HIGH PERFORMANCE COMPUTING

SEMESTER 6, 2022

Netaji Subhas University of Technology

New Delhi-1100784

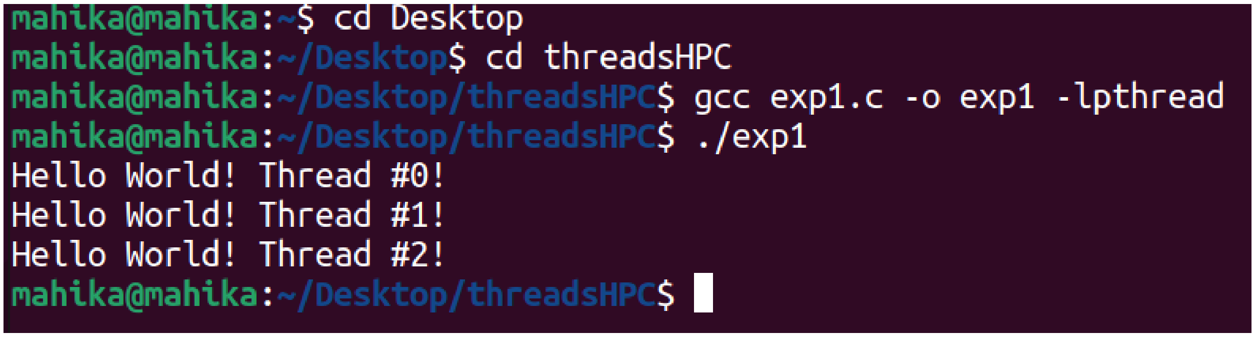
Mahika Kushwaha

2020UCO1659

Section : COE 3

Experiment 1:
Run a basic hello World program using pthreads

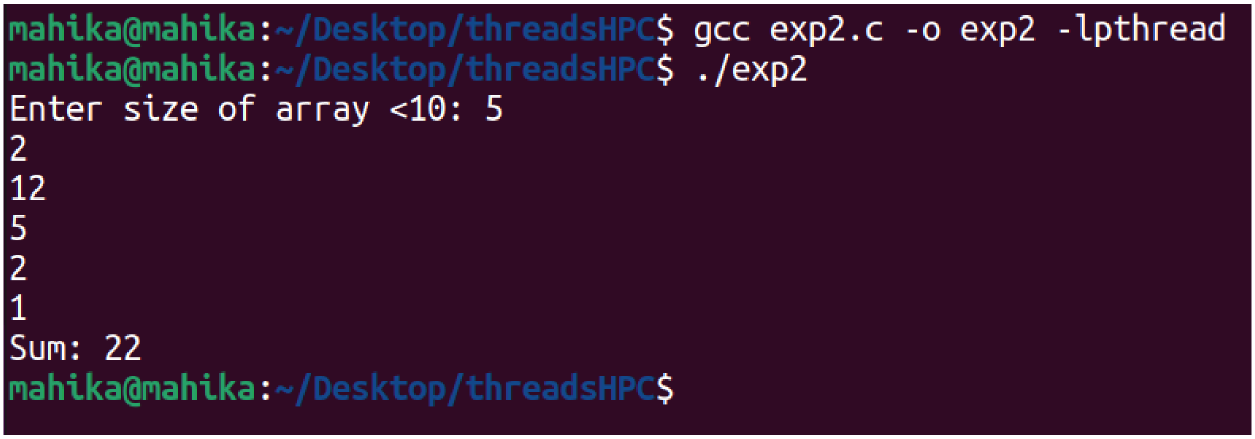
```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 3

void *PrintHello(void *threadid)

{
   long tid;
   tid = (long)threadid;
   printf("Hello World! Thread #%ld!\n", tid);
   pthread_exit(NULL);
}

int main(int argc, char *argv[])

{
   pthread_t threads[NUM_THREADS];
   int rc;
   long t;

   for (t = 0; t < NUM_THREADS; t++)
   {
      rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
      if (rc)
      {
         printf("ERROR; return code from pthread_create() is %d\n", rc);
         exit(-1);
      }
   }

   /* Last thing that main() should do */
   pthread_exit(NULL);
}
```



```
mahika@mahika:~$ cd Desktop
mahika@mahika:~/Desktop$ cd threadsHPC
mahika@mahika:~/Desktop/threadsHPC$ gcc exp1.c -o exp1 -lpthread
mahika@mahika:~/Desktop/threadsHPC$ ./exp1
Hello World! Thread #0!
Hello World! Thread #1!
Hello World! Thread #2!
mahika@mahika:~/Desktop/threadsHPC$ 
```

