嵌入式系統軟體設計

Embedded System Software Design Project 2

教授：陳雅淑

學號：M11007328

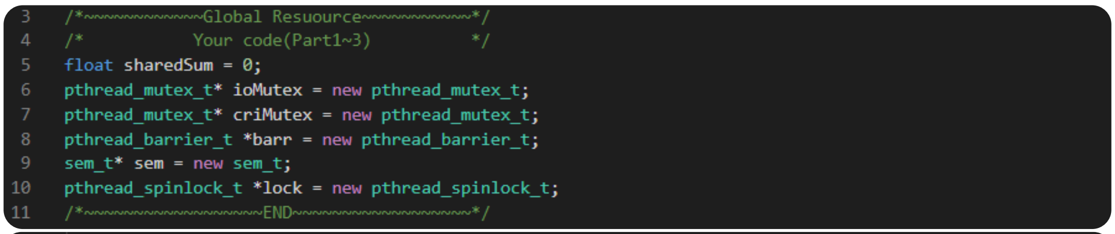
姓名：陳柏華

Part1 (Mutex and Barrier):

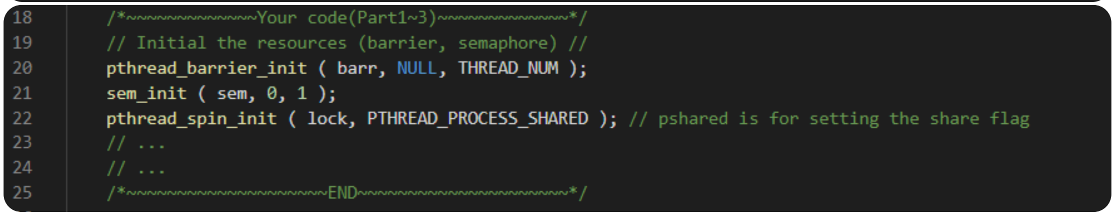
In part 1, to protect the shared resource by using a mutex, and to synchronize threads by using a barrier in order to obtain the correct result.

a) Describe how to protect share resources and how to synchronize threads in your report.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ system.cpp~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

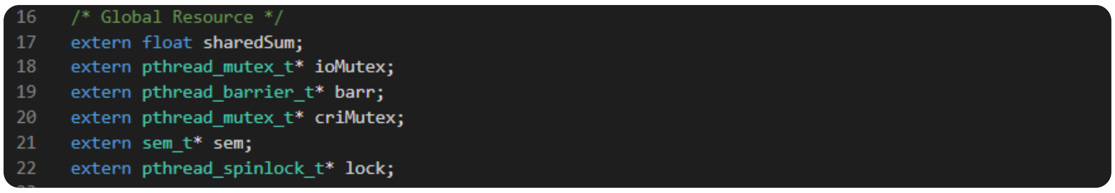


```
/*~~~~~~~~~~~~~Global Resuource~~~~~~~~~~~~*/
/*           Your code(Part1~3)          */
float sharedSum = 0;
pthread_mutex_t* ioMutex = new pthread_mutex_t;
pthread_mutex_t* criMutex = new pthread_mutex_t;
pthread_barrier_t *barr = new pthread_barrier_t;
sem_t* sem = new sem_t;
pthread_spinlock_t *lock = new pthread_spinlock_t;
/*~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~*/
```



```
    /*~~~~~~~~~~~~~~Your code(Part1~3)~~~~~~~~~~~~~~*/
    // Initial the resources (barrier, semaphore) //
    pthread_barrier_init ( barr, NULL, THREAD_NUM );
    sem_init ( sem, 0, 1 );
    pthread_spin_init ( lock, PTHREAD_PROCESS_SHARED ); // pshared is for setting the share flag
    // ...
    // ...
    /*~~~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~~~*/
```

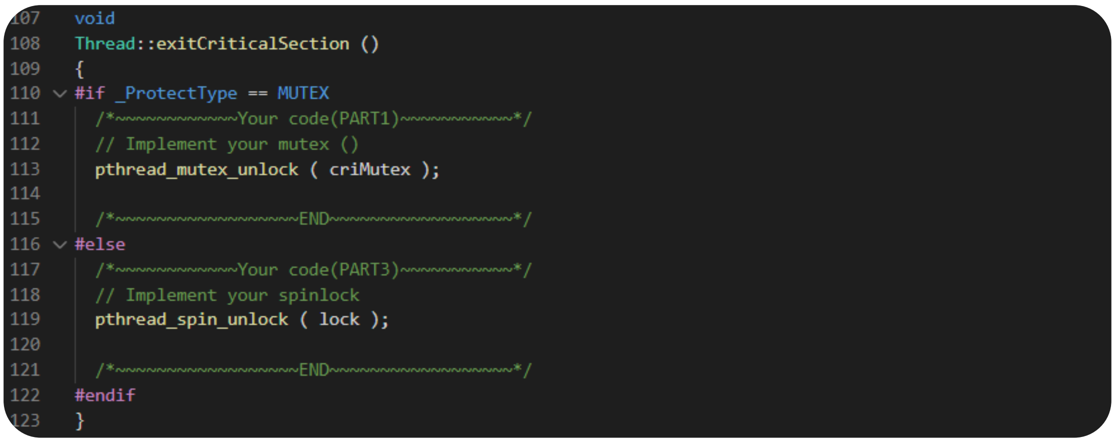
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ thread.h~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~



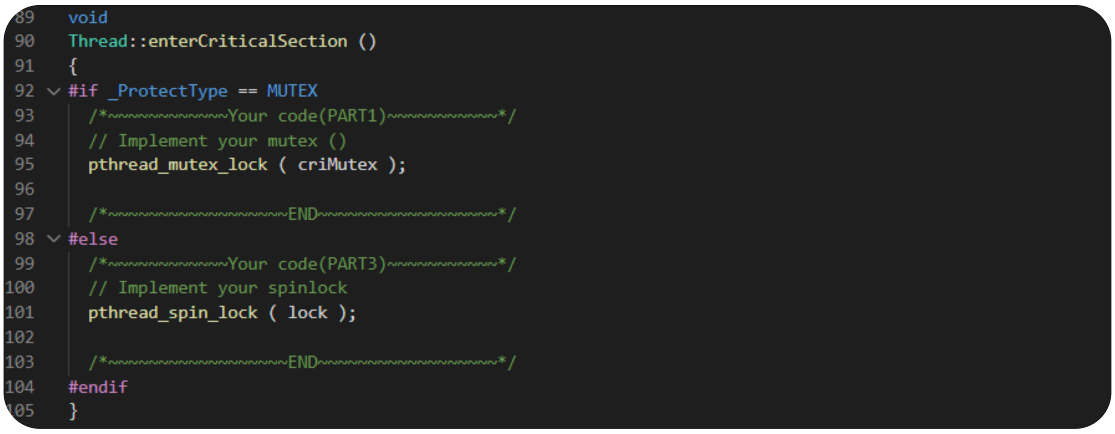
```
/* Global Resource */
extern float sharedSum;
extern pthread_mutex_t* ioMutex;
extern pthread_barrier_t* barr;
extern pthread_mutex_t* criMutex;
extern sem_t* sem;
extern pthread_spinlock_t* lock;
```

首先在 system.cpp 中完成 Mutex 以及 Barrier 的初始設定，Mutex 不需要事先定義參數所以在第二張圖片中沒有定義，在 Barrier 的參數設定中給定當前使用的 Thread 數量，代表每一次執行 Barrier 都需要等所有 Thread 抵達才能執行。此外也需要至 thread.h 設定完成才能夠正常使用 Mutex 以及 Barrier。

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ thread.cpp~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

