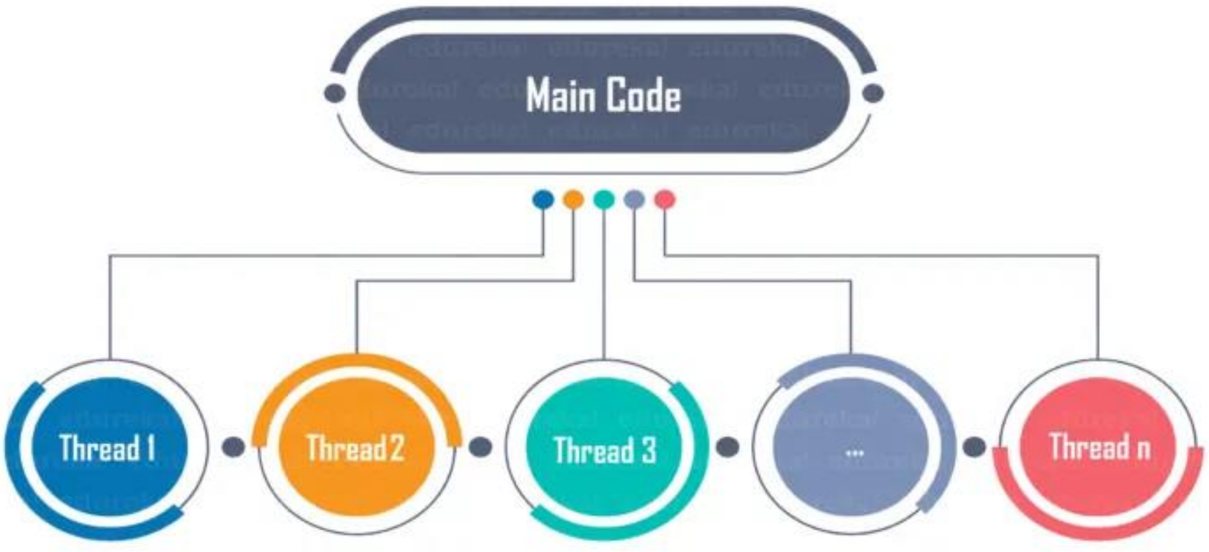


PROJECT 01 MULTITHREADED PROGRAMMING AND SYNCHRONIZATION

CECS 326 - OPERATING SYSTEMS

PROFESSOR: HAILU XU

DATE: FEBRUARY 21, 2021

STUDENTS: FIONA LE , THANH NGUYEN

Step 1: Simple Multi-Thread Programming without Synchronization

- **Code:**

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

int SharedVariable = 0; // is shared by threads

void SimpleThread(int which)
/*
    This is the provided function. The purpose of this function
is seeing which thread is called at which time.

    Input: The ID of the thread
*/
{
    int num, val;

    for (num = 0; num < 20; num++)
    {
        if (random() > RAND_MAX / 2)
            usleep(500);

        val = SharedVariable;
        printf("*** thread %d sees value %d\n", which, val);
        SharedVariable = val + 1;
    }
    val = SharedVariable;
    printf("Thread %d sees final value %d\n", which, val);
}

void* singleThread(void* threadID)
```

```
/*  This is the sub function that is executed by the thread. The
return type and parameter type of the sub function must be of
type void* because the pointer to this sub function should be
passed to the function thread_create()
*/
{
    SimpleThread((intptr_t)(threadID));

    // Terminate a thread
    pthread_exit(NULL);
}

int main(int argc, char* argv[])
{
    // Check if the input is correct: output_file
number_of_threads
    if (argc != 2)
    {
        printf("ERROR: usage: output <number of threads>\n");
        return EXIT_FAILURE;
    }

    // The input is correct: argc == 2
    // Convert the number_of_thread from string to number
    int numberOfThreads = atoi(argv[1]);

    // Check if the number of thread is positive (greater than
0)
    if (numberOfThreads <= 0)
    {
        printf("ERROR: usage: output <number of threads>\n");
        return EXIT_FAILURE;
    }

    // Create threads
    int rc;
    pthread_t threads[numberOfThreads];
```

```
 // Use pthread_create to create thread and pass a pointer to a
sub function called singleThread. Check if it is fait to create
the thread, then return ERROR.
    for (long i = 0; i < numberOfThreads; i++)
        if (rc = pthread_create(&threads[i], NULL, singleThread,
(void*) i))
        {
            printf("ERROR: Failed to create Thread %ld\n
Error number = %d\n", i, rc);
            return EXIT_FAILURE;
        }

 // Wait for the thread to exit
    for (long i = 0; i < numberOfThreads; i++)
        if (rc = pthread_join(threads[i], NULL))
        {
            printf("ERROR: Failed to join Thread %ld\n
Error number = %d\n", i, rc);
            return EXIT_FAILURE;
        }

  // Exit threads
    pthread_exit(NULL);

