6CS005
High Performance Computing
Week 3 Workshop

Revision on Multithreading

Student Number: 2408869

Name: Narayan Lohani

Group: L6CG15

Table of Contents

Explanation

3. Write a multithreaded C program that creates multiple threads (prompt user for number of threads to use) to find all prime numbers between 1 and 10,000. Each thread must count how many primes it finds and returns that count to the main thread using pthread_exit(). The main thread should then print how many prime numbers each thread finds.

To solve this problem, a thread routine was created in which an integer variable, namely count, was dynamically allocated and initialized to 0. The thread routine then typecasted the argument from which the start and stop of number to find the primes from were extracted. Iteratively from start to stop, a user-defined function is_prime() was called which returns 0 if the number is composite and 1 if the number is prime. If the iterating number is prime, then the value count is increased by 1. When the iteration was completed, the total number of primes found would be stored in count variable and the routine would exit with the count variable using pthread_exit().

The program begins by prompting the user to enter the number of threads to create. Memory is allocated for arrays to store thread identifiers and their respective number ranges. The total range of numbers from 1 to 10,000 is divided among the threads so that each thread processes a roughly equal subset. The last thread handles any remaining numbers to ensure the entire range is covered. Each thread is created using pthread_create() and executes the prime-finding routine in parallel.

After all threads are started, the main function waits for their completion using pthread_join(). A pointer is used to receive the count returned by each thread. The program prints the number of primes found by each thread and then frees the dynamically allocated memory for the count variable. Finally, the memory allocated for the thread and range arrays is also released to prevent memory leaks.

Output of Program 3 Source of Program 3

4. Write a multithreaded C program that uses multiple threads (prompt user for number of threads to use) to find prime numbers between 1 and 10,000. As soon as the 5th prime number is found, the program should cancel all other threads as soon as possible using pthread_cancel().

To solve this problem, following components were used:

Global Variables

- prime_counts (int): to count number of prime numbers found combinedly by all threads
- lock (pthread_mutex_t): to protect global variables from race condition

Structures

- typedef – start_stop: start (int), stop (int), threads (pthread_t*), total_threads (int)

User Defined Function

- is_prime(int n): returns 1 if a number is prime, 0 otherwise
- main(): handles flow of the program

Thread Routine

- print_five_primes(void* args)

The main function prompts user for number of threads to use. Using malloc memory for threads and ranges(start_stop) are created. Number to be calculated each thread are divided roughly equally, the remaining numbers are taken by last thread. It also initializes the mutex using pthread_mutex_init().

For each thread, range which store start and stop numbers, threads pointer, thread_size are calculated and assigned. Then finally threads are created using pthread_create() and respective ranges are passed as arguments.

The thread routine firstly typecasts the argument into start_stop. Then cancellation point is set at the beginning of the loop. Inside loop, for each number between start and stop, is_prime() function is called. If the number is not prime, then it simply skips this number and goes to next one. However, if the number is prime then,

mutex is locked, and prime_count is checked if it is less than 5. If the prime_count is less than 5 then, prime_count is increased by one. After increasing the prime_count again, the prime_count is checked, has the recent increment cause the prime_count to be 5? Because our goal is to print only first 5 print numbers. If the recent increment indeed was the fifth prime number, then in a loop, call pthread_cancel all other threads using pthread_cancel. Unlock the mutex and exit itself too using pthread_exit().

Output of Program 4 Source of Program 4

Output of Program 3

```
lhn_nryn@DESKTOP-CJ2TM4F: /mnt/d/narayan/hpc_herald/week3/workshop/submission
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/submission$ ls
 3.exe   4.exe   NarayanLohani_week3_hpc_workshop.docx   q3.c   q4.c  '~$rayanLohani_week3_hpc_wor
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/submission$ gcc q3.c -o 3 -lm
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/submission$ ./3
Number of threads : 7
Thread 0 found 225 prime numbers
Thread 1 found 189 prime numbers
Thread 2 found 174 prime numbers
Thread 3 found 164 prime numbers
Thread 4 found 162 prime numbers
Thread 5 found 153 prime numbers
Thread 6 found 162 prime numbers
```

