嵌入式系統軟體設計

Embedded System Software Design

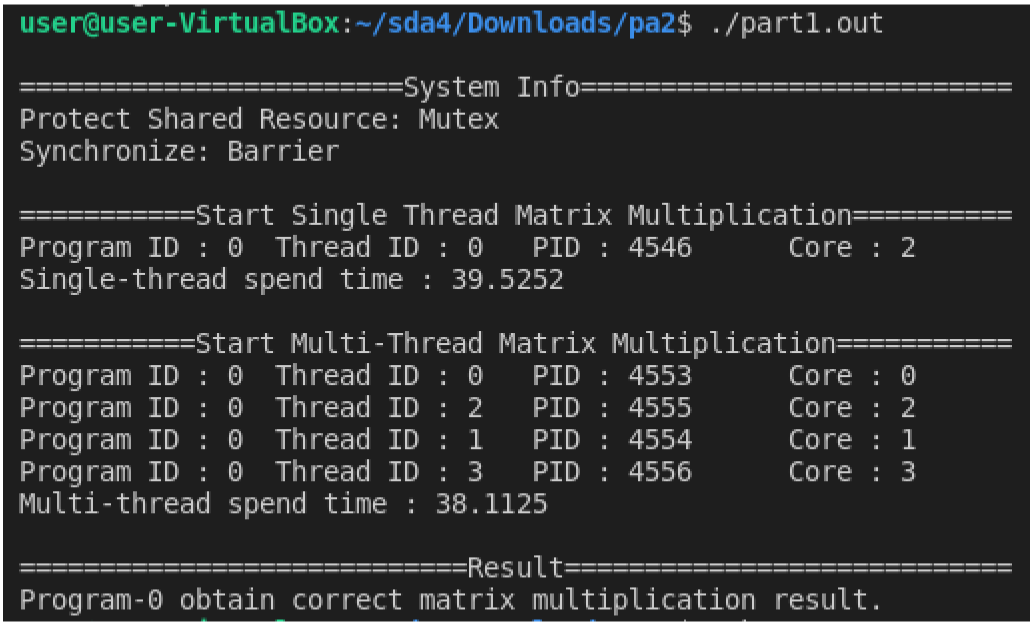
PA2

指導教授: 陳雅淑 教授

課程學生: M10907314 張祐銓

- Part 1(35%)

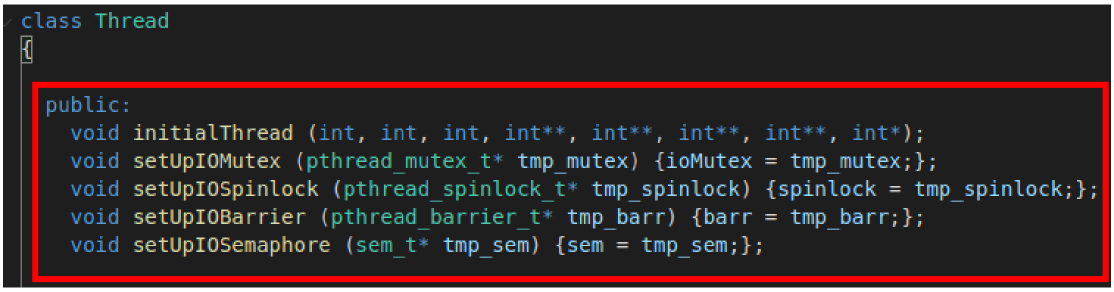
  - ➢ Execution result of using mutex and barrier. 20%



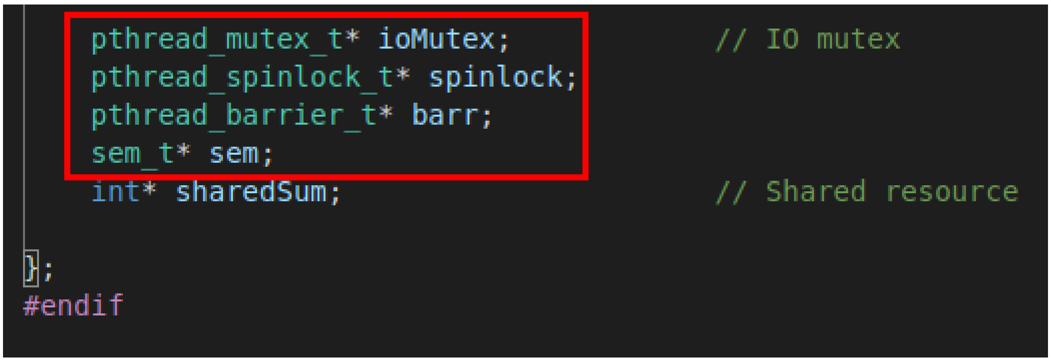
```
user@user-VirtualBox:~/sda4/Downloads/pa2$ ./part1.out

========================System Info===========================
Protect Shared Resource: Mutex
Synchronize: Barrier

===========Start Single Thread Matrix Multiplication==========
Program ID : 0  Thread ID : 0   PID : 4546      Core : 2
Single-thread spend time : 39.5252

===========Start Multi-Thread Matrix Multiplication===========
Program ID : 0  Thread ID : 0   PID : 4553      Core : 0
Program ID : 0  Thread ID : 2   PID : 4555      Core : 2
Program ID : 0  Thread ID : 1   PID : 4554      Core : 1
Program ID : 0  Thread ID : 3   PID : 4556      Core : 3
Multi-thread spend time : 38.1125

============================Result============================
Program-0 obtain correct matrix multiplication result.
```

  - ➢ Describe how to synchronize thread. 10%

在 Thread.h 中宣告 spinlock、barrier、semaphore 變數以及設定的 function。

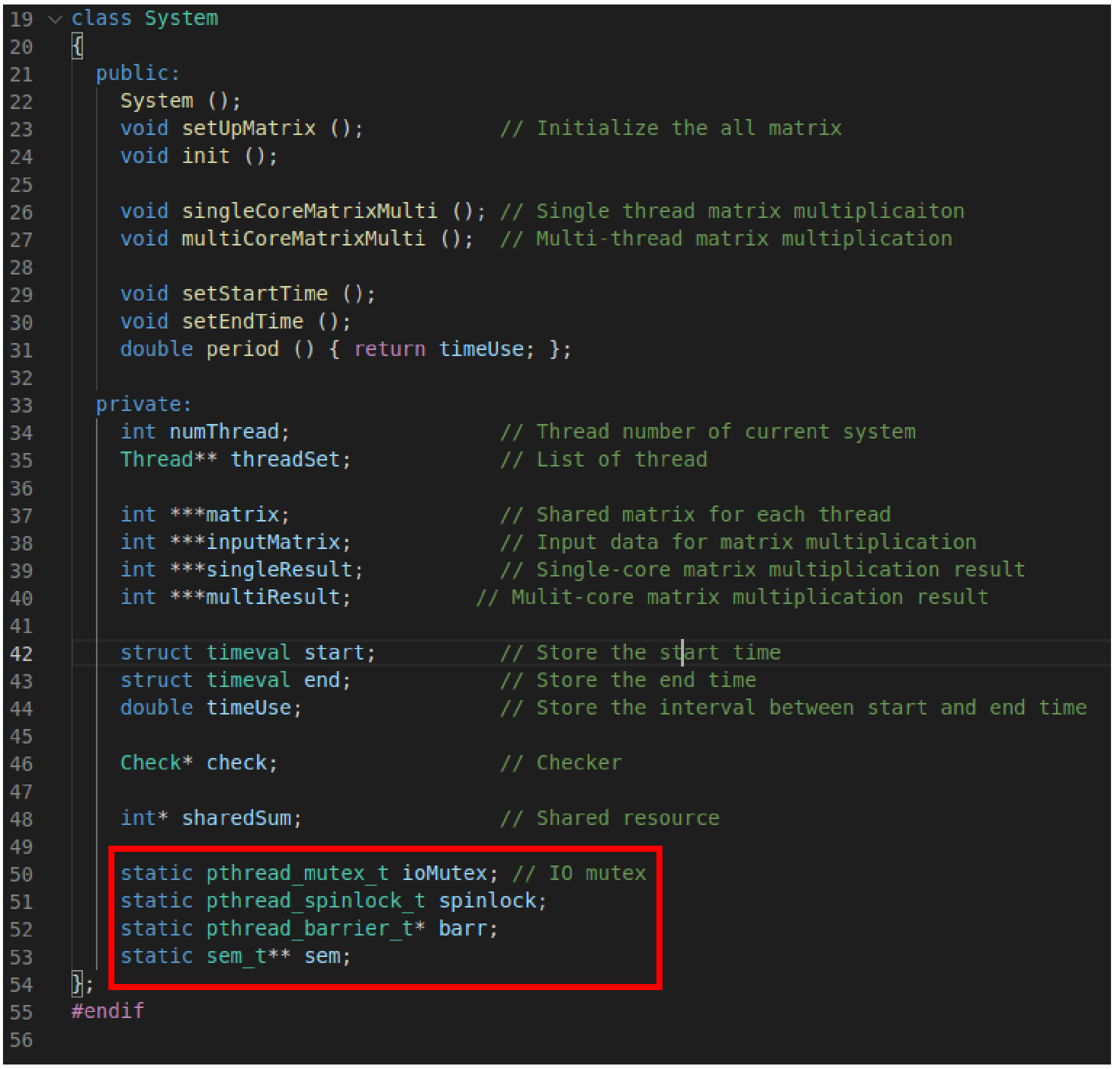


```
class Thread
{

public:
    void initialThread (int, int, int, int**, int**, int**, int**, int*);
    void setUpIOMutex (pthread_mutex_t* tmp_mutex) {ioMutex = tmp_mutex;};
    void setUpIOSpinlock (pthread_spinlock_t* tmp_spinlock) {spinlock = tmp_spinlock;};
    void setUpIOBarrier (pthread_barrier_t* tmp_barr) {barr = tmp_barr;};
    void setUpIOSemaphore (sem_t* tmp_sem) {sem = tmp_sem;};
```



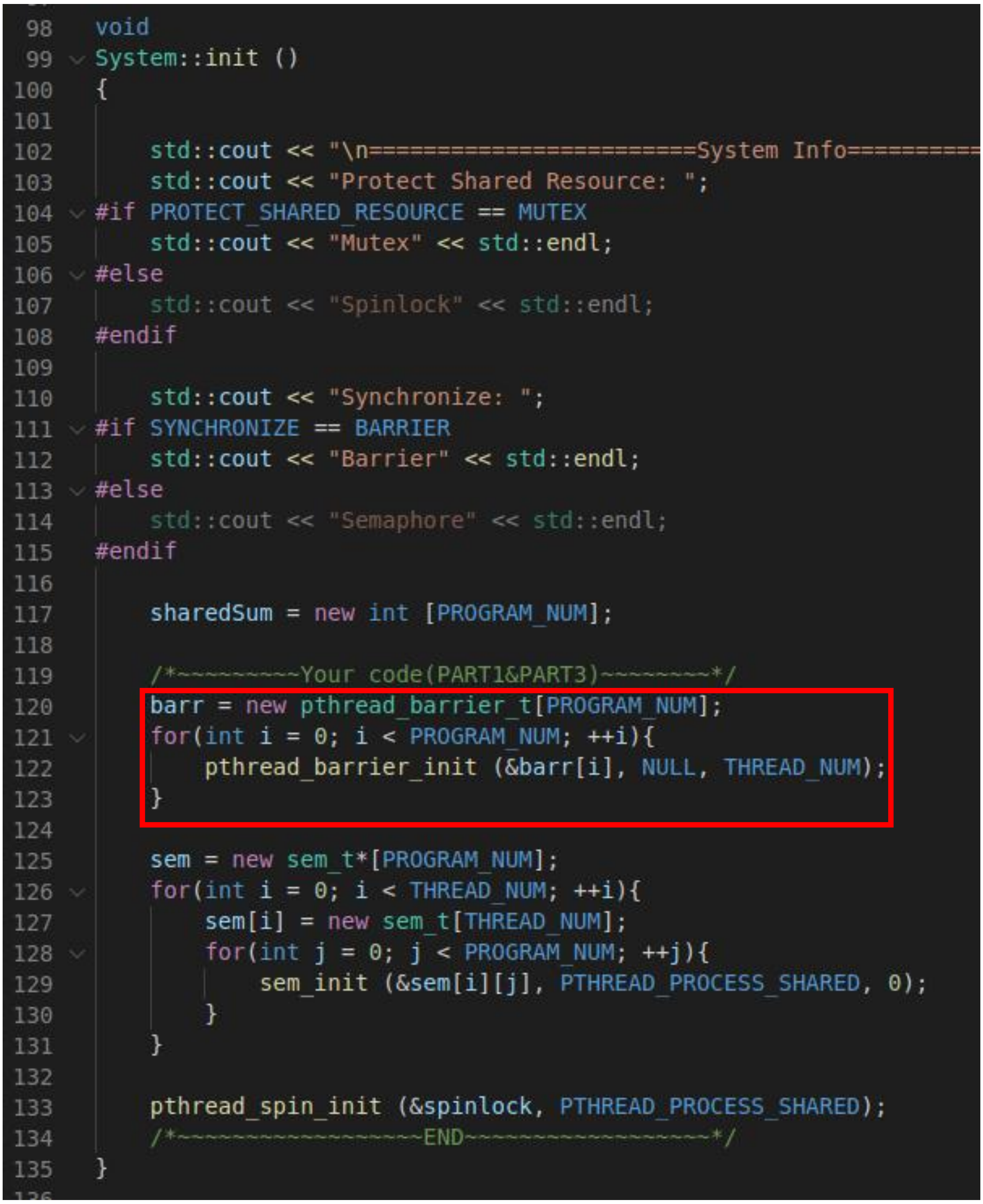
```
    pthread_mutex_t* ioMutex;           // IO mutex
    pthread_spinlock_t* spinlock;
    pthread_barrier_t* barr;
    sem_t* sem;
    int* sharedSum;                     // Shared resource

};
#endif
```