    return EXIT_SUCCESS;
}
```

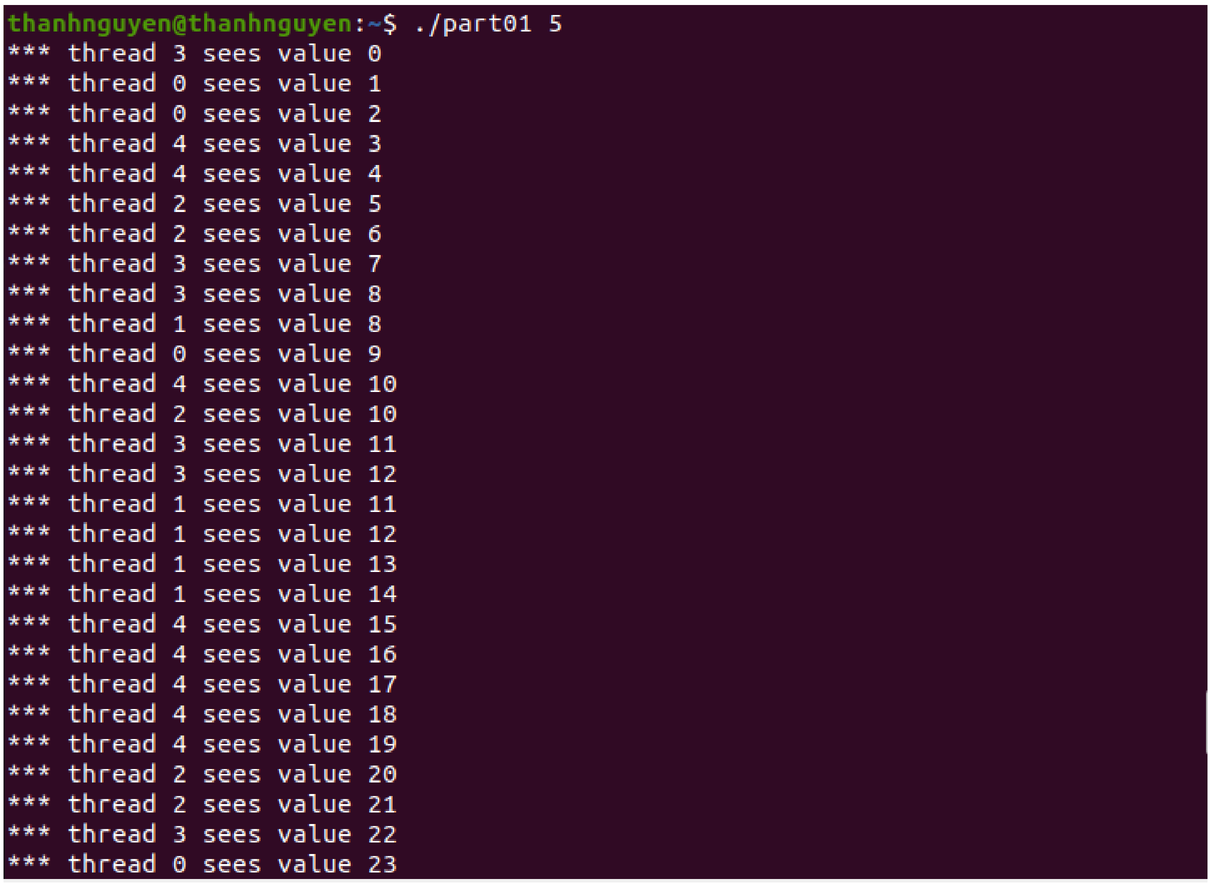
❖ **Code Description:**

- **SimpleThread function:** the provided function
- **single-thread function**: we wanted to maintain the provided function above, then we create another function. This function will call the SimpleThread function by using the command: ***SimpleThread((intptr_t)(threadID))***. It uses **void*** for the return and parameter variables to pass to the function ***thread_create()***.
- **Main function:**

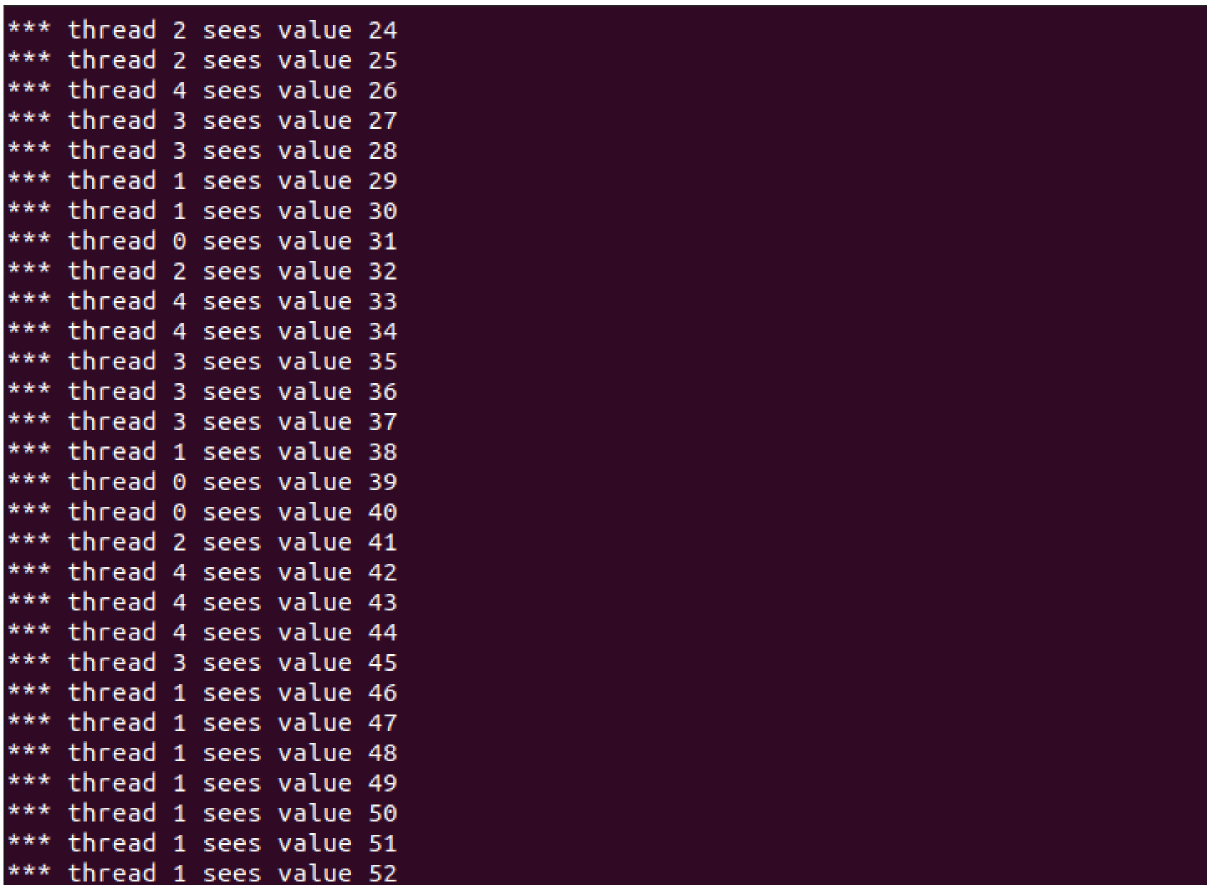
- At first, we check the input from the command line: check if the syntax is correct, the number of threads is available and greater than 0 or not.
- If the first step passes, we create the array of threads, which contains the number of threads from the input using ***pthread_create(&threads[i], NULL, singleThread, (void*) i)***.
- We also use ***pthread_join(threads[i], NULL)*** to wait for the thread to exit.

❖ **Result:**

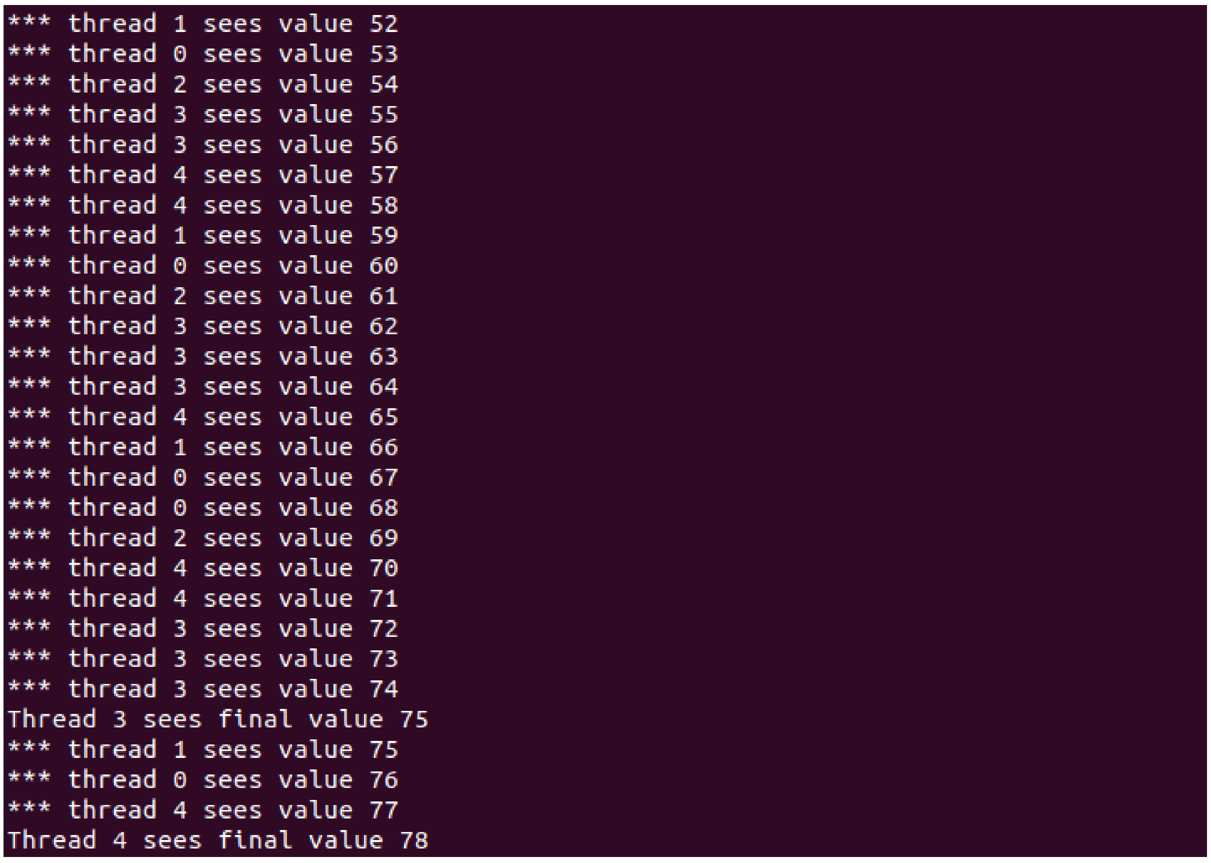
- **Create 5 threads:**



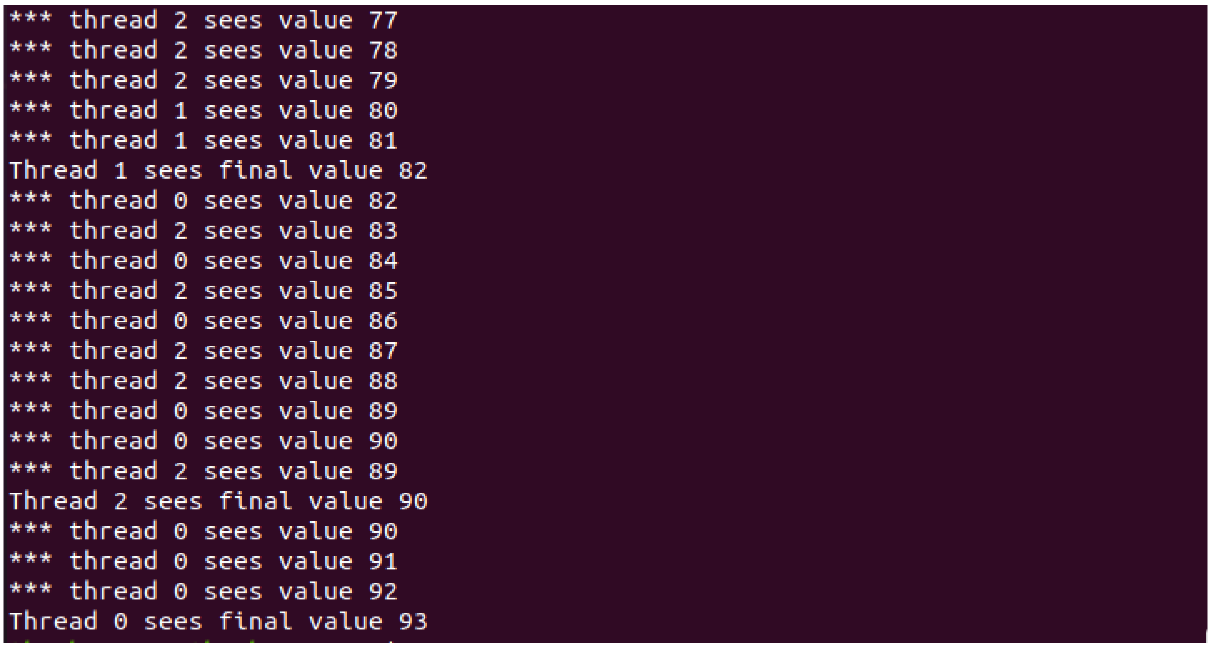
```
thanhnguyen@thanhnguyen:~$ ./part01 5
*** thread 3 sees value 0
*** thread 0 sees value 1
*** thread 0 sees value 2
*** thread 4 sees value 3
*** thread 4 sees value 4
*** thread 2 sees value 5
*** thread 2 sees value 6
*** thread 3 sees value 7
*** thread 3 sees value 8
*** thread 1 sees value 8
*** thread 0 sees value 9
*** thread 4 sees value 10
*** thread 2 sees value 10
*** thread 3 sees value 11
*** thread 3 sees value 12
*** thread 1 sees value 11
*** thread 1 sees value 12
*** thread 1 sees value 13
*** thread 1 sees value 14
*** thread 4 sees value 15
*** thread 4 sees value 16
*** thread 4 sees value 17
*** thread 4 sees value 18
*** thread 4 sees value 19
*** thread 2 sees value 20
*** thread 2 sees value 21
*** thread 3 sees value 22
*** thread 0 sees value 23
```



