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

1. (a) **7 marks**

   ```
   L1 = L2 = L3 = 0;
   COBEGIN
   BEGIN S1; V(L1); S3; END
   BEGIN S2; V(L2); S5; V(L3); END
   BEGIN P(L1); P(L2); S4; P(L3); S6; END
   COEND
   S7;
   ```

(b) **4 marks**

   - Entering readers cannot barge ahead of waiting writers.
     The last exiting reader only starts a waiting writer so reader groups alternate with waiting writers.
   - Entering writers always wait if there were waiting tasks because exiting tasks ensure waiting tasks start.
     Exiting writers always start a waiting reader first so writers alternate with waiting reader groups.

(c) **2 marks**

   - W 12:30 leaves and R 1:00 and 2:00 enter, so R 2:00 reads stale W 12:30 data rather than fresher W 1:30 data
   - W 12:30 leaves and W 1:30 enters, so R 1:00 reads fresh W 1:30 data rather than W 12:30 data

   - **1 mark** Block readers and writes on same lock in temporal order.
   - **1 mark** Cannot tell reader from writer when blocked on same lock.

(d) **2 marks**

   - Task calling P atomically blocks on s and m is V ed
   - **2 marks** Preemption allows race between m.V() and s.P() so temporal blocking not preserved
   - **2 marks** Race removed because preemption cannot occur between operations because of atomicity
2. (a) **2 marks** In live lock no task is using the critical section, while in starvation some tasks are using the critical section but some are not.

(b) **3 marks** No, the Banker’s algorithm assumes all tasks require their maximum resource allocations to run until completion, which does not happen in practice.

(c) i. **2 marks** Synchronization deadlock occurs when there is a failure in cooperation so that a blocked task is never woken. Mutual exclusion deadlock occurs when a requested resource protected by mutual exclusion is never acquired.

ii. **1 mark** Mutual exclusion deadlock can only be prevented if one of the rules needed to create it can be removed.

(d) **4 marks**

---

![Diagram](image-url)
3. (a) **2 marks** Explicit scheduling is when the programmer controls the next task executing in the monitor.

*Implicit* scheduling is when the monitor controls the next task executing in the monitor.

(b) **2 marks** The advantage is optimizing the common occurrence of a task immediately leaving the monitor after signalled a waiting task.

The disadvantage is that it prevents multiple signalling or signal/wait (dating service unblock girl/wait on exchange) scenarios.

(c) i. **1 mark** A/S stack
ii. **1 mark** P1
iii. **1 mark** _Accept_( remove )
iv. **1 mark** A/S stack
v. **1 mark** C2
vi. **1 mark** P1
vii. **1 mark** BB

(d) i. **1 mark** A continuous stream of C’s prevents W and S from unblocking.
ii. **1 mark** Randomly starting W and S makes synchronization within the monitor difficult.
iii. **1 mark** C can barge ahead of a W or S.
iv. **1 mark** C cannot barge ahead of a signalled W or S.

4. (a) **2 marks** If a task blocks on a semaphore, the monitor’s mutual-exclusion lock is not released and no further calls are allowed, i.e., deadlock.

Need to use a synchronization/condition/uCondition lock instead.

(b) **1 mark** When it is necessary to communication with the task (monitor object) after it has terminated.

(c) i. **3 marks** thread, stack, mutual exclusion (monitor)
ii. **1 mark** it does not accept calls to the listed members
iii. **1 mark** mutual exclusion
iv. **1 mark** add more tasks
v. **1 mark** buffering

5. (a) **2 marks** Because the processor stalls until the data reaches the registers, where all computations occur (especially on a RISC computer).

Because data may be overwritten multiple times so only the last write needs to move all the way to memory.

(b) **2 marks** Read serving is false, set intent to true, and miss seeing my serving set to true.

(c) **2 marks** LL instruction loads (reads) value from memory into register and establishes *watchpoint* (Oracle) on load address. SC instruction stores (writes) new value back to original or another memory location but only if no write has occurred at LL *watchpoint*.

(d) **2 marks** Both use typed channels/mailboxes and both use a select/case statement to access messages of different type.
6. (a) 10 marks

```c
L1:
1 bool first = true;

L2:
1 pb = id;
1 if ( first ) {
1 first = false;
1 _Accept( JY );
1 first = true;
}
1 return jy;

L3:
1 jy = id;
1 if ( first ) {
1 first = false;
1 _Accept( PB );
1 first = true;
}
1 return pb;
```

(b) 11 marks

```c
L1:
1 uCondition PBs, JYs;

L2:
1 if ( JYs.empty() ) {
1 PBs.wait( id );
1 pb = id;
} else {
1 pb = id;
1 JYs.signalBlock();
} // if
1 return jy;

L3:
1 if ( PBs.empty() ) {
1 JYs.wait();
1 jy = id;
} else {
1 jy = id;
} // if
1 return pb;
```

(c) 10 marks

```c
L1:
1 AUTOMATIC_SIGNAL;

L2:
1 pb.push_back( id );
1 WAITUNTIL( ! jy.empty(), , );
1 id = jy.front();
1 jy.pop_front();
1 EXIT();
1 return id;

L3:
1 jy.push_back( id );
1 WAITUNTIL( ! pb.empty(), , );
1 id = pb.front();
1 pb.pop_front();
1 EXIT();
1 return id;
```
7. 23 marks

```c
void fillShuttle(unsigned int noOfClients) {
    Shuttle * shuttleId = (Shuttle *)shuttles.front();
    shuttles.signalBlock();
    for (unsigned int i = 0; i < noOfClients; i += 1) {
        clients.front().delivery(shuttleId);
        clients.pop_front();
    } // for
}

void main() {
    for (;;) {
        _Accept(~Coordinator) {
            break;
        } or _Accept(timeUp) {
            if (!shuttles.empty() && numClientsWaiting > 0) fillShuttle(numClientsWaiting);
        } or _Accept(checkIn) {
            if (numClientsWaiting >= ShuttleSize) fillShuttle(ShuttleSize);
        } or _Accept(getRide) {
            if (!shuttles.empty() && numClientsWaiting >= ShuttleSize) fillShuttle(ShuttleSize);
        } // _Accept
    } // for

    // can shutdown in any order
    shuttingDown = true;
    for (unsigned int i = 0; i < NumClients; i += 1) {
        if (clients.empty()) _Accept(getRide);
        clients.front().exception(new Closed);
        clients.pop_front();
    } // for

    for (unsigned int i = 0; i < NumShuttles; i += 1) {
        if (shuttles.empty()) _Accept(checkIn);
        shuttles.signalBlock();
    } // for

    _Accept(timeUp);
}
```