在 System 中同樣宣告 spinlock、barrier、semaphore 變數。



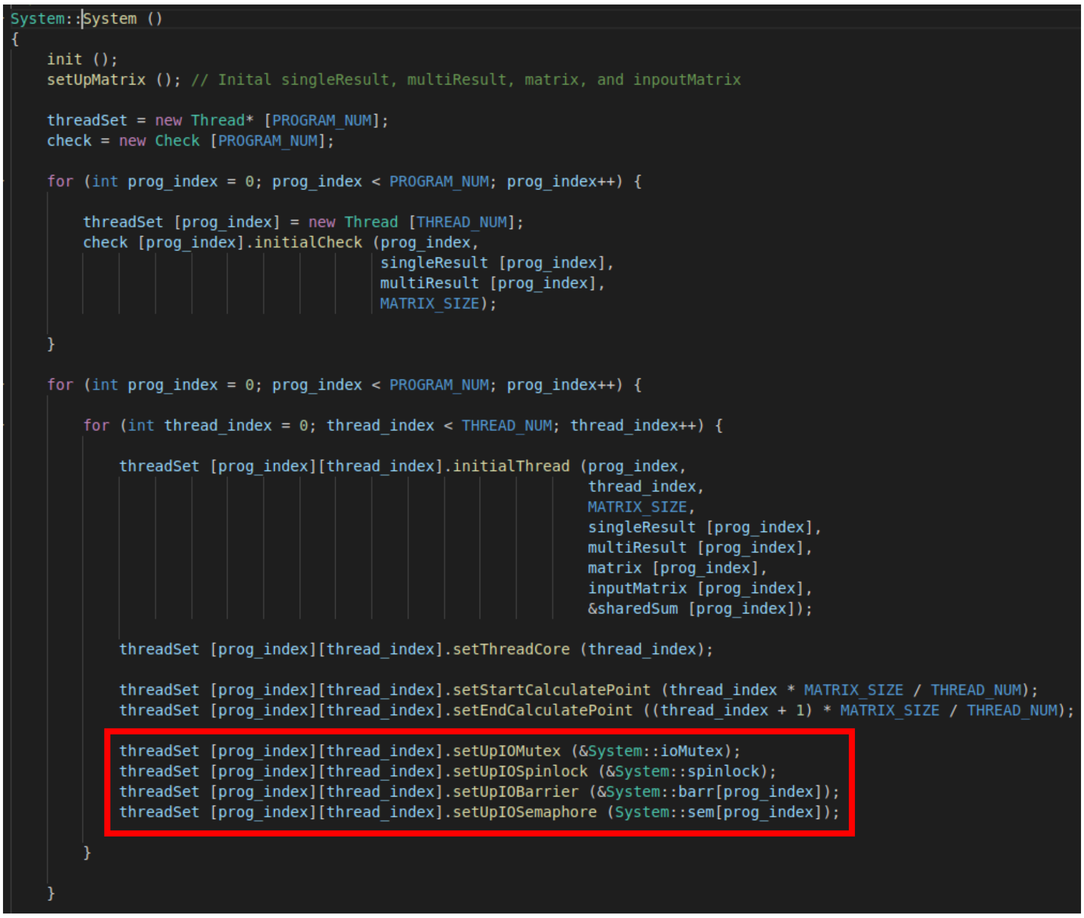
```
class System
{
  public:
    System ();
    void setUpMatrix ();           // Initialize the all matrix

    void init ();

    void singleCoreMatrixMulti (); // Single thread matrix multiplicaiton
    void multiCoreMatrixMulti ();  // Multi-thread matrix multiplication

    void setStartTime ();
    void setEndTime ();
    double period () { return timeUse; };

  private:
    int numThread;                 // Thread number of current system
    Thread** threadSet;            // List of thread

    int ***matrix;                 // Shared matrix for each thread
    int ***inputMatrix;            // Input data for matrix multiplication
    int ***singleResult;           // Single-core matrix multiplication result
    int ***multiResult;          // Mulit-core matrix multiplication result

    struct timeval start;          // Store the start time
    struct timeval end;            // Store the end time
    double timeUse;                // Store the interval between start and end time

    Check* check;                  // Checker

    int* sharedSum;                // Shared resource

    static pthread_mutex_t ioMutex; // IO mutex
    static pthread_spinlock_t spinlock;
    static pthread_barrier_t* barr;
    static sem_t** sem;
};
#endif
```

因為會根據 PROGRAM_NUM 這個變數決定說要計算幾個矩陣，而不同矩陣也會分配 THREAD_NUM 個執行緒去同步計算，所以 barrier 會創建 PROGRAM_NUM 個，並將每個 Barrier 初始化為 THREAD_NUM 次，因為要等到所有執行同個矩陣的 Thread 到達才繼續而一個矩陣會有 THREAD_NUM 個同時做。



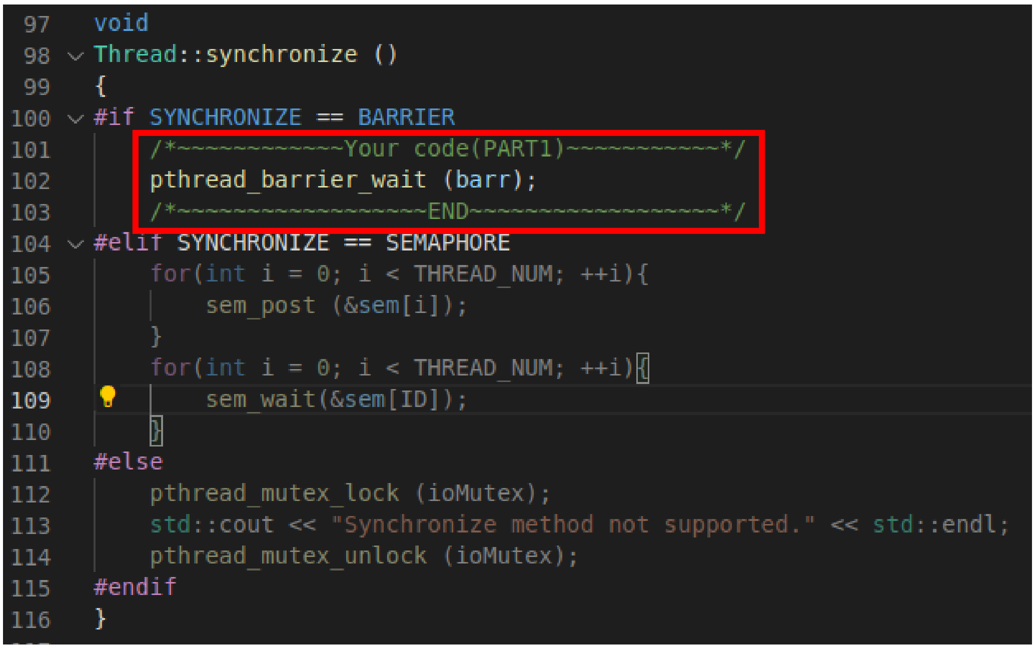
```
void
System::init ()
{

    std::cout << "\n========================System Info=========
    std::cout << "Protect Shared Resource: ";
#if PROTECT_SHARED_RESOURCE == MUTEX
    std::cout << "Mutex" << std::endl;
#else
    std::cout << "Spinlock" << std::endl;
#endif

    std::cout << "Synchronize: ";
#if SYNCHRONIZE == BARRIER
    std::cout << "Barrier" << std::endl;
#else
    std::cout << "Semaphore" << std::endl;
#endif

    sharedSum = new int [PROGRAM_NUM];

    /*~~~~~~~~~Your code(PART1&PART3)~~~~~~~~*/
    barr = new pthread_barrier_t[PROGRAM_NUM];
    for(int i = 0; i < PROGRAM_NUM; ++i){
        pthread_barrier_init (&barr[i], NULL, THREAD_NUM);
    }

    sem = new sem_t*[PROGRAM_NUM];
    for(int i = 0; i < THREAD_NUM; ++i){
        sem[i] = new sem_t[THREAD_NUM];
        for(int j = 0; j < PROGRAM_NUM; ++j){
            sem_init (&sem[i][j], PTHREAD_PROCESS_SHARED, 0);
        }
    }

    pthread_spin_init (&spinlock, PTHREAD_PROCESS_SHARED);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~*/
}
```