```
*** thread 2 sees value 24
*** thread 2 sees value 25
*** thread 4 sees value 26
*** thread 3 sees value 27
*** thread 3 sees value 28
*** thread 1 sees value 29
*** thread 1 sees value 30
*** thread 0 sees value 31
*** thread 2 sees value 32
*** thread 4 sees value 33
*** thread 4 sees value 34
*** thread 3 sees value 35
*** thread 3 sees value 36
*** thread 3 sees value 37
*** thread 1 sees value 38
*** thread 0 sees value 39
*** thread 0 sees value 40
*** thread 2 sees value 41
*** thread 4 sees value 42
*** thread 4 sees value 43
*** thread 4 sees value 44
*** thread 3 sees value 45
*** thread 1 sees value 46
*** thread 1 sees value 47
*** thread 1 sees value 48
*** thread 1 sees value 49
*** thread 1 sees value 50
*** thread 1 sees value 51
*** thread 1 sees value 52
```

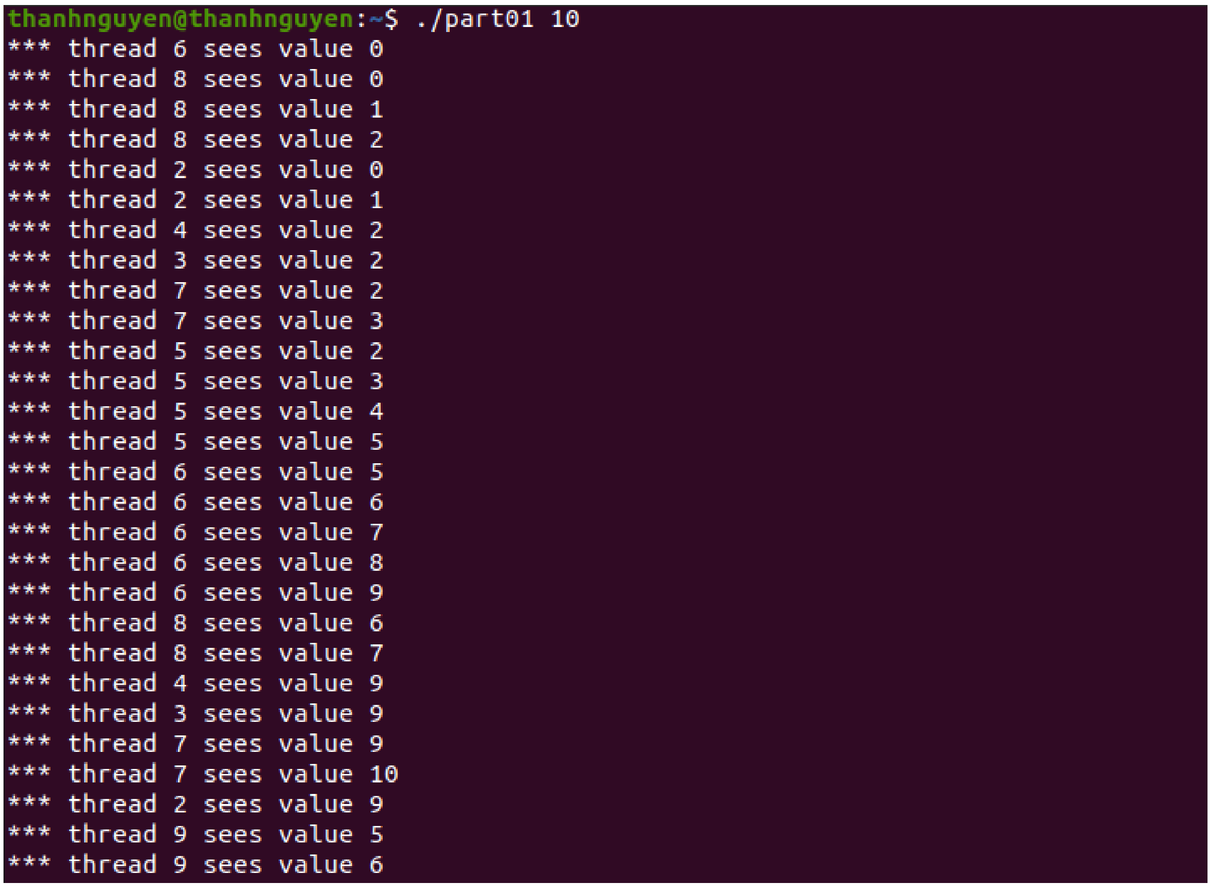


```
*** thread 1 sees value 52
*** thread 0 sees value 53
*** thread 2 sees value 54
*** thread 3 sees value 55
*** thread 3 sees value 56
*** thread 4 sees value 57
*** thread 4 sees value 58
*** thread 1 sees value 59
*** thread 0 sees value 60
*** thread 2 sees value 61
*** thread 3 sees value 62
*** thread 3 sees value 63
*** thread 3 sees value 64
*** thread 4 sees value 65
*** thread 1 sees value 66
*** thread 0 sees value 67
*** thread 0 sees value 68
*** thread 2 sees value 69
*** thread 4 sees value 70
*** thread 4 sees value 71
*** thread 3 sees value 72
*** thread 3 sees value 73
*** thread 3 sees value 74
Thread 3 sees final value 75
*** thread 1 sees value 75
*** thread 0 sees value 76
*** thread 4 sees value 77
Thread 4 sees final value 78
```



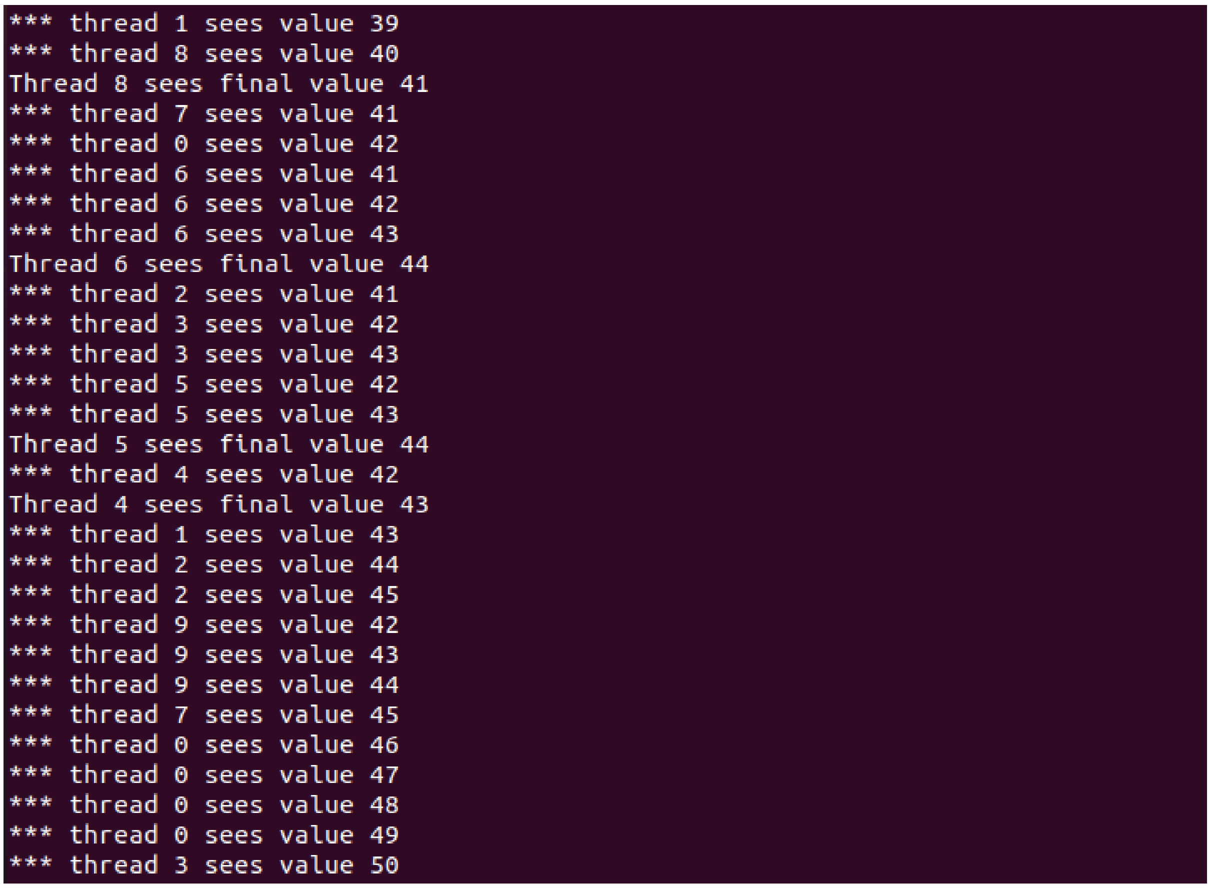
```
*** thread 2 sees value 77
*** thread 2 sees value 78
*** thread 2 sees value 79
*** thread 1 sees value 80
*** thread 1 sees value 81
Thread 1 sees final value 82
*** thread 0 sees value 82
*** thread 2 sees value 83
*** thread 0 sees value 84
*** thread 2 sees value 85
*** thread 0 sees value 86
*** thread 2 sees value 87
*** thread 2 sees value 88
*** thread 0 sees value 89
*** thread 0 sees value 90
*** thread 2 sees value 89
Thread 2 sees final value 90
*** thread 0 sees value 90
*** thread 0 sees value 91
*** thread 0 sees value 92
Thread 0 sees final value 93
```

- **Create 10 threads:**



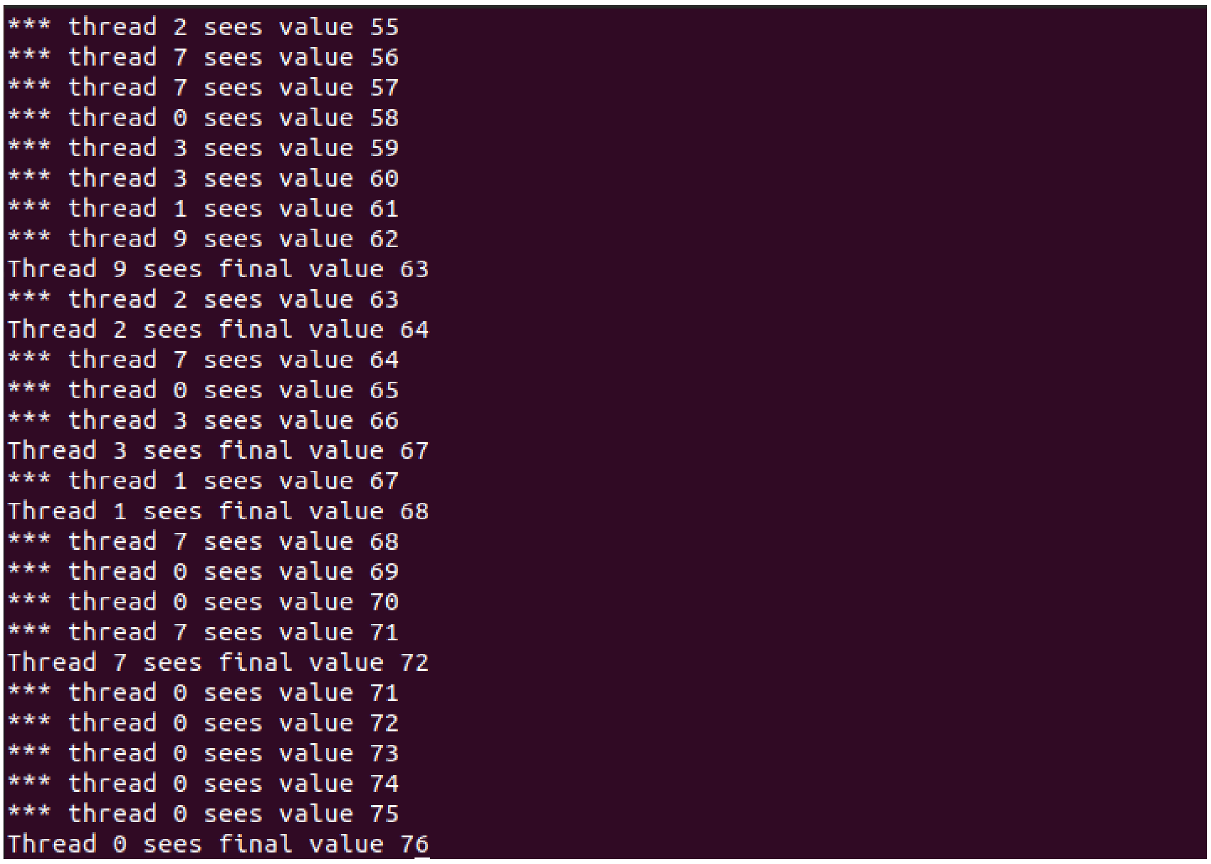
```
thanhnguyen@thanhnguyen:~$ ./part01 10
*** thread 6 sees value 0
*** thread 8 sees value 0
*** thread 8 sees value 1
*** thread 8 sees value 2
*** thread 2 sees value 0
*** thread 2 sees value 1
*** thread 4 sees value 2
*** thread 3 sees value 2
*** thread 7 sees value 2
*** thread 7 sees value 3
*** thread 5 sees value 2
*** thread 5 sees value 3
*** thread 5 sees value 4
*** thread 5 sees value 5
*** thread 6 sees value 5
*** thread 6 sees value 6
*** thread 6 sees value 7
*** thread 6 sees value 8
*** thread 6 sees value 9
*** thread 8 sees value 6
*** thread 8 sees value 7
*** thread 4 sees value 9
*** thread 3 sees value 9
*** thread 7 sees value 9
*** thread 7 sees value 10
*** thread 2 sees value 9
*** thread 9 sees value 5
*** thread 9 sees value 6
```

(few more lines)



