
  pthread_create(&Thread2,NULL,Compute_Sum,(void*)m2);
  pthread_join(Thread1,&total1);
  pthread_join(Thread2,&total2);
  Display("\nSum of Student 1 marks : %d \n",(int*)total1);
  Display("\nSum of Student 2 marks : %d \n",(int*)total2);
  if (total1>total2)
    Display("Student 1 scored Highest Marks");
  else if(total1==total2)
    Display("Student 1 and Student 2 have Equal Total Marks\n");
  else
     Display("Student 1 scored Highest Marks\n");
}
```

## 6. Programs

### 1.Hello World message Program by creating 3 different threads

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *thread(void*arg){
    printf("%s by Child thread\n" ,(char*)arg);
    pthread_exit("Bye Have a nice day");

}
int main(void) {
    int retcode1,retcode2,retcode3,i;
    pthread_t t1,t2,t3;
    void*exitstatus;
    retcode1=pthread_create(&t1,NULL,thread,"Hello World");
    retcode2=pthread_create(&t2,NULL,thread,"Hello World");
    retcode3=pthread_create(&t3,NULL,thread,"Hello World");
    pthread_join(t1,&exitstatus);
    pthread_join(t2,&exitstatus);
    pthread_join(t3,&exitstatus);
    printf("From Child Thread = %s\n",(char*)exitstatus);



    return (EXIT_SUCCESS);
}
```

### 2.Program for Student Problem

```c
// Program Done By Deepak R 18ETCS002041
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void input(int arr[5]){
    for(int i=0;i<5;i++){
        printf("Enter Subject %d marks (out of 10) : ",i+1);
        scanf("%d",&arr[i]);
    }
}
void *Compute_Sum(void *arg){//Chid Thread execution Block
    int sum=0;
    int *arr=(int*)arg;
    for(int i=0;i<5;i++)
        sum+=arr[i];
    pthread_exit(sum);
}
int main() {//main process Block]
    pthread_t Thread1,Thread2;
    int m1[5],m2[5];
    void *total1,*total2;
    printf("Student 1\n");
    input(m1);//Inputing marks of Student 1
    pthread_create(&Thread1,NULL,Compute_Sum,(void*)m1);
    printf("Student 2\n");
    input(m2);//Inputing marks of Student 2
    pthread_create(&Thread2,NULL,Compute_Sum,(void*)m2);
    pthread_join(Thread1,&total1);
    pthread_join(Thread2,&total2);
    printf("\nSum of Student 1 marks : %d \n",(int*)total1);
    printf("\nSum of Student 2 marks : %d \n",(int*)total2);

    if (total1>total2)
        printf("Student 1 scored Highest Marks\n");
    else if(total1==total2)
        printf("Student 1 and Student 2 have Equal Total Marks\n");

    else
        printf("Student 2 scored Highest Marks\n");
    return (EXIT_SUCCESS);
}
```

## 7. Presentation of Results

### Result of Hello world Program by creating 3 different threads

```
Hello World by Child thread
Hello World by Child thread
Hello World by Child thread
From Child Thread = Bye Have a nice day
```

**Test case 1** when Student 1 Scores More marks than Student 2

```
Student 1
Enter Subject 1 marks (out of 10) : 6
Enter Subject 2 marks (out of 10) : 4
Enter Subject 3 marks (out of 10) : 5
Enter Subject 4 marks (out of 10) : 3
Enter Subject 5 marks (out of 10) : 6
Student 2
Enter Subject 1 marks (out of 10) : 3
Enter Subject 2 marks (out of 10) : 5
Enter Subject 3 marks (out of 10) : 6
Enter Subject 4 marks (out of 10) : 2
Enter Subject 5 marks (out of 10) : 5

Sum of Student 1 marks : 24

Sum of Student 2 marks : 21
Student 1 scored Highest Marks
```

**Test case 2** when Student 2 scores more than Student 1

```
Student 1
Enter Subject 1 marks (out of 10) : 7
Enter Subject 2 marks (out of 10) : 5
Enter Subject 3 marks (out of 10) : 3
Enter Subject 4 marks (out of 10) : 2
Enter Subject 5 marks (out of 10) : 4
Student 2
Enter Subject 1 marks (out of 10) : 8
Enter Subject 2 marks (out of 10) : 8
Enter Subject 3 marks (out of 10) : 7
Enter Subject 4 marks (out of 10) : 6
Enter Subject 5 marks (out of 10) : 9

Sum of Student 1 marks : 21

Sum of Student 2 marks : 38
Student 2 scored Highest Marks
```

**Test case 3** when both Students score equal marks

```
Student 1
Enter Subject 1 marks (out of 10) : 5
Enter Subject 2 marks (out of 10) : 4
Enter Subject 3 marks (out of 10) : 5
Enter Subject 4 marks (out of 10) : 3
Enter Subject 5 marks (out of 10) : 5
Student 2
Enter Subject 1 marks (out of 10) : 2
Enter Subject 2 marks (out of 10) : 4
Enter Subject 3 marks (out of 10) : 5
Enter Subject 4 marks (out of 10) : 7
Enter Subject 5 marks (out of 10) : 4

Sum of Student 1 marks : 22

Sum of Student 2 marks : 22
Student 1 and Student 2 have Equal Total Marks
```