將指標設定給每個 Thread，執行同個矩陣的 thread 會拿到同個 barrier。



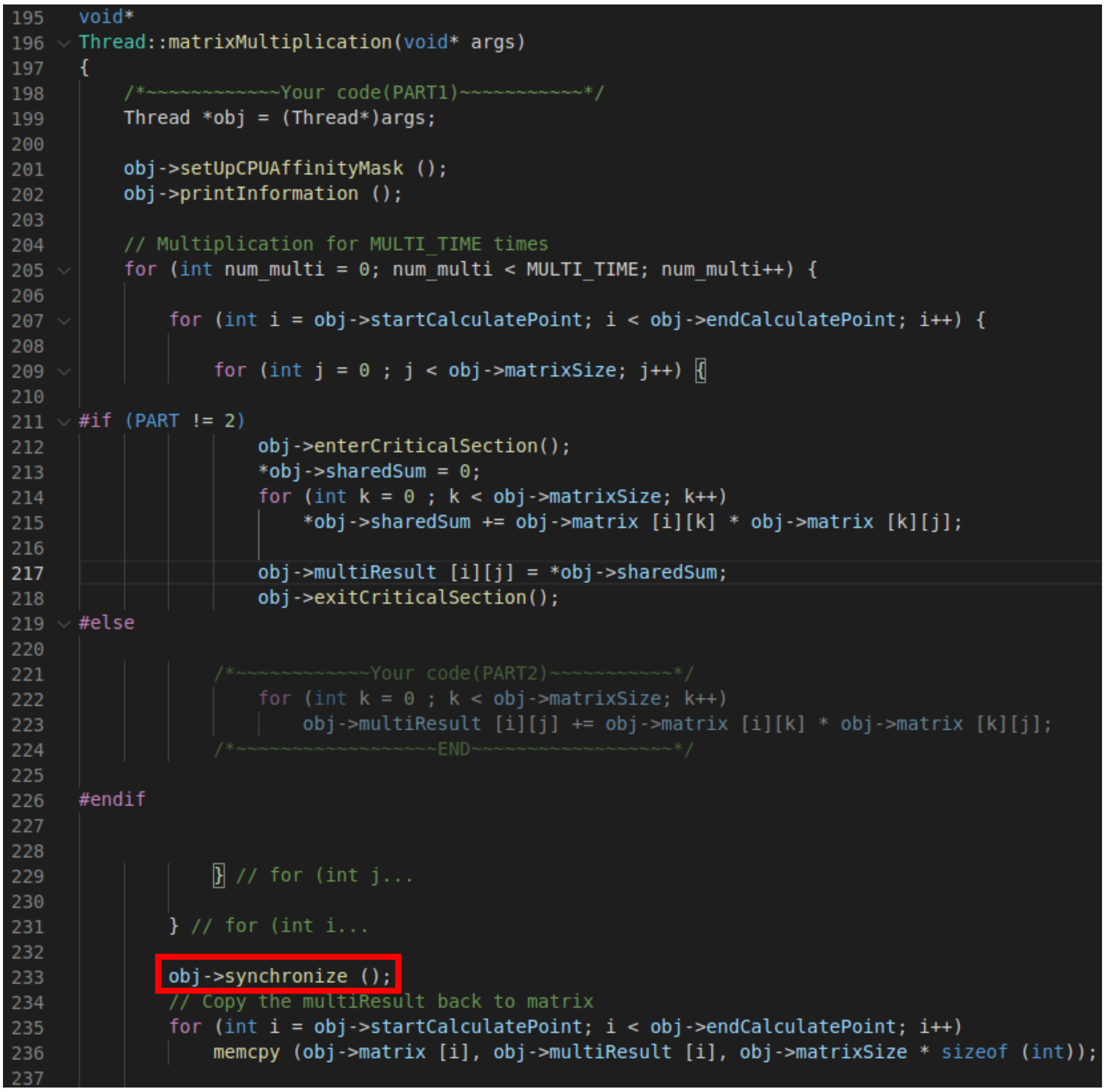
```cpp
System::System ()
{
    init ();
    setUpMatrix (); // Inital singleResult, multiResult, matrix, and inpoutMatrix

    threadSet = new Thread* [PROGRAM_NUM];
    check = new Check [PROGRAM_NUM];

    for (int prog_index = 0; prog_index < PROGRAM_NUM; prog_index++) {

        threadSet [prog_index] = new Thread [THREAD_NUM];
        check [prog_index].initialCheck (prog_index,
                                         singleResult [prog_index],
                                         multiResult [prog_index],
                                         MATRIX_SIZE);

    }

    for (int prog_index = 0; prog_index < PROGRAM_NUM; prog_index++) {

        for (int thread_index = 0; thread_index < THREAD_NUM; thread_index++) {

            threadSet [prog_index][thread_index].initialThread (prog_index,
                                                                thread_index,
                                                                MATRIX_SIZE,
                                                                singleResult [prog_index],
                                                                multiResult [prog_index],
                                                                matrix [prog_index],
                                                                inputMatrix [prog_index],
                                                                &sharedSum [prog_index]);

            threadSet [prog_index][thread_index].setThreadCore (thread_index);

            threadSet [prog_index][thread_index].setStartCalculatePoint (thread_index * MATRIX_SIZE / THREAD_NUM);
            threadSet [prog_index][thread_index].setEndCalculatePoint ((thread_index + 1) * MATRIX_SIZE / THREAD_NUM);

            threadSet [prog_index][thread_index].setUpIOMutex (&System::ioMutex);
            threadSet [prog_index][thread_index].setUpIOSpinlock (&System::spinlock);
            threadSet [prog_index][thread_index].setUpIOBarrier (&System::barr[prog_index]);
            threadSet [prog_index][thread_index].setUpIOSemaphore (System::sem[prog_index]);

        }

    }
```

synchronize 中就會執行 barrier wait 等待子其他 Thread 同步。



```cpp
void
Thread::synchronize ()
{
#if SYNCHRONIZE == BARRIER
    /*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
    pthread_barrier_wait (barr);
    /*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#elif SYNCHRONIZE == SEMAPHORE
    for(int i = 0; i < THREAD_NUM; ++i){
        sem_post (&sem[i]);
    }
    for(int i = 0; i < THREAD_NUM; ++i){
        sem_wait(&sem[ID]);
    }
#else
    pthread_mutex_lock (ioMutex);
    std::cout << "Synchronize method not supported." << std::endl;
    pthread_mutex_unlock (ioMutex);
#endif
}
```

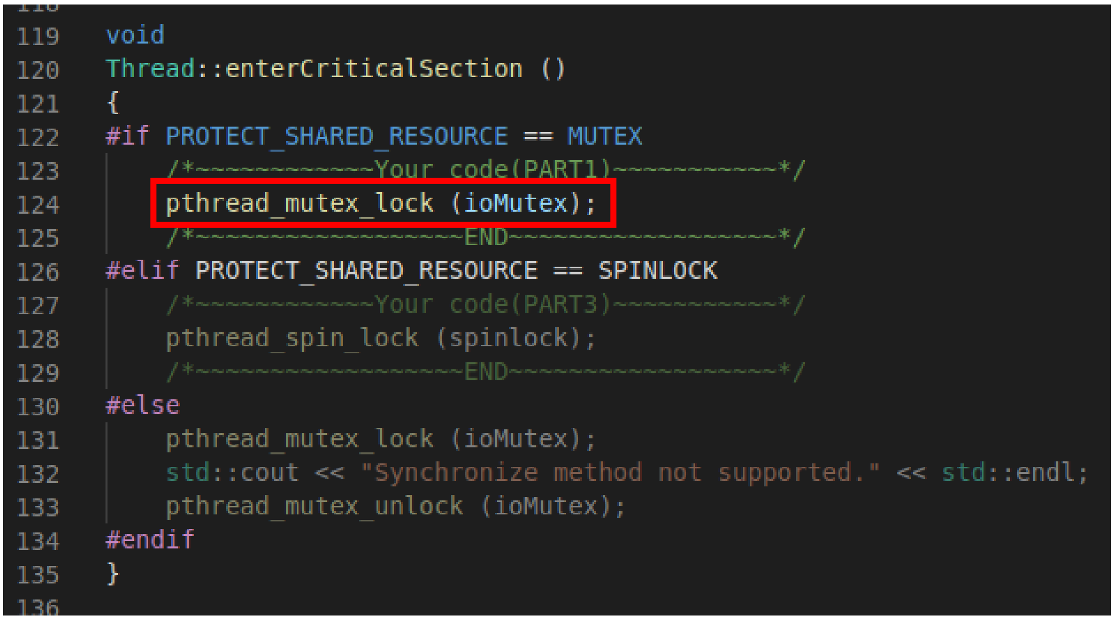
要在寫入 matrix 前同步，因為若不同步而其他 Thread 先改變 matrix 的值的話上面其他還在計算的 Thread 會出問題，因為 matrix 的值已經被改寫了而矩陣計算會拿 matrix 的值計算所以結果會出錯。



```
void*
Thread::matrixMultiplication(void* args)
{
    /*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
    Thread *obj = (Thread*)args;

    obj->setUpCPUAffinityMask ();
    obj->printInformation ();

    // Multiplication for MULTI_TIME times
    for (int num_multi = 0; num_multi < MULTI_TIME; num_multi++) {

        for (int i = obj->startCalculatePoint; i < obj->endCalculatePoint; i++) {

            for (int j = 0 ; j < obj->matrixSize; j++) {

#if (PART != 2)
                obj->enterCriticalSection();
                *obj->sharedSum = 0;
                for (int k = 0 ; k < obj->matrixSize; k++)
                    *obj->sharedSum += obj->matrix [i][k] * obj->matrix [k][j];

                obj->multiResult [i][j] = *obj->sharedSum;
                obj->exitCriticalSection();
#else

            /*~~~~~~~~~~~~Your code(PART2)~~~~~~~~~~~*/
                for (int k = 0 ; k < obj->matrixSize; k++)
                    obj->multiResult [i][j] += obj->matrix [i][k] * obj->matrix [k][j];
            /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/

#endif


            } // for (int j...

        } // for (int i...

        obj->synchronize ();
        // Copy the multiResult back to matrix
        for (int i = obj->startCalculatePoint; i < obj->endCalculatePoint; i++)
            memcpy (obj->matrix [i], obj->multiResult [i], obj->matrixSize * sizeof (int));

```

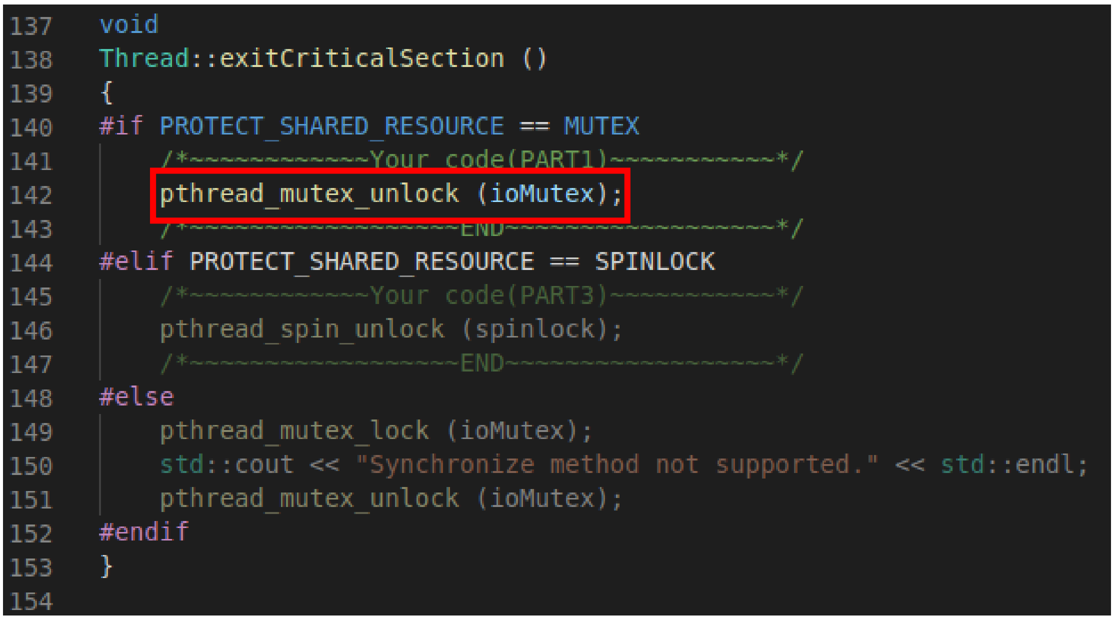
- ➢ Describe how to protect a shared resource. 5%

在 enterCriticalSection 中執行 pthread_mutex_lock 將 mutex 鎖住。



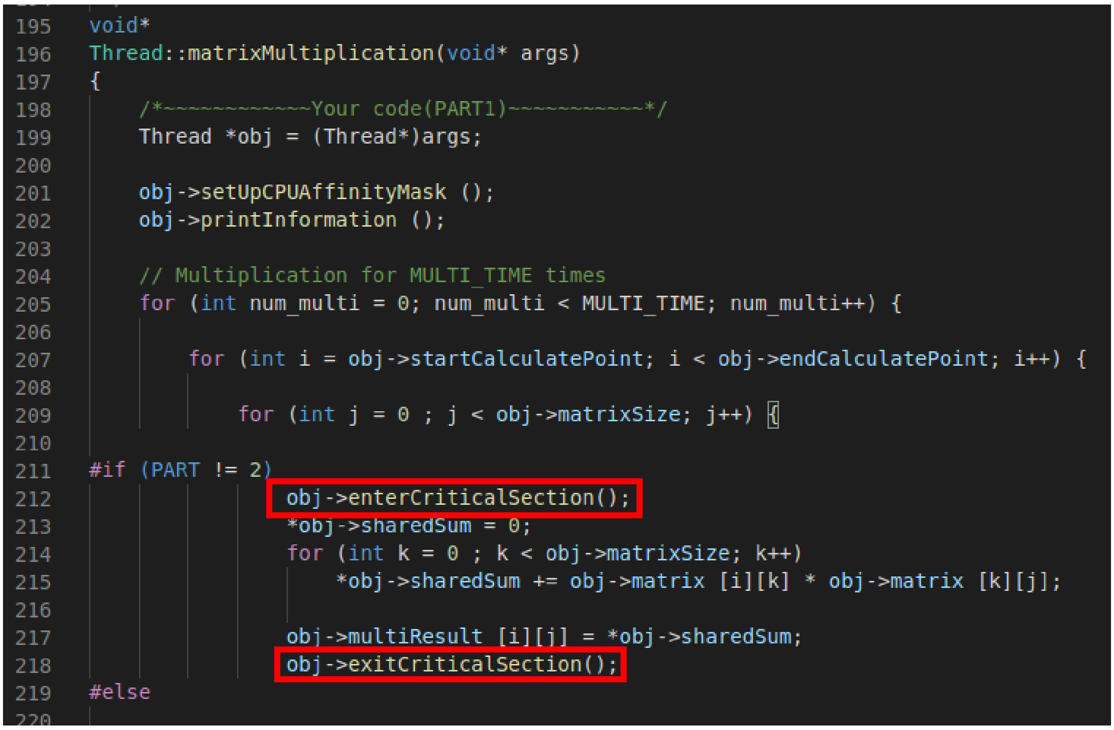
```
void
Thread::enterCriticalSection ()
{
#if PROTECT_SHARED_RESOURCE == MUTEX
    /*~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
    pthread_mutex_lock (ioMutex);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#elif PROTECT_SHARED_RESOURCE == SPINLOCK
    /*~~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~~*/
    pthread_spin_lock (spinlock);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#else
    pthread_mutex_lock (ioMutex);
    std::cout << "Synchronize method not supported." << std::endl;
    pthread_mutex_unlock (ioMutex);
#endif
}
```