```
*** thread 1 sees value 39
*** thread 8 sees value 40
Thread 8 sees final value 41
*** thread 7 sees value 41
*** thread 0 sees value 42
*** thread 6 sees value 41
*** thread 6 sees value 42
*** thread 6 sees value 43
Thread 6 sees final value 44
*** thread 2 sees value 41
*** thread 3 sees value 42
*** thread 3 sees value 43
*** thread 5 sees value 42
*** thread 5 sees value 43
Thread 5 sees final value 44
*** thread 4 sees value 42
Thread 4 sees final value 43
*** thread 1 sees value 43
*** thread 2 sees value 44
*** thread 2 sees value 45
*** thread 9 sees value 42
*** thread 9 sees value 43
*** thread 9 sees value 44
*** thread 7 sees value 45
*** thread 0 sees value 46
*** thread 0 sees value 47
*** thread 0 sees value 48
*** thread 0 sees value 49
*** thread 3 sees value 50
```

(few more lines)



```
*** thread 2 sees value 55
*** thread 7 sees value 56
*** thread 7 sees value 57
*** thread 0 sees value 58
*** thread 3 sees value 59
*** thread 3 sees value 60
*** thread 1 sees value 61
*** thread 9 sees value 62
Thread 9 sees final value 63
*** thread 2 sees value 63
Thread 2 sees final value 64
*** thread 7 sees value 64
*** thread 0 sees value 65
*** thread 3 sees value 66
Thread 3 sees final value 67
*** thread 1 sees value 67
Thread 1 sees final value 68
*** thread 7 sees value 68
*** thread 0 sees value 69
*** thread 0 sees value 70
*** thread 7 sees value 71
Thread 7 sees final value 72
*** thread 0 sees value 71
*** thread 0 sees value 72
*** thread 0 sees value 73
*** thread 0 sees value 74
*** thread 0 sees value 75
Thread 0 sees final value 76
```

Step 2: Simple Multi-Thread Programming with Synchronization

- **Code:**

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

// MACRO (These functions will be called only if Macro is defined)
#ifdef PTHREAD_SYNC
    pthread_mutex_t mutex;
    pthread_barrier_t barrier;
#endif

int SharedVariable = 0; // is shared by threads

void SimpleThread(int which)
/*
    This is the provided function. The purpose of this function is
        seeing which thread is called at which time.

    Input: The ID of the thread
*/
{
    int num, val;

    #ifdef PTHREAD_SYNC
        // Lock a mutex object
        pthread_mutex_lock(&mutex);
    #endif
    for (num = 0; num < 20; num++)
    {
        if (random() > RAND_MAX / 2)
            usleep(500);
        val = SharedVariable;
        printf("*** thread %d sees value %d\n", which, val);
        SharedVariable = val + 1;
    }
    #ifdef PTHREAD_SYNC
        // Release a mutex object
        pthread_mutex_unlock(&mutex);
        // Allow all threads to wait for the last thread to exit
the loop.
        pthread_barrier_wait(&barrier);
    #endif
    val = SharedVariable;
    printf("Thread %d sees final value %d\n", which, val);
}

void* singleThread(void* threadID)
/*  This is the sub-function that is executed by the thread. The
return type and parameter type of the sub-function must be of type
void* because the pointer to this sub-function should be
passed to the function thread_create()
*/
{
    SimpleThread((intptr_t)(threadID));

    // Terminate the thread
    pthread_exit(NULL);
}

int main(int argc, char* argv[])
{
    // Check if the input is correct: output_file number_of_threads
    if (argc != 2)
    {
        printf("ERROR: usage: output <number of threads>\n");
        return EXIT_FAILURE;
    }

    // The input is correct: argc == 2
    // Convert the number_of_thread from string to number
    int numberOfThreads = atoi(argv[1]);

    // Check if the number of thread is positive (greater than
0)
    if (numberOfThreads <= 0)
    {
        printf("ERROR: usage: output <number of threads>\n");
        return EXIT_FAILURE;
    }

    pthread_t threads[numberOfThreads];

    #ifdef PTHREAD_SYNC
        // Initialize the mutex
        pthread_mutex_init(&mutex, NULL);
        // Initialize the barrier
        pthread_barrier_init(&barrier, NULL, numberOfThreads);
    #endif

    // Create threads
    for (long i = 0; i < numberOfThreads; i++)
        pthread_create(&threads[i], NULL, singleThread, (void*) i);

    // Wait for the thread to exit
    for (long i = 0; i < numberOfThreads; i++)
        pthread_join(threads[i], NULL);

    // Exit threads
    pthread_exit(NULL);

    return EXIT_SUCCESS;
}
```

❖ **Describe code:**

The basic code is similar to part 01. We just added the definition of PTHREAD_SYNC macro, pthread mutex variable, and barrier to synchronize the output of the program (each thread will see 20 values before another thread accesses the counter, and all the threads get the same final value, for example, 100 for 4 threads because each thread sees its final value every 20 values)

- **PTHREAD_SYNC macro:** The code inside the macro will be called only if Macro is defined

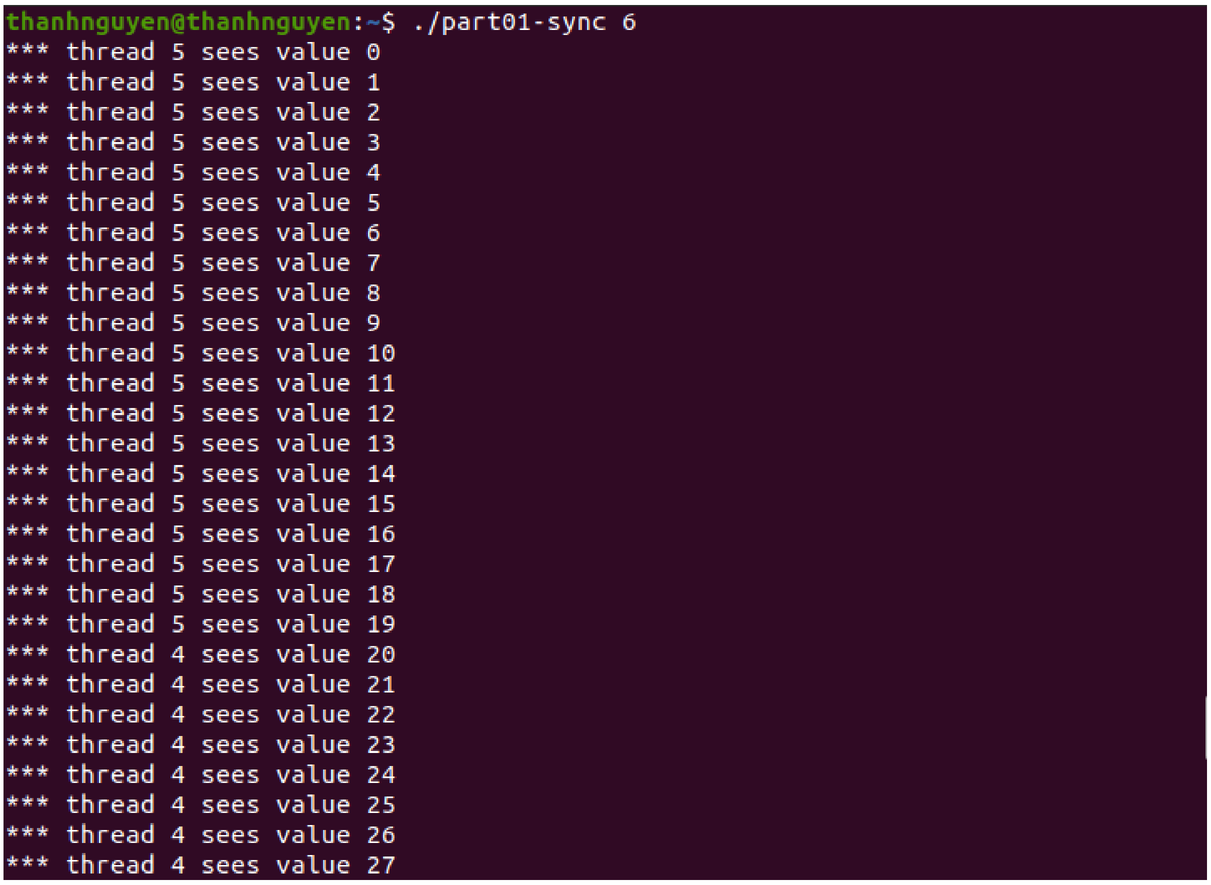
For executing the macro, we use the commands: *gcc -Wall -D PTHREAD_SYNC <code_file>* to run the code.