Experiment 2:
Find the sum of all elements of an array using two processors using pthreads

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define ARRAY_SIZE 10// size of the array
#define NUM_THREADS 2 // number of threads

int array[ARRAY_SIZE]; // the array to sum
int partial_sums[NUM_THREADS]; // partial sums for each thread
pthread_mutex_t lock; // mutex for accessing partial_sums

// thread function to compute partial sum
void *partial_sum(void *thread_id) {
    int id = *(int *)thread_id;
    int start = id * (ARRAY_SIZE / NUM_THREADS);
    int end = (id + 1) * (ARRAY_SIZE / NUM_THREADS);

    int sum = 0;
    for (int i = start; i < end; i++) {
        sum += array[i];
    }

    pthread_mutex_lock(&lock);
    partial_sums[id] = sum;
    pthread_mutex_unlock(&lock);

    pthread_exit(NULL);
}

int main() {
    int n;
    printf("Enter size of array <10: ");
    scanf("%d",&n);
    for (int i = 0; i < n; i++) {
        scanf("%d",&array[i]);
    }

    // initialize the mutex
    pthread_mutex_init(&lock, NULL);

    pthread_t threads[NUM_THREADS];
    int thread_ids[NUM_THREADS];

    // create the threads
    for (int i = 0; i < NUM_THREADS; i++) {
        thread_ids[i] = i;
        pthread_create(&threads[i], NULL, partial_sum, (void *)&thread_ids[i]);
    }

    // join the threads
    for (int i = 0; i < NUM_THREADS; i++) {
        pthread_join(threads[i], NULL);
    }

    // compute the final sum
    int sum = 0;
    for (int i = 0; i < NUM_THREADS; i++) {
        sum += partial_sums[i];
    }

    // print the final sum
    printf("Sum: %d\n", sum);

    // destroy the mutex
    pthread_mutex_destroy(&lock);