在 exitCriticalSection 中執行 pthread_mutex_unlock 將 mutex 釋放。



```
void
Thread::exitCriticalSection ()
{
#if PROTECT_SHARED_RESOURCE == MUTEX
    /*~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
    pthread_mutex_unlock (ioMutex);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#elif PROTECT_SHARED_RESOURCE == SPINLOCK
    /*~~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~~*/
    pthread_spin_unlock (spinlock);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#else
    pthread_mutex_lock (ioMutex);
    std::cout << "Synchronize method not supported." << std::endl;
    pthread_mutex_unlock (ioMutex);
#endif
}
```

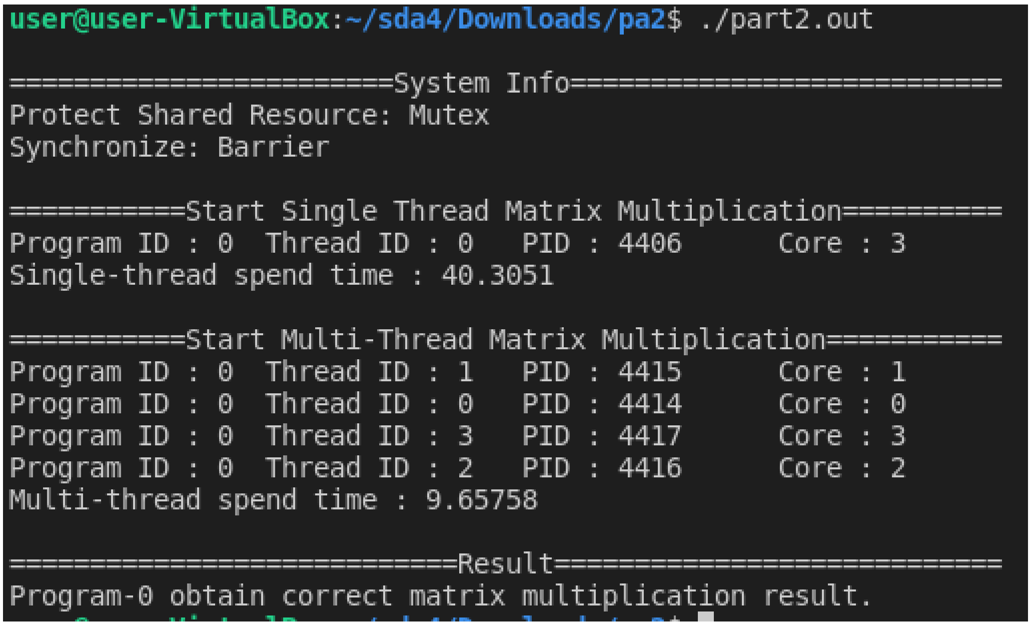
sharedSum 是所有 Thread 的共用變數會先儲存到這裡再放到 multiResult 中，所以當在只用 sharedSum 前就要執行 enterCriticalSection 將變數保護，sharedSum 寫入 multiResult 後就可以執行 exitCriticalSection 釋放掉了。



```
void*
Thread::matrixMultiplication(void* args)
{
    /*~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
    Thread *obj = (Thread*)args;

    obj->setUpCPUAffinityMask ();
    obj->printInformation ();

    // Multiplication for MULTI_TIME times
    for (int num_multi = 0; num_multi < MULTI_TIME; num_multi++) {

        for (int i = obj->startCalculatePoint; i < obj->endCalculatePoint; i++) {

            for (int j = 0 ; j < obj->matrixSize; j++) {

#if (PART != 2)
                obj->enterCriticalSection();
                *obj->sharedSum = 0;
                for (int k = 0 ; k < obj->matrixSize; k++)
                    *obj->sharedSum += obj->matrix [i][k] * obj->matrix [k][j];

                obj->multiResult [i][j] = *obj->sharedSum;
                obj->exitCriticalSection();
#else
```

- Part 2(30%)

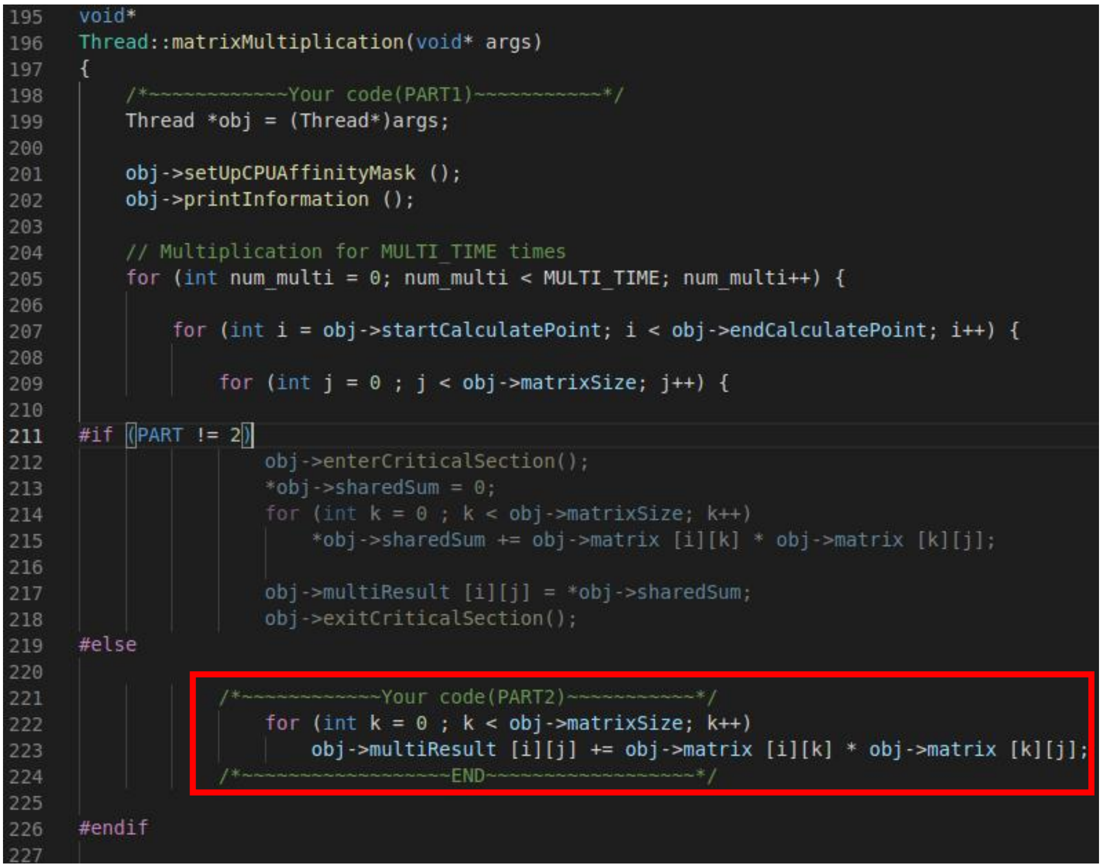
➢ Execution resultof using reentrant function.15%



```
user@user-VirtualBox:~/sda4/Downloads/pa2$ ./part2.out

========================System Info===========================
Protect Shared Resource: Mutex
Synchronize: Barrier

===========Start Single Thread Matrix Multiplication==========
Program ID : 0  Thread ID : 0   PID : 4406      Core : 3
Single-thread spend time : 40.3051

===========Start Multi-Thread Matrix Multiplication===========
Program ID : 0  Thread ID : 1   PID : 4415      Core : 1
Program ID : 0  Thread ID : 0   PID : 4414      Core : 0
Program ID : 0  Thread ID : 3   PID : 4417      Core : 3
Program ID : 0  Thread ID : 2   PID : 4416      Core : 2
Multi-thread spend time : 9.65758

============================Result============================
Program-0 obtain correct matrix multiplication result.
```

- Describe how to modify non-reentrant function into reentrant function. 10%

將計算結果直接寫入 multiResult 中就可以不使用共用變數。



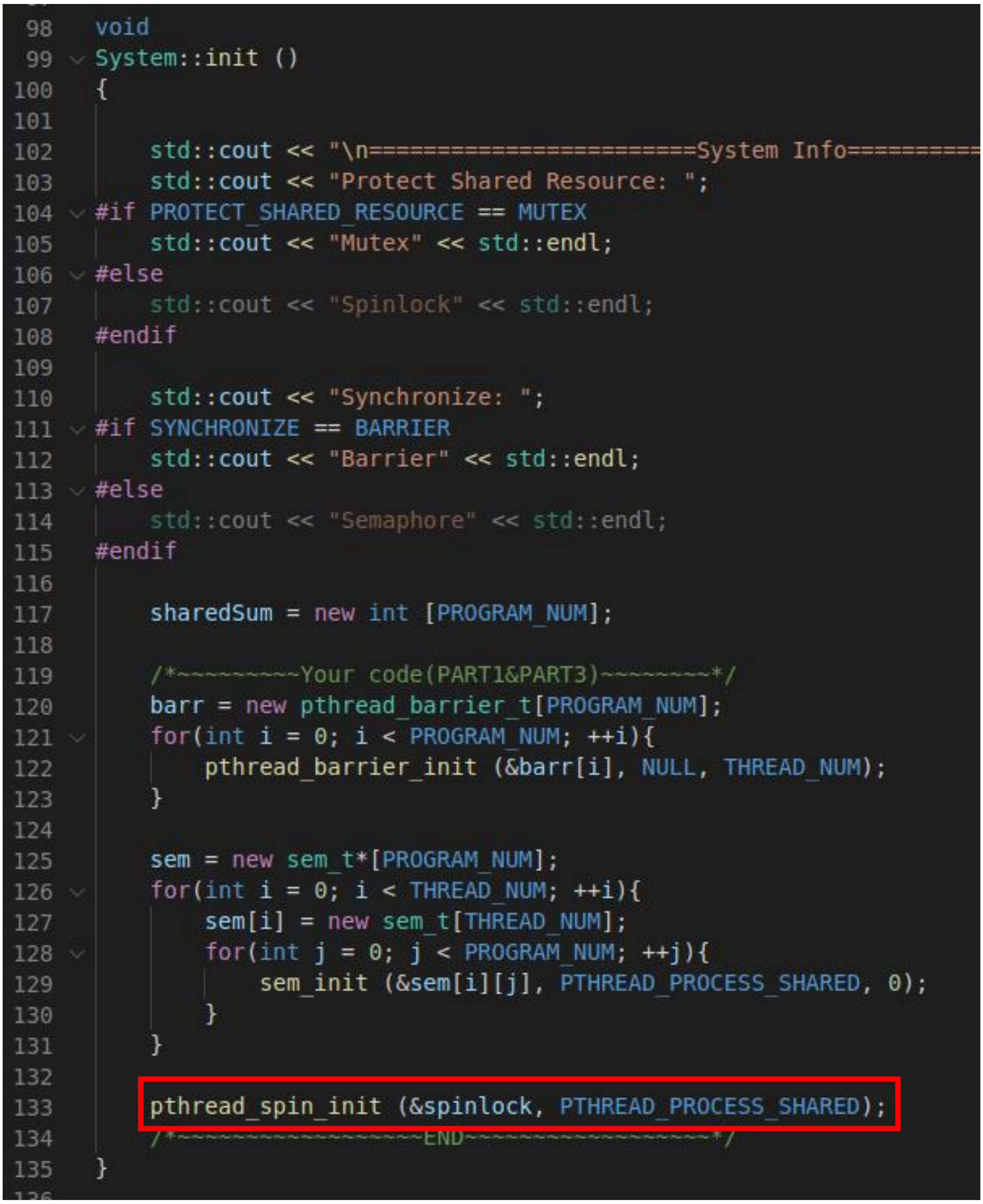
```
void*
Thread::matrixMultiplication(void* args)
{
    /*~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
    Thread *obj = (Thread*)args;

    obj->setUpCPUAffinityMask ();
    obj->printInformation ();

    // Multiplication for MULTI_TIME times
    for (int num_multi = 0; num_multi < MULTI_TIME; num_multi++) {

        for (int i = obj->startCalculatePoint; i < obj->endCalculatePoint; i++) {

            for (int j = 0 ; j < obj->matrixSize; j++) {

#if (PART != 2)
                obj->enterCriticalSection();
                *obj->sharedSum = 0;
                for (int k = 0 ; k < obj->matrixSize; k++)
                    *obj->sharedSum += obj->matrix [i][k] * obj->matrix [k][j];

                obj->multiResult [i][j] = *obj->sharedSum;
                obj->exitCriticalSection();
#else

            /*~~~~~~~~~~~~~Your code(PART2)~~~~~~~~~~~*/
                for (int k = 0 ; k < obj->matrixSize; k++)
                    obj->multiResult [i][j] += obj->matrix [i][k] * obj->matrix [k][j];
            /*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/

#endif