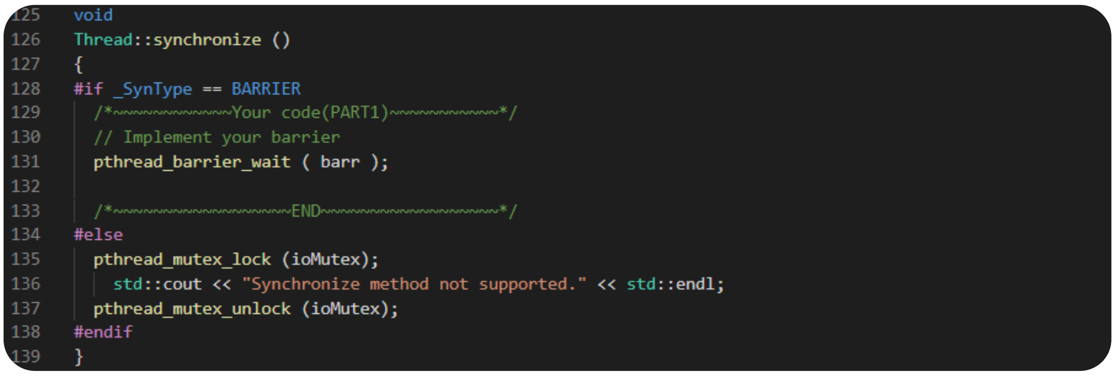


```
void
Thread::exitCriticalSection ()
{
#if _ProtectType == MUTEX
    /*~~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~~~*/
    // Implement your mutex ()
    pthread_mutex_unlock ( criMutex );

    /*~~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~~*/
#else
    /*~~~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~~~~*/
    // Implement your spinlock
    pthread_spin_unlock ( lock );

    /*~~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~~*/
#endif
}
```



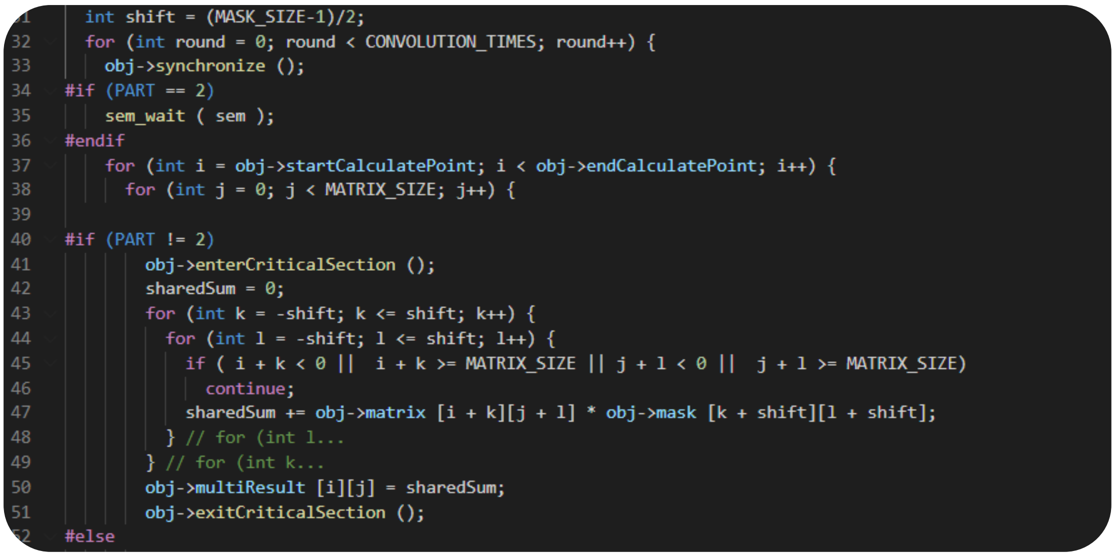
```
void
Thread::enterCriticalSection ()
{
#if _ProtectType == MUTEX
    /*~~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~~~*/
    // Implement your mutex ()
    pthread_mutex_lock ( criMutex );

    /*~~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~~*/
#else
    /*~~~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~~~~*/
    // Implement your spinlock
    pthread_spin_lock ( lock );

    /*~~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~~*/
#endif
}
```

接著是 CriticalSection 的設定，在 thread.cpp 中定義這兩個 Function 分別代表進入 CriticalSection 以及離開 CriticalSection。在進入 CriticalSection 時需要透過 Mutex 鎖起來以確保共用的資源不會被其他 Thread 搶佔，一次只讓一個 Thread 執行，離開 CriticalSection 時需要解鎖 Mutex，其他就緒的 Thread 才能進到 CriticalSection 使用共用資源。

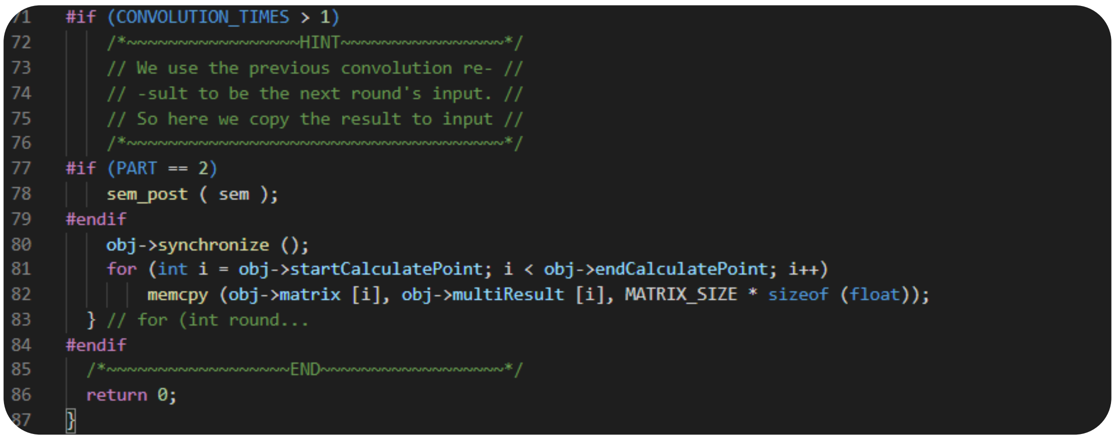


```
void
Thread::synchronize ()
{
#if _SynType == BARRIER
    /*~~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~~~*/
    // Implement your barrier
    pthread_barrier_wait ( barr );

    /*~~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~~*/
#else
    pthread_mutex_lock (ioMutex);
        std::cout << "Synchronize method not supported." << std::endl;
    pthread_mutex_unlock (ioMutex);
#endif
}
```

這邊是 synchronize 的設定，當每一次呼叫 synchronize 時都需要等到所有設定的 Thread 抵達才能繼續往下執行。
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~



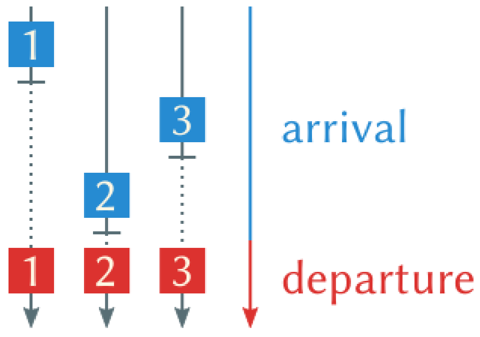
```
31      int shift = (MASK_SIZE-1)/2;
32      for (int round = 0; round < CONVOLUTION_TIMES; round++) {
33          obj->synchronize ();
34  #if (PART == 2)
35          sem_wait ( sem );
36  #endif
37          for (int i = obj->startCalculatePoint; i < obj->endCalculatePoint; i++) {
38              for (int j = 0; j < MATRIX_SIZE; j++) {
39
40  #if (PART != 2)
41                  obj->enterCriticalSection ();
42                  sharedSum = 0;
43                  for (int k = -shift; k <= shift; k++) {
44                      for (int l = -shift; l <= shift; l++) {
45                          if ( i + k < 0 ||  i + k >= MATRIX_SIZE || j + l < 0 ||  j + l >= MATRIX_SIZE)
46                              continue;
47                          sharedSum += obj->matrix [i + k][j + l] * obj->mask [k + shift][l + shift];
48                      } // for (int l...
49                  } // for (int k...
50                  obj->multiResult [i][j] = sharedSum;
51                  obj->exitCriticalSection ();
52  #else
```



```
71  #if (CONVOLUTION_TIMES > 1)
72          /*~~~~~~~~~~~~~~~~~~~~HINT~~~~~~~~~~~~~~~~~~~*/
73          // We use the previous convolution re- //
74          // -sult to be the next round's input. //
75          // So here we copy the result to input //
76          /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
77  #if (PART == 2)
78          sem_post ( sem );
79  #endif
80          obj->synchronize ();
81          for (int i = obj->startCalculatePoint; i < obj->endCalculatePoint; i++)
82              memcpy (obj->matrix [i], obj->multiResult [i], MATRIX_SIZE * sizeof (float));
83      } // for (int round...
84  #endif
85      /*~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~*/
86      return 0;
87  }
```