Output of Program 4

```
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/submission$ gcc q4.c -o 4 -lm
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/submission$ ./4
Number of threads: 3
1. Thread 130585181812416 found 2
2. Thread 130585181812416 found 3
3. Thread 130585181812416 found 5
4. Thread 130585181812416 found 7
5. Thread 130585181812416 found 11
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/submission$
```

Source Code of Program 3

hpc_herald - q3.c

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

// Upper limit: find primes in [1..MAX]
#define MAX  10000

// Structure describing a range [start, stop] for a thread to process
typedef struct {
    int start, stop;
} start_stop;

// Check if n is prime; Returns 1 if prime, 0 otherwise.
int is_prime(int n) {
    if (n < 2) return 0;
    int limit = (int)sqrt(n);
    for (int i = 2; i <= limit; i++) {
        if (n % i == 0) {
            return 0;
        }
    }
    return 1;
}

/*
    Thread routine: count primes in the assigned range.
    Allocates an int on the heap, stores the count there and returns
    the pointer via pthread_exit.
*/
void* prime_finder(void* arg) {
    int* count = (int*)malloc(sizeof(int));
    *count = 0;
    start_stop* s_s = (start_stop*)arg;
    // Iterate inclusive from start to end and increment count for primes
    for (int i = s_s->start; i <= s_s->stop; i++) {
        if (is_prime(i)) {
            (*count)++;
        }
    }
    // Return pointer to the result
    pthread_exit(count);
}
```

hpc_herald - q3.c

```c
int main() {
    int threads_size;

    // Ask user for number of threads to use
    printf("Number of threads : ");
    scanf("%d", &threads_size);

    // Allocate arrays for pthread_t handles and start_stop endpoints
    pthread_t* threads = malloc(threads_size * sizeof(pthread_t));
    start_stop* range = malloc(threads_size * sizeof(start_stop));

    /* Divide the interval [1..MAX] into roughly equal blocks.
       Last thread takes the  remainder threads */
    int block_size = MAX / threads_size;

    for (int i = 0; i < threads_size; i++) {
        /* Compute inclusive start and stop for this thread's block */
        range[i].start = (i * block_size) + 1;
        range[i].stop = (i == threads_size - 1) ? MAX : (i + 1) * block_size;

        /* Create thread to count primes in range[i] */
        pthread_create(&threads[i], NULL, prime_finder, &range[i]);
    }

    /*
    Comment to verify prime count | uncomment lines with : // C3
    */
    // int total_prime_number = 0; // C3
    for (int i = 0; i < threads_size; i++) {
        void* retval;
        pthread_join(threads[i], &retval);
        printf("Thread %d found %d prime numbers\n", i, *(int*)retval);
        // total_prime_number += *(int*)retval; // C3
        free(retval); // free the heap-allocated count returned by thread
    }
    // Print final total and free allocated arrays
    // printf("\nAt the end, total prime numbers : %d\n", total_prime_number); // C3
    free(threads);
    free(range);