```

- Describe the reason why using a non-reentrant function or a reentrant function could obtain better performance. 5%

因為直接將結果寫入 multiResult 的話就可以不使用共用變數 shareResult，可以避免掉因為需要同時使用 shareResult 互相等待的問題，就可以減少等待的時間產生更好的效率。

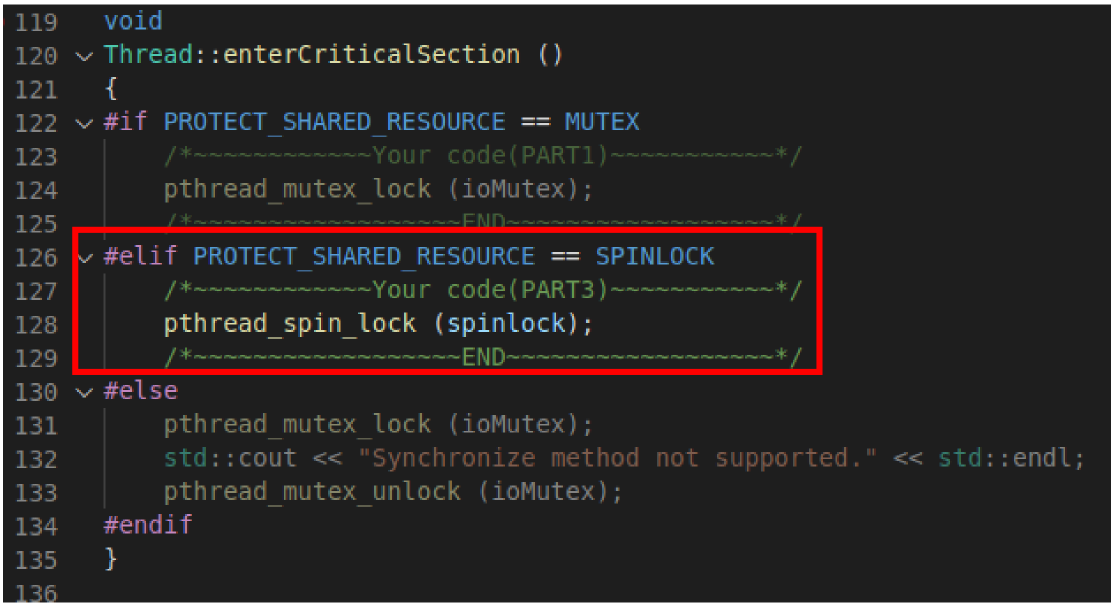
- ## Part 3(35%)
  - Execution result of using spinlock. 10%

初始化 spinlock 並設定成 PTHREAD_PROCESS_SHARED。



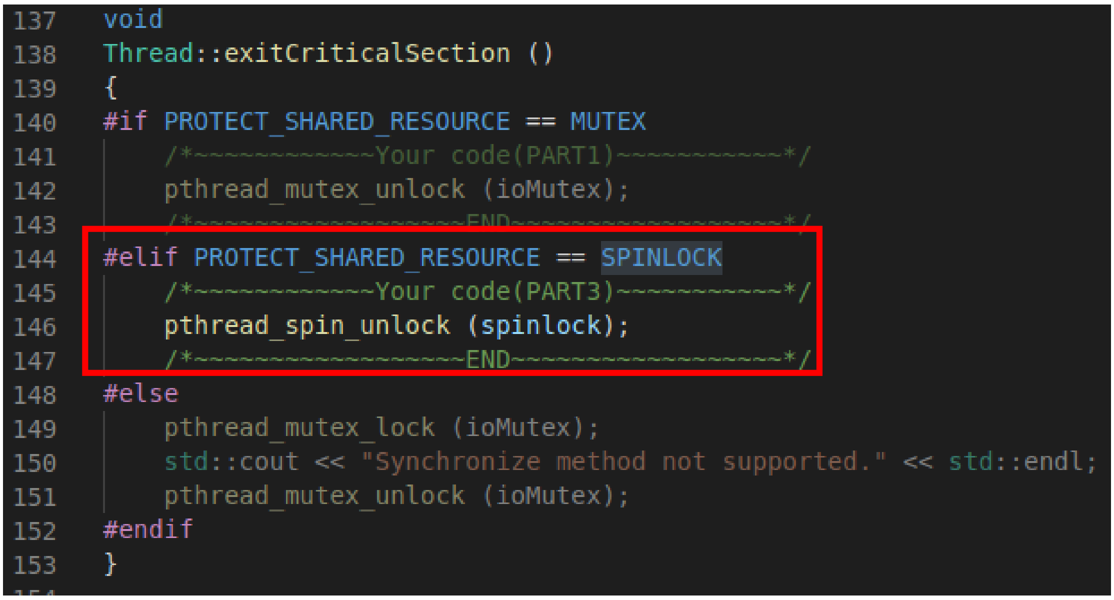
```
void
System::init ()
{

    std::cout << "\n========================System Info==========
    std::cout << "Protect Shared Resource: ";
#if PROTECT_SHARED_RESOURCE == MUTEX
    std::cout << "Mutex" << std::endl;
#else
    std::cout << "Spinlock" << std::endl;
#endif

    std::cout << "Synchronize: ";
#if SYNCHRONIZE == BARRIER
    std::cout << "Barrier" << std::endl;
#else
    std::cout << "Semaphore" << std::endl;
#endif

    sharedSum = new int [PROGRAM_NUM];

    /*~~~~~~~~~~Your code(PART1&PART3)~~~~~~~~*/
    barr = new pthread_barrier_t[PROGRAM_NUM];
    for(int i = 0; i < PROGRAM_NUM; ++i){
        pthread_barrier_init (&barr[i], NULL, THREAD_NUM);
    }

    sem = new sem_t*[PROGRAM_NUM];
    for(int i = 0; i < THREAD_NUM; ++i){
        sem[i] = new sem_t[THREAD_NUM];
        for(int j = 0; j < PROGRAM_NUM; ++j){
            sem_init (&sem[i][j], PTHREAD_PROCESS_SHARED, 0);
        }
    }

    pthread_spin_init (&spinlock, PTHREAD_PROCESS_SHARED);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
}
```

在 enterCriticalSection 中執行 pthread_spin_lock 將 spinlock 鎖住。



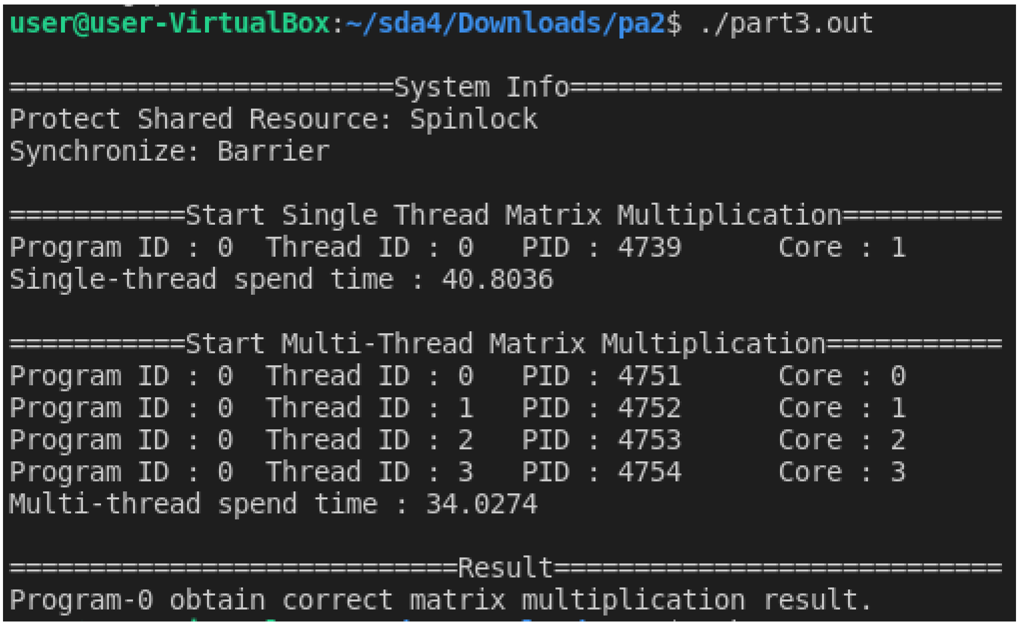
```
void
Thread::enterCriticalSection ()
{
#if PROTECT_SHARED_RESOURCE == MUTEX
    /*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
    pthread_mutex_lock (ioMutex);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#elif PROTECT_SHARED_RESOURCE == SPINLOCK
    /*~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~~*/
    pthread_spin_lock (spinlock);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#else
    pthread_mutex_lock (ioMutex);
    std::cout << "Synchronize method not supported." << std::endl;
    pthread_mutex_unlock (ioMutex);
#endif
}
```

在 exitCriticalSection 中執行 pthread_spin_unlock 將 spinlock 釋放。



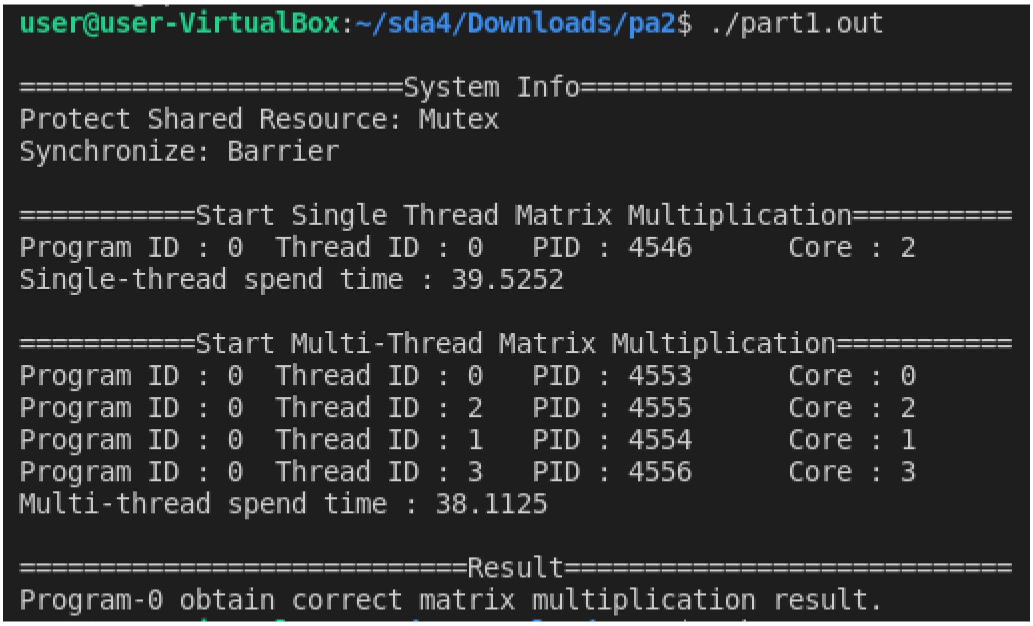
```
void
Thread::exitCriticalSection ()
{
#if PROTECT_SHARED_RESOURCE == MUTEX
    /*~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
    pthread_mutex_unlock (ioMutex);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#elif PROTECT_SHARED_RESOURCE == SPINLOCK
    /*~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~~*/
    pthread_spin_unlock (spinlock);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#else
    pthread_mutex_lock (ioMutex);
    std::cout << "Synchronize method not supported." << std::endl;
    pthread_mutex_unlock (ioMutex);
#endif
}
```