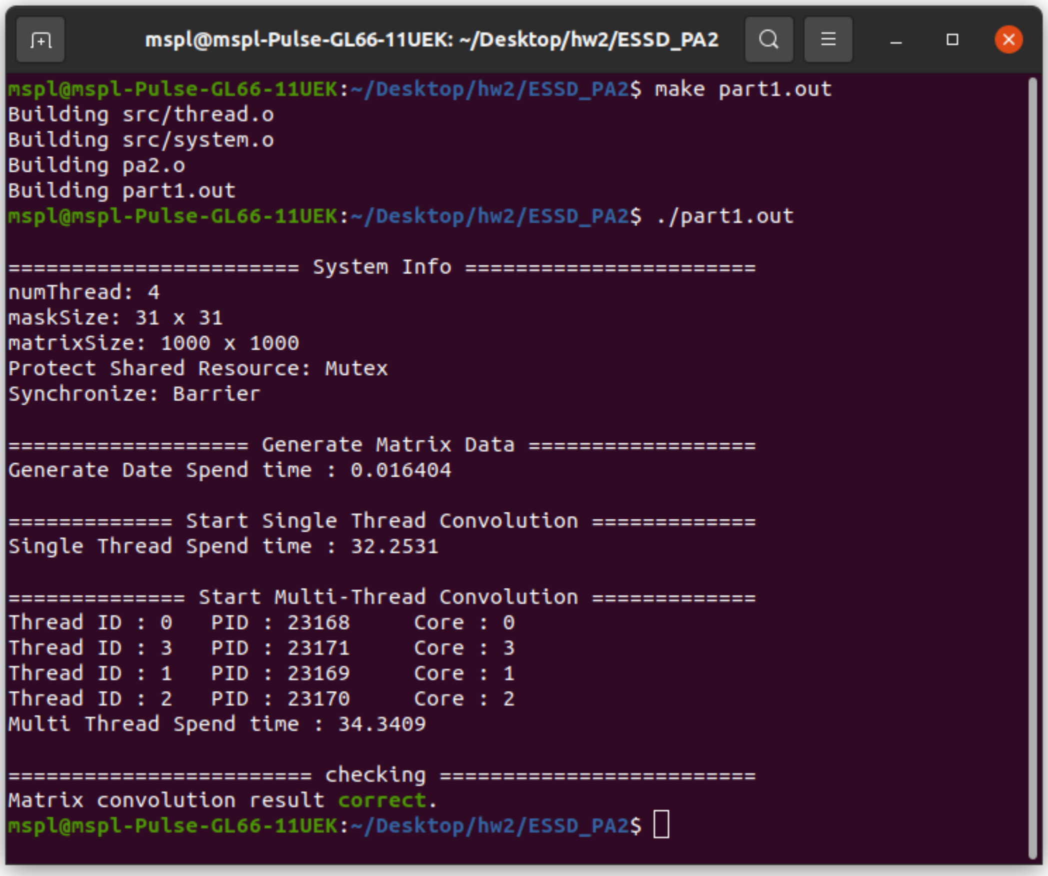
在 convolution 的地方我主要是將 CriticalSection 界定在 41 行 ~ 51 行，主要原因為，在這個範圍內有使用到全域變數 sharedSum 而且是主要運算的地方，加上之後可以確保 sharedSum 在累加過程中不會被其他趕來的 Thread 干擾，並且當 sharedSum 將累加完成的數值輸出給 mutiResult 後才算結束 CriticalSection。

Synchronize 的部分我使用 barrier 加在第 33 行與 80 行，主要是確保每一次進入 convolution 時是在每個 Thread 都抵達的情況下才做，以及每一次將輸出值傳遞給下一輪時也都是在每個 Thread 都抵達的情況下才做。(下圖為 barrier 是意圖)



(網址: https://6xq.net/barrier-intro/)

b) Show the execution result in Figure 2 in your report.



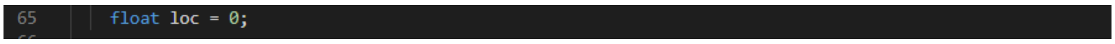
```
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/hw2/ESSD_PA2$ make part1.out
Building src/thread.o
Building src/system.o
Building pa2.o
Building part1.out
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/hw2/ESSD_PA2$ ./part1.out

======================= System Info =======================
numThread: 4
maskSize: 31 x 31
matrixSize: 1000 x 1000
Protect Shared Resource: Mutex
Synchronize: Barrier

=================== Generate Matrix Data ==================
Generate Date Spend time : 0.016404

============= Start Single Thread Convolution =============
Single Thread Spend time : 32.2531

============== Start Multi-Thread Convolution =============
Thread ID : 0   PID : 23168     Core : 0
Thread ID : 3   PID : 23171     Core : 3
Thread ID : 1   PID : 23169     Core : 1
Thread ID : 2   PID : 23170     Core : 2
Multi Thread Spend time : 34.3409

======================== checking =========================
Matrix convolution result correct.
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/hw2/ESSD_PA2$ 
```

Part 2 (Reentrant Function):

In part 2, please modify the non-reentrant function into the reentrant function.

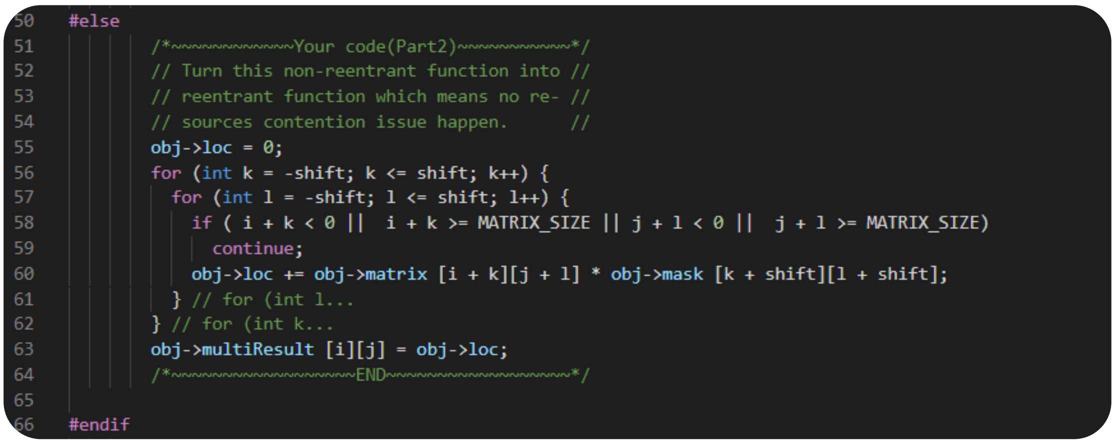
a) Describe how to modify the non-reentrant function to the reentrant function in your report.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ thread.h~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~



```
float loc = 0;
```

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ thread.cpp~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