    return 0;
}
```

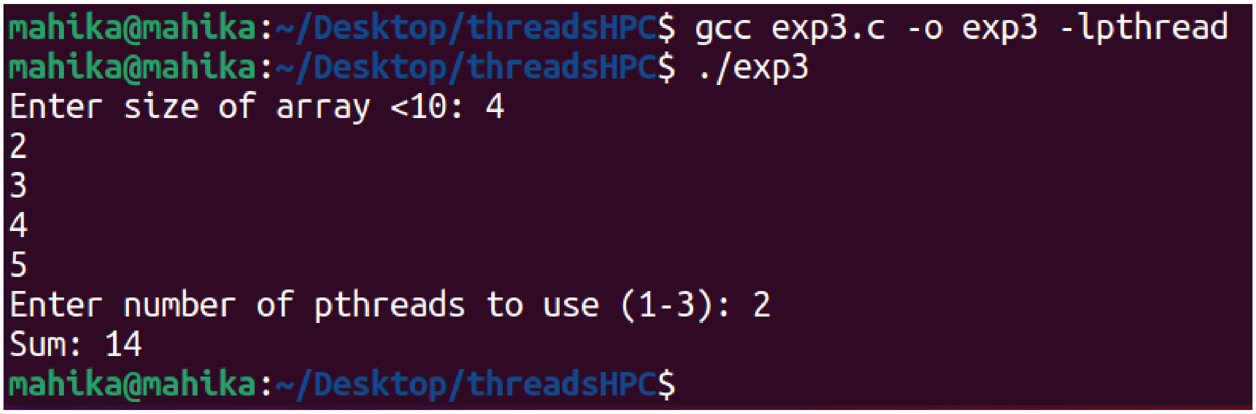


```
mahika@mahika:~/Desktop/threadsHPC$ gcc exp2.c -o exp2 -lpthread
mahika@mahika:~/Desktop/threadsHPC$ ./exp2
Enter size of array <10: 5
2
12
5
2
1
Sum: 22
mahika@mahika:~/Desktop/threadsHPC$
```

Experiment 3:
Find sum of all elements using p processors by using pthreads

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#define ARRAY_SIZE 10 // size of the array
int array[ARRAY_SIZE]; // the array to sum
int sum_global = 0; // global variable to store the sum
pthread_mutex_t lock; // mutex for accessing sum

int pthread_num; // global variable to store number of pthreads

pthread_once_t once_control = PTHREAD_ONCE_INIT; // control variable for
pthread_once

// function to initialize pthread_num using sysconf
```

```
void initialize_pthread_num() {
    pthread_num = sysconf(_SC_NPROCESSORS_ONLN);
}

// thread function to compute partial sum
void *partial_sum(void *thread_id) {
    int id = *(int *)thread_id;

    int start = id * (ARRAY_SIZE / pthread_num);
    int end = (id + 1) * (ARRAY_SIZE / pthread_num);

    int sum = 0;
    for (int i = start; i < end; i++) {
        sum += array[i];
    }

    pthread_mutex_lock(&lock);
    sum_global += sum;
    pthread_mutex_unlock(&lock);

    pthread_exit(NULL);
}

int main() {
    // initialize the array with random values
    int n;
    printf("Enter size of array <10: ");
    scanf("%d",&n);
    for (int i = 0; i < n; i++) {
        scanf("%d",&array[i]);
    }

    // initialize the mutex
    pthread_mutex_init(&lock, NULL);

    // initialize pthread_num using pthread_once
    pthread_once(&once_control, initialize_pthread_num);
    // prompt user for number of pthreads to use
    printf("Enter number of pthreads to use (1-%d): ", pthread_num);
    scanf("%d", &pthread_num);
    if (pthread_num < 1 || pthread_num > pthread_num) {
        printf("Invalid number of pthreads\n");
        return 1;
    }

    pthread_t threads[pthread_num];
    int thread_ids[pthread_num];

    // create the threads
    for (int i = 0; i < pthread_num; i++) {
        thread_ids[i] = i;
        pthread_create(&threads[i], NULL, partial_sum, (void *)&thread_ids[i]);
    }

    // join the threads
    for (int i = 0; i < pthread_num; i++) {
        pthread_join(threads[i], NULL);
    }

    // print the final sum
    printf("Sum: %d\n", sum_global);

    // destroy the mutex
    pthread_mutex_destroy(&lock);

    return 0;
}
```



```
mahika@mahika:~/Desktop/threadsHPC$ gcc exp3.c -o exp3 -lpthread
mahika@mahika:~/Desktop/threadsHPC$ ./exp3
Enter size of array <10: 4
2
3
4
5
Enter number of pthreads to use (1-3): 2
Sum: 14
mahika@mahika:~/Desktop/threadsHPC$
```

Experiment 4:
Illustrate basic mpi communication routines

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[]) {
    int rank, size, data;
    MPI_Status status;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    // Send and receive data
    if (rank == 0) {
        data = 123;
        MPI_Send(&data, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        printf("Process %d sent data %d to process 1\n", rank, data);

        MPI_Recv(&data, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &status);
        printf("Process %d received data %d from process 1\n", rank, data);
    }
    else if (rank == 1) {
        MPI_Recv(&data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
        printf("Process %d received data %d from process 0\n", rank, data);
        data = 456;
        MPI_Send(&data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
        printf("Process %d sent data %d to process 0\n", rank, data);
    }

    // Barrier
    MPI_Barrier(MPI_COMM_WORLD);
    printf("Process %d passed the barrier\n", rank);

    // Finalize
    MPI_Finalize();
    printf("Process %d finalized\n", rank);

    return 0;
}
```

Experiment 5:
Design a parallel program for summing up an array, matrix multiplication and show logging and tracing mpi activity