Name: Logan Roe
Section: 1
University ID: 367542190

Programming Project 2 Report

Summary (10 pts):

Through this project, I was able to significantly improve on my understanding of pthreads in general. For example, I learned how to handle large arrays of pthreads, and related variables, such as mutexes, and how to signal them properly between numerous different threads and conditions that had to be met. Further, I learned how to endlessly loop workers and how to consistently signal them as requirements were met.

Beyond pthreads, I learned how to deal with the time of day and various pointer related C coding concepts, such as dealing with pointers to make tables, linked lists, etc. In general, my C coding abilities and understanding has significantly improved due to this project and what was required through it. Similarly, my conceptual understanding of the related information has also been greatly improved. Overall, my pthread knowledge and C knowledge, specifically dealing with points, has greatly improved due to this project.

6.2:

(5pts) Average runtime for each program (use the "real" time)

Fine-Granularity	
Run #	Runtime (s)
1	55.893
2	55.893
3	55.895
4	55.896
5	55.891
6	55.893
7	55.895
8	55.893
Total:	447.149
Average:	55.894

Coarse-Granularity	
Run #	Runtime (s)
1	69.458
2	68.554
3	69.250
4	66.320
5	67.779
6	70.365
7	67.411
8	68.553
Total:	547.69
Average:	68.461

The left table shows the fine-granularity run numbers with associated runtimes and the right table shows the coarse-granularity run numbers with associated runtimes. Both tables show a total runtime, simply a sum of all runtimes, and an average runtime over the eight runs.

6.3:

1. **(3 pts) Which technique was faster – coarse or fine-grained locking?**
   Fine-grained locking was significantly faster. The average runtime of the fine-grained locking implementation was 55.894 seconds whereas the average runtime of the coarse-grained locking implementation was 68.461 seconds. The fine-grained locking implementation was 18.36% faster than the coarse-grained implementation.
2. **(3 pts) Why was this technique faster?**
   The fine-grained locking technique is faster because it allows the workers to access accounts in parallel, assuming they are accessing different accounts. This allows for process X to run at the same time as process Y without any interference (assuming different accounts in each). However, with coarse-grained locking, Y would have to wait for X to complete its processing before running itself.
3. **(3 pts) Are there any instances where the other technique would be faster?**
   When there is only one process, or very few, in which all processes have a lot of accounts being accessed, it is quite likely that coarse-grained locking would lead to faster overall execution time. This is because fewer overall mutexes have to be locked and unlocked, which clearly takes time in itself.
4. **(3 pts) What would happen to the performance if a lock was used for every 10 accounts? Why?**
   Assuming the same run scenarios, 10 runners and 1000 accounts, then having a lock for every 10 accounts would likely have a faster overall runtime than coarse-grained locking as it would allow for some parallelization to occur. It is hard to say whether or not it would be better than fine-grained locking because it is heavily dependent on how much the accounts in the transaction requests overlap. However, in general, for 1000 accounts, a lock for every 10 accounts would likely be faster than a lock for every account. This is because this leads to less overall locking and unlocking of accounts and it is unlikely for many more collisions to occur due to having a lock for 10 accounts instead of just one.
5. **(3 pts) What is the optimal locking granularity (fine, coarse or medium)?**
   This can change on a case-to-case basis between fine and medium granularity. For fine granularity, it will be better when dealing with programs that access every single “node” numerous times in quick succession. However, if a lot of “nodes” are not being accessed, or at least not at the same time, a medium granularity would likely be better. This would allow for the advantage of fewer mutex locks and unlocks but also avoid the majority of polling that would occur with coarse granularity. Overall, medium granularity will likely be best in the general case, however, fine granularity will prove to be better in other cases.