Solutions
15-410, Fall 2005, Homework Assignment 1.

1 Run Away (10 pts.)

1.1 5pts

Explain roughly how `flee()` works.

The job of `flee()` is to return execution to the point where `we_fled()` was first called, with a non-zero return code. Looking at it another way, `flee()` is supposed to cause `we_fled()` to "return" a second time. Thus the stack needs to be the same height as if `we_fled()` were returning, the program counter needs to be set to `we_fled()`'s return address, and callee-save registers need to be restored to the correct values. Oh, yes, `%eax` needs to be something other than zero. Overall, a bunch of registers need to be set to particular values and a `ret` instruction needs to be executed.

1.2 5pts

What are the fields of a `flight_path`? Be specific if you can.

You need `%esp`, `%ebp`, `we_fled()`'s return address, and the values of the callee-save registers. You may include some sanity-checking information to catch certain invalid invocations of `flee()`, and it's up to you whether or not you believe signal-handling information should be included (it's been done both ways in `setjmp()/longjmp()`).

2 Racy (15 pts.)

This question is due to Dijkstra via Nutt.

You should show mutual exclusion doesn't hold with an execution trace.

Execution Trace

time	Thread 0	Thread 1
0	3:`critical[1]==0`	
1		3:`critical[0]==0`
2	5:`critical[0]=1`	
3		5:`critical[1]=1`
4	6:`...begin...`	
5		6:`...begin...`

You can argue that progress holds as follows. Since the polling loop at lines 3 and 4 only tests values but doesn't modify them, the only way for the system to get stuck there with nobody entering the critical section would be for both "critical" flags to be on simultaneously (which would be a deadlock, certainly no progress possible). But there is no way for a particular process to enter the loop with its own flag on–the flags are initially zero and are always set to zero by each process at the end of its critical section, which is the same thing as before the loop. So if both processes are in the loop then both flags are *off*, and both will proceed into the critical section, which is what we were complaining about above.

Bounded waiting does not hold, since one thread can enter over and over as long as the other thread samples at just the right/wrong time. That is, you can run the execution trace below an infinite number of times back-to-back and Thread 0 will never enter the critical section.

Execution Trace

time	Thread 0	Thread 1
0		3:`critical[0]==0`
1		5:`critical[1]=1`
2		6:`...begin...`
3		6:`...end...`
4	3:`critical[1]==1`	
5		5:`critical[1]=0`

3 Fascinating (10 pts.)

3.1 4 pts

Is the following state safe? Why or why not?

A safe sequence is: give the remaining tape drive to Process A, at which point you have two tape drives to satisfy Process B's possible request for one, at which point you have three drives to satisfy Process C's possible request for two. So the state is safe.

3.2 4 pts

Is the following state safe? Why or why not?

There is no safe sequence. Observe that any requesting process must sleep, since there are no free resources. Observe further that all three processes are allowed to request. If all do, then each will be holding non-preemptible mutual-exclusion resources while waiting for each other.

Of course, if only two request, and the third exits without requesting its second tape drive, then there will not be a deadlock–this time.

3.3 2 pts

Is there something odd about this system?

It depends on your perspective, but *safety* effectively reduces the amount of parallelism available in the hardware from three concurrent processes' worth to two. If you run a deadlock-avoidance allocator, one process will always be idle.

Assuming that the processes *typically* use one tape drive and only *rarely* need the second one, a deadlock-avoidance resource manager will rarely allocate the third tape drive. Your manager will walk into the machine room and see two tape drives spinning and one idle, and one process idle, and will probably be unhappy even after you draw lots of circles and arrows. You can argue that all you need to get the third tape drive spinning is for your manager to buy you a *fourth* one... but when should you mention that number four will hardly ever spin?

Of course, utilization is even worse if a simpler approach, deadlock prevention in the form of all-at-once allocation, is used.