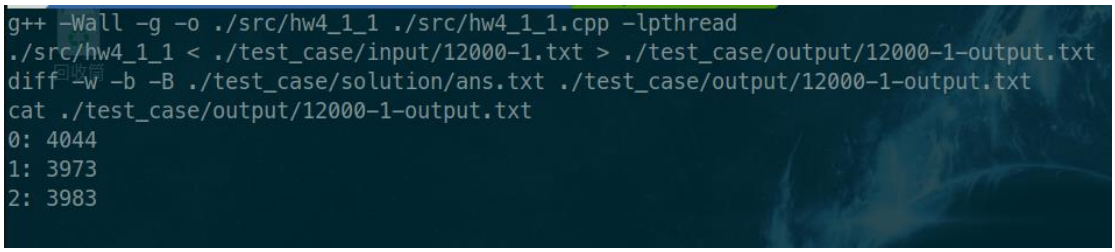
OS HW4 report

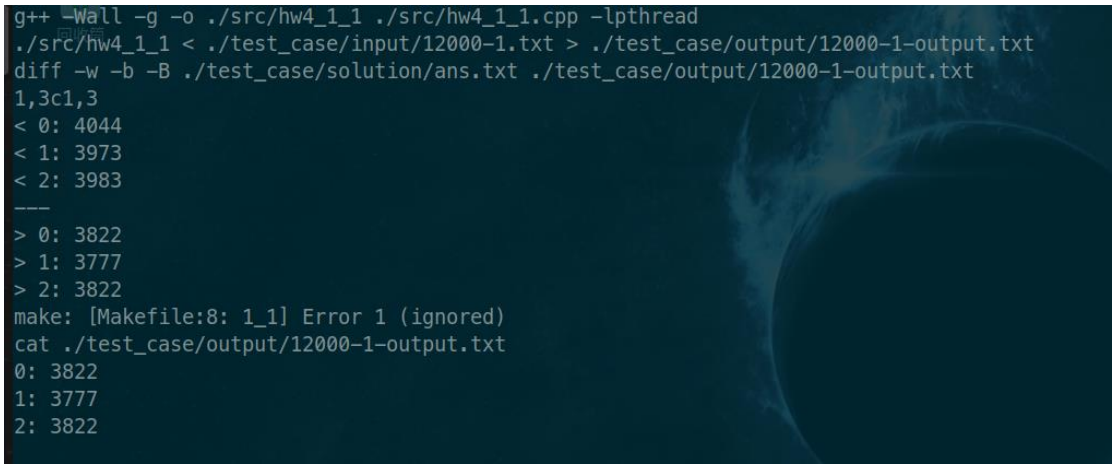
Name: 楊子民
Student ID: 0816147

Q1:
Compare results between hw4_1_1 with/without synchronization.

with synchronization



without synchronization



每個 thread 平分字串去計算裡面的 012 數量。
要記得用 mutex lock 保護全域變數更新的這個 critical section，不然值會變得不可預期。

Q2:

Compare results between hw4_1_2 with/without synchronization.

with synchronization

```
g++ -Wall -g -o ./src/hw4_1_2 ./src/hw4_1_2.cpp -lpthread
./src/hw4_1_2 < ./test_case/input/12000-2.txt > ./test_case/output/12000-2-output.txt
diff -w -b -B ./test_case/solution/ans.txt ./test_case/output/12000-2-output.txt
cat ./test_case/output/12000-2-output.txt
0: 4044
1: 3973
2: 3983
```

without synchronization

```
g++ -Wall -g -o ./src/hw4_1_2 ./src/hw4_1_2.cpp -lpthread
./src/hw4_1_2 < ./test_case/input/12000-2.txt > ./test_case/output/12000-2-output.txt
diff -w -b -B ./test_case/solution/ans.txt ./test_case/output/12000-2-output.txt
1c1,2
< 0: 4044
---
> 0: 1381
> 2: 3886
3d3
< 2: 3983
make: [Makefile:15: 1_2] Error 1 (ignored)
cat ./test_case/output/12000-2-output.txt
0: 1381
2: 3886
1: 3973
```

這邊 without synchronization 也保留了 mutex lock 保護全域變數更新的這個 critical section。

但要記得用 semaphore 讓 thread1 等其他 2 個 thread 計算完之後再輸出，不然值可能會錯，且 thread2 3 要記得等前一個 thread 輸出完之後再輸出，不然輸出順序會不可預期。

Q3:

Compare results between hw4_2 with/without synchronization.

with synchronization

```
g++ -Wall -g -o ./src/hw4_2   ./src/hw4_2.cpp   -lpthread
./src/hw4_2  < ./test_case/input/pi.txt > ./test_case/output/4_2-output.txt
cat ./test_case/output/4_2-output.txt
get: 78521
Pi: 3.140840
```

without synchronization

```
g++ -Wall -g -o ./src/hw4_2   ./src/hw4_2.cpp   -lpthread
./src/hw4_2  < ./test_case/input/pi.txt > ./test_case/output/4_2-output.txt
cat ./test_case/output/4_2-output.txt
get: 77140
Pi: 3.085600
```

每個 thread 平分要實作 Monte Carlo Method 的點的數量。

要記得用 mutex lock 保護全域變數更新的這個 critical section，不然值會變得不可預期。

但因為有用到亂數這個函式所以值本來就不可預期，所以 mutex 的影響比較難看出來我多試幾次刻意找一個值差比較多一點的放上來。

Q4:

Some problems you meet and how to resolve.

or some Reflections.

Hw4-1-2 我想只寫一個 function 達到 3 個依序輸出的目的，而不用寫 3 個很像的 function 只差在最後輸出，所以我想了一陣子決定用 4 個 semaphore 來進行同步，前 2 個用來讓 thread1 等 thread2 3 算完，後 2 個用來讓 thread 2 3 等前一個 thread 輸出完再輸出。