|  |  |  |  |
| --- | --- | --- | --- |
| **Lab No** | 08 | **Reg. No** | 224921 |
| **Student Name** | Muhammad Saad Tariq | **Section** | A |

|  |
| --- |
| Solution Task 1 |
| Code:  /\*  How to run code:  - Save code in a .c file : task1.c  - Compile using lpthread : gcc task1.c -o task1.out -lpthread  - Run by entering the arguments of dividing the array into four parts  ./task1.out 0 25 50 75  \*/  #include <stdio.h>  #include <pthread.h>  #define array\_size 100  void \*calc\_sum(void \*param);  int numbers[array\_size];  int sum = 0;  pthread\_mutex\_t mtx = PTHREAD\_MUTEX\_INITIALIZER;  int main(int argc, void \*argv[]){  int i;  for(i = 0; i < array\_size; i++){  numbers[i] = i+1;  }    pthread\_t tid1, tid2, tid3, tid4;  pthread\_create(&tid1, NULL, calc\_sum, argv[1]);  pthread\_create(&tid2, NULL, calc\_sum, argv[2]);  pthread\_create(&tid3, NULL, calc\_sum, argv[3]);  pthread\_create(&tid4, NULL, calc\_sum, argv[4]);  pthread\_join(tid1, NULL);  pthread\_join(tid2, NULL);  pthread\_join(tid3, NULL);  pthread\_join(tid4, NULL);  printf("\n");  printf("Sum of first 100 numbers: %d\n", sum);  return 0;  }  void \*calc\_sum(void \*param){  pthread\_mutex\_lock(&mtx);  int starting = atoi(param);  int limit = (starting + 25);  for(starting; starting < limit; starting++){  sum += numbers[starting];  }  pthread\_mutex\_unlock(&mtx);  return NULL;  }  Output: |

|  |
| --- |
| Solution Task 2 |
| Code:  /\*  How to run code:  - Save code in a .c file : task1.c  - Compile using lpthread : gcc task2.c -o task2.out -lpthread  - Run by entering the arguments of dividing the array into four parts  ./task1.out 0 25 50 75  \*/  #include <stdio.h>  #include <pthread.h>  #define array\_size 100  void \*calc\_sum(void \*param);  int numbers[array\_size];  int total\_sum = 0;  int main(int argc, void \*argv[]){  int i;  for(i = 0; i < array\_size; i++){  numbers[i] = i+1;  }    pthread\_t tid1, tid2, tid3, tid4;  pthread\_create(&tid1, NULL, calc\_sum, argv[1]);  pthread\_create(&tid2, NULL, calc\_sum, argv[2]);  pthread\_create(&tid3, NULL, calc\_sum, argv[3]);  pthread\_create(&tid4, NULL, calc\_sum, argv[4]);  pthread\_join(tid1, NULL);  pthread\_join(tid2, NULL);  pthread\_join(tid3, NULL);  pthread\_join(tid4, NULL);  printf("\n");  printf("Sum of first 100 numbers: %d\n", total\_sum);  return 0;  }  void \*calc\_sum(void \*param){  int sum = 0;  int starting = atoi(param);  int start\_counter = starting;  int limit = (starting + 25);  for(starting; starting < limit; starting++){  sum += numbers[starting];  }  printf("\nSum (%d to %d): %d", start\_counter, limit, sum);  total\_sum += sum;  pthread\_exit(0);  }  Output: |

|  |
| --- |
| Solution Task 3 |
| The approach without using synchronization primitives would be faster. This is because when we use synchronization primitives, each thread has to wait to gain access of the lock first and they can only gain access once the previous thread possessing the access unlocks it first. Thus, unless the previous controlling thread does not release the lock variable the new thread cannot execute, therefore context switch being of no use. This causes the program to work slowly as all threads have to wait to gain access of the locked variable and then execute.  On the other hand, in case of using local sums, context switches can take place allowing all threads to execute themselves and store results in their local sums, later forwarding it to the main function, thus being faster. |