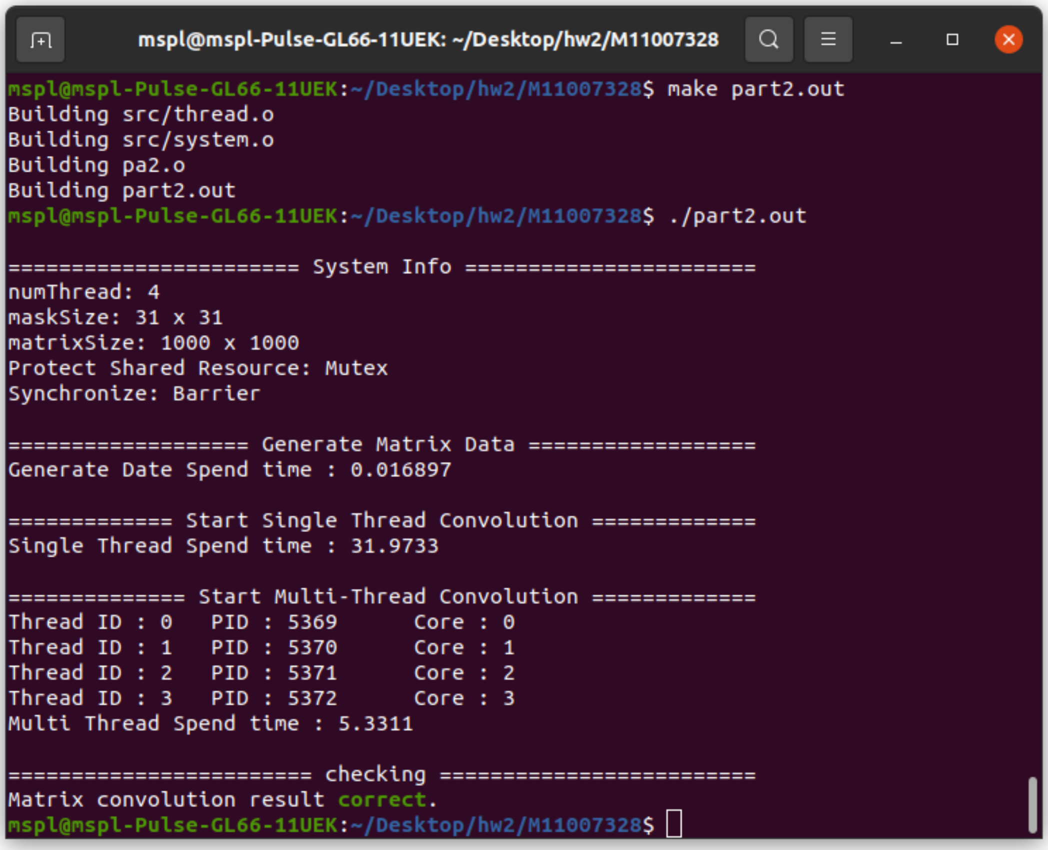


```
#else
            /*~~~~~~~~~~~~Your code(Part2)~~~~~~~~~~~*/
            // Turn this non-reentrant function into //
            // reentrant function which means no re- //
            // sources contention issue happen.      //
            obj->loc = 0;
            for (int k = -shift; k <= shift; k++) {
              for (int l = -shift; l <= shift; l++) {
                if ( i + k < 0 ||  i + k >= MATRIX_SIZE || j + l < 0 ||  j + l >= MATRIX_SIZE)
                  continue;
                obj->loc += obj->matrix [i + k][j + l] * obj->mask [k + shift][l + shift];
              } // for (int l...
            } // for (int k...
            obj->multiResult [i][j] = obj->loc;
            /*~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/

#endif
```

將 non-reentrant function 改成 reentrant function 可以使用 Local variable 將每一次 Thread 使用的累加變數 loc 都設在 Thread 上，這樣在使用 convolution 做運算時就不會產生因為是 Global variable 而互相搶佔的情況。

程式中使用的 Local variable 名為 loc 是作為累加使用，取代了原先的 Global variable 也就是 shareSum，因為這個區域變數在每個 Thread 上都有所以也不需要進行保護。

b) Show the execution result of multi-thread with reentrant function as Figure 3.



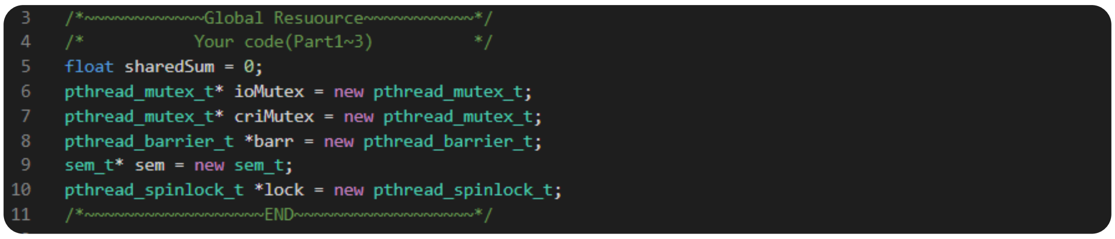
mspl@mspl-Pulse-GL66-11UEK: ~/Desktop/hw2/M11007328

```
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/hw2/M11007328$ make part2.out
Building src/thread.o
Building src/system.o
Building pa2.o
Building part2.out
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/hw2/M11007328$ ./part2.out

======================= System Info =======================
numThread: 4
maskSize: 31 x 31
matrixSize: 1000 x 1000
Protect Shared Resource: Mutex
Synchronize: Barrier

=================== Generate Matrix Data ==================
Generate Date Spend time : 0.016897

============= Start Single Thread Convolution =============
Single Thread Spend time : 31.9733

============== Start Multi-Thread Convolution =============
Thread ID : 0   PID : 5369     Core : 0
Thread ID : 1   PID : 5370     Core : 1
Thread ID : 2   PID : 5371     Core : 2
Thread ID : 3   PID : 5372     Core : 3
Multi Thread Spend time : 5.3311

======================== checking =========================
Matrix convolution result correct.
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/hw2/M11007328$ 
```

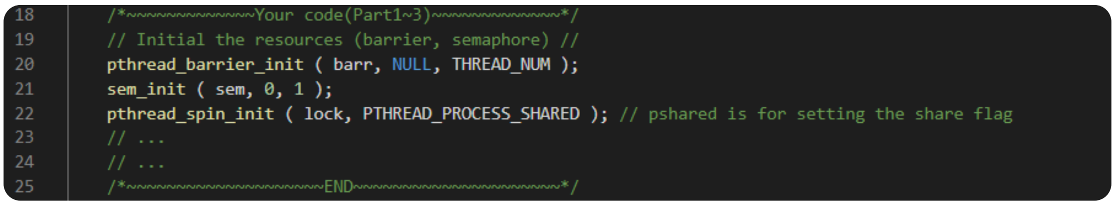
c) Analyze the execution time of the non-reentrant function and reentrant function, and compare them. (Please use your experimental result to support your discussion)

我使用的方法是用 binary semaphore 讓 non-reentrant function 可以顯示正確答案再進一步與 reentrant function 做比較。

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ system.cpp ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

