# Lab 4 report


## 6314061 – Quang Tuan Le


## February 2023

## 1 Task 1: Monte Carlo Estimation of PI

- The execution time of my program with number of threads (4, 8, 16, and 32) and the sample_points_per_thread is 10,000 (which is relatively 40,000/4 – 80,000/8 – 160,000/16 – 320,000/32 sample points).

| 4 threads | 8 threads | 16 threads | 32 threads |
|---|---|---|---|
| 0.000466 (s) | 0.000530 (s) | 0.000960 (s) | 0.001722 (s) |

## 2 Task 2: Bank account simulation

- The output when I ran the bank account simulation code for each of 2, 4, 8, 16, and 32 threads without fixing code.

| Number of threads | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|
| 1st Output | $0.00 | $22900.00 | $22700.00 | $99300.00 | $85000.00 |
| 2nd Output | $0.00 | $0.00 | $127300.00 | $80000.00 | $-61800.00 |
| 3rd Output | $0.00 | $1800.00 | $20400.00 | $-57100.00 | $237100.00 |

- The problem of the code is related to confliction when multiple threads execute deposit and withdraw which leads to logical error. So, I use pthread_mutex_lock and pthread_mutex_unlock to protect these critical sections (the code is attached to my submission). Here is the result after fixing the problem.

| Number of threads | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|
| Output | $0.00 | $0.00 | $0.00 | $0.00 | $0.00 |

# 3 Task 3: A read-write lock based on Pthread mutex and condition variables

- Outputs are almost same after replacing pthread_rwlock with my read-write lock. (They are same about the purpose).
- I only modified the functions rdlockThread and wrlockThread, so the main code is my test case. Here is the results: (test03 is modified, test03_1 is the original)
- 2 threads

```
Enter test case - ./test03
Main, initialize the read write lock
Main, create the read lock thread
Main, create the write lock thread
Entered thread, getting read lock
got the rwlock read lock
Entered thread, getting write lock
Main, wait for the threads
unlock the read lock
Next thread unlocked
Got the rwlock write lock, now unlock
Next thread unlocked
Main completed
```

```
Enter test case - ./test03_1
Main, initialize the read write lock
Main, create the read lock thread
Main, create the write lock thread
Entered thread, getting read lock
got the rwlock read lock
Entered thread, getting write lock
unlock the read lock
Next thread unlocked
Main, wait for the threads
Got the rwlock write lock, now unlock
Next thread unlocked
Main completed
```

- 4 threads

```
Enter test case - ./test03
Main, initialize the read write lock
Main, create the read lock thread
Main, create the read lock thread
Entered thread, getting read lock
got the rwlock read lock
Main, create the write lock thread
Main, create the write lock thread
Entered thread, getting write lock
Entered thread, getting read lock
Entered thread, getting write lock
unlock the read lock
Next thread unlocked
Got the rwlock write lock, now unlock
Next thread unlocked
got the rwlock read lock
Main, wait for the threads
unlock the read lock
Next thread unlocked
Got the rwlock write lock, now unlock
Next thread unlocked
Main completed
```

```
Enter test case - ./test03_1
Main, initialize the read write lock
Main, create the read lock thread
Main, create the read lock thread
Main, create the write lock thread
Entered thread, getting read lock
got the rwlock read lock
Main, create the write lock thread
Entered thread, getting read lock
got the rwlock read lock
Entered thread, getting write lock
Entered thread, getting write lock
unlock the read lock
Next thread unlocked
Main, wait for the threads
unlock the read lock
Next thread unlocked
Got the rwlock write lock, now unlock
Next thread unlocked
Got the rwlock write lock, now unlock
Next thread unlocked
Main completed
```

- So, basically both of implementation are supporting Read-Write locks when multiple threads are created and these threads can frequent read but only one can writes at a time. And both of them are working properly without any deadlock for 2 or 4 threads.