- **Mutex**: A mutex is a lock that we set before using a shared resource and release after using it. When the lock is set, no other thread can access the locked region of code.
    - A mutex is initialized in the body of the main function inside the PTHREAD_SYNC macro using *pthread_mutex_int(pthread_mutex_t &mutex, NULL).*
    - The mutex is locked in the "SimpleThread" function while using the shared resource called SharedVariable using *pthread_mutex_lock(pthread_mutex_t*)* before the for loop, then the other threads cannot access the SharedVariable until the current thread finishes its task (sees 20 values).
    - After the for loop in the "SimpleThread" function, unlock the same mutex using *pthread_mutex_unlock(pthread_mutex_t*)* so that another thread can access the SharedVariable.
- **Barrier**: A barrier holds one or multiple threads until all threads participating in the barrier have reached the barrier point.
    - The *pthread_barrier_init(&barrier, NULL, unsigned count)* function allocates any resources required to use the barrier referenced by the barrier and initializes the barrier with the defaults attributes. The count specifies the number of threads that must call pthread_barrier_wait().

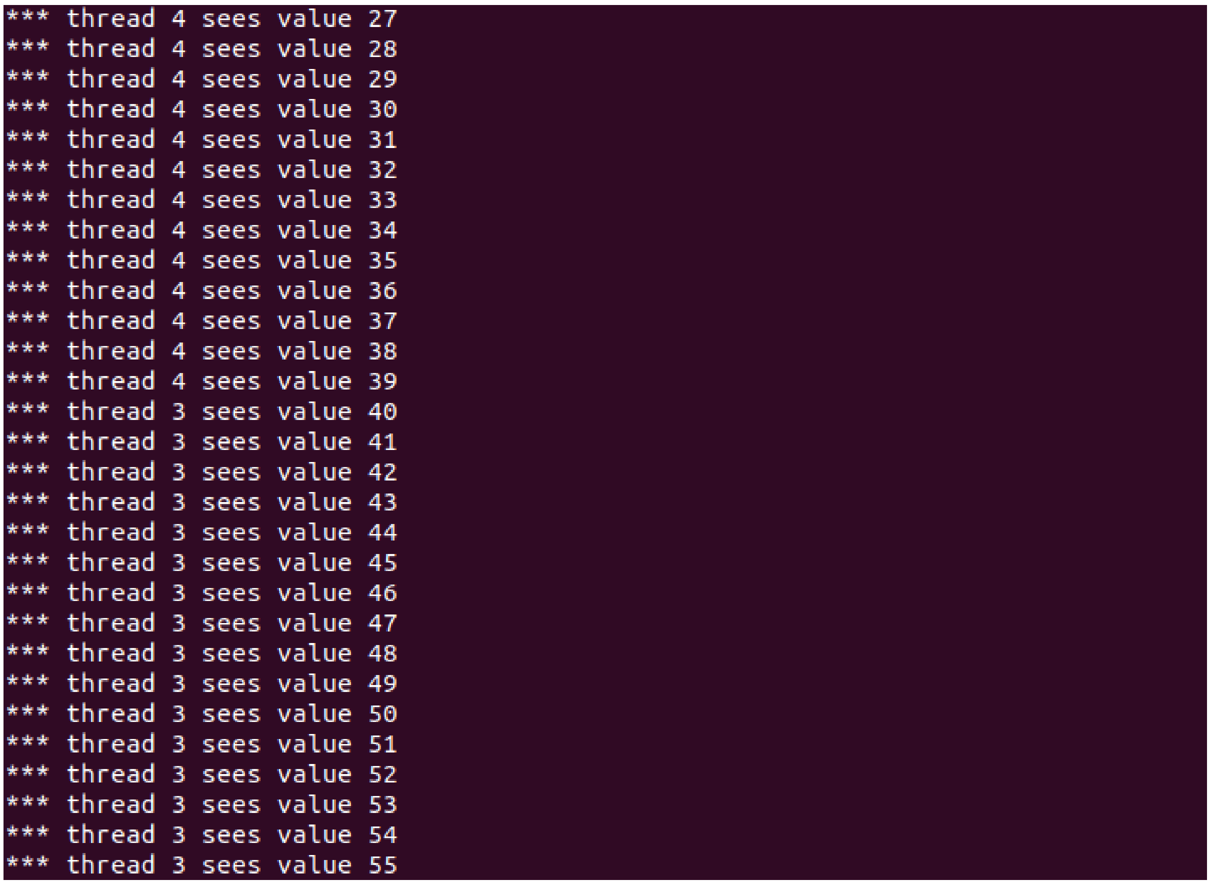
    - Using *pthread_barrier_wait()* before checking the final value in the "SimpleThread" function ensures that no thread will proceed until all threads have reached the barrier. So, all the threads can get the same final results at the end.