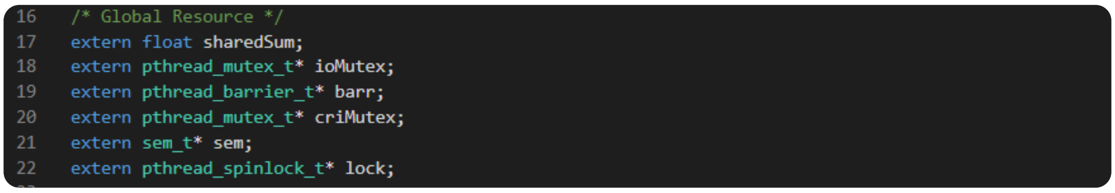


```
/*~~~~~~~~~~~~~Global Resuource~~~~~~~~~~~~*/
/*            Your code(Part1~3)          */
float sharedSum = 0;
pthread_mutex_t* ioMutex = new pthread_mutex_t;
pthread_mutex_t* criMutex = new pthread_mutex_t;
pthread_barrier_t *barr = new pthread_barrier_t;
sem_t* sem = new sem_t;
pthread_spinlock_t *lock = new pthread_spinlock_t;
/*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
```



```
    /*~~~~~~~~~~~~~~Your code(Part1~3)~~~~~~~~~~~~~~*/
    // Initial the resources (barrier, semaphore) //
    pthread_barrier_init ( barr, NULL, THREAD_NUM );
    sem_init ( sem, 0, 1 );
    pthread_spin_init ( lock, PTHREAD_PROCESS_SHARED ); // pshared is for setting the share flag
    // ...
    // ...
    /*~~~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~~~*/
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ thread.h~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

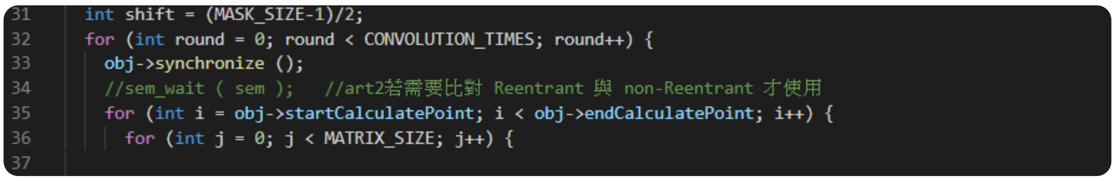


```
16  /* Global Resource */
17  extern float sharedSum;
18  extern pthread_mutex_t* ioMutex;
19  extern pthread_barrier_t* barr;
20  extern pthread_mutex_t* criMutex;
21  extern sem_t* sem;
22  extern pthread_spinlock_t* lock;
```

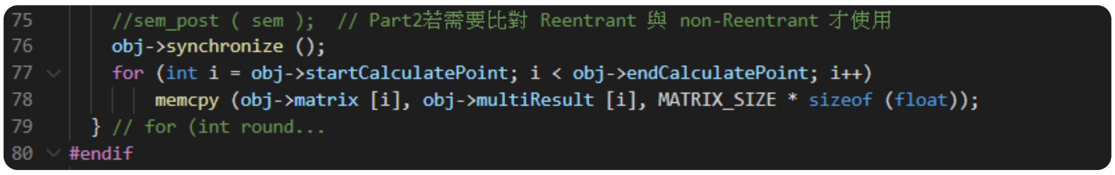
binary semaphore 值只能是 0 或 1，在邏輯上相當於一個 mutex(互斥鎖)。mutex 使用上與 binary semaphore 具有相同功能，但是 mutex 主要設計是防止兩個 process 同時間執行相同的一段 code 或存取同一資料，而 binary semaphore 設計上則是限制同時間存取同一資源，因此很適合用於這題。

首先我先在 system.cpp 定義 binary semaphore 的相關參數 pshared 設為 0 且 value 設為 1 因為是 binary semaphore。

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ thread.cpp~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

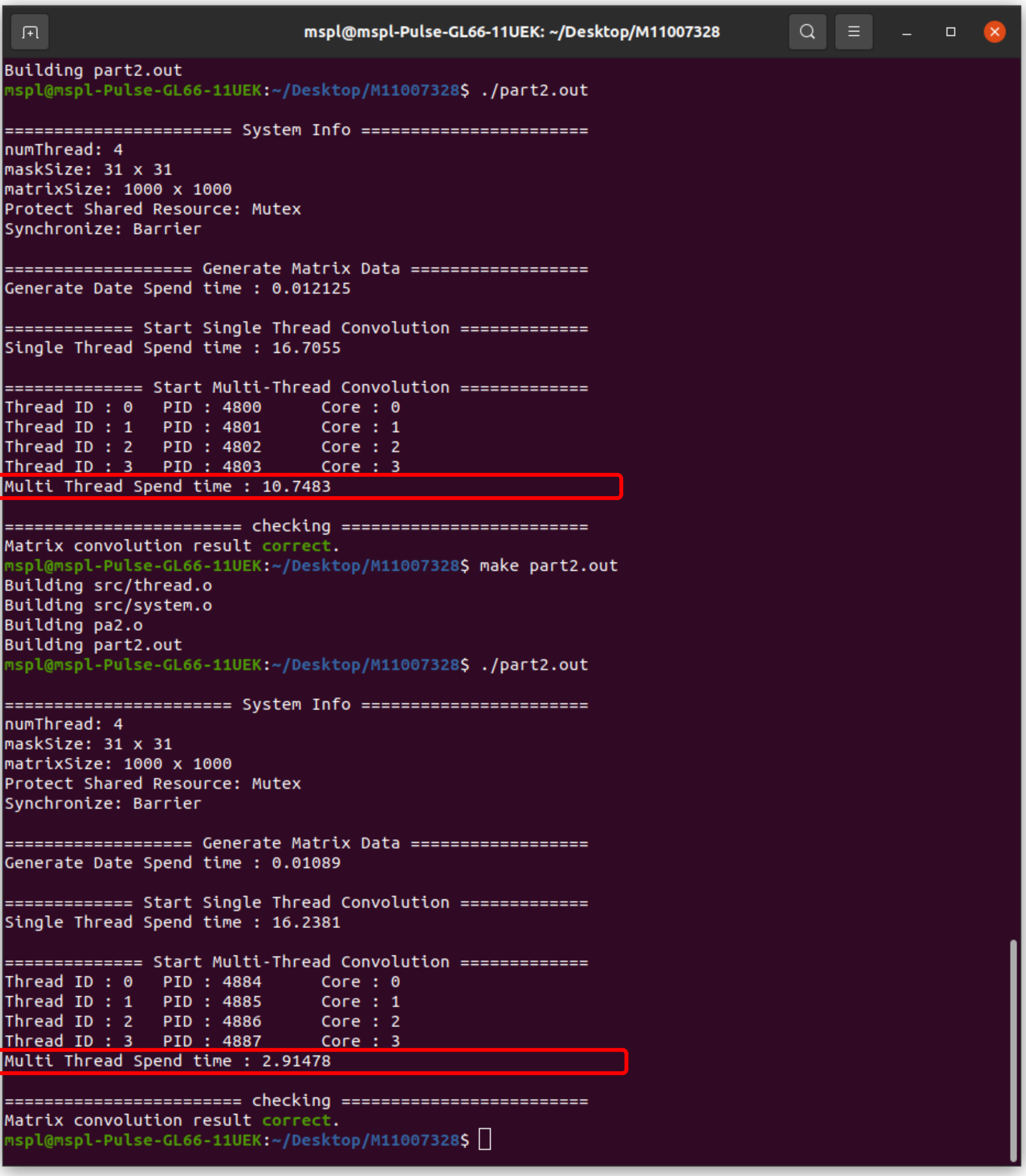


```
31  int shift = (MASK_SIZE-1)/2;
32  for (int round = 0; round < CONVOLUTION_TIMES; round++) {
33      obj->synchronize ();
34      //sem_wait ( sem );   //art2若需要比對 Reentrant 與 non-Reentrant 才使用
35      for (int i = obj->startCalculatePoint; i < obj->endCalculatePoint; i++) {
36          for (int j = 0; j < MATRIX_SIZE; j++) {
37
```



```
75          //sem_post ( sem );  // Part2若需要比對 Reentrant 與 non-Reentrant 才使用
76          obj->synchronize ();
77          for (int i = obj->startCalculatePoint; i < obj->endCalculatePoint; i++)
78              memcpy (obj->matrix [i], obj->multiResult [i], MATRIX_SIZE * sizeof (float));
79      } // for (int round...
80  #endif
```

這部分是設定的方法，首先我將 binary semaphore 的 sem_wait 設置在每一次新的 convolution 開始的地方也就是 35 行並且用 #if 以及 #endif 區隔其他題目，以及在每一次存取 convolution 結果給下一輪做使用之前也會執行 binary semaphore 的 sem_post。sem_wait 會將當前 binary semaphore 的值減 1 並且讓要求通過的 Thread 順利通過，Thread 通過後 sem_wait 會回歸到 0 並等待下一次 binary semaphore 值等於 1 的時候才讓等待的 Thread 通行，而 binary semaphore 要加 1 需要倚靠 sem_post 他能夠讓 binary semaphore 的當前數值加 1。



