**CSS 330 - Operating System: ASSIgnment 2**

**MISBAH NOOR AWAn (333811)**

**FAiq jamal (344994)**

**BESE 11A**

**SHORT REPORT ON LAB 10:**

The subject of the lab was about Programming threads and goal was to learn the threads and their features programmatically. We worked on it using Pthreads which are standard threads API for creating and manipulating threads in Unix based environment.

We learnt that Pthread API are grouped into thread management, Mutexes, Conditional variables and Synchronization. Further we studied creation of threads, Attributes of threads, the terminating threads and then the joining of threads.

Task 1 of the lab was to learn how to create threads and is explained below with the help of various colors.

|  |
| --- |
| Creation Code Explained |
|  |
|  |

The second task was to use mutexes in threads. The code is explained below using color coding.

|  |
| --- |
| Mutexes Code Explained |
|  |
|  |

The 3rd lab task was to join the threads so one thread only executes when previous thread is done executing.

|  |
| --- |
| Joins Code Explained |
|  |
|  |

**SHORT REPORT ON LAB 11:**

The subject of the lab was about Programming threads and goal was to learn the threads and their features programmatically. We worked on it using Pthreads which are standard threads API for creating and manipulating threads in Unix based environment.

We learnt that Pthread API are grouped into thread management, Mutexes, Conditional variables and Synchronization. Further we studied creation of threads, Attributes of threads, the terminating threads and then the joining of threads.

Task 1 of the lab was about to implement a simple program using for loop to add first million positive integers elements in an array. The program was implemented in C language whose code and output is given below.

**Task1 code and output**:

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <time.h>    static long long int sum;  static int arr[100000000];    int main()  {  *//fill array*  for (int i = 0; i < 100000000; i++)  arr[i] = i + 1;    *//sum all members of array*  for (int i = 0; i < 100000000; i++)  sum += arr[i];    *//print sum*  printf("sum of first 100,000,000 positive integers: %lld**\n**",sum);    return 0;  } |

|  |
| --- |
|  |

Task2 demanded the conversion of previous code into multi-threaded program using Pthread library to show 4 threads running in parallel, code and output of which is given below

**Task2 code and output:**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>    #define totalThreads 4  #define totalElements 10000000    static int arr[totalElements];  static long long int sum;    *// define node for thread and its number*  typedef struct arg\_data {  int thread\_number;  } arg\_data;      void\* worker\_sum(void\* arg)  {  arg\_data\* current\_thread\_data = (arg\_data\*)arg;  int endpart = (current\_thread\_data->thread\_number) \* (totalElements / totalThreads);  int startpart = endpart - (totalElements / totalThreads);    long long int current\_thread\_sum = 0;  for (int i = startpart; i < endpart; i++) {  current\_thread\_sum += arr[i];  }  sum += current\_thread\_sum;    return NULL;  }    int main()  {  *//fill array*  for (int i = 0; i < totalElements; i++)  arr[i] = i + 1;    *//pthread objects*  pthread\_t id[totalThreads];    *//argument data to send in worker functions*  arg\_data arg\_arr[totalThreads];    *//total number of threads we will create*  int no\_of\_threads = totalThreads;    int thread\_no = 1;    *//creating the child threads*  for (thread\_no = 1; thread\_no <= totalThreads; thread\_no++) {  arg\_arr[thread\_no - 1].thread\_number = thread\_no;  pthread\_create(&id[thread\_no - 1], NULL, worker\_sum, &arg\_arr[thread\_no - 1]);  }    *//joining the threads one by one*  for (int i = 1; i <= totalThreads; i++)  pthread\_join(id[i - 1], NULL);  printf("sum of first 100,000,000 positive integers: %lld**\n**", sum);    return 0;  } |
|  |

The comparison of above two outputs are same because in both the cases elements of array are same and hence add up to 50000005000000.

Task3 was about to run the above code several time and verify that threads concurrently in each run. The output is shown below:

**Tas3 output:**

|  |
| --- |
|  |

Finally the task 4 was about using the code implemented in task 1 and task 2 to analyze and print the time each program takes to execute and the explanation over the results achieved.

For this purpose the only change made to the codes was introduction of clock using the following lines of code:

|  |
| --- |
| clock\_t start, end;  double cpu\_time\_taken;  start = clock(); This initiates the clock added before loop  end = clock();  cpu\_time\_taken = ((double)(end - start)) / CLOCKS\_PER\_SEC;  This is to stop the clock and calculate time lapse in seconds, added after the program concludes. |

**Clocked Task1 output:**

|  |
| --- |
|  |

**Clocked Task2 output:**

|  |
| --- |
|  |

**RESULT EXPLANATION:**

**The above output shows that the simple task of adding array elements too more time of 0.616754 seconds while same array elements in multi threads took only 0.021460 seconds. This shows that the multi-threading works faster. This is because in task1, the summing is done sequentially whereas in multithreading the threads are running concurrently. multithreading is extremely useful as you can utilize your CPU properly.**