Part3 的執行結果。

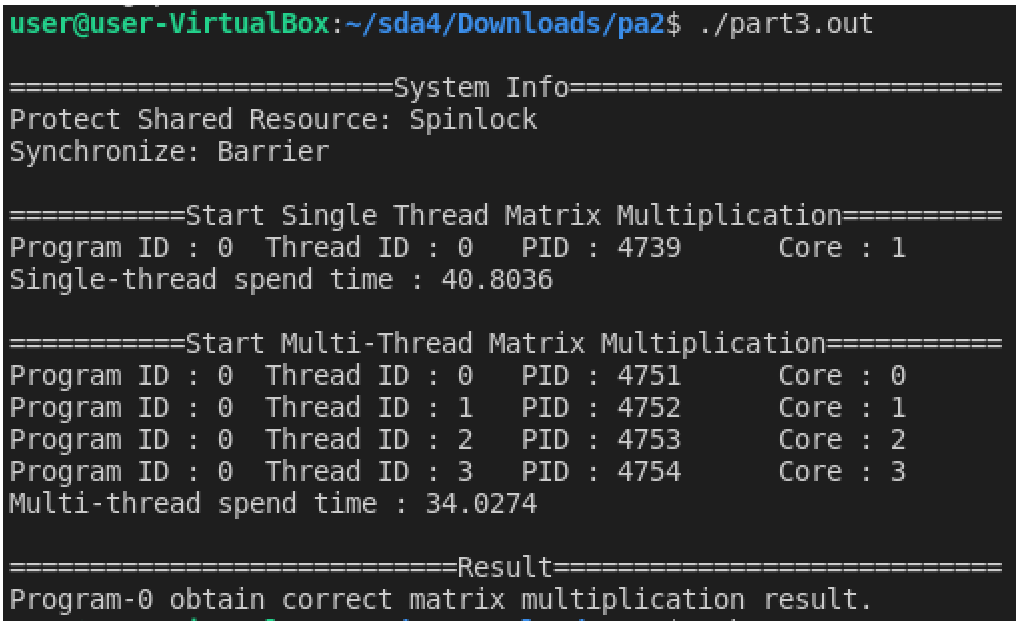


```
user@user-VirtualBox:~/sda4/Downloads/pa2$ ./part3.out

========================System Info===========================
Protect Shared Resource: Spinlock
Synchronize: Barrier

===========Start Single Thread Matrix Multiplication==========
Program ID : 0  Thread ID : 0   PID : 4739      Core : 1
Single-thread spend time : 40.8036

===========Start Multi-Thread Matrix Multiplication===========
Program ID : 0  Thread ID : 0   PID : 4751      Core : 0
Program ID : 0  Thread ID : 1   PID : 4752      Core : 1
Program ID : 0  Thread ID : 2   PID : 4753      Core : 2
Program ID : 0  Thread ID : 3   PID : 4754      Core : 3
Multi-thread spend time : 34.0274

============================Result============================
Program-0 obtain correct matrix multiplication result.
```

- ➢ Describe which method (mutex and spinlock) could obtain better performance under the benchmark we provided (5%)and why (5%).



```
user@user-VirtualBox:~/sda4/Downloads/pa2$ ./part1.out

========================System Info===========================
Protect Shared Resource: Mutex
Synchronize: Barrier

===========Start Single Thread Matrix Multiplication==========
Program ID : 0  Thread ID : 0   PID : 4546      Core : 2
Single-thread spend time : 39.5252

===========Start Multi-Thread Matrix Multiplication===========
Program ID : 0  Thread ID : 0   PID : 4553      Core : 0
Program ID : 0  Thread ID : 2   PID : 4555      Core : 2
Program ID : 0  Thread ID : 1   PID : 4554      Core : 1
Program ID : 0  Thread ID : 3   PID : 4556      Core : 3
Multi-thread spend time : 38.1125

============================Result============================
Program-0 obtain correct matrix multiplication result.
```

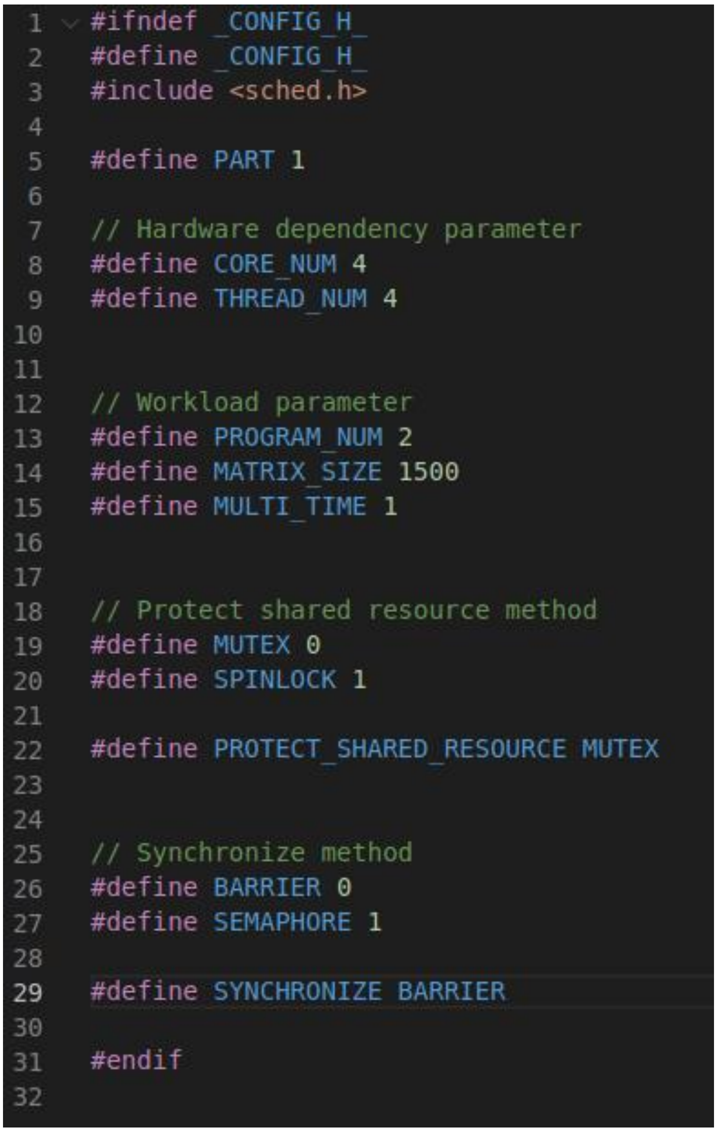


```
user@user-VirtualBox:~/sda4/Downloads/pa2$ ./part3.out

========================System Info==========================
Protect Shared Resource: Spinlock
Synchronize: Barrier

===========Start Single Thread Matrix Multiplication==========
Program ID : 0  Thread ID : 0   PID : 4739      Core : 1
Single-thread spend time : 40.8036

===========Start Multi-Thread Matrix Multiplication===========
Program ID : 0  Thread ID : 0   PID : 4751      Core : 0
Program ID : 0  Thread ID : 1   PID : 4752      Core : 1
Program ID : 0  Thread ID : 2   PID : 4753      Core : 2
Program ID : 0  Thread ID : 3   PID : 4754      Core : 3
Multi-thread spend time : 34.0274

============================Result===========================
Program-0 obtain correct matrix multiplication result.
```

從兩張分別使用 mutex 和 spinlock 的結果圖發現 spinlock 的執行速度會比 mutex 還要來的快。

因為 mutex 在會先切換到其他 Task(不是這個程式中的 task)去做，當可以使用共用資源的時候再 context switch 回來而造成會額外增加時間，而 spinlock 因為等待時是採用 busy waiting 的方式所以會一直停留在同個 task 不會切換到其他工作執行，所以當可以使用共用變數的時候就可以立刻繼續執行下去。

- Show the benchmark your used (5%), explain the properties of such benchmark(5%)and the execution results(5%).



```
#ifndef _CONFIG_H_
#define _CONFIG_H_
#include <sched.h>

#define PART 1

// Hardware dependency parameter
#define CORE_NUM 4
#define THREAD_NUM 4


// Workload parameter
#define PROGRAM_NUM 2
#define MATRIX_SIZE 1500
#define MULTI_TIME 1


// Protect shared resource method
#define MUTEX 0
#define SPINLOCK 1

#define PROTECT_SHARED_RESOURCE MUTEX


// Synchronize method
#define BARRIER 0
#define SEMAPHORE 1

#define SYNCHRONIZE BARRIER

#endif

```

此配置會讓 Mutex 處理速度快於 Spinlock 處理速度

CORE_NUM: 程式內無用到。

THREAD_NUM; 設定有多少個 thread 並行處理一個矩陣的運算。

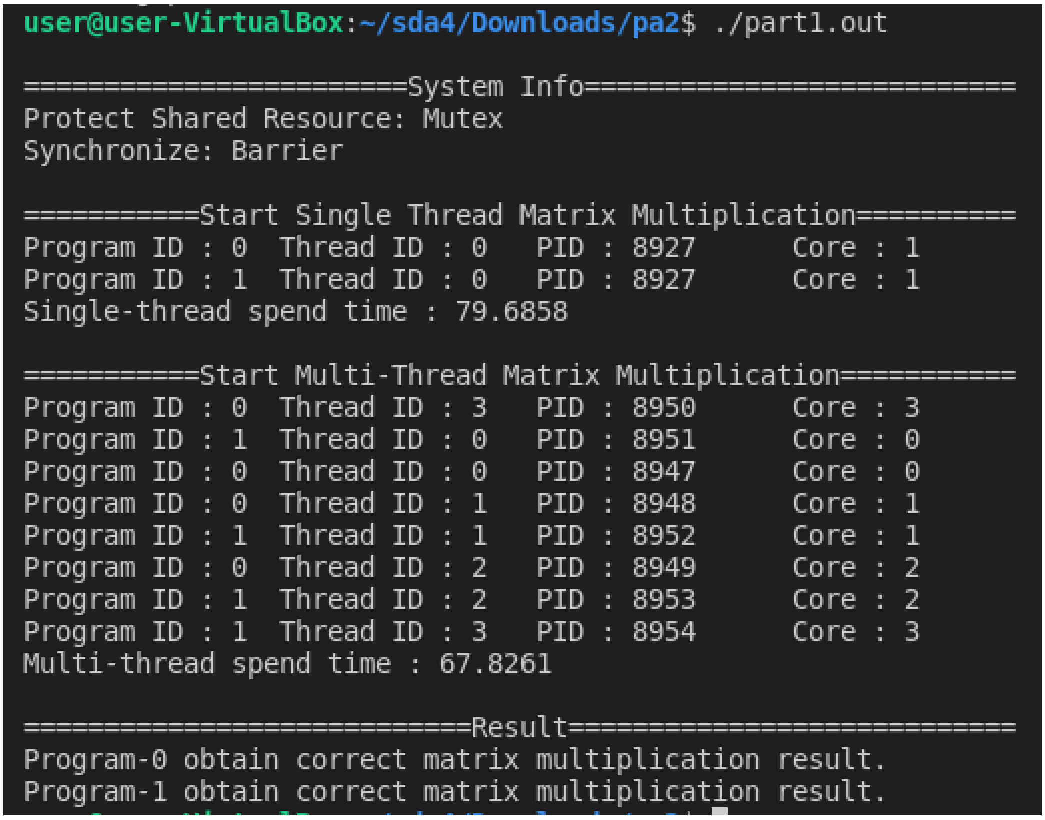
PROGRAM_NUM: 共有幾個矩陣要進行運算，所以 Thread 的數量就會變成 THREAD_NUM* PROGRAM_NUM 個。