    return 0;
}

```

Source Code of Program 4

hpc_herald - q4.c

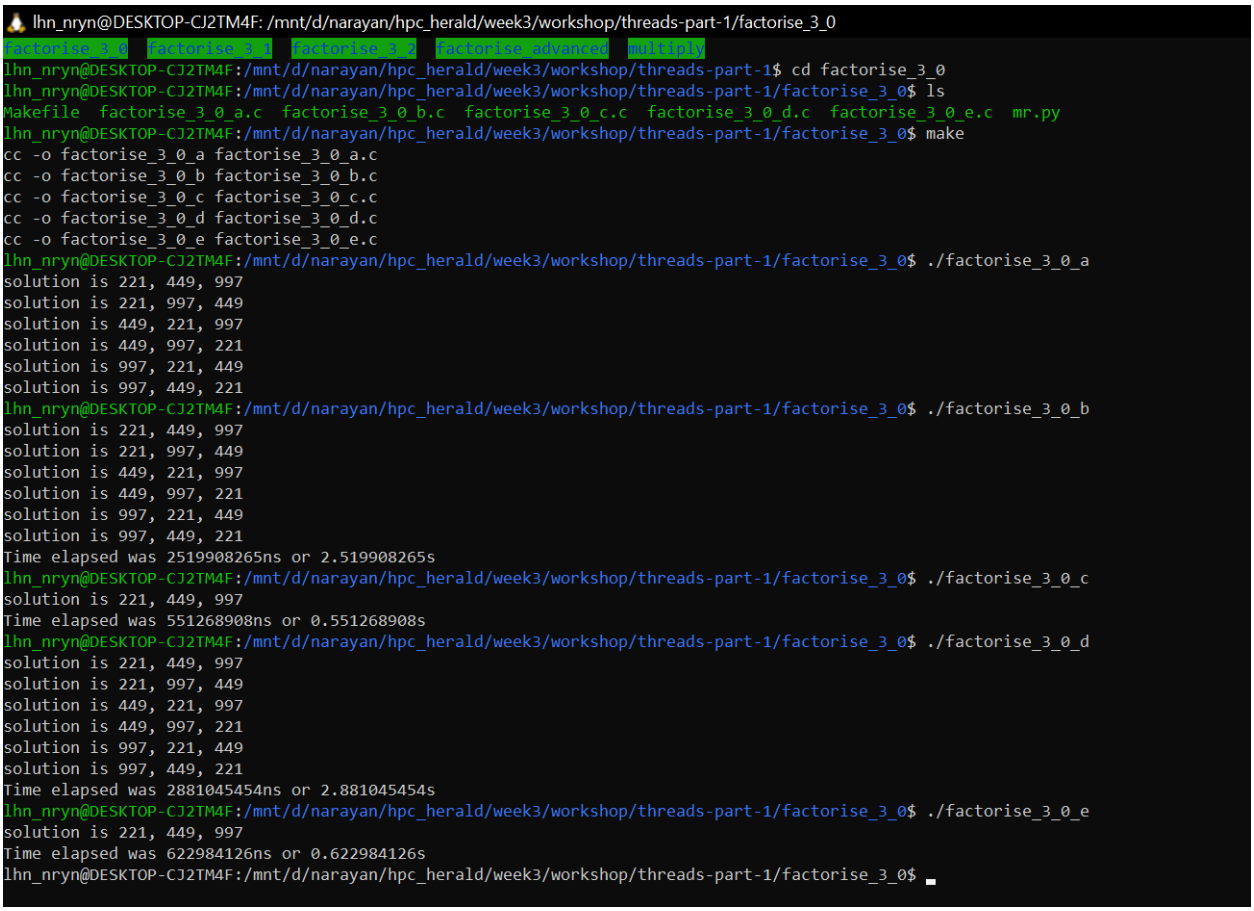
```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <math.h>

#define MAX 10000

int prime_counts = 0;
pthread_mutex_t lock;

typedef struct {
    int start, stop;
    pthread_t* threads;
    int total_threads;
} start_stop;

// Check if n is prime
int is_prime(int n) {
    if (n < 2) return 0;
    int limit = (int)sqrt(n);
    for (int i = 2; i <= limit; i++)
        if (n % i == 0)
            return 0;
    return 1;
}

// Thread function
void* print_five_primes(void* arg) {
    start_stop* s_s = (start_stop*)arg;

    for (int i = s_s->start; i <= s_s->stop; i++) {
        pthread_testcancel(); // cancellation checkpoint

        if (is_prime(i)) {
            pthread_mutex_lock(&lock);

