Homework 9

Handout 27
CS242: Autumn 2002
5 December

Solutions

1. .... Actor computing

(a) A sequence number can be added to each task. When B first receives a task, it can check the sequence number. If this is not the first task, or the next one to be processed, then B should store the task and process it later in order.

(b) One protocol, similar to TCP, is for B to acknowledge each received message (by sequence number). To avoid flooding the communication mechanism, A can send a few messages, then wait for acknowledgements to arrive before proceeding with additional messages. If A receives acknowledgements for several messages with sequence numbers greater than n, then A can suspect that message n is delayed and resend it.

(c) The *I'm done* message should have a sequence number too.

2. .... Concurrent access to objects

(a) Two threads might change `top` at the same time.

(b) Use synchronized methods.

(c) No, because it's okay to access one side of the queue while another thread is accessing the other side of the queue. One alters `back` while the other alters `front`.

3. .... Java syncronized objects

(a) The thread calling `put` waits until someone calls `get` and makes space available in the buffer.

(b) Wake up any thread waiting to call the synchronized method.

(c) Two threads could call `get` at the same time. Both could increment `count` before either store, causing both assignments `buffer[putIn] = value` to store into the same location. Since the second assignment overwrites the first, data is lost.

(d) It should be possible to allow the two to run in parallel if $|\texttt{putIn} - \texttt{takeOut}| > 1$, where difference is computed modulo `numSlots`.

(e) We need to change the test $|\texttt{putIn} - \texttt{takeOut}| > 1$ in `put`.

4. .... Resources and Java Garbage Collection

(a) When the object is garbage collected (which is when the `finalize` method is called).

(b) Suppose one object tries to acquire the camera while the object which holds the camera is uncollected garbage. If the garbage collector only runs when there is a memory shortage, there will be a deadlock because the program will not make any further progress and, thus, will not consume any more memory thereby never reaching a situation where the collector decides to collect outstanding garbage.

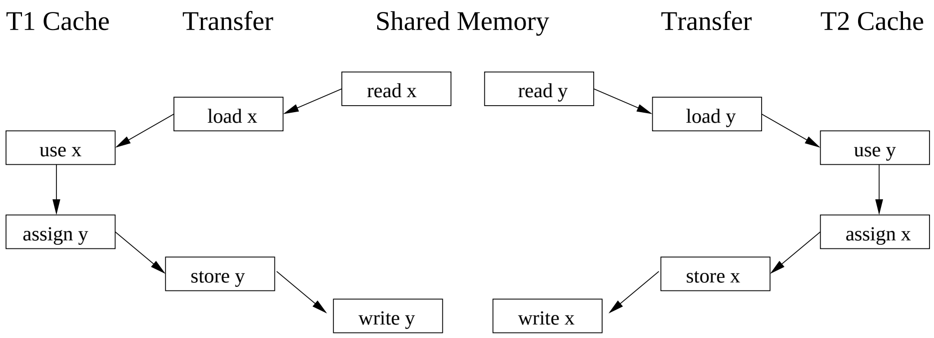
(c) No, forcing the call to garbage collection only lets the JVM know it's a good time to do it, it doesn't force garbage collection to actually happen.

(d) Add another method called (e.g., `dispose`), and call this explicitly when the camera should be released. This may also require adding some internal flags to make sure the camera doesn't get released twice.

(e) A multi-threaded JVM could probably avoid this problem, since presumably if all the other threads were suspended (due to being blocked), then the garbage collector thread would run and would collect the object, freeing the lock. But since not all JVM's are multi-threaded, a portable implementation should not rely on this behavior.

5. .... Java memory model

(a) This is the straight forward mapping of the code to the dependency graph. Notice that a and b do not show up since they are thread local variables and do not change the shared memory state.



(b) The only difference between this and part a) is that the constraint between the assign and store for both x and y are removed. The prescient store means that these could actually take place before the read of x and y happen. This is emphasized in the drawing by moving the store and write for x and y to the top.