15-410, Fall 2015, Homework Assignment 1.

Due Monday, October 12, 20:59:59 p.m.

Please observe the non-standard submission time... As we intend to make solutions available on the web site immediately thereafter for exam-study purposes, please turn your solutions in on time.

Homework must be submitted in either PostScript or PDF format (not: Microsoft Word, Word Perfect, Apple Works, LaTeX, XyWrite, WordStar, etc.). Submit your answers by placing them in the appropriate hand-in directory, e.g., `/afs/cs.cmu.edu/academic/class/15410-f15-users/$USER/hw1/$USER.ps` or `/afs/cs.cmu.edu/academic/class/15410-f15-users/$USER/hw1/$USER.pdf`. A plain text file (.text or .txt) is also acceptable, though it must conform to Unix expectations, meaning lines of no more than 120 characters separated by newline characters (note that this is *not* the Windows convention or the MacOS convention). Please avoid creative filenames such as `hw1/my_15-410_homework.PdF`.

1 Tape drives (4 pts.)

Consider a system with three processes and three tape drives. The maximal needs of each process are declared below:

Resource Declarations

Process A
3 tape drives

Process B
3 tape drives

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	-

(Continued on next page)

2 "Go Ahead" (6 pts.)

Consider the following critical-section protocol, which is based on the "registering interest" approach presented in our "Synchronization #1" lecture. As you may recall, that attempt suffered from the problem that if the two threads declared interest "at the same time" a progress failure would result. This code addresses that issue with a per-thread "go ahead" signal (set by each thread if it observes that the other thread wants to enter). Of course, it is possible that the two threads might "simultaneously" set their "want" flags and then "simultaneously" observe each other's "want" flags; the code below uses thread i.d.'s as a tie-breaker.

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

 1.    do {
 2.       ...remainder section...
 3.       want[i] = 1;
 4.       if (want[j]) {
 5.         if (i == 0) // tie breaker
 6.           goahead[j] = 1;
 7.       }
 8.       while (want[j] && !goahead[i])
 9.         continue;
10.       ...begin critical section...
11.       ...end critical section...
12.       want[i] = 0;
13.       goahead[i] = 0;
14.    } while (1);
```

(This protocol is presented in "standard form," i.e., if thread 0 is running this code, $i == 0$ and $j == 1$; if thread 1 is running this code, $i == 1$ and $j == 0$.)

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. You may use more or fewer lines in your trace.

Execution Trace

time	Thread 0	Thread 1
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		