❖ **Result:**

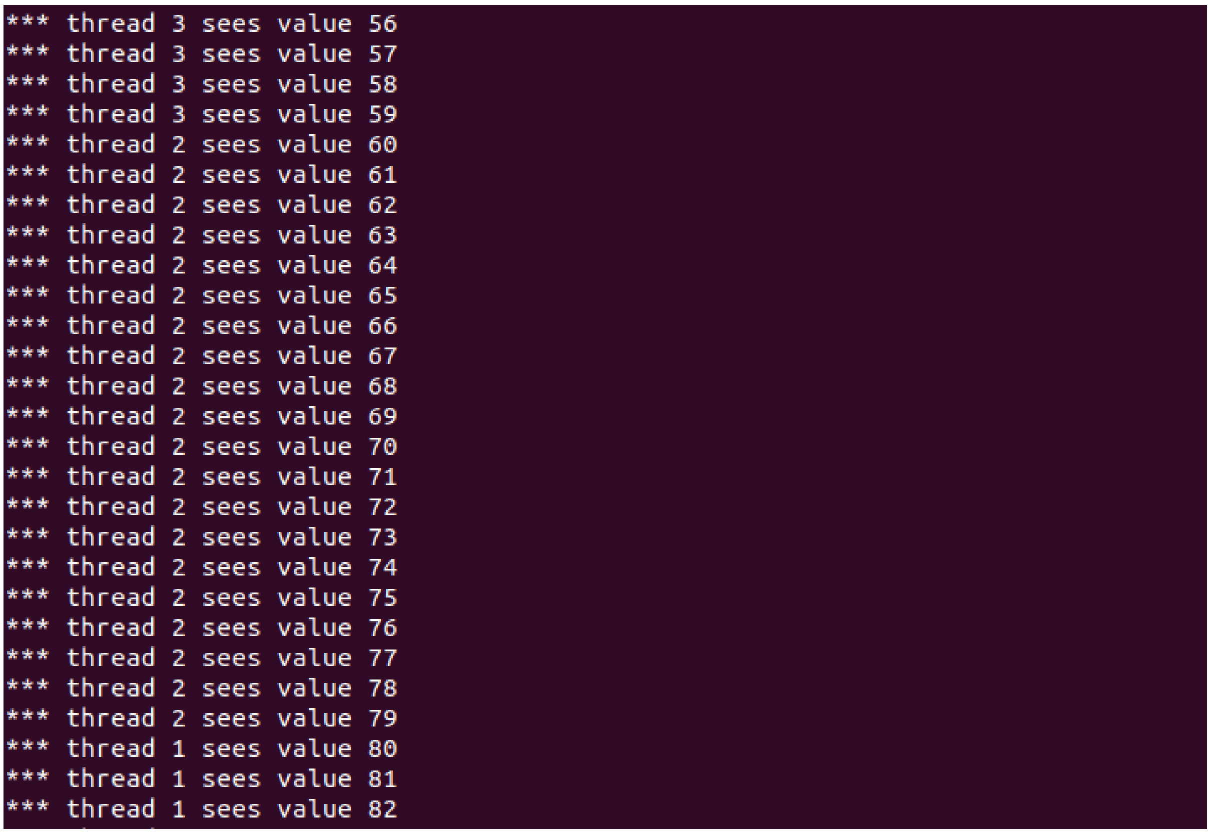
- **Create 6 threads:**



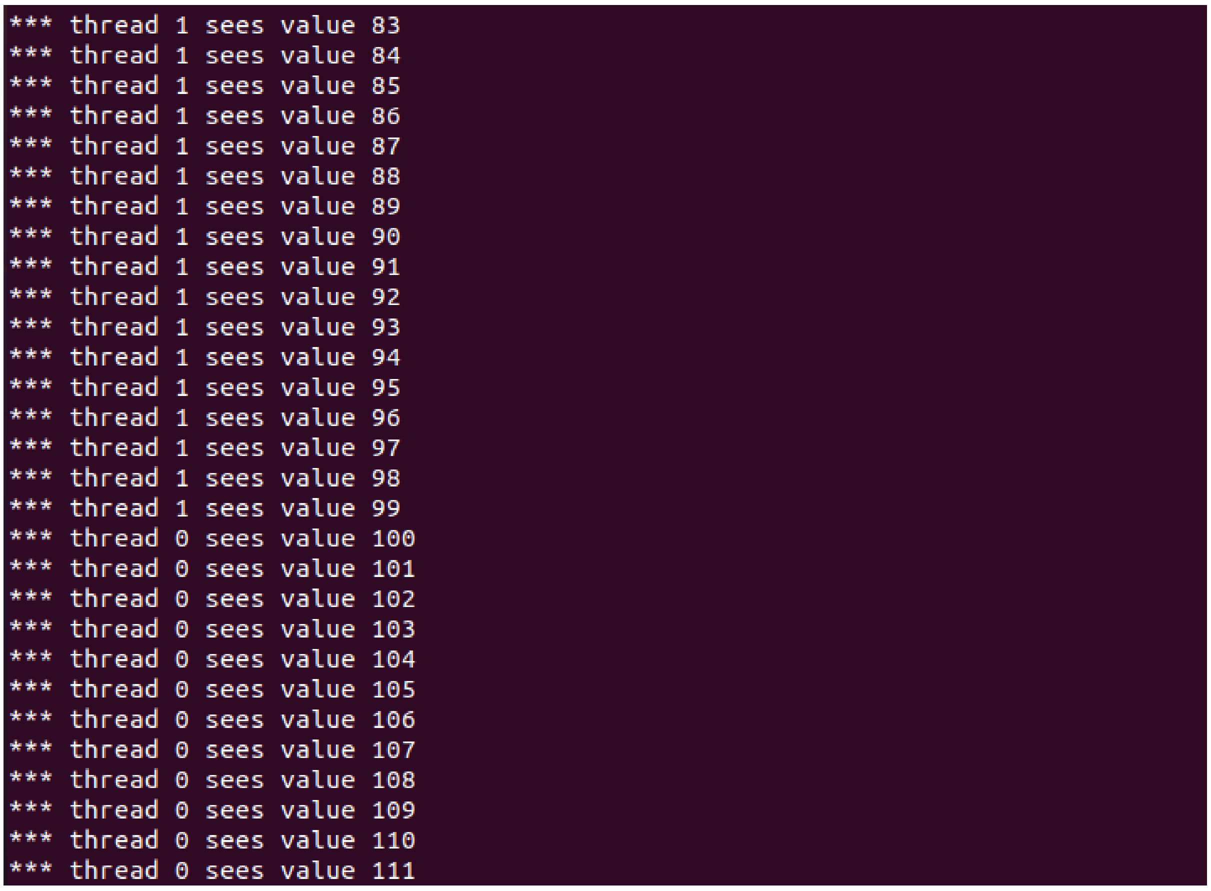
```
thanhnguyen@thanhnguyen:~$ ./part01-sync 6
*** thread 5 sees value 0
*** thread 5 sees value 1
*** thread 5 sees value 2
*** thread 5 sees value 3
*** thread 5 sees value 4
*** thread 5 sees value 5
*** thread 5 sees value 6
*** thread 5 sees value 7
*** thread 5 sees value 8
*** thread 5 sees value 9
*** thread 5 sees value 10
*** thread 5 sees value 11
*** thread 5 sees value 12
*** thread 5 sees value 13
*** thread 5 sees value 14
*** thread 5 sees value 15
*** thread 5 sees value 16
*** thread 5 sees value 17
*** thread 5 sees value 18
*** thread 5 sees value 19
*** thread 4 sees value 20
*** thread 4 sees value 21
*** thread 4 sees value 22
*** thread 4 sees value 23
*** thread 4 sees value 24
*** thread 4 sees value 25
*** thread 4 sees value 26
*** thread 4 sees value 27
```



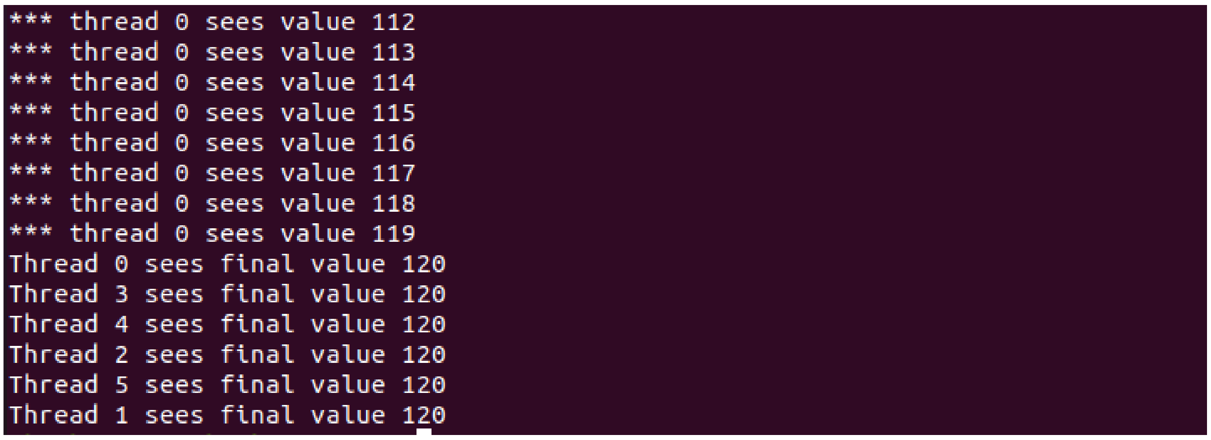
```
*** thread 4 sees value 27
*** thread 4 sees value 28
*** thread 4 sees value 29
*** thread 4 sees value 30
*** thread 4 sees value 31
*** thread 4 sees value 32
*** thread 4 sees value 33
*** thread 4 sees value 34
*** thread 4 sees value 35
*** thread 4 sees value 36
*** thread 4 sees value 37
*** thread 4 sees value 38
*** thread 4 sees value 39
*** thread 3 sees value 40
*** thread 3 sees value 41
*** thread 3 sees value 42
*** thread 3 sees value 43
*** thread 3 sees value 44
*** thread 3 sees value 45
*** thread 3 sees value 46
*** thread 3 sees value 47
*** thread 3 sees value 48
*** thread 3 sees value 49
*** thread 3 sees value 50
*** thread 3 sees value 51
*** thread 3 sees value 52
*** thread 3 sees value 53
*** thread 3 sees value 54
*** thread 3 sees value 55
```



```
*** thread 3 sees value 56
*** thread 3 sees value 57
*** thread 3 sees value 58
*** thread 3 sees value 59
*** thread 2 sees value 60
*** thread 2 sees value 61
*** thread 2 sees value 62
*** thread 2 sees value 63
*** thread 2 sees value 64
*** thread 2 sees value 65
*** thread 2 sees value 66
*** thread 2 sees value 67
*** thread 2 sees value 68
*** thread 2 sees value 69
*** thread 2 sees value 70
*** thread 2 sees value 71
*** thread 2 sees value 72
*** thread 2 sees value 73
*** thread 2 sees value 74
*** thread 2 sees value 75
*** thread 2 sees value 76
*** thread 2 sees value 77
*** thread 2 sees value 78
*** thread 2 sees value 79
*** thread 1 sees value 80
*** thread 1 sees value 81
*** thread 1 sees value 82
```

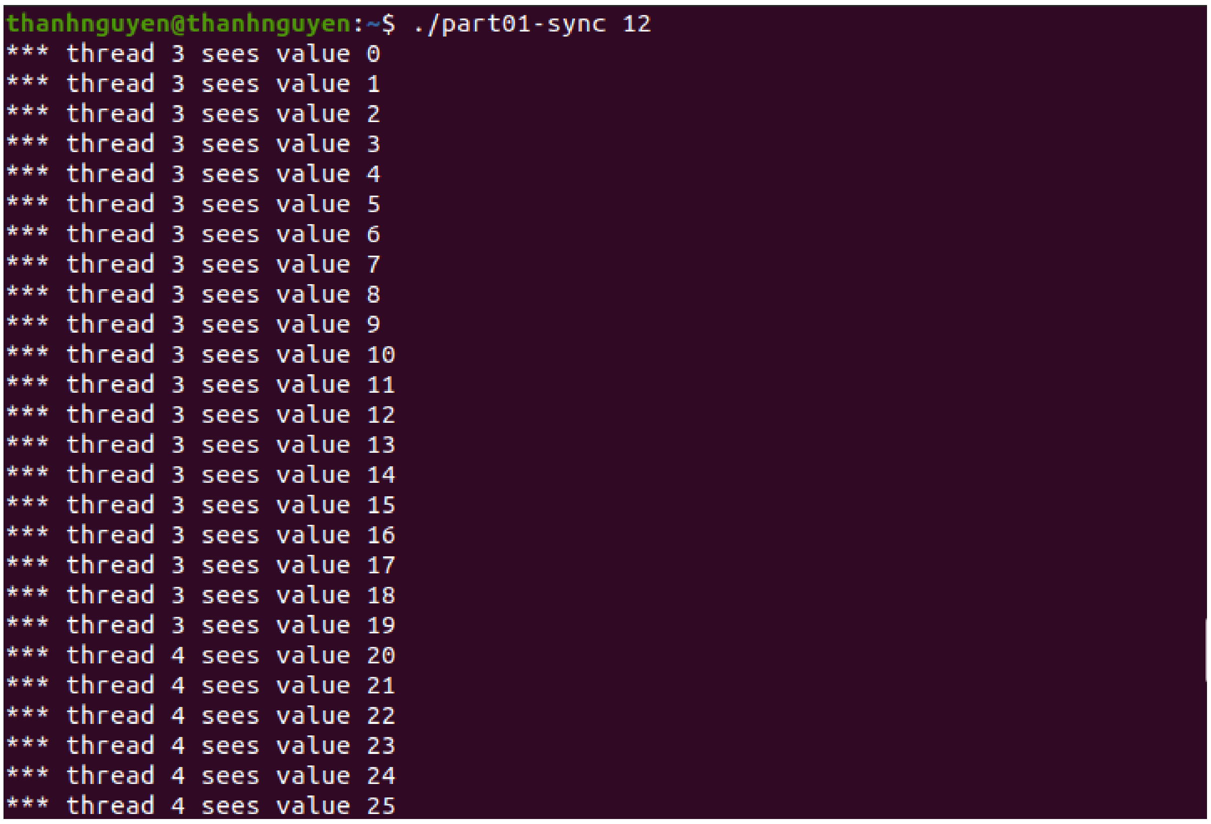


```
*** thread 1 sees value 83
*** thread 1 sees value 84
*** thread 1 sees value 85
*** thread 1 sees value 86
*** thread 1 sees value 87
*** thread 1 sees value 88
*** thread 1 sees value 89
*** thread 1 sees value 90
*** thread 1 sees value 91
*** thread 1 sees value 92
*** thread 1 sees value 93
*** thread 1 sees value 94
*** thread 1 sees value 95
*** thread 1 sees value 96
*** thread 1 sees value 97
*** thread 1 sees value 98
*** thread 1 sees value 99
*** thread 0 sees value 100
*** thread 0 sees value 101
*** thread 0 sees value 102
*** thread 0 sees value 103
*** thread 0 sees value 104
*** thread 0 sees value 105
*** thread 0 sees value 106
*** thread 0 sees value 107
*** thread 0 sees value 108
*** thread 0 sees value 109
*** thread 0 sees value 110
*** thread 0 sees value 111
```



