Final Exam Answers – CS 343 Winter 2023

Instructor: Caroline Kierstead

April 13, 2023

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

Part A - Multiple Choice

Elided for reuse.

Part B - Short Answer and Code

1. (a) **2 marks** Can only be done by removing synchronization, which is required for communication i.e. reduces class of solvable problems to ones that can be solved independently.

   (b) **2 marks** *Deadlock prevention* removes one of the conditions necessary for deadlock, thus ensuring that deadlock cannot occur. *Deadlock avoidance* allows a thread/task to move into a potentially unsafe state, but the system prevents deadlock from occurring by refusing requests that would (conservatively) lead to deadlock.

   (c) **1 mark** One of either *Banker's Algorithm* or *Allocation Graphs*.

   (d) **1 mark** The *ordered resource policy* is the only practical method.

2. (a) **3 marks**

```
Monitor M {
1     MutexLock m;
      void foo() {
1           m.acquire();
            ...
1           m.release();
      }
};
```

   (b) **3 marks**

```
int Monitor::bar() {
      m.acquire();
      ...
2     int local_value = value;        // make copy before drop lock
      m.release();
1     return local_value;             // return copy
}
```

   (c) **3 marks**

```
Monitor M {
      void foo() {
            ...
            bench.wait(m)
1           // NO REACQUIRE
            ...
      }
      int bar() {
            ...
1           if ( ! bench.signal() )
1                 m.release();
            ...
      }
```

Part C – Long Answer

1. (a) **10 marks**

```
L1:
1   bool taxiWaiting = false, clientWaiting = false;

L2:
1   clientWaiting = true;
1   xclient = x; yclient = y;
–   clientId = id;

1   _When ( ! taxiWaiting ) _Accept( getClient ) {}

1   clientWaiting = false;

L3:
1   taxiWaiting = true;
1   taxiId = id;

1   _When ( ! clientWaiting ) _Accept( getTaxi ) {}

1   taxiWaiting = false;
1   x = xclient; y = yclient; // taxi returns client info
```

(b) **14 marks** If not using shadow queue in uCondition, need more complex exchange protocol.

```
L1:
1   uCondition waitingTaxis, waitingClients;

L2:
1   if ( ! waitingTaxis.empty() ) {
1       xclient = x; yclient = y;
1       clientId = id;
1       taxiId = waitingTaxis.front();
1       waitingTaxis.signalBlock();
    } else {
1       waitingClients.wait( id );
1       xclient = x; yclient = y;
1       clientId = id;
    } // if

L3:
1   if ( ! waitingClients.empty() ) {
1       taxiId = id;
1       waitingClients.signalBlock();
    } else {
1       waitingTaxis.wait( id );
    } // if
1   x = xclient; y = yclient; // taxi returns client info
```

(c) **11 marks**

```
L1:
1   int waitingTaxis = 0, waitingClients = 0;
1   AUTOMATIC_SIGNAL;

L2:
1   waitingClients++;
1   if ( waitingTaxis == 0 ) {
1       WAITUNTIL( waitingTaxis != 0„ );
1       waitingClients--;
–       waitingTaxis--;
0.5     xclient = x; yclient = y;
–       clientId = id;
1       exchange = false;
    } else {
0.5     xclient = x; yclient = y;
–       clientId = id;
        exchange = true;
1       WAITUNTIL( ! exchange, , );
    } // if

1   EXIT();
    return taxiId; // given

L3:
// code is symmetric to client
1   waitingTaxis++;
    if ( waitingClients == 0 ) {
        WAITUNTIL( waitingClients != 0„ );

        waitingClients--;
        waitingTaxis--;
0.5     taxiId = id;
        exchange = false;
    } else {
0.5     taxiId = id;
        exchange = true;
        WAITUNTIL( ! exchange, , );
    } // if
1   x = xclient; y = yclient;
    EXIT();
    return clientId; // given
```

2. **25 marks**

```
void MapleLeafTaxiDispatcher::main() {
1    Taxi * taxitasks[NoOfTaxi];

1    for ( int id = 0; id < NoOfTaxi; id += 1 ) {
1        taxitasks[id] = new Taxi( *this, id );              // allocate taxis
     }
1    for ( ;; ) {
1        _Accept( close ) {
1            break;
1        } or _Accept( getClient || getTaxi ) {
1            if ( taxis.size() > 0 && clients.size() > 0 ) {
1                LocnClient *n = clients.front();
-                clients.pop_front();
1                xclient = n->x; yclient = n->y;
1                list<LocnTaxi *>::iterator nearest = nearestTaxi( n, taxis ); // find closest taxi
1                n->ftaxi.delivery( (*nearest)->id );
1                delete n;                                   // allocated in getTaxi
1                (*nearest)->idle.signalBlock();
1                taxis.erase( nearest );
             }
         }
     }
1    osacquire( cout ) << "Closed for the day." << endl;
1    for ( int i = 0; clients.size() != 0; i += 1 ) {        // notify potentially waiting clients
1        LocnClient *client = clients.front();
-        clients.pop_front();
1        client->ftaxi.delivery( new Closed );               // raise exception
1        delete client;                                      // allocated in getTaxi
     }
1    closed = true;                                          // tell taxi tasks to go home
1    for ( int i = 0; i < NoOfTaxi; i += 1 ) {
1        if ( taxis.empty() ) _Accept( getClient );          // wait for taxi
1        taxis.front()->idle.signalBlock();
-        taxis.pop_front(); // unblock with closed
     }
1    for ( int i = 0; i < NoOfTaxi; i += 1 ) delete taxitasks[i]; // delete taxis
}
```