Fullname: Đào Ngọc Thành – 20163693

Labwork – Distributed System - Chapter 6: SYNCHRONIZATION

Question 1: Launch this program several times. What do you notice? Explain it!

Each time I run this program I got a different results, but the results always greater than 1000. Because, the resource is not synchronized between Threads.

Question 2: Change the code of the general executable program by replacing ThreadedWorkerWithoutSyncwith ThreadedWorkerWithSyncto initiate three instances worker1-3. What is the difference between the outputofthis program and that of question 1? Explain it!

Every time I ran this program, I got the same result 3000. Because the resource have been synchronized between Threads.

Each time I run this program I get the result 3000, compare to the different result from question 1. Because It’s another way to synchronized the resource.

Question 4: Complete this file above (in the part YOUR-CODE-HERE) with a loop to increase the variable shared by 1 for 5 seconds.

void \*fun(void \**args*)

{

time\_t start = time(NULL);

time\_t end = start + 5; //run for 5 seconds

while (start != end)

{

shared++;

start = time(NULL);

}

return

Question 5: Try to increase the value of threads and the value of the constant NUM\_TRANS after each execution time until you obtain the different results between Balanceand INIT\_BALANCE+credits-debits. Explain why do you get this difference.

I could not choose the value of the constant NUM\_TRANS to obtain the different results between Balanceand INIT\_BALANCE+credits-debits. Because the resource was not synchronized between threads.

Question 6: Try to build and run this program. Launch it repeatedly until you see the difference between Shared and Expect values. Analyze the source code to understand the problem that leads to this difference.

The problem comes in when number of threads becomes large, so the percent that the check lock section of incrementor function run in the same time when lock = 1 increase. Therefor, sometime I run this program the Shareda nd Expect values different.

Question 7: Now, you have to modify the code of the file without-lock.c in implementing the mutex lockabove (you can name it differently like mutex-lock-banking.c). Try to launch it repeatedly and evaluate the obtained output. What is the improvement after using mutex lock?

The value of Shared and Expect variable are the same despite of how many time I run it.

Question 8: compare the run times of the two strategies to prove that Fine Locking is faster and much faster on larger load sets.

* Testing with *NUM\_TRANS* = 100000, *n\_threads* = 5, it can be seen that fine lock is faster than coarse lock
  + Coarse lock: A screenshot of a cell phone

    Description automatically generated
  + Fine lock: A screenshot of a cell phone

    Description automatically generated
* Testing with *NUM\_TRANS* = 10000000, *n\_threads* = 5, it can be seen that fine lock is faster than coarse lock:
  + Coarse lock: A screenshot of a cell phone

    Description automatically generated
  + Fine lock:
* In real banking system, each transaction takes a few seconds to process. To simulate this, a small delay is added before releasing each lock using a function:

A screen shot of a computer

Description automatically generated

* Testing results with *NUM\_TRANS* = 5, *n\_threads* = 5, balance calculation delay = 1s, credit and deposit calculation delay = 2s. Fine lock is much faster than coarse lock in this case:
  + Coarse lock: A screenshot of a cell phone

    Description automatically generated
  + Fine lock: A screen shot of a smart phone

    Description automatically generated

Question 9: Run this program and what do you get as output? Explain what the deadlock is.

The deadlock

pthread\_create(&thread\_1, NULL, fun\_1, NULL);

pthread\_create(&thread\_2, NULL, fun\_2, NULL);

because two functions also need to wait the a, b free of lock to continue to run. So at any time it’s can encounter the situation that two function wait for each other.