Solutions
15-410, Fall 2015, Homework Assignment 1.

1 Tape drives (4 pts.)

Imagine the system is in the state depicted below. List one request which the system should grant right away, and one request which the system should react to by blocking the process making the request. Briefly justify each of your answers. Also, briefly describe something that is odd about using deadlock avoidance for this set of applications.

Who	Max	Has	Room
A	3	1	2
B	3	0	3
System	3	2	-

If Process A requests one tape drive, the request should be granted, because in that state it would be possible for Process A to request its third tape drive, run to completion, and then Process B could definitely run to completion. That is, if Process A requests its second tape drive, the request should be granted because of the safe sequence (A,B).

In the other direction, if Process B requests one tape drive, the system should block Process B before granting the request, because granting the request would move the system to an unsafe state as depicted below.

Who	Max	Has	Room
A	3	1	2
B	3	1	2
System	3	1	-

This state is unsafe because the system has only one free tape drive, but each process is entitled to request more than one. If the system were in this state, it might well happen that some process would exit without requesting anything further. However, it could also happen that both Process A and Process B would request another tape drive. In that case, the system would have no choice but to block both of them; because each of the processes would be blocked waiting for tape drives held by the other, that would form a deadlock. Formally speaking, in the state depicted above no process can request resources up to its maximum without blocking, so no process can request resources up to its maximum and complete in a timely fashion, thus there is no first item in a safe sequence, thus no safe sequence can exist.

The odd thing about using deadlock avoidance for this set of processes is that the only safe way for the system to operate is without hold&wait: whichever process has any request granted first must get all the other resources, i.e., the first allocation effectively forces the other allocations, so allocation is effectively all-at-once, which is the opposite of hold&wait, so for these two processes deadlock avoidance works the same way as deadlock prevention, except that the resource manager does pointless computations.

(Continued on next page)

2 “Go Ahead” (6 pts.)

```
int want[2] = { 0, 0 };
int goahead[2] = { 0, 0 };

do {
   ...remainder section...
   want[i] = 1;
   if (want[j]) {
     if (i == 0) // tie breaker
       goahead[j] = 1;
   }
   while (want[j] && !goahead[i])
     continue;
   ...begin critical section...
   ...end critical section...
   want[i] = 0;
   goahead[i] = 0;
} while (1);
```

There is a problem with this critical-section protocol. Identify a required property which this protocol does not have and then present a trace which supports your claim.

This protocol does not ensure progress (despite the fact that it was designed to!).

Execution Trace

time	Thread 0	Thread 1
0	want[0] = 1	
1	want[1]? (no)	
2		want[1] = 1
3		want[0]? (yes)
4	while (1 && !0)	i==0? (no)
5	while (1 && !0)	while (1 && !0)
6	while (1 && !0)	while (1 && !0)
7	...	...

If you believe this protocol also doesn’t ensure one of the other properties, you are probably right. For example, it also doesn’t provide bounded waiting. Just make sure that all of the things you write in your trace can actually happen. In particular, if you assert a loop will run indefinitely, it must be the case that the loop condition obviously can be true indefinitely.

In terms of difficulty, this problem is probably a little easier than an exam question on a similar topic. However, this question represents the sort of reasoning we expect you to carry out.