MATRIX_SIZE: 矩陣大小 MATRIX_SIZE* MATRIX_SIZE。

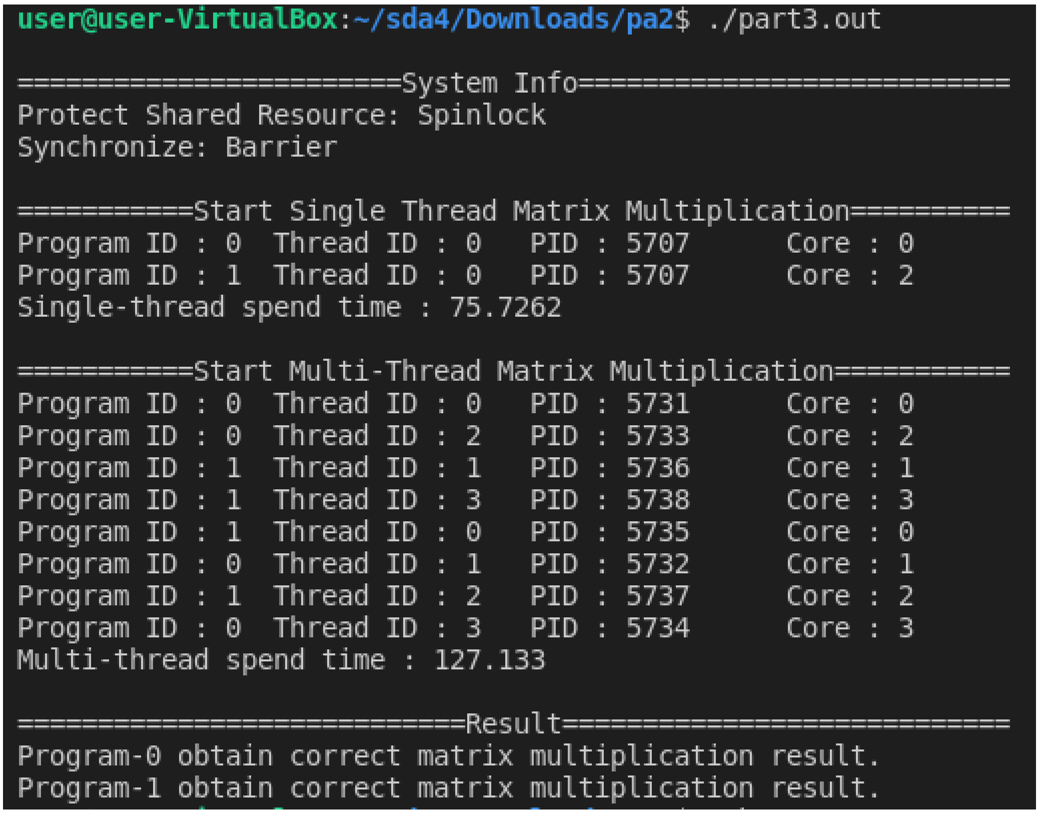
MULTI_TIME: 設定要做幾次矩陣運算。

PROTECT_SHARED_RESOURCE: 設定要使用 Mutex 還是 Spinlock 保護共用變數。

SYMCHRONIZE:設定要使用 Barrier 還是 Semaphore 同步 Thread。



```
user@user-VirtualBox:~/sda4/Downloads/pa2$ ./part1.out

========================System Info===========================
Protect Shared Resource: Mutex
Synchronize: Barrier

===========Start Single Thread Matrix Multiplication==========
Program ID : 0  Thread ID : 0   PID : 8927      Core : 1
Program ID : 1  Thread ID : 0   PID : 8927      Core : 1
Single-thread spend time : 79.6858

===========Start Multi-Thread Matrix Multiplication===========
Program ID : 0  Thread ID : 3   PID : 8950      Core : 3
Program ID : 1  Thread ID : 0   PID : 8951      Core : 0
Program ID : 0  Thread ID : 0   PID : 8947      Core : 0
Program ID : 0  Thread ID : 1   PID : 8948      Core : 1
Program ID : 1  Thread ID : 1   PID : 8952      Core : 1
Program ID : 0  Thread ID : 2   PID : 8949      Core : 2
Program ID : 1  Thread ID : 2   PID : 8953      Core : 2
Program ID : 1  Thread ID : 3   PID : 8954      Core : 3
Multi-thread spend time : 67.8261

============================Result============================
Program-0 obtain correct matrix multiplication result.
Program-1 obtain correct matrix multiplication result.
```



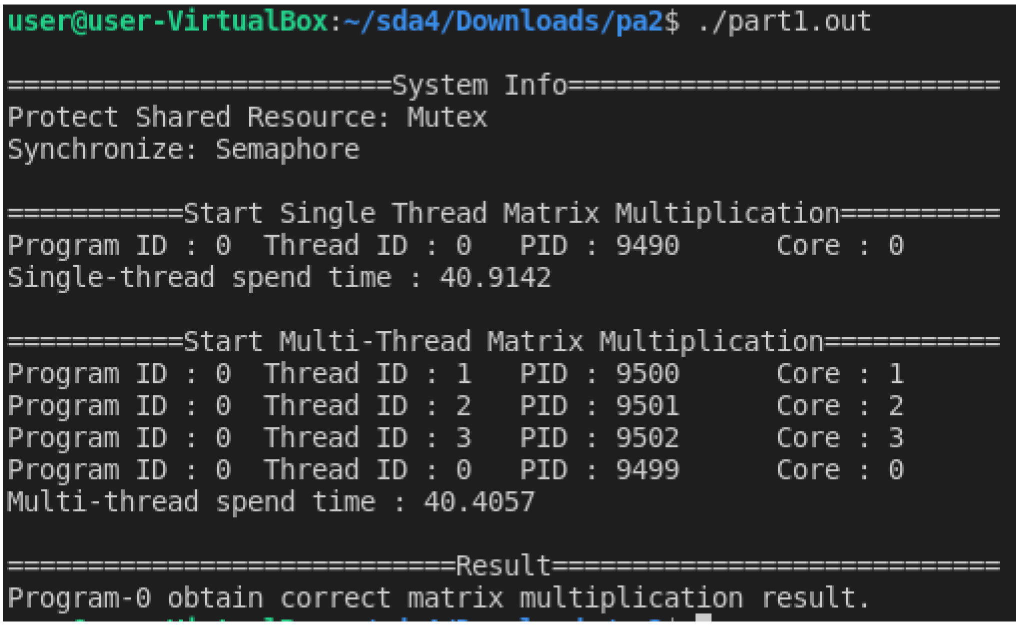
```
user@user-VirtualBox:~/sda4/Downloads/pa2$ ./part3.out

========================System Info===========================
Protect Shared Resource: Spinlock
Synchronize: Barrier

===========Start Single Thread Matrix Multiplication==========
Program ID : 0  Thread ID : 0   PID : 5707      Core : 0
Program ID : 1  Thread ID : 0   PID : 5707      Core : 2
Single-thread spend time : 75.7262

===========Start Multi-Thread Matrix Multiplication===========
Program ID : 0  Thread ID : 0   PID : 5731      Core : 0
Program ID : 0  Thread ID : 2   PID : 5733      Core : 2
Program ID : 1  Thread ID : 1   PID : 5736      Core : 1
Program ID : 1  Thread ID : 3   PID : 5738      Core : 3
Program ID : 1  Thread ID : 0   PID : 5735      Core : 0
Program ID : 0  Thread ID : 1   PID : 5732      Core : 1
Program ID : 1  Thread ID : 2   PID : 5737      Core : 2
Program ID : 0  Thread ID : 3   PID : 5734      Core : 3
Multi-thread spend time : 127.133

============================Result============================
Program-0 obtain correct matrix multiplication result.
Program-1 obtain correct matrix multiplication result.
```

我將 PROGRAM_NUM 從 1 調成 2 使 Mutex 速度快於 Spinlock，

PROGRAM_NUM 會計算兩個矩陣，每個矩陣會有 THREAD_NUM 個

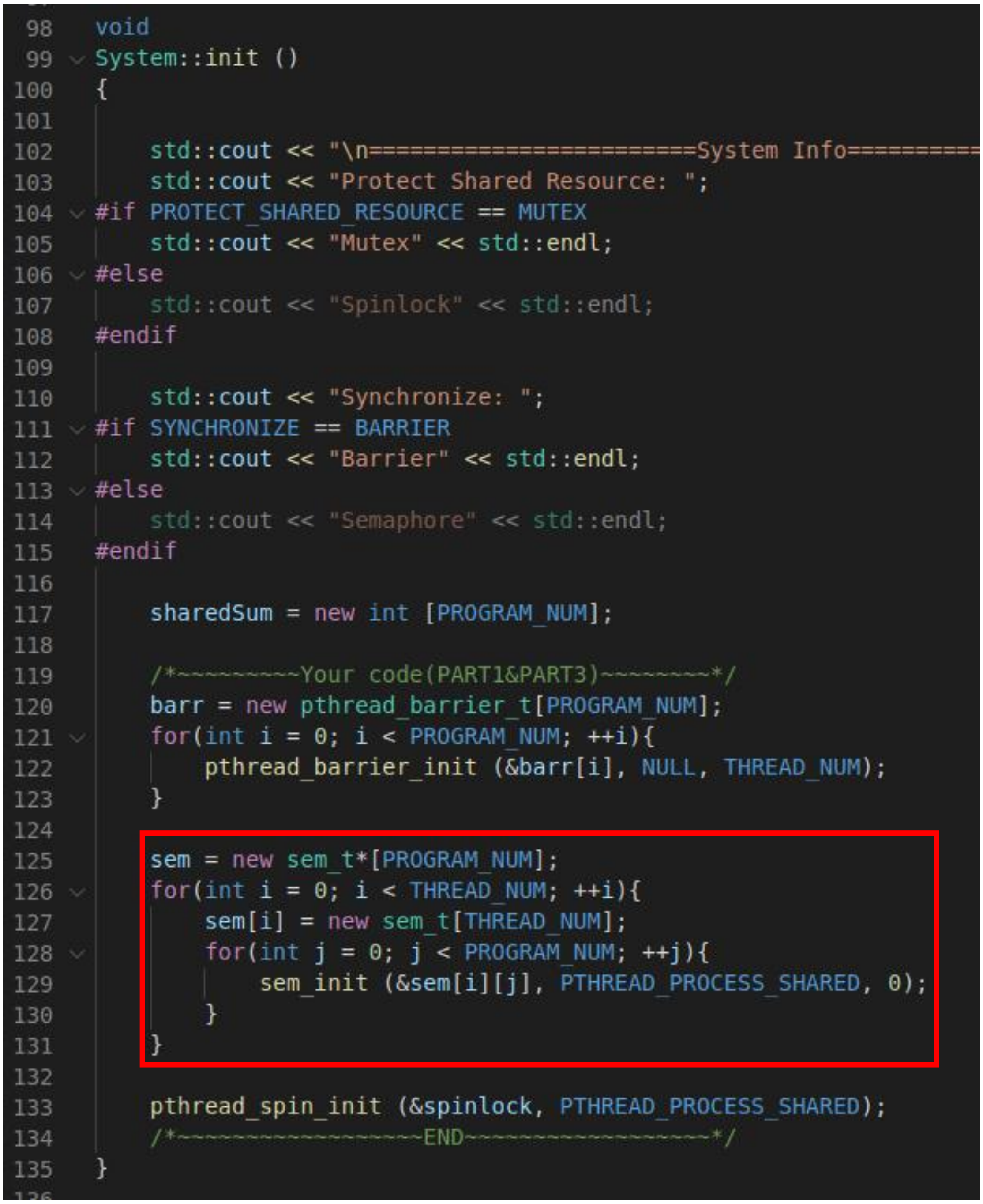
Thread 分工計算，而且會有 2 個同 ID 的 Thread 被綁在同一個 Core 上，所以當其中一個 Thread 被 Mutex 擋住時會切換到同個 Core 上的另外一個矩陣運算 Thread 執行，而 Spinlock 會直接 busy waiting 不會切換到另外一個 Thread 工作，所以 Mutex 會快於 Spinlock。

- Bonus Question(semaphore synchronize)
  - Execution result of using semaphore.