```
Building part2.out
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/M11007328$ ./part2.out

====================== System Info ======================
numThread: 4
maskSize: 31 x 31
matrixSize: 1000 x 1000
Protect Shared Resource: Mutex
Synchronize: Barrier

================== Generate Matrix Data ==================
Generate Date Spend time : 0.012125

============ Start Single Thread Convolution ============
Single Thread Spend time : 16.7055

============= Start Multi-Thread Convolution ============
Thread ID : 0   PID : 4800      Core : 0
Thread ID : 1   PID : 4801      Core : 1
Thread ID : 2   PID : 4802      Core : 2
Thread ID : 3   PID : 4803      Core : 3
Multi Thread Spend time : 10.7483

======================= checking =======================
Matrix convolution result correct.
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/M11007328$ make part2.out
Building src/thread.o
Building src/system.o
Building pa2.o
Building part2.out
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/M11007328$ ./part2.out

====================== System Info ======================
numThread: 4
maskSize: 31 x 31
matrixSize: 1000 x 1000
Protect Shared Resource: Mutex
Synchronize: Barrier

================== Generate Matrix Data ==================
Generate Date Spend time : 0.01089

============ Start Single Thread Convolution ============
Single Thread Spend time : 16.2381

============= Start Multi-Thread Convolution ============
Thread ID : 0   PID : 4884      Core : 0
Thread ID : 1   PID : 4885      Core : 1
Thread ID : 2   PID : 4886      Core : 2
Thread ID : 3   PID : 4887      Core : 3
Multi Thread Spend time : 2.91478

======================= checking =======================
Matrix convolution result correct.
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/M11007328$
```

	Reentrant	**Non-reentrant**
Spend time (s)	2.91478	10.7483

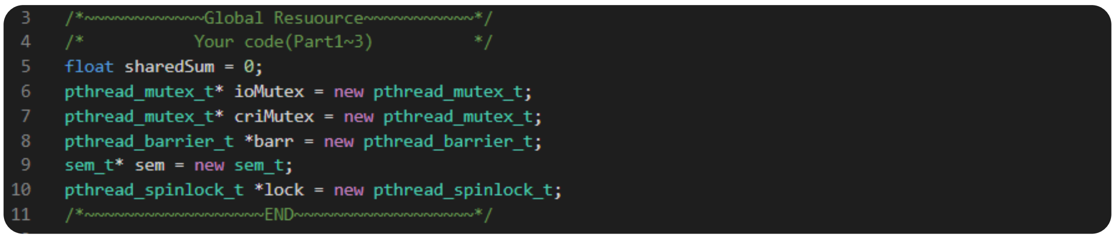
從結果可以看出 Reentrant 在速度上可以比 non-Reentrant 快好幾倍，可能的原因是 Reentrant 的方法因為採用了 Local variable 的方式做運算所以可以支援多 Thread 一起執行達到平行處理資料，但是 non-Reentrant 因為在運算過程中人須保護裡面的資料所以一次只能接受一個 Thread 處理，所以速度上會相較於 Reentrant 的方式慢。

Part3 (Spinlock):

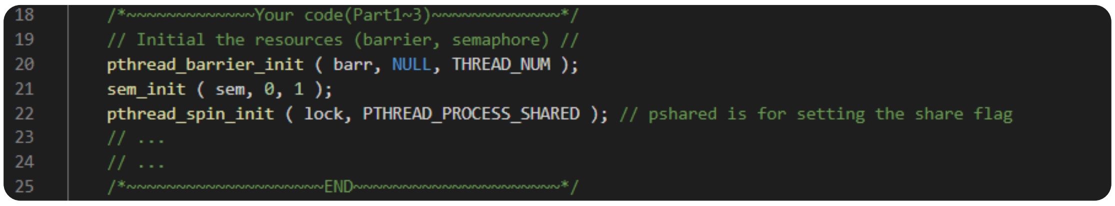
In part 3, try to protect the shared resource by using spinlock and synchronize the thread by using a barrier in order to obtain the correct result. Please modify the code based on part 1.

a) Describe how to protect the shared resource by using spinlock.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ system.cpp ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

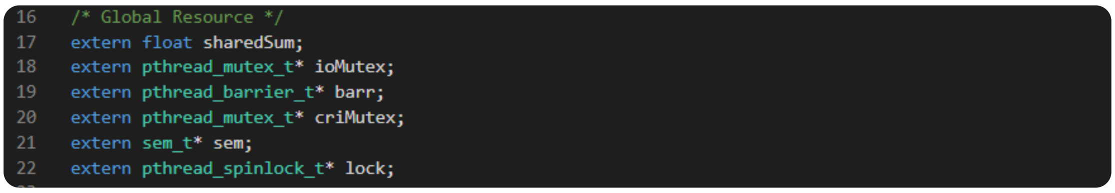


```
/*~~~~~~~~~~~~~Global Resuource~~~~~~~~~~~~*/
/*            Your code(Part1~3)           */
float sharedSum = 0;
pthread_mutex_t* ioMutex = new pthread_mutex_t;
pthread_mutex_t* criMutex = new pthread_mutex_t;
pthread_barrier_t *barr = new pthread_barrier_t;
sem_t* sem = new sem_t;
pthread_spinlock_t *lock = new pthread_spinlock_t;
/*~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~*/
```



