Final Exam Answers – CS 343 Fall 2022

Instructor: Peter Buhr

December 10, 2022

These are not the only answers that are acceptable, but these answers come from the notes or lectures.

1. (a) **2 marks** The Coordinator can accumulate results (subtotals) while Workers are reinitialize.

   (b) **2 marks** member block waits for Nth thread, and then unblocks all waiting threads.
   member last is implicitly called by the Nth thread that triggers the barrier release.

   (c) **3 marks** The semaphore is initialized to 0.
   The P is in one thread before S2 and the V in the other thread after the S1.

   (d) **2 marks** A *simple* critical section has only one thread in it.
   A *complex* critical section can have multiple threads in it.

   (e) **7 marks**

   ```
   1   COBEGIN
   1       BEGIN S1; S3; END
   1       S2;
       COEND
   1   S4;
   1   COBEGIN
   1       S5;
   1       S6;
       COEND
   ```

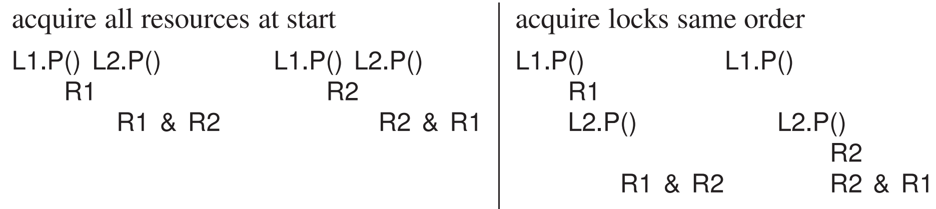
   (f) **1 mark** Yes.

   (g) **2 marks** A *shadow queue* contains information about the kind of blocked thread waiting on the lock.

   (h) **2 marks** Barging between reader/writer threads and barging among the writer threads.

2. (a) **2 marks** synchronization or mutual exclusion

(b) **6 marks**



(c) i. **3 marks**

11	Total Resources	P1	0
-10	Used		2
1	Available for allocation		

The state is NOT safe as there are insufficient resources for any process to execute so no sequence of execution is possible after this point.

ii. **3 marks**

11	Total Resources	P4	0
-10	Used		8
1	Available for allocation		

The state is safe as any sequence of execution after this point is safe.

iii. **4 marks**

11	Total Resources	P1	1
-9	Used		3
2	Available for allocation	P2	0
			5
		P3	1
			9
		P4	3
			11

The state is safe as this particular sequence of execution is safe.

3. (a) **2 marks** A thread can be preempted between the lock release and returning v, allowing v to change before the preemption continues with the wrong v.
   Copy v to a local temporary, release the lock, return the temporary.
   (b) **2 marks** The signal is *delayed* because the signaller holds the monitor lock.
   (c) **2 marks** For signal, the signalled thread is postponed and signaller thread continues.
   For signalBlock the signaller thread is postponed and signalled thread continues.
   (d) **1 mark** signalBlock
   (e) **2 marks** read-only member to access data.
   combining multi-step protocol into a single call that still requires a complex critical-section.
   (f) **2 marks** The signaller queue is optimized away because the signaller has the highest priority so it does not need to block.
   The signalled queue is changed to a stack needed for **_Accept**.

4. (a) **2 marks** Adding **_When** clauses before each **_Accept** / **_Select** clause and a statement afterwards.
   (b) **2 marks** If an accepted member fails with an exception, the acceptor is notified with a RendezvousFailure exception.
   (c) **2 marks** A courier carries message between administrators so the administrators do not make calls to communicate with each other.
   (d) **1 mark** client side
   (e) **2 marks** The **_Select** blocks until either future f1 or both f2 and f3 are available.

5. (a) **1 mark** No
   (b) **2 marks** Data values are replicated from memory through the cache levels into registers.
   (c) **2 marks** **volatile** ensures variables are loaded/stored frequently to/from registers.
   (d) **1 mark** ABA
   (e) **1 mark** channels
   (f) i. **3 marks** Nth task does notifyAll, leaves monitor and performs its ith step, and then races back (barging) into the barrier before any notified task restarts. It sees count still at N and incorrectly starts its ith+1 step before the current tasks have completed their ith step.
   ii. **1 mark** Nth task sets count to 0 (barging avoidance).
   iii. **2 marks** Spurious wakeup may spontaneously unblock a waiting thread from a condition variable.