            if (prime_counts < 5) {
                prime_counts++;
                printf("%d. Thread %ld found %d\n", prime_counts, pthread_self(), i);
            }
            // If 5th prime found, cancel all other threads and exit
            if (prime_counts >= 5) {
                for (int t = 0; t < s_s->total_threads; t++) {
                    if (!pthread_equal(s_s->threads[t], pthread_self())) {
                        pthread_cancel(s_s->threads[t]);
                    }
                }
                pthread_mutex_unlock(&lock);
                pthread_exit(NULL);
            }
            pthread_mutex_unlock(&lock);
        }
    }
    return NULL;
}
```

hpc_herald - q4.c

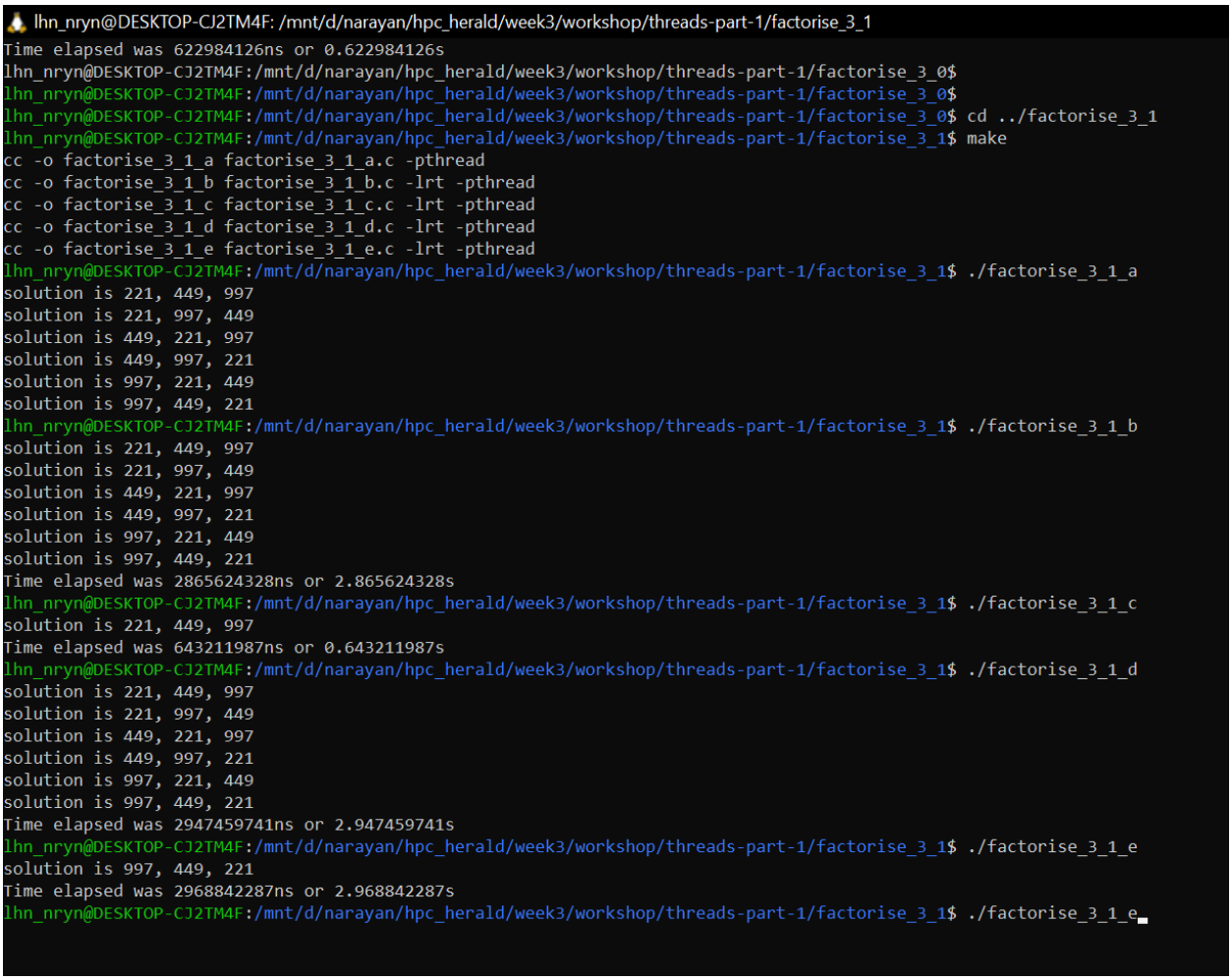
```
56
57  int main() {
58      int threads_size;
59      printf("Number of threads: ");
60      scanf("%d", &threads_size);
61
62      pthread_t* threads = malloc(threads_size * sizeof(pthread_t));
63      start_stop* range = malloc(threads_size * sizeof(start_stop));
64
65      int block_size = MAX / threads_size;
66
67      pthread_mutex_init(&lock, NULL);
68
69      for (int i = 0; i < threads_size; i++) {
70          range[i].start = i * block_size + 1;
71          range[i].stop = (i == threads_size - 1) ? MAX : (i + 1) * block_size;
72          range[i].threads = threads;
73          range[i].total_threads = threads_size;
74          pthread_create(&threads[i], NULL, print_five_primes, &range[i]);
75      }
76
77      for (int i = 0; i < threads_size; i++)
78          pthread_join(threads[i], NULL);
79
80      pthread_mutex_destroy(&lock);
81      free(threads);
82      free(range);
83      return 0;
84  }
85
```

Uni file execution

Factorise_3_0



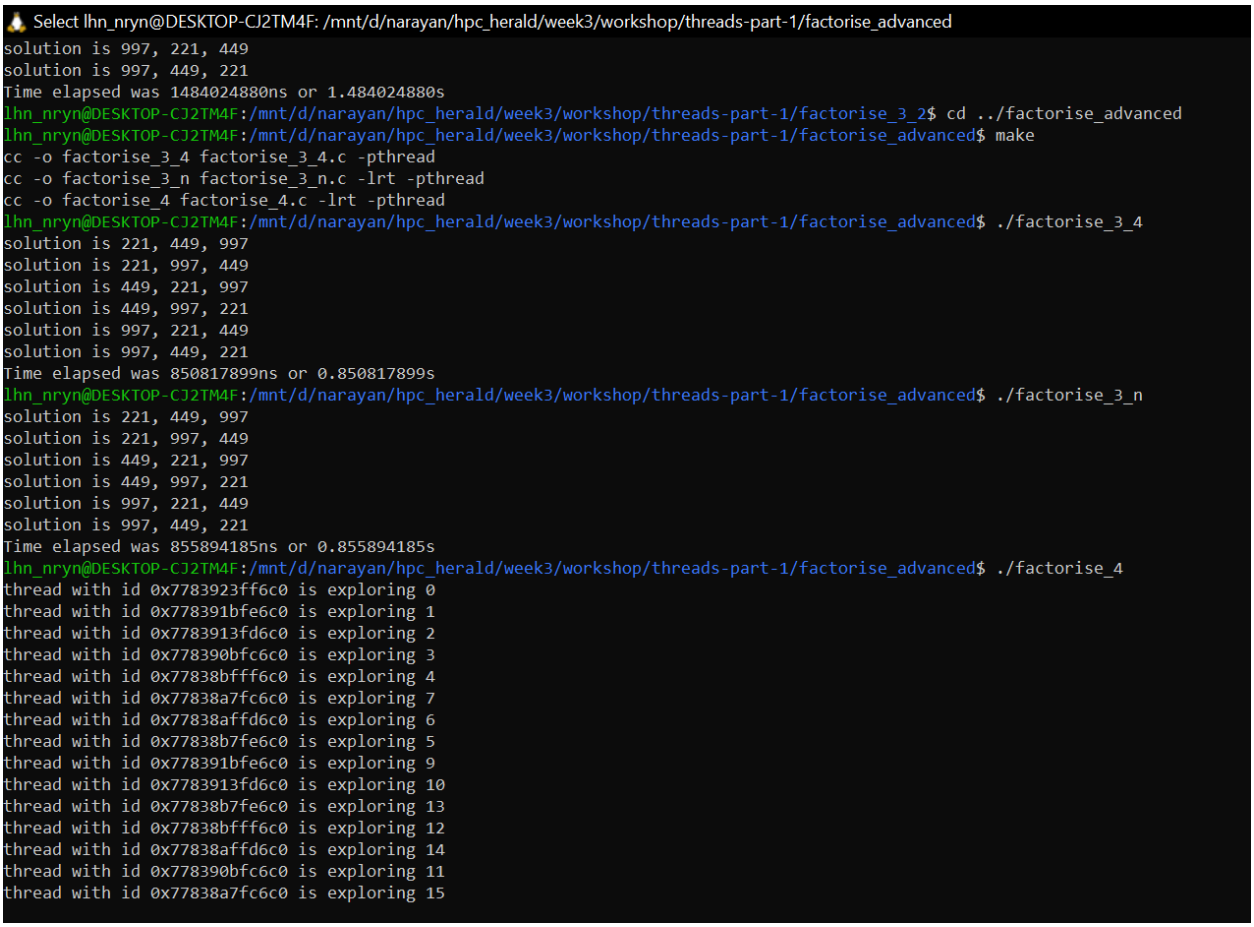
Factorise_3_1



Factorise_3_2