```
    /*~~~~~~~~~~~~~~Your code(Part1~3)~~~~~~~~~~~~~~*/
    // Initial the resources (barrier, semaphore) //
    pthread_barrier_init ( barr, NULL, THREAD_NUM );
    sem_init ( sem, 0, 1 );
    pthread_spin_init ( lock, PTHREAD_PROCESS_SHARED ); // pshared is for setting the share flag
    // ...
    // ...
    /*~~~~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~~~~~*/
```

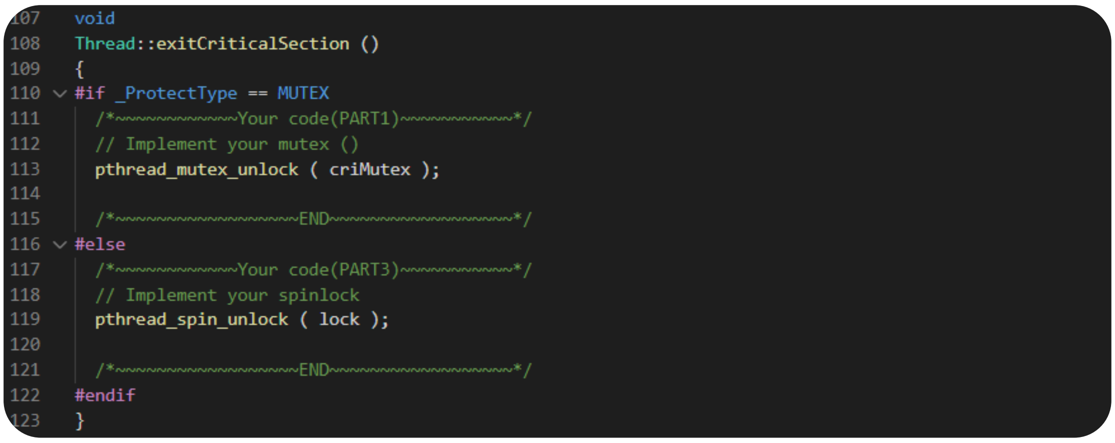
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ thread.h~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~



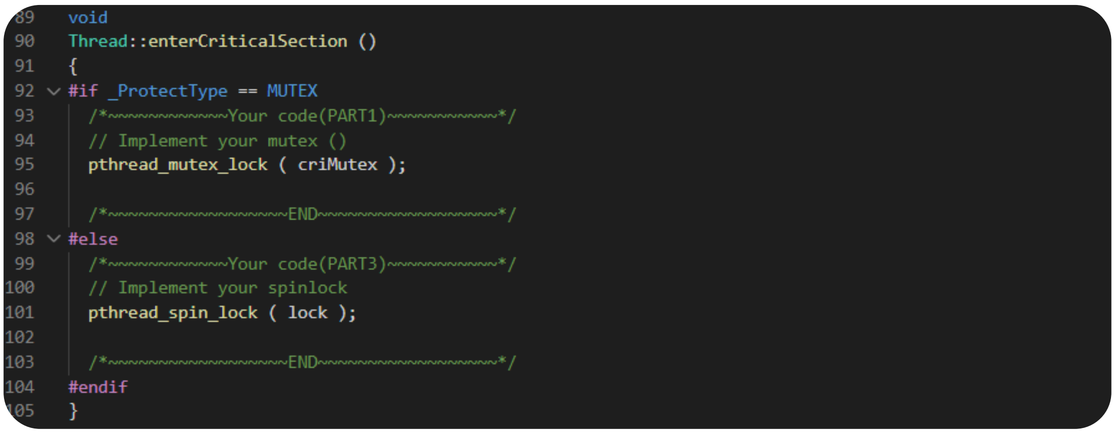
```
/* Global Resource */
extern float sharedSum;
extern pthread_mutex_t* ioMutex;
extern pthread_barrier_t* barr;
extern pthread_mutex_t* criMutex;
extern sem_t* sem;
extern pthread_spinlock_t* lock;
```

Spinlock 的設置跟前面提到的幾種方法很相似，這部分的宣告與上述相同，pshared 的參數我設為 *PTHREAD_PROCESS_SHARED*。

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ thread.cpp~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~



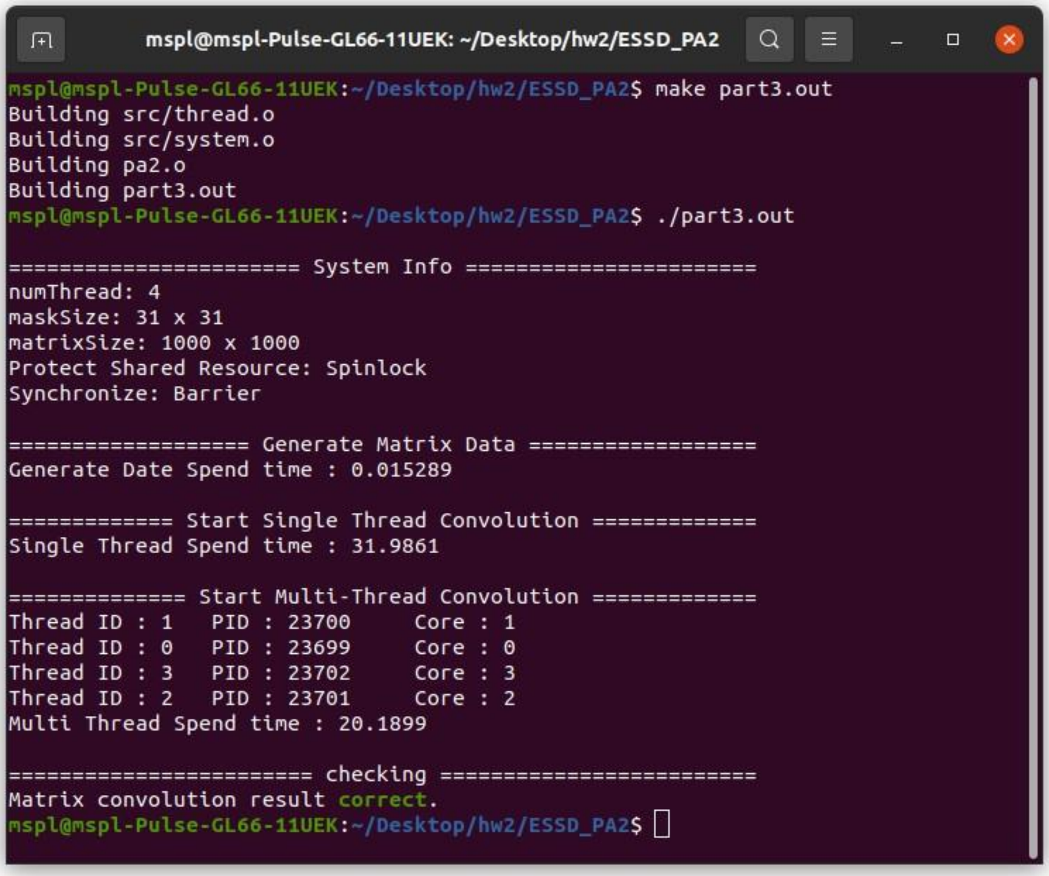
```
void
Thread::exitCriticalSection ()
{
#if _ProtectType == MUTEX
    /*~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
    // Implement your mutex ()
    pthread_mutex_unlock ( criMutex );

    /*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#else
    /*~~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~~*/
    // Implement your spinlock
    pthread_spin_unlock ( lock );

    /*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#endif
}
```



```
void
Thread::enterCriticalSection ()
{
#if _ProtectType == MUTEX
    /*~~~~~~~~~~~~~Your code(PART1)~~~~~~~~~~~*/
    // Implement your mutex ()
    pthread_mutex_lock ( criMutex );

    /*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#else
    /*~~~~~~~~~~~~~Your code(PART3)~~~~~~~~~~~*/
    // Implement your spinlock
    pthread_spin_lock ( lock );

    /*~~~~~~~~~~~~~~~~~~~END~~~~~~~~~~~~~~~~~~*/
#endif
}
```

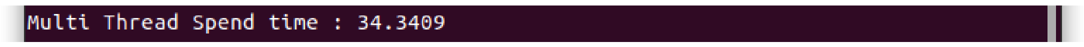
這邊設定的方法我一樣設定在 CriticalSaction 中，進入 CriticalSaction 時呼叫 lock 離開時呼叫 unlock，這樣的方法可以保護資料不會被其他插隊進來的 Thread 搶佔，優缺點在後面會再提到。
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

b) Show the execution result in Figure 4.

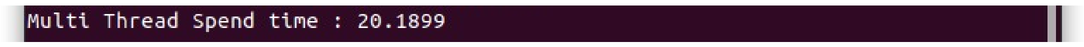


```
mspl@mspl-Pulse-GL66-11UEK: ~/Desktop/hw2/ESSD_PA2
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/hw2/ESSD_PA2$ make part3.out
Building src/thread.o
Building src/system.o
Building pa2.o
Building part3.out
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/hw2/ESSD_PA2$ ./part3.out

======================= System Info =======================
numThread: 4
maskSize: 31 x 31
matrixSize: 1000 x 1000
Protect Shared Resource: Spinlock
Synchronize: Barrier

=================== Generate Matrix Data ==================
Generate Date Spend time : 0.015289

============ Start Single Thread Convolution =============
Single Thread Spend time : 31.9861

============= Start Multi-Thread Convolution =============
Thread ID : 1   PID : 23700     Core : 1
Thread ID : 0   PID : 23699     Core : 0
Thread ID : 3   PID : 23702     Core : 3
Thread ID : 2   PID : 23701     Core : 2
Multi Thread Spend time : 20.1899

======================= checking ==========================
Matrix convolution result correct.
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/hw2/ESSD_PA2$
```

c) Compare to part 1, please observe which method could obtain better performance under the benchmark we provided and explain why. Please use the execution results to support your discussion. (Show both the execution results of using mutex and spinlock respectively to support your discussion.)