6. (a) **12 marks**

```
1     unsigned int winner;
1     bool shutdownStarted = false;

      TallyBets::Payout TallyBets::placeBet( BetSlip slip ) {
1       if ( shutdownStarted ) _Throw Leave();
1           try {
1                 _Accept( done ) {
1                 } or _Accept( race ) {
1                 } or _Accept( placeBet ) {
                  }
1           } catch( uMutexFailure::RendezvousFailure & ) {}
1       if ( shutdownStarted ) _Throw Leave();
1           return tally( slip );
      } // TallyBets::placeBet

      void TallyBets::race( unsigned int winner ) {
1           TallyBets::winner = winner;
      } // TallyBets::race

      void done() {
1           shutdownStarted = true;
      } // TallyBets::done
```

(b) **7 marks**

```
1     uCondition bench;

      TallyBets::Payout TallyBets::placeBet( BetSlip slip ) {
1       if ( shutdownStarted ) _Throw Leave();
1           bench.wait();
1           bench.signal();
1       if ( shutdownStarted ) _Throw Leave();
1           return tally( slip );
      } // TallyBets::placeBet

      void TallyBets::race( unsigned int winner ) {
–           TallyBets::winner = winner;
1           bench.signal();
      } // TallyBets::race

      void done() {
–           shutdownStarted = true;
      } // TallyBets::done
```

(c) **7 marks**

```
1     AUTOMATIC_SIGNAL;
1     unsigned int numBets = 0;

      TallyBets::Payout TallyBets::placeBet( BetSlip slip ) {
–       if ( shutdownStarted ) _Throw Leave();
1           numBets += 1;
1           WAITUNTIL( numBets == 0, , );
–       if ( shutdownStarted ) _Throw Leave();
1           EXIT();
–           return tally( slip );
      } // TallyBets::placeBet

      void TallyBets::race( unsigned int winner ) {
–           TallyBets::winner = winner;
1           numBets = 0;
1           EXIT();
      } // TallyBets::race

      void done() {
–           shutdownStarted = true;
      } // TallyBets::done
```

7. **25 marks**

```
      struct Work {
1         BetSlip slip;
1         FPayout fpayout;
1         Work( BetSlip slip ) : slip( slip ) {}
      };
1     FPayout placeBet( BetSlip slip );
      Work * node;
1     list<Work *> students;                       // students waiting for race results
1     unsigned int numStuds;

1     TallyBets( unsigned int numStuds ) : numStuds( numStuds ) {}
      TallyBets::FPayout TallyBets::placeBet( BetSlip slip ) {
1         node = new Work( slip );
1         return node->fpayout;
      } // TallyBets::placeBet
-     void TallyBets::race( unsigned int winner )  // same as for monitor
-     void TallyBets::done()                       // same as for monitor

      void TallyBets::main() {
1         for ( ;; ) {
1             _Accept( done ) {
1                 break;
1             } or _Accept( race ) {
1                 while ( ! students.empty() ) {
1                     Work * n = students.front();
1                     students.pop_front();
1                     n->fpayout.delivery( tally( n->slip ) );
1                     delete n;
                  } // for
              } or _Accept( placeBet ) {
1                 students.push_back( node );      // store future
              } // _Accept
          } // for
          // Students not waiting on futures.
1         unsigned int rem = numStuds - 1 - students.size();
1         for ( unsigned int b = 0; b < rem; b += 1 ) {
1             _Accept( placeBet ) {
-                 students.push_back( node );      // store future
-                 Work * n = students.front();
-                 students.pop_front();
1                 n->fpayout.delivery( new Leave() );
-                 delete n;
1             } or _Accept( done ) {}
          } // for
          // Students waiting on futures.
1         while ( ! students.empty() ) {
-             Work * n = students.front();
-             students.pop_front();
-             n->fpayout.delivery( new Leave() );
-             delete n;
          } // while
      } // TallyBets::main
```

Alternatively, put students.push_back(node) in placeBet().