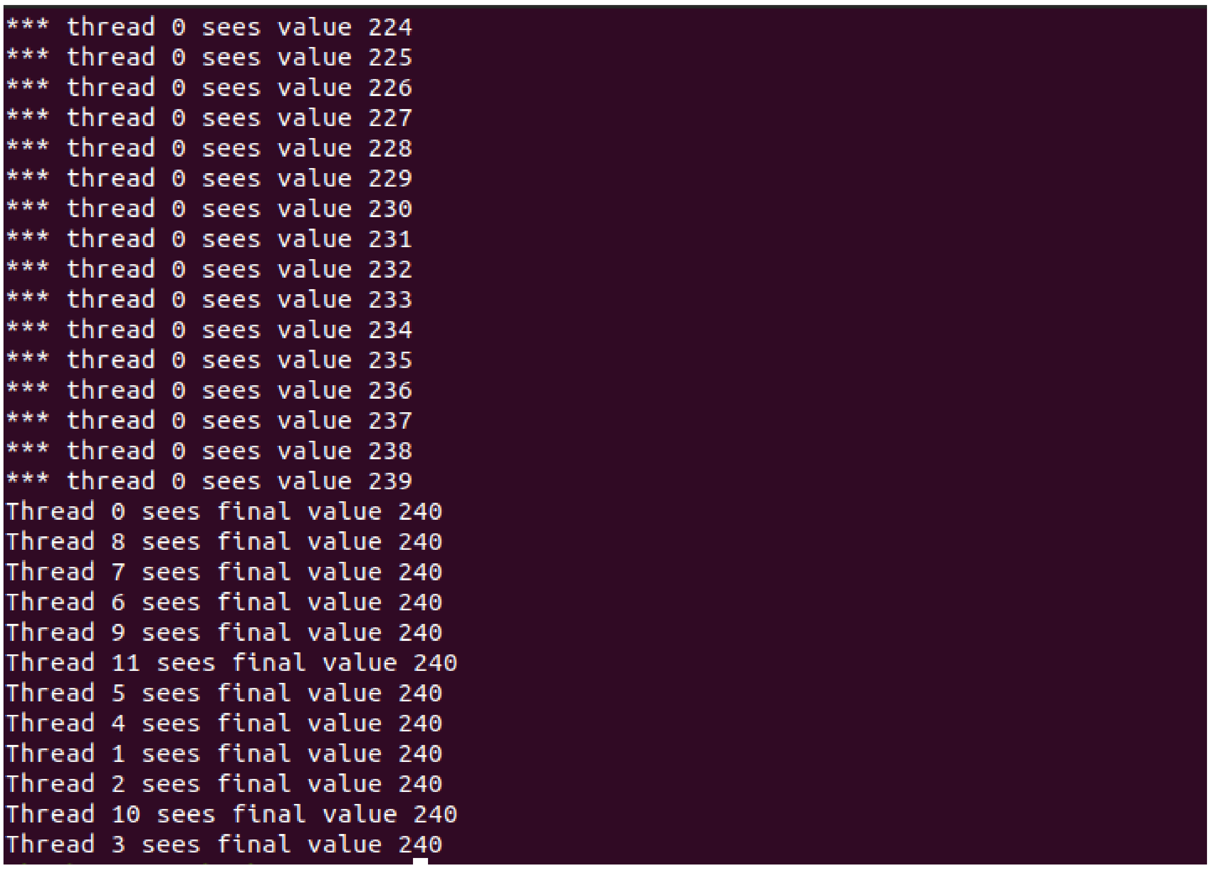
```
*** thread 0 sees value 112
*** thread 0 sees value 113
*** thread 0 sees value 114
*** thread 0 sees value 115
*** thread 0 sees value 116
*** thread 0 sees value 117
*** thread 0 sees value 118
*** thread 0 sees value 119
Thread 0 sees final value 120
Thread 3 sees final value 120
Thread 4 sees final value 120
Thread 2 sees final value 120
Thread 5 sees final value 120
Thread 1 sees final value 120
```

- **Create 12 threads:**



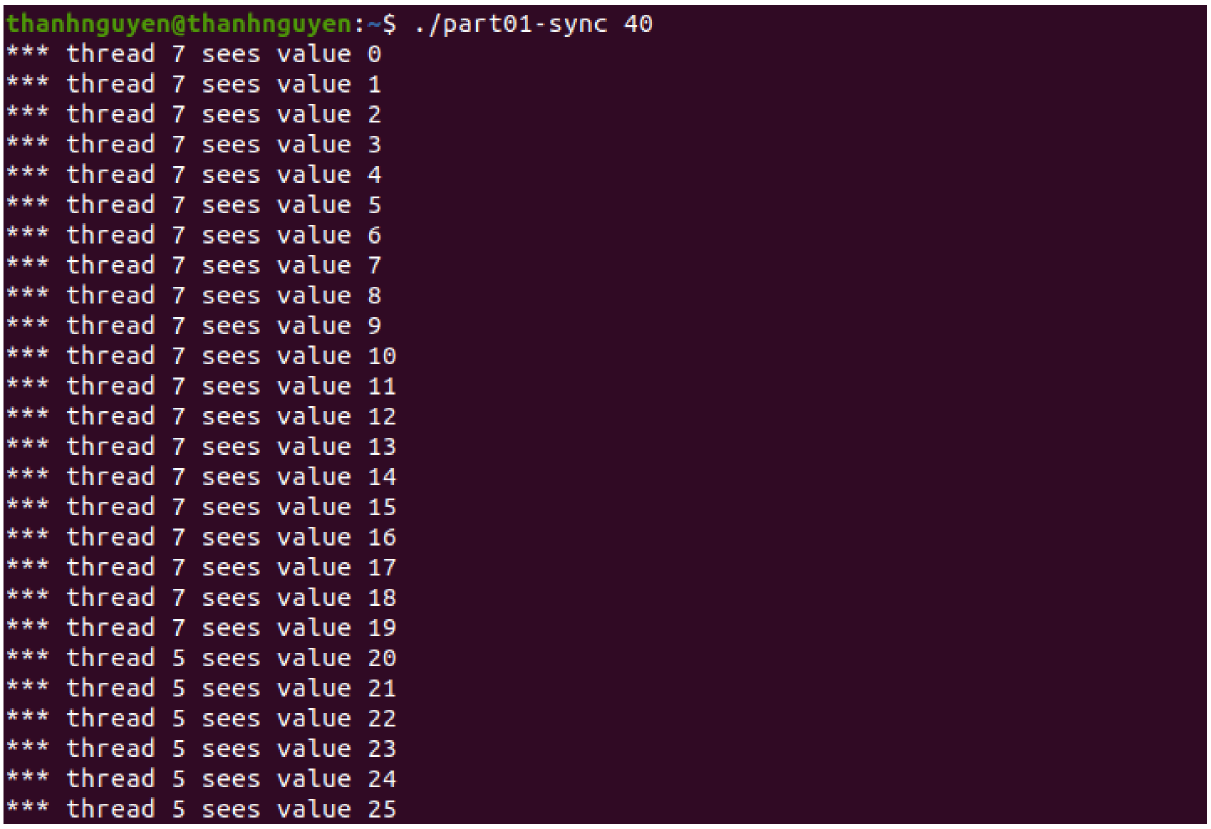
```
thanhnguyen@thanhnguyen:~$ ./part01-sync 12
*** thread 3 sees value 0
*** thread 3 sees value 1
*** thread 3 sees value 2
*** thread 3 sees value 3
*** thread 3 sees value 4
*** thread 3 sees value 5
*** thread 3 sees value 6
*** thread 3 sees value 7
*** thread 3 sees value 8
*** thread 3 sees value 9
*** thread 3 sees value 10
*** thread 3 sees value 11
*** thread 3 sees value 12
*** thread 3 sees value 13
*** thread 3 sees value 14
*** thread 3 sees value 15
*** thread 3 sees value 16
*** thread 3 sees value 17
*** thread 3 sees value 18
*** thread 3 sees value 19
*** thread 4 sees value 20
*** thread 4 sees value 21
*** thread 4 sees value 22
*** thread 4 sees value 23
*** thread 4 sees value 24
*** thread 4 sees value 25
```

(few more lines)