```
Multi Thread Spend time : 34.3409
```



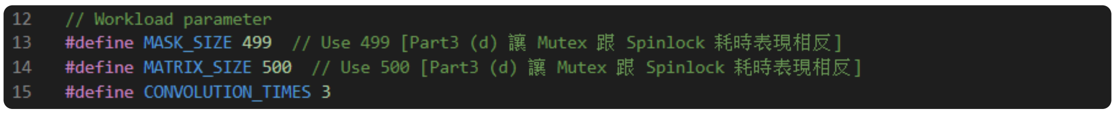
```
Multi Thread Spend time : 20.1899
```

	Mutex	**Spinlock**
Spend time (s)	34.3409	20.1899

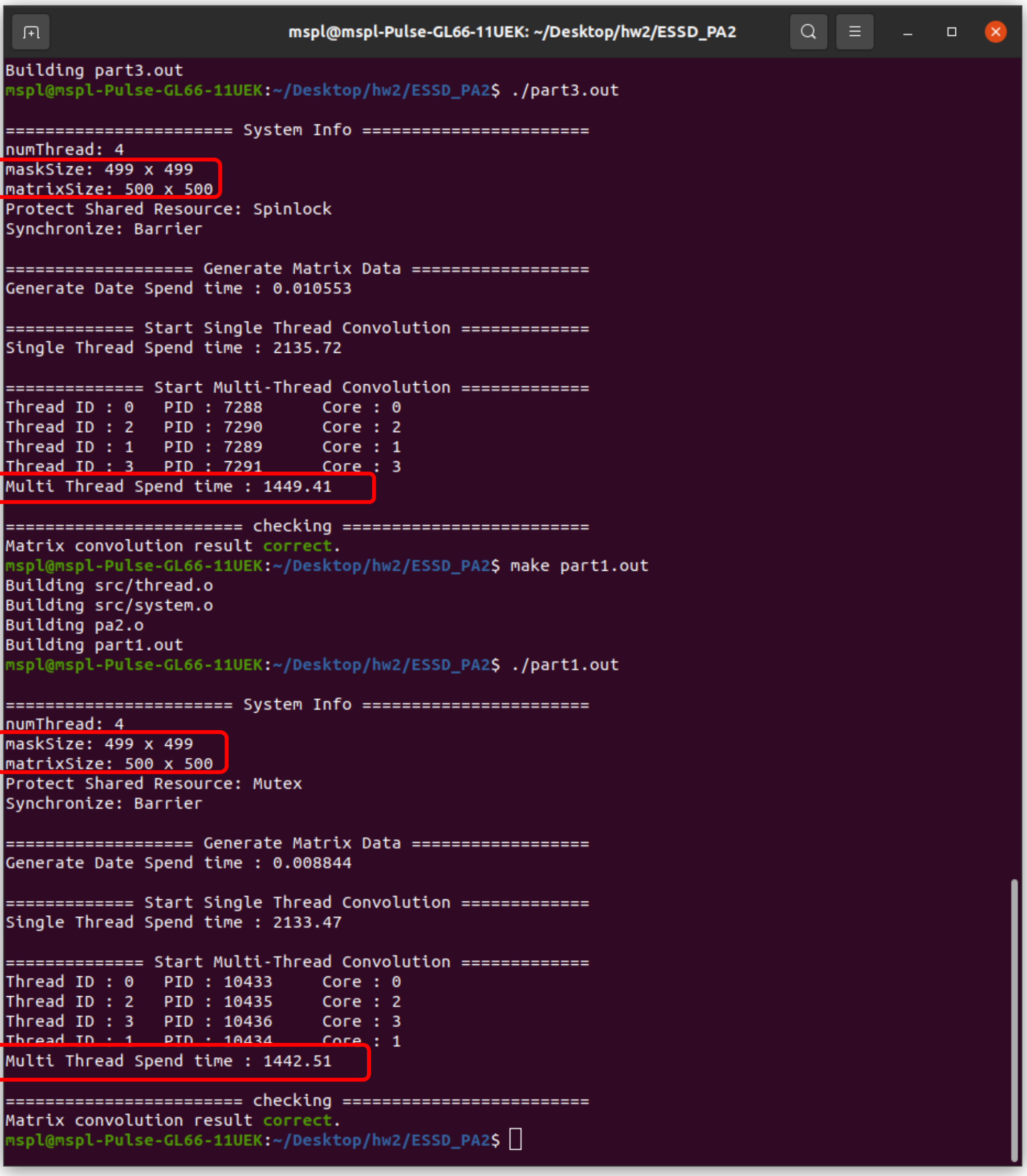
這部分使用的是先前提到的兩張結果圖使用的時間，由時間進行比較可以得出，在 maskSize 設為 31 且 matrixSize 設為 1000 時是 Spinlock 的表現比較快，主要原因是因為在執行 CriticalSaction 的期間過短，導致 Spinlock 的等待時間小於 Mutex 卡住的時間，而在程式當中我執行 CriticalSaction 的區域是在計算每一個被 Mask 罩住的區域所以反之如果要讓 Mutex 效果優於 Spinlock 我們需要將 CriticalSaction 的時間拉長。

d) Following (c), please modify the configuration of the benchmark (maskSize and matrixSize) such that the performance results are opposite to the results of (c). Show the configuration of your benchmark by the screenshot of a file (config.h) and describe the property of the configuration. (Show both the execution results of using mutex and spinlock respectively to support your discussion.)

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ config.h ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~



```
// Workload parameter
#define MASK_SIZE 499  // Use 499 [Part3 (d) 讓 Mutex 跟 Spinlock 耗時表現相反]
#define MATRIX_SIZE 500  // Use 500 [Part3 (d) 讓 Mutex 跟 Spinlock 耗時表現相反]
#define CONVOLUTION_TIMES 3
```



```
Building part3.out
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/hw2/ESSD_PA2$ ./part3.out

====================== System Info ======================
numThread: 4
maskSize: 499 x 499
matrixSize: 500 x 500
Protect Shared Resource: Spinlock
Synchronize: Barrier

================== Generate Matrix Data =================
Generate Date Spend time : 0.010553

============ Start Single Thread Convolution ============
Single Thread Spend time : 2135.72

============= Start Multi-Thread Convolution ============
Thread ID : 0   PID : 7288      Core : 0
Thread ID : 2   PID : 7290      Core : 2
Thread ID : 1   PID : 7289      Core : 1
Thread ID : 3   PID : 7291      Core : 3
Multi Thread Spend time : 1449.41

======================= checking ========================
Matrix convolution result correct.
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/hw2/ESSD_PA2$ make part1.out
Building src/thread.o
Building src/system.o
Building pa2.o
Building part1.out
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/hw2/ESSD_PA2$ ./part1.out

====================== System Info ======================
numThread: 4
maskSize: 499 x 499
matrixSize: 500 x 500
Protect Shared Resource: Mutex
Synchronize: Barrier

================== Generate Matrix Data =================
Generate Date Spend time : 0.008844

============ Start Single Thread Convolution ============
Single Thread Spend time : 2133.47

============= Start Multi-Thread Convolution ============
Thread ID : 0   PID : 10433     Core : 0
Thread ID : 2   PID : 10435     Core : 2
Thread ID : 3   PID : 10436     Core : 3
Thread ID : 1   PID : 10434     Core : 1
Multi Thread Spend time : 1442.51

======================= checking ========================
Matrix convolution result correct.
mspl@mspl-Pulse-GL66-11UEK:~/Desktop/hw2/ESSD_PA2$ 
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

	Mutex	**Spinlock**
Spend time (s)	1442.51	1449.41

經實驗佐證當 CriticalSaction 的時間過短容易造成 Spinlock 的優勢，因此若要讓結果反轉就必須讓 CriticalSaction 的時間加長，要做到加長需要將 maskSize 加大，因為我使用的 CriticalSaction 是在 mask 的影響範圍內的計算，所以加大 maskSize 能有效增加 CriticalSaction 的持續時間，但是這樣是不夠的，為了加大 Mutex 的優勢，我會將 matrixSize 也減少，這樣除了可以減少運算等待的時間，也可以減少 CriticalSaction 以外的變因。