```
lhn_nryn@DESKTOP-CJ2TM4F: /mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/factorise_3_2
Time elapsed was 2968842287ns or 2.968842287s
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/factorise_3_1$ cd ../factorise_3_2
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/factorise_3_2$ make
cc -o factorise_3_2_a factorise_3_2_a.c -pthread
cc -o factorise_3_2_b factorise_3_2_b.c -lrt -pthread
cc -o factorise_3_2_c factorise_3_2_c.c -lrt -pthread
cc -o factorise_3_2_d factorise_3_2_d.c -lrt -pthread
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/factorise_3_2$ ./factorise_3_2_a
solution is 221, 449, 997
solution is 221, 997, 449
solution is 449, 221, 997
solution is 449, 997, 221
solution is 997, 221, 449
solution is 997, 449, 221
Time elapsed was 1533079237ns or 1.533079237s
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/factorise_3_2$ ./factorise_3_2_b
solution is 221, 449, 997
solution is 221, 997, 449
solution is 449, 221, 997
solution is 449, 997, 221
solution is 997, 221, 449
solution is 997, 449, 221
Time elapsed was 1512837600ns or 1.512837600s
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/factorise_3_2$ ./factorise_3_2_c
solution is 221, 449, 997
solution is 221, 997, 449
solution is 449, 221, 997
solution is 449, 997, 221
solution is 997, 221, 449
solution is 997, 449, 221
Time elapsed was 1498760773ns or 1.498760773s
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/factorise_3_2$ ./factorise_3_2_d
solution is 221, 449, 997
solution is 221, 997, 449
solution is 449, 221, 997
solution is 449, 997, 221
solution is 997, 221, 449
solution is 997, 449, 221
Time elapsed was 1484024880ns or 1.484024880s
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/factorise_3_2$
```

Factorise_advanced



Multiply