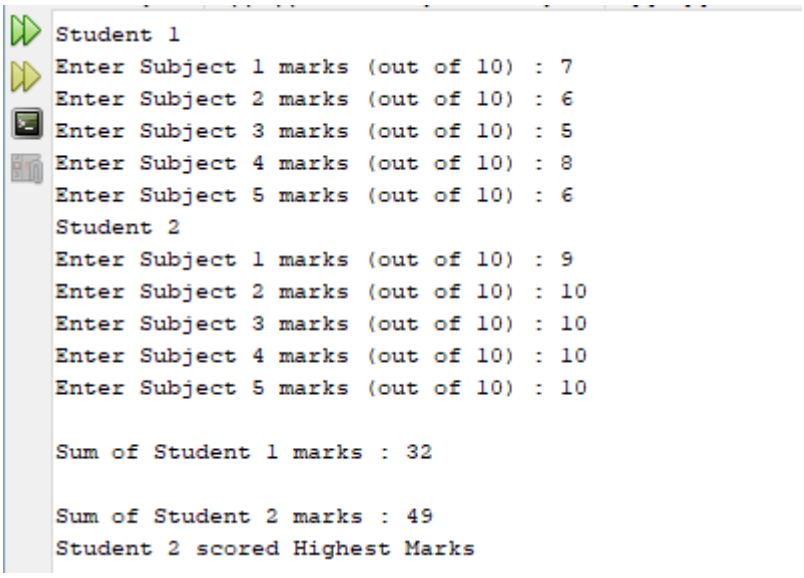
**Test case 4** when Student 1 Scores More marks than Student 2

```
Student 1
Enter Subject 1 marks (out of 10) : 6
Enter Subject 2 marks (out of 10) : 8
Enter Subject 3 marks (out of 10) : 7
Enter Subject 4 marks (out of 10) : 8
Enter Subject 5 marks (out of 10) : 9
Student 2
Enter Subject 1 marks (out of 10) : 4
Enter Subject 2 marks (out of 10) : 3
Enter Subject 3 marks (out of 10) : 2
Enter Subject 4 marks (out of 10) : 5
Enter Subject 5 marks (out of 10) : 6

Sum of Student 1 marks : 38

Sum of Student 2 marks : 20
Student 1 scored Highest Marks
```

**Test case 5** when Student 2 Scores More marks than Student 1



```
Student 1
Enter Subject 1 marks (out of 10) : 7
Enter Subject 2 marks (out of 10) : 6
Enter Subject 3 marks (out of 10) : 5
Enter Subject 4 marks (out of 10) : 8
Enter Subject 5 marks (out of 10) : 6
Student 2
Enter Subject 1 marks (out of 10) : 9
Enter Subject 2 marks (out of 10) : 10
Enter Subject 3 marks (out of 10) : 10
Enter Subject 4 marks (out of 10) : 10
Enter Subject 5 marks (out of 10) : 10

Sum of Student 1 marks : 32

Sum of Student 2 marks : 49
Student 2 scored Highest Marks
```

## 8. Analysis and Discussions

In Hello Program 3 different threads with different threads ID's are generated to print Hello World, follows the program output for the student marks calculation, 2 threads are created since 2 students are taken into consideration for now, which can be later changed in the program parameters, each of the thread calculates the total marks of their respective student and stores the result in thr_data, this is the structure we passed as a reference to the thread function to have a sort of multiple arguments, which can be later dereferenced in the thread function to obtain the members of the structure. The program then waits for all the threads to finish their job and the main thread then selects the student with the maximum marks and displays it to stdout.

## 9. Conclusions

| pthread_create - create a new thread |
|---|
| **#include <pthread.h>**<br><br>**int pthread_create(pthread_t ****thread***, const pthread_attr_t ****attr***,**<br>**void *(****start_routine***) (void *), void ****arg***);**<br><br>Compile and link with *-pthread*. |
| Description:<br>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*().<br><br>If attr is NULL, then the thread is created with default attributes.<br>The initial value of the new thread's CPU-time clock is 0 |

| pthread_join - join with a terminated thread |
|---|
| **#include <pthread.h>**<br><br>**int pthread_join(pthread_t** *thread***, void *****retval***);**<br><br>Compile and link with *-pthread*. |
| Description:<br>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.<br>If retval is not NULL, then pthread_join() copes the exit status of the target thread. |

## 10. Comments

### 1. Limitations of Experiments

The number of threads is assumed to be equal to the number of students, this will become inefficient when the number of students increase, the experiment can be further enhanced to distribute the work among the threads, such as giving a batch job to each thread. Furthermore, the work of the main thread can be reduced by using a critical section variable such as max_marks, which can be updated as soon as any of the thread finished.

### 2. Limitations of Results

The results are limited to the fact that the potential of multithreading wasn't used, the program has to be simulated for a fair amount of threads and students to get a satisfiable result, which can then be compared to single threaded application.

### 3. Learning happened

The use of POSIX threads was learnt, such as creating and destroying threads, and passing data to and from threads.

### 4. Recommendations

Load balance the threads for better performance by giving batch jobs and also reduce the work on the main thread.