COL100: Lab 11
Threads

March 31, 2017

Part 1: Threads

1. (a) **List three reasons for wanting to use multiple threads in a process.**
   (b) **If a process exits and there are still threads of that process executing, will they continue to execute? Explain.**
   (c) **What is one advantage and one disadvantage of a process using user-level threads over a process using kernel-level threads?**
   (d) **Suppose two processes are executing, both of which are using multiple kernellevel threads. Is a context switch between two threads within one processes less work than a context switch between two threads in different processes? Explain.**
   (e) **Suppose two processes are executing, both of which are using multiple userlevel threads. Is a context switch between two threads within one processes less work than a context switch between two threads in different processes? Explain.**

   Ans:

   (a) simplifies program structure improves performance in presence of blocking make use of multiple processing units
   (b) No. When a process exits, it takes everything with it. This includes the process structure, the memory space, etc. including threads. (The process is the execution environment; without it, a thread can not execute.)
   (c) advantage: don't need kernel to understand threads faster switching, since kernel not involved
   disadvantage: if one thread blocks, all threads blocked process can't take advantage of multiple processors
   (d) YES! More state information needs to be switched when threads are in different processes. Context switch between two threads in the same process: only need to store/load the TCB information context switch between two threads in different processes: above, plus need to store/load the PCB information
   (e) YES! Process switch occurs when threads are in different processes. Context switch between two threads in the same process: Handled by the dispatcher within your program OS doesn't need to do anything context switch between two threads in different processes: must process switch between two processes

2. **What would be the impact of multithreading on a uni-processor system?**
   (a) Increase throughput
   (b) Degrade performance

(c) Reduce execution time

(d) Improve performance

Ans: Degrade performance

3. **Define race condition?**
   Ans: A race condition is when non-deterministic behavior results from threads accessing shared data or resources without following a defined synchronization protocol for serializing such access.

4. **What is a critical section? What are the requirements for correctly solving a critical section problem?**
   Ans: A section of code in which only 1 process can be executing at any given time because it uses shared variables. To solve this problem, you are essentially solving a locking/mutual exclusion problem, so you must be able to

   (a) lock the process before entering the critical section

   (b) unlock the process when you are done executing the critical section

   (c) wait to use the critical section of code if it is already locked

   (d) never have to wait 'forever' when trying to get to your critical section.

5. **Both busy waiting and blocking methods can be used as means to address critical section problems and process synchronization. Describe how blocking is different from busy waiting, and discuss the advantages and disadvantages of each.**
   Ans: Busy waiting means a process simply spins (does nothing but continue to test its entry condition) while it is waiting to enter its critical section. This continues to use (waste) CPU cycles, which is inefficient. Additionally, if a low priority process is in its critical section, and a high priority process goes into busy waiting, it may never get to its critical section., On a single CPU, busy waiting becomes a circular issue, so blocking would make more sense, as it only requires a single call to the OS to change state to Ready/Run, once it is notified that blocking process is complete.

6. **You are designing a data structure for efficient dictionary lookup in a multithreaded application. The design uses a hash table that consists of an array of pointers each corresponding to a hash bin. The array has 1001 elements, and a hash function takes an item to be searched and computes an entry between 0 and 1000. The pointer at the computed entry is either null, in which case the item is not found, or it points to a doubly linked list of items that you would search sequentially to see if any of them matches the item you are searching for. There are three functions defined on the hash table: Insertion (if an item is not there already), Lookup (to see if an item is there), and deletion (to remove an item from the table). Considering the need for synchronization, would you:**

   (a) **Use a mutex over the entire table?**

   (b) **Use a mutex over each hash bin?**

   (c) **Use a mutex over each hash bin and a mutex over each element in the doubly linked list?**

   Justify your answer.

   Ans: A mutex over the entire table is undesirable since it would unnecessarily restrict concurrency. Such a design would only permit a single insert, lookup or delete operation to be outstanding at any given time, even if they are to different hash bins. A mutex over each element in the doubly linked list would permit the greatest concurrency, but a correct, deadlock-free implementation has to ensure that all elements involved in a delete or insert operation, namely, up to three elements for an delete, or two elements and

the hash bin for inserts/some deletes, are acquired in a well-defined order. A mutex over each hash bin is a compromise between these two solutions – it permits more concurrency than solution 1, and is easier to implement correctly than solution 2.

7. **What is the disadvantage of shortest job first or shortest remaining time first scheduling (there is a subtle difference between the two, SJF is non preemptive while SRTF is preemptive)**
   Ans: You need to have a good approximation for how much time a process will take in order for these algorithms to work. (One can never know how much time an incoming process will take without actually running it

Part 2: Programming Questions

1. Write a Python program to add two n*n matrices using n threads.
2. Write a Python program to multiply two n*n matrices using n threads.
3. Implement a thread safe list in python by using any synchronization mechanism of your choice.