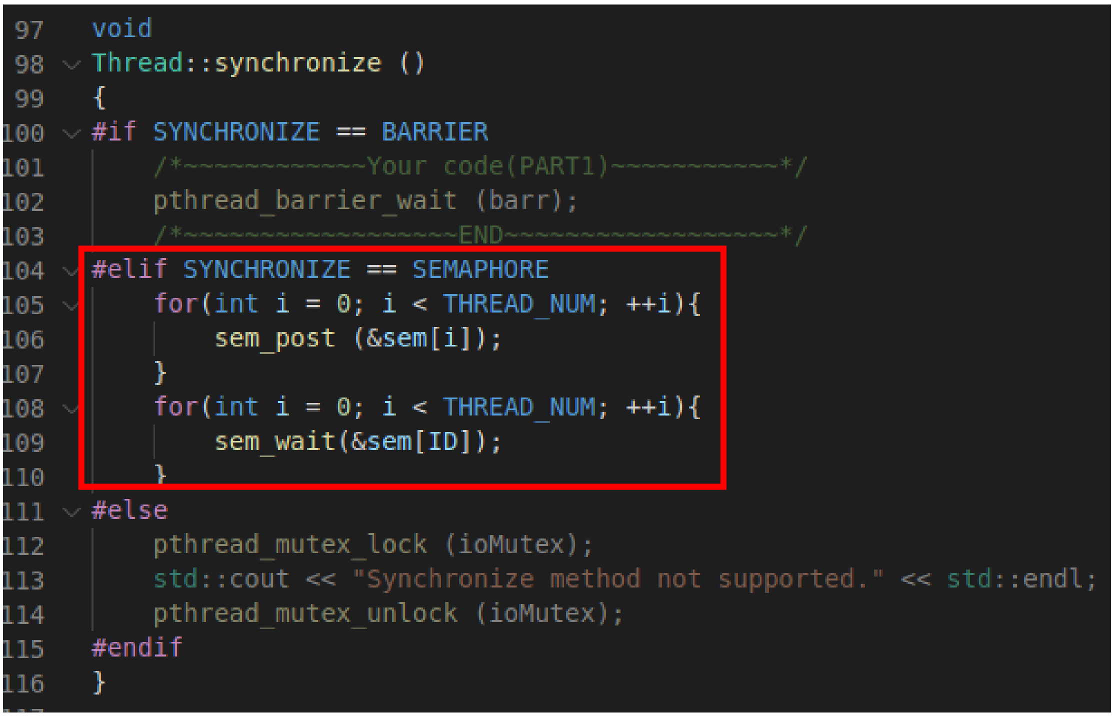
```
user@user-VirtualBox:~/sda4/Downloads/pa2$ ./part1.out

========================System Info===========================
Protect Shared Resource: Mutex
Synchronize: Semaphore

===========Start Single Thread Matrix Multiplication==========
Program ID : 0  Thread ID : 0   PID : 9490      Core : 0
Single-thread spend time : 40.9142

===========Start Multi-Thread Matrix Multiplication===========
Program ID : 0  Thread ID : 1   PID : 9500      Core : 1
Program ID : 0  Thread ID : 2   PID : 9501      Core : 2
Program ID : 0  Thread ID : 3   PID : 9502      Core : 3
Program ID : 0  Thread ID : 0   PID : 9499      Core : 0
Multi-thread spend time : 40.4057

============================Result============================
Program-0 obtain correct matrix multiplication result.
```

- Describe how to synchronize with semaphore



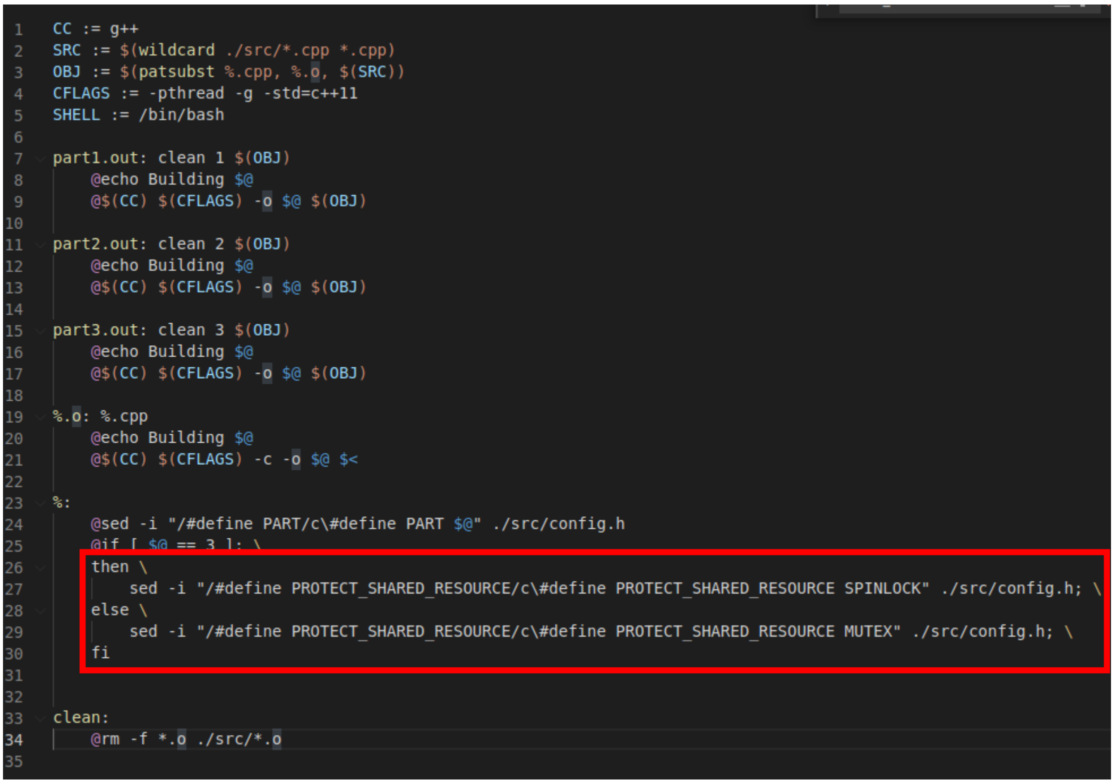
```
void
System::init ()
{

    std::cout << "\n=======================System Info=========
    std::cout << "Protect Shared Resource: ";
#if PROTECT_SHARED_RESOURCE == MUTEX
    std::cout << "Mutex" << std::endl;
#else
    std::cout << "Spinlock" << std::endl;
#endif

    std::cout << "Synchronize: ";
#if SYNCHRONIZE == BARRIER
    std::cout << "Barrier" << std::endl;
#else
    std::cout << "Semaphore" << std::endl;
#endif

    sharedSum = new int [PROGRAM_NUM];

    /*~~~~~~~~~~Your code(PART1&PART3)~~~~~~~~*/
    barr = new pthread_barrier_t[PROGRAM_NUM];
    for(int i = 0; i < PROGRAM_NUM; ++i){
        pthread_barrier_init (&barr[i], NULL, THREAD_NUM);
    }

    sem = new sem_t*[PROGRAM_NUM];
    for(int i = 0; i < THREAD_NUM; ++i){
        sem[i] = new sem_t[THREAD_NUM];
        for(int j = 0; j < PROGRAM_NUM; ++j){
            sem_init (&sem[i][j], PTHREAD_PROCESS_SHARED, 0);
        }
    }

    pthread_spin_init (&spinlock, PTHREAD_PROCESS_SHARED);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
}
```

sem_t 會先創建出 PROGRAM_NUM*THREAD_NUM 個並將所有 sem_t 都初始化次數為 0。



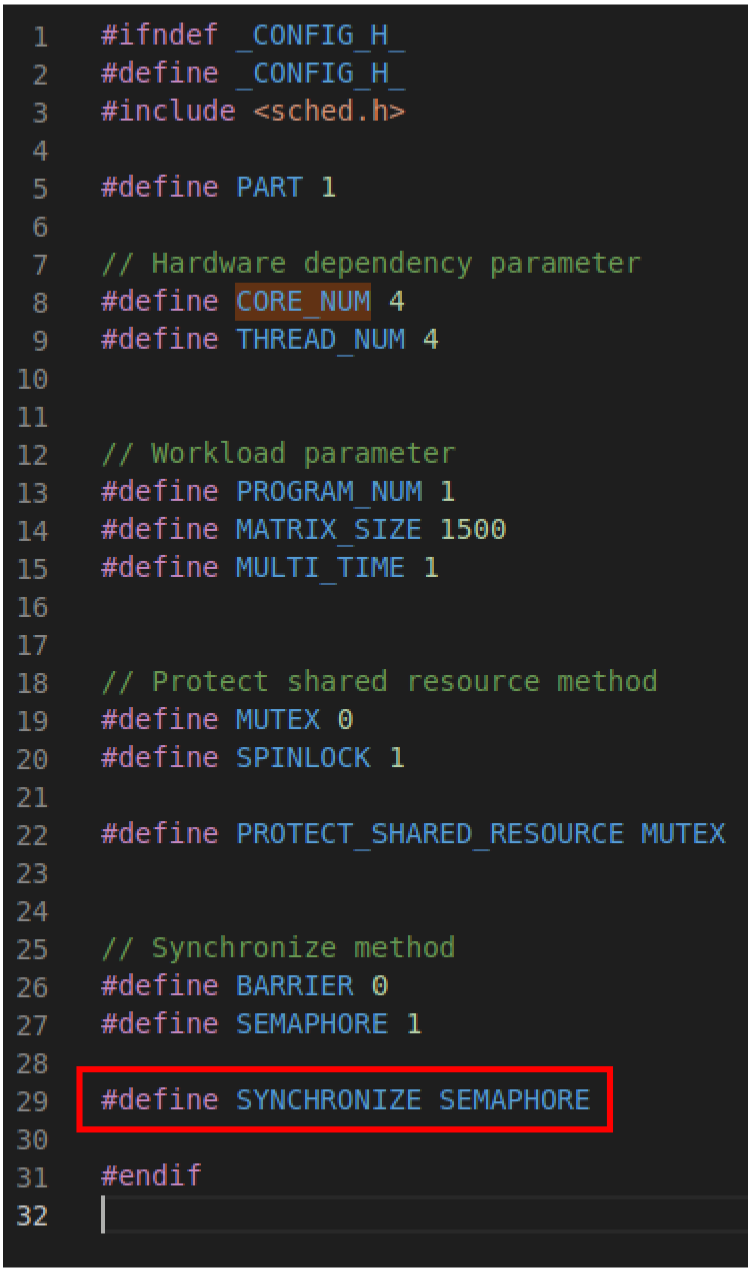
```
void
Thread::synchronize ()
{
#if SYNCHRONIZE == BARRIER
    /*~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
    pthread_barrier_wait (barr);
    /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#elif SYNCHRONIZE == SEMAPHORE
    for(int i = 0; i < THREAD_NUM; ++i){
        sem_post (&sem[i]);
    }
    for(int i = 0; i < THREAD_NUM; ++i){
        sem_wait(&sem[ID]);
    }
#else
    pthread_mutex_lock (ioMutex);
    std::cout << "Synchronize method not supported." << std::endl;
    pthread_mutex_unlock (ioMutex);
#endif
}
```

在 synchonize 內實現，當有某個 Thread 先抵達時就會先 post 給跟自己執行同個矩陣的 Thread 的 semaphore 一次，之後 wait 自己的 semaphore THREAD_NUM 次這樣就可以同步所有執行同個矩陣的 Thread。



```makefile
CC := g++
SRC := $(wildcard ./src/*.cpp *.cpp)
OBJ := $(patsubst %.cpp, %.o, $(SRC))
CFLAGS := -pthread -g -std=c++11
SHELL := /bin/bash

part1.out: clean 1 $(OBJ)
	@echo Building $@
	@$(CC) $(CFLAGS) -o $@ $(OBJ)

part2.out: clean 2 $(OBJ)
	@echo Building $@
	@$(CC) $(CFLAGS) -o $@ $(OBJ)

part3.out: clean 3 $(OBJ)
	@echo Building $@
	@$(CC) $(CFLAGS) -o $@ $(OBJ)

%.o: %.cpp
	@echo Building $@
	@$(CC) $(CFLAGS) -c -o $@ $<

%:
	@sed -i "/#define PART/c\#define PART $@" ./src/config.h
	@if [ $@ == 3 ]; \
	then \
		sed -i "/#define PROTECT_SHARED_RESOURCE/c\#define PROTECT_SHARED_RESOURCE SPINLOCK" ./src/config.h; \
	else \
		sed -i "/#define PROTECT_SHARED_RESOURCE/c\#define PROTECT_SHARED_RESOURCE MUTEX" ./src/config.h; \
	fi


clean:
	@rm -f *.o ./src/*.o
```

因為 make 會將 config.h 內的 SYNCHRONIZE 都改成 Barrier 所以將 makefile 中的兩行刪掉。



```
#ifndef _CONFIG_H_
#define _CONFIG_H_
#include <sched.h>

#define PART 1

// Hardware dependency parameter
#define CORE_NUM 4
#define THREAD_NUM 4


// Workload parameter
#define PROGRAM_NUM 1
#define MATRIX_SIZE 1500
#define MULTI_TIME 1


// Protect shared resource method
#define MUTEX 0
#define SPINLOCK 1

#define PROTECT_SHARED_RESOURCE MUTEX


// Synchronize method
#define BARRIER 0
#define SEMAPHORE 1

#define SYNCHRONIZE SEMAPHORE

#endif

```

要使用 Semaphore 同步時只要將 config.h 裡的 SYNCHRONIZE 改成 SEMAPHORE 即可。