```
lhn_nryn@DESKTOP-CJ2TM4F: /mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/multiply
thread with id 0x778391bfe6c0 is exploring 617
^C
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/factorise_advanced$ cd ../multiply
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/multiply$ make
cc -o multiply0 multiply0.c -pthread
cc -o multiply1 multiply1.c -pthread
cc -o multiply2 multiply2.c -pthread
cc -o multiply3 multiply3.c -pthread
cc -o multiply4 multiply4.c -pthread
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/multiply$ ./multiply0
thread id is 29908
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/multiply$ ./multiply1
thread id is 29910
thread id is 29911
thread id is 29912
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/multiply$ ./multiply2
thread 29914 received 7 and 5
thread 29914 calculated 35
7 * 5 = 35
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/multiply$ ./multiply3
thread 29916 received 7 and 5
thread 29916 calculated 35
thread 29917 received 6 and 4
thread 29917 calculated 24
thread 29918 received 2 and 9
thread 29918 calculated 18
7 * 5 = 35
6 * 4 = 24
2 * 9 = 18
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/multiply$ ./multiply4
thread 29920 received 77777 and 5
thread 29921 received 666 and 4
thread 29921 calculated 2664
thread 29920 calculated 388885
thread 29922 received 99999 and 9
thread 29922 calculated 899991
77777 * 5 = 388885
666 * 4 = 2664
99999 * 9 = 899991
lhn_nryn@DESKTOP-CJ2TM4F:/mnt/d/narayan/hpc_herald/week3/workshop/threads-part-1/multiply$
```