Christian Shinkle

7.12: The four necessary conditions:

- Mutual Exclusion: locks can only be acquired when no other thread is using it
- Hold-and-Wait: a lock can be held by threads while waiting for another
- No preemption: locks held by a thread cannot be preempted by an thread
- Circular Wait: With multiple threads and multiple locks, a circular condition is possible to arise.
  With all conditions satisfied, deadlock can occur in this situation.

7.17: This system is deadlock free because with 4 resources and 3 processes, in the worst case scenario, two processes will have one resource and the last process will have two. This means two processes will have to wait for the third to finish execution. Once it does, those two resources will be released and the other two processes can use them. Therefore, deadlock is not possible.

7.23: State 0:

Available			
A	B	C	D
3	3	2	1

	Allocation				Max				Need			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	2	0	0	1	4	2	1	2	2	2	1	1
P1	3	1	2	1	5	2	5	2	2	1	3	1
P2	2	1	0	3	2	3	1	6	0	2	1	3
P3	1	3	1	2	1	4	2	4	0	1	1	2
P4	1	4	3	2	3	6	6	5	2	2	3	3

Following the Banker's Algorithm, P0 is the only process whose need is less than available, so we allocate the resources to it.

State 1:

Available			
A	B	C	D
5	3	2	2

	Allocation				Max				Need			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	Finished in state 1											
P1	3	1	2	1	5	2	5	2	2	1	3	1
P2	2	1	0	3	2	3	1	6	0	2	1	3
P3	1	3	1	2	1	4	2	4	0	1	1	2
P4	1	4	3	2	3	6	6	5	2	2	3	3

Next, P3 is now the only process whose need is less than available.

State 2:

Available			
A	B	C	D
6	6	3	4

	Allocation				Max				Need			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	Finished in state 1											
P1	3	1	2	1	5	2	5	2	2	1	3	1
P2	2	1	0	3	2	3	1	6	0	2	1	3
P3	Finished in state 2											
P4	1	4	3	2	3	6	6	5	2	2	3	3

Next, P4’s need is less than available.

State 3:

Available			
A	B	C	D
7	10	6	6

	Allocation				Max				Need			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	Finished in state 1											
P1	3	1	2	1	5	2	5	2	2	1	3	1
P2	2	1	0	3	2	3	1	6	0	2	1	3
P3	Finished in state 2											
P4	Finished in state 3											

Next, P1’s need is less than available.

State 4:

Available			
A	B	C	D
10	11	8	7

	Allocation				Max				Need			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	Finished in state 1											
P1	Finished in state 4											
P2	2	1	0	3	2	3	1	6	0	2	1	3
P3	Finished in state 2											
P4	Finished in state 3											

State 5:

Available			
A	B	C	D
12	12	8	10

	Allocation				Max				Need			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	Finished in state 1											
P1	Finished in state 4											
P2	Finished in state 5											
P3	Finished in state 2											
P4	Finished in state 3											

The order of execution is: P0, P3, P4, P1, P2

7.24: The optimistic assumption is that resource allocation will not occur often because the deadlock-detection algorithm will need to run every time a resource is allocated to ensure no deadlocks will occur. This assumption could easily be violated by a system that need to allocated resource often. This will cause the algorithm to be ran often and will cripple the runtime of the system.

7.25:

```
monitor BridgeCrossing
{
        enum{WAITING, CROSSING, FARMING} state[2];
        condition town[2];

        void enter_bridge(int i) {
                state[i] = WAITING;
                test(i);
                if (state[i] != CROSSING)
                        town[i].wait();
        }

        void leave_bridge(int i) {
                state[i] = FARMING;
                test((i+1)%2);
        }

        void test(int i) {
                if ((state[(i + 1) % 5] != CROSSING) &&
                        (state[i] == WAITING)) {
                        state[i] = CROSSING;
                        self[i].signal();
                }
        }
}
```