Document Title: Laboratory Sheet 5

Subject: CPP (B.CSE VI SEM)

---

1. Write a program where 2 threads communicate using a single global variable “balance” and initialized to 1000..Thread 1 deposits amount = 50 for for 50 times and prints the balance amount and thread 2 withdrawals amount=20 for 20 times and prints the final balance .Execution of thread 1 and thread 2 should not interleave.

2. Write a program where 2 threads operate on a global variable “account” initialized to 1000. There is a deposit function which deposits a given amount in this “account”: int deposit(int amount)

   There is a withdrawal function which withdraws a given amount from the “account”:
   int withdrawal(int amount)

   However there is a condition: withdrawal function should block the calling thread when the amount in the “account” is less than 1000, i.e. you can’t withdraw if the “account” value is less than 1000. Threads calling the deposit function should indicate to the withdrawing threads when the amount is greater than 1000.
3. Write a thread program which demonstrates how to "wait" for thread completions by using the Pthread join routine. Since some implementations of Pthreads may not create threads in a joinable state, therefore explicitly created attribute in a joinable state so that they can be joined later. Created thread should perform the calculation of sum =sum + sin(i) + Tan(i), where i =0 to 10000. Print the out in the following manner

Main: creating thread 0
Main: creating thread 1
Thread 0 starting...
Main: creating thread 2
Thread 1 starting...
Main: creating thread 3
Thread 2 starting...
Thread 3 starting...
Thread 1 done. Result = -3.153838e+06
Thread 0 done. Result = -3.153838e+06
Main: completed join with thread 0 having a status of 0
Main: completed join with thread 1 having a status of 1
Thread 3 done. Result = -3.153838e+06
Thread 2 done. Result = -3.153838e+06
Main: completed join with thread 2 having a status of 2
Main: completed join with thread 3 having a status of 3
Main: program completed. Exiting.

4. Implementation of Boss/Worker Model: Here the idea is to have a single boss thread that creates work and several worker threads that process the work. Typically the boss thread creates a certain number of workers right away -even before any work has arrived. The worker threads form a thread pool and are all programmed to block

immediately. When the boss thread generates some work, it arranges to have one worker thread unblock to handle it. When all workers be busy the boss thread might do one of the following by taking request from the user

1. Queue up the work to be handled later as soon as a worker is free.
2. Create more worker threads.
3. Block until a worker is free to take the new work.

If no work has arrived recently and there are an excessive number of worker threads in the thread pool, the boss thread might terminate a few of them to recover resources. In any case, since creating and terminate threads is relatively expensive (compared to, say, blocking on a mutex) it is generally better to avoid creating a thread for each unit of work produced.