Concurrent and Distributed Systems - 2015–2016

Supervision 1: Semaphores, generalised producer-consumer, and priorities

Q0 Semaphores

(a) Counting semaphores are initialised to a value — 0, 1, or some arbitrary n. For each case, list one situation in which that initialisation would make sense.

(b) Write down two fragments of pseudo-code, to be run in two different threads, that experience deadlock as a result of poor use of mutual exclusion.

(c) Deadlock is not limited to mutual exclusion; it can occur any time its preconditions (especially hold-and-wait, cyclic dependence) occur. Describe a situation in which two threads making use of semaphores for condition synchronisation (e.g., in producer-consumer) can deadlock.

```
int buffer[N]; int in = 0, out = 0;
spaces = new Semaphore(N);
items  = new Semaphore(0);
guard  = new Semaphore(1);   // for mutual exclusion

// producer threads
while(true) {
  item = produce();
  wait(spaces);
  wait(guard);
    buffer[in] = item;
    in = (in + 1) % N;
  signal(guard);
  signal(items);
}

// consumer threads
while(true) {
  wait(items);
  wait(guard);
    item = buffer[out];
    out =(out+1) % N;
  signal(guard);
  signal(spaces);
  consume(item);
}
```

Figure 1: Pseudo-code for a producer-consumer queue using semaphores.

(d) In Figure 1, `items` and `spaces` are used for condition synchronisation, and `guard` is used for mutual exclusion. Why will this implementation become unsafe in the presence of multiple consumer threads or multiple producer threads, if we remove `guard`?

(e) Semaphores are introduced in part to improve efficiency under contention around critical sections by preferring blocking to spinning. Describe a situation in which this might not be the case; more generally, under what circumstances will semaphores hurt, rather than help, performance?

(f) The implementation of semaphores themselves depends on two classes of operations: increment/decrement of an integer, and blocking/waking up threads. As such, semaphore operations are themselves composite operations. What might go wrong if `wait()`'s integer operation and scheduler operation are non-atomic? How about `signal()`?

Q1 Priority and work distribution

The implementation in Figure 1 makes no particular effort to prioritise which consumer threads receive items.

(a) Round-robin work distribution would distribute work evenly across all consumer threads. Provide pseudo-code for an implementation based on a fixed, known number of threads.

(b) Prioritised work distribution prefers threads in certain classes (e.g., high priority) over those in other classes (e.g., low priority). Provide pseudo-code for an implementation based on an unknown number of threads each belonging to one of three priority bands. Assume that "peeking" at semaphore values and queue lengths is allowed.

(c) Most recently used (MRU) work distribution prefers the most recently used thread over less recently used threads. Provide pseudo-code for an implementation based on an unknown number of threads — does the problem become easier if we can modify the semaphore implementation?

(d) For each of the above work distribution schemes, describe a scenario in which each scheme might be used in order to improve performance, and explain why.

Q2 Contention

The implementation suffers from unnecessary contention between producers and consumers due to the shared guard lock.

(a) Provide pseudo-code for an implementation that eliminates that contention.

(b) How can we convince ourselves that the solution is correct?

Q3 Priority inversion

A system using thre generalised producer-consumer implementation in Figure 1 suffers from priority inversion.

(a) A portion of the priority inversion arises from low-priority producers starving high-priority consumers waiting for one another via 'guard'. Why might using a mutex for mutual exclusion, rather than a semaphore, make this an easier problem to mitigate?

(b) Another portion of the priority inversion arises from low-priority consumers starving high-priority consumers. How might this problem be addressed?