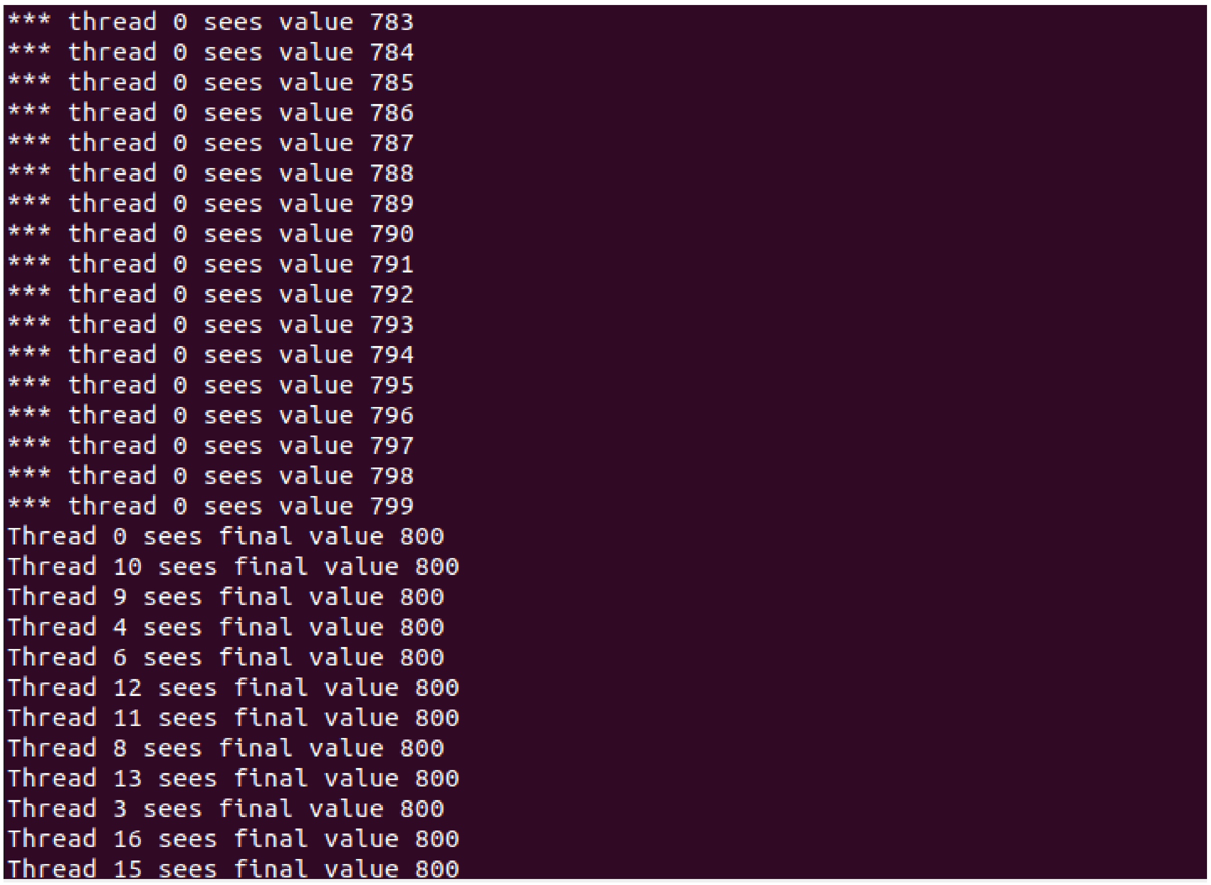
```
*** thread 0 sees value 224
*** thread 0 sees value 225
*** thread 0 sees value 226
*** thread 0 sees value 227
*** thread 0 sees value 228
*** thread 0 sees value 229
*** thread 0 sees value 230
*** thread 0 sees value 231
*** thread 0 sees value 232
*** thread 0 sees value 233
*** thread 0 sees value 234
*** thread 0 sees value 235
*** thread 0 sees value 236
*** thread 0 sees value 237
*** thread 0 sees value 238
*** thread 0 sees value 239
Thread 0 sees final value 240
Thread 8 sees final value 240
Thread 7 sees final value 240
Thread 6 sees final value 240
Thread 9 sees final value 240
Thread 11 sees final value 240
Thread 5 sees final value 240
Thread 4 sees final value 240
Thread 1 sees final value 240
Thread 2 sees final value 240
Thread 10 sees final value 240
Thread 3 sees final value 240
```

- **Create 40 threads:**

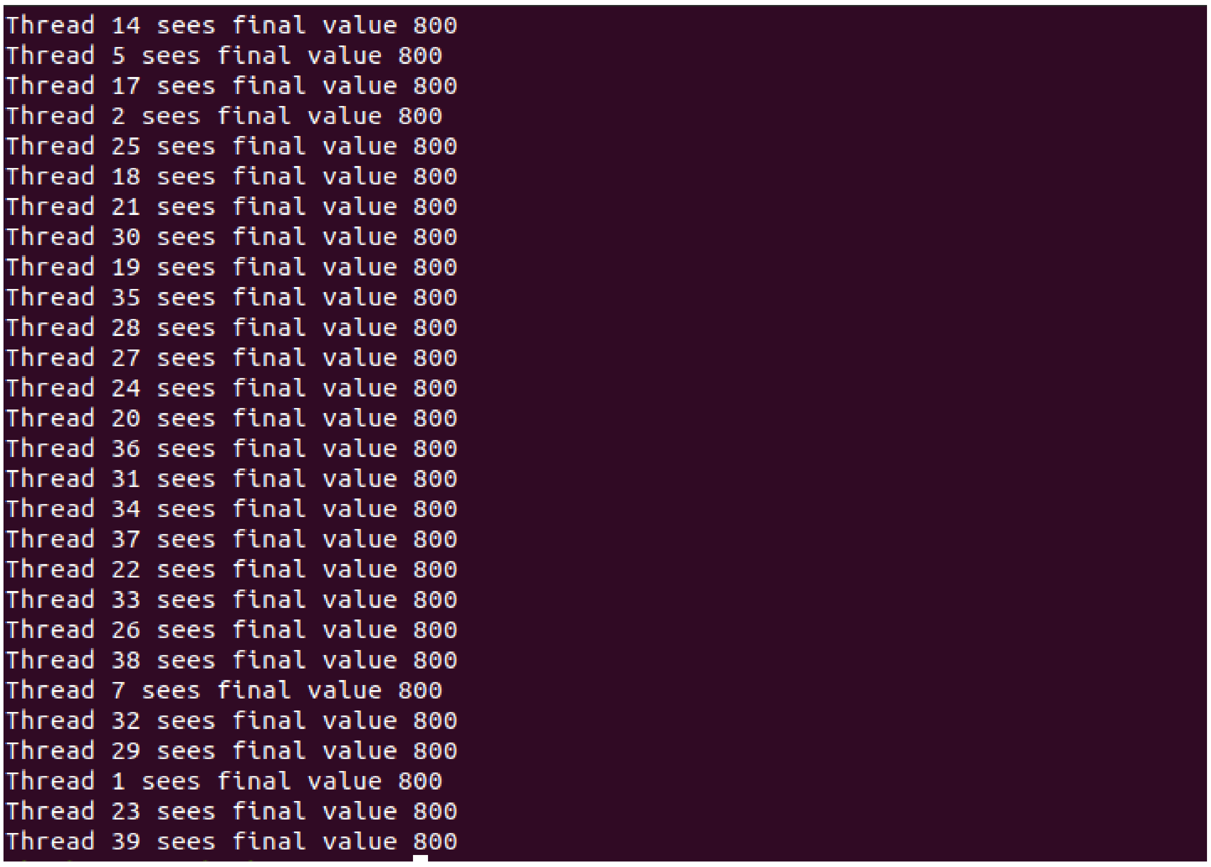


```
thanhnguyen@thanhnguyen:~$ ./part01-sync 40
*** thread 7 sees value 0
*** thread 7 sees value 1
*** thread 7 sees value 2
*** thread 7 sees value 3
*** thread 7 sees value 4
*** thread 7 sees value 5
*** thread 7 sees value 6
*** thread 7 sees value 7
*** thread 7 sees value 8
*** thread 7 sees value 9
*** thread 7 sees value 10
*** thread 7 sees value 11
*** thread 7 sees value 12
*** thread 7 sees value 13
*** thread 7 sees value 14
*** thread 7 sees value 15
*** thread 7 sees value 16
*** thread 7 sees value 17
*** thread 7 sees value 18
*** thread 7 sees value 19
*** thread 5 sees value 20
*** thread 5 sees value 21
*** thread 5 sees value 22
*** thread 5 sees value 23
*** thread 5 sees value 24
*** thread 5 sees value 25
```

(few more lines)



```
*** thread 0 sees value 783
*** thread 0 sees value 784
*** thread 0 sees value 785
*** thread 0 sees value 786
*** thread 0 sees value 787
*** thread 0 sees value 788
*** thread 0 sees value 789
*** thread 0 sees value 790
*** thread 0 sees value 791
*** thread 0 sees value 792
*** thread 0 sees value 793
*** thread 0 sees value 794
*** thread 0 sees value 795
*** thread 0 sees value 796
*** thread 0 sees value 797
*** thread 0 sees value 798
*** thread 0 sees value 799
Thread 0 sees final value 800
Thread 10 sees final value 800
Thread 9 sees final value 800
Thread 4 sees final value 800
Thread 6 sees final value 800
Thread 12 sees final value 800
Thread 11 sees final value 800
Thread 8 sees final value 800
Thread 13 sees final value 800
Thread 3 sees final value 800
Thread 16 sees final value 800
Thread 15 sees final value 800
```



```
Thread 14 sees final value 800
Thread 5 sees final value 800
Thread 17 sees final value 800
Thread 2 sees final value 800
Thread 25 sees final value 800
Thread 18 sees final value 800
Thread 21 sees final value 800
Thread 30 sees final value 800
Thread 19 sees final value 800
Thread 35 sees final value 800
Thread 28 sees final value 800
Thread 27 sees final value 800
Thread 24 sees final value 800
Thread 20 sees final value 800
Thread 36 sees final value 800
Thread 31 sees final value 800
Thread 34 sees final value 800
Thread 37 sees final value 800
Thread 22 sees final value 800
Thread 33 sees final value 800
Thread 26 sees final value 800
Thread 38 sees final value 800
Thread 7 sees final value 800
Thread 32 sees final value 800
Thread 29 sees final value 800
Thread 1 sees final value 800
Thread 23 sees final value 800
Thread 39 sees final value 800
```

Step 3: Compare 2 steps:

❖ **Describe the difference:**

- For part 1 (without Synchronization): All the threads can access the SharedVariable variable at any time. Sometimes, some threads see the same value. When all threads finish the for loop, the final value of the last thread does not see the 20*number_of_threads value. And each thread sees the different final value from the others.

- For part 2 (with Synchronization): Only one thread can access the SharedVariable at the time, then each value will be seen by only one thread. Moreover, just after the current thread sees all 20 values, another thread can access SharedVariable, so there is no alternation. In addition, all the threads will see the same final value at the end, and this value should be equal to 20*number_of_threads.

    ➢ **Reason:** In the part 02, we use mutex to lock the accession of other threads to the SharedVariable value before running the for loop, and release it after the for loop finishes. Therefore, only one thread uses the shared resources and gives access when it completes its task. We also use a barrier to hold the threads until all threads finish seeing 20 values, then they will see the same final values at the end.

Individual Contributions:

Task	**Contribution**
singleThread function	Thanh Nguyen
Check input	Fiona Le
pthread_create(), pthread_join() macro definition	Fiona Le
mutex	Thanh Nguyen

barrier	
README file	Fiona Le
MAKEFILE	Fiona Le
Recording	Thanh Nguyen
Report	Thanh Nguyen , Fiona Le