EDA092 - Lab 2 Report
Group A27

Henrik Hugo & Simon Fransson
(hhugo@student.chalmers.se, frsimon@student.chalmers.se)

December 10, 2013

1 One lock

Measurements done with one lock and proofing why the solution does not yield any deadlocks.

1.1 Measurements

As shown in Figure 1 there is a difference between 2 and 4 threads. The measurements are done over 10 executions and taking the average of the execution time. The results shows that it takes more time for the program to finish with more threads. These results are intriguing as one would think that an increased amount of threads would yield in less execution time. This is reasonable because more work should be done in less time. This is however not the case and the reason is because of collisions between the different threads as they try to access the critical section at the same time. This leaves threads waiting for the mutual exclusion variable to become available and not doing anything in the meantime.

Another thing delaying the execution is the system taking care of other processes as well. In an ideal execution, our own process would be the only one running and this is not the case. The program is also not considering which thread started to wait first for the token (lock) if the token is already in use. This results in some threads end up waiting very long until they actually get access to the critical section which prolongs execution time.

This is the reason for the increase in execution time because the program is spending more time waiting than it is executing.

Table 1: Measurements for one lock

	One-lock (2 threads)[ms]	One-lock (4 threads)[ms]
	56.89	163.29
	58.14	142.33
	141.17	117.92
	56.78	81.12
	99.03	102.67
	170.22	146.45
	123.22	98.71
	124.49	96.47
	74.85	151.64
	155.68	170.47
Average:	106.047	127.107

1.2 Deadlock

Question: is deadlock possible to occur? Answer: No.
There are four conditions that needs to be met in order for deadlock to occur:

- Mutual exclusion: only one process at a time can use a resource.
  - True: Because the `pthread_mutex_lock` is only allowing one thread at a time to use the queue for enqueuing or dequeuing.

- Hold and wait: a process holding some resource can request additional resources and wait for them if they are held by other processes.
  - False: The enqueue and dequeue functions will not try to request additional resources inside the lock. Also there is only one resource to handle.
- No preemption: a resource can only be release by the process holding it. After that process has completed its task.
  - True: The `pthread_mutex_unlock` is used for this.
- Circular wait: there exists a circular chain of 2 or more blocked processes, each waiting for a resource held by the next process in the chain.
  - False: Since the resources in our programs are independent of each other, no process will hold a resource whilst waiting for another. The resources used by the process will be released when the work is done, and the work does not contain acquiring any other resource than the one already acquired.

Conclusion: Since at least one of the conditions is broken, deadlock can not occur.

2 Two locks

Measurements done with two locks and proofing why the solution still does not yield any deadlocks.

2.1 Measurements

As the results display in Figure 2, there was an improvement for the average execution time using two locks. The reason is because there are not as many collisions anymore. The threads are not always contesting for the same lock as they were before since there is a second lock. The two locks are also independent of each other so the threads can dequeue and enqueue concurrently without any troubles. In other words, instead of having just 1 thread executing in one lock we can now have two threads executing concurrently by each thread executing in its own critical section.

Table 2: Measurements for one lock

	Two-lock (2 threads)[ms]	Two-lock (4 threads)[ms]
	32.22	48.25
	16.32	35.07
	93.45	79.74
	128.46	45.15
	54.3	49.63
	65.46	16.4
	149.18	73
	104.2	128.8
	55.14	48.77
	54.1	49.1
Average:	75.283	57.391

2.2 Deadlock

Question: is deadlock possible to occur? Answer: No.
The same conclusion drawn from using one lock can be drawn again. The difference with the two locks does not introduce any new complications. This is because the two locks are independent of each other. It could have been a problem if once one lock had been acquired to try and acquire the other lock as well. However since the implementation is not done in this way, the problem does not arises.