Computer Science 15-410: Operating Systems
Mid-Term Exam (B), Spring 2005

1. **Please read the entire exam before starting to write. This should help you avoid getting bogged down on one problem.**
2. Be sure to put your name and Andrew ID below *and also* put your Andrew ID at the top of *each* following page.
3. This is a closed-book in-class exam. You may not use any reference materials during the exam.
4. If you have a clarification question, please write it down on the card we have provided. Please don't ask us questions of the form "If I answered like this, would it be ok?" or "Are you looking for ...?"
5. The weight of each question is indicated on the exam. Weights of question *parts* are estimates which may be revised during the grading process and are for your guidance only.
6. Please be concise in your answers. You will receive partial credit for partially correct answers, but truly extraneous remarks may count against your grade.
7. **Write legibly even if you must slow down to do so!**. If you spend some time to *think clearly* about a problem, you will probably have time to write your answer legibly.

Andrew Username	
Full Name	

Question	Max	Points	Grader
1.	10		
2.	20		
3.	10		
4.	15		
5.	20		
	75		

I have not received advance information on the content of this 15-410 mid-term exam by discussing it with anybody who took part in a conflict exam session or via any other avenue.

Signature: ____ Date ___________________________

1. $\boxed{\text{10 points}}$ Short answer.

   (a) $\boxed{\text{4 points}}$ How are “user mode” and “kernel mode” different? List up to four differences (try to rank them from most important to least important).

Give a *brief* definition of each of the following terms as they apply to this course. You may add a sentence providing an example or a clarification if you wish, but there is no need for long answers.

(b) $\boxed{\text{2 points}}$ runnable

(c) $\boxed{\text{2 points}}$ progress

(d) $\boxed{\text{2 points}}$ bus lock

2. $\boxed{\text{20 points}}$ Deadlock

A new server clustering architecture, named ClusterF (a successor to the ill-fated ClusterA through ClusterE protocols), is developed. ClusterF connects up to 32 machines via a high-speed ring-shaped network (each node is connected to a clockwise neighbor and a counter-clockwise neighbor). The ring allows traffic in both directions.

A small switch connects each node to its two neighbors; its primary job is to scan incoming traffic from both directions to decide whether each incoming packet is addressed to its node and should be captured or should instead be forwarded onward to the neighbor on the other side.

Due to DWDM, fiber optics, and other network mumbo-jumbo, the link speed is effectively limitless, but the switching hardware connecting each node to its two neighbors can handle only a total of 10Gb/s traffic (as a total in both directions). Control messages use a special low-throughput, high-reliability, high-priority channel, so you may assume that control messages always arrive at their destinations and are processed immediately.

The ClusterF networking protocol allows reservation of fixed one-directional "circuits" between any pair of machines. Any circuit established by this protocol can run in either direction around the ring, can specify a guaranteed bandwidth up to the maximum bandwidth possible through the ring (10Gb/s) and is guaranteed to stay connected until the originating node releases the circuit or something catastrophic happens (e.g., server room catches fire, melting optical fiber). If a circuit is established at a given bandwidth it cannot be closed by the system nor can its bandwidth be reduced. These properties of the system are part of the ClusterF API and must be guaranteed.

In order to establish a one-directional circuit between two machines, the protocol works as follows:

Protocol to establish a circuit from node N_{start} to node N with bandwidth guarantee B:

1. Determine the appropriate direction around the network ring (shortest path; clockwise if distance around network ring is a tie).
2. Set `next_node` to be node N_{start}
3. If `next_node` is node N, return success
4. Set `next_node` to be `next_node`'s neighbor in the direction determined in the first step.
5. Send a request to `next_node` to allow our existing partially-set-up circuit to add an incoming leg with bandwidth B to `next_node`. `next_node` will grant our request if its switch has enough remaining processing ability (i.e., wouldn't be processing more than a total of 10 Gb/s worth of incoming traffic) and will reject it otherwise.
   (a) If `next_node` grants our request, go to Step 3 (success check).
   (b) If `next_node` denies our request, sleep for a short period and return to Step 5 ("Send a request").

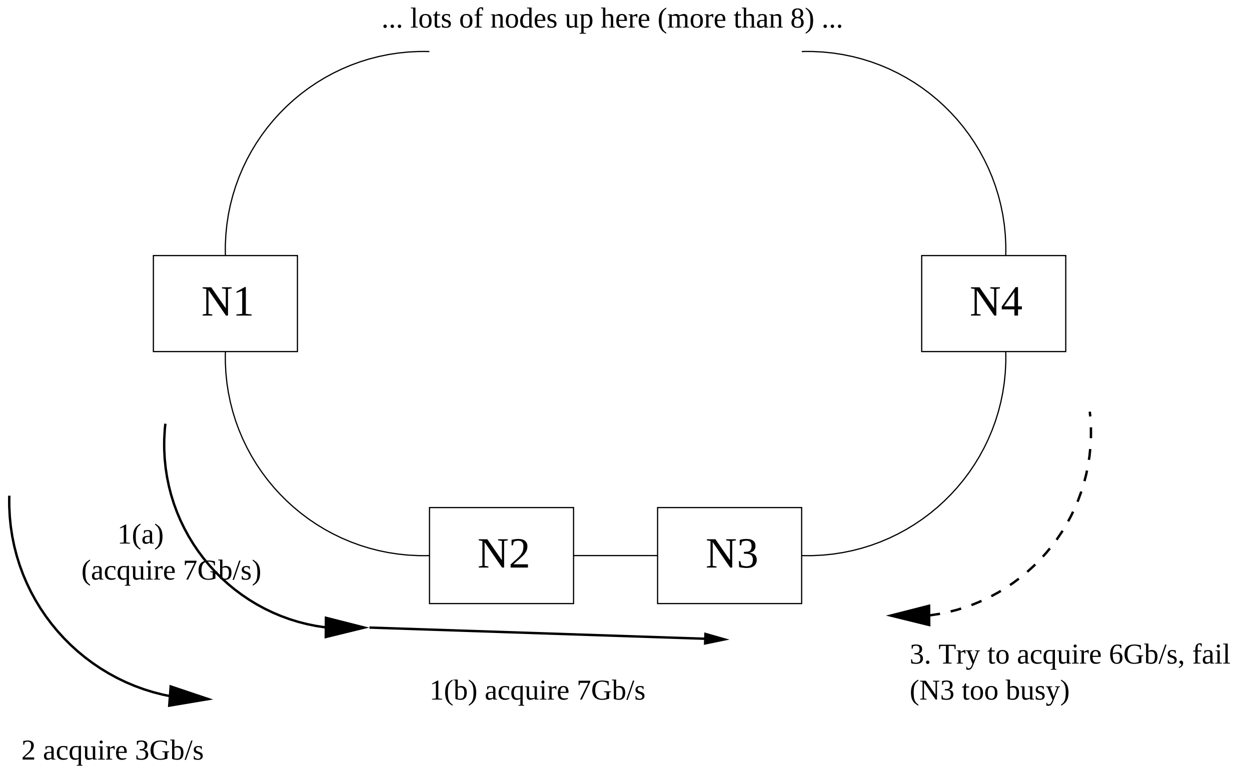

Figure 1: Network (shown at Step 3)

Here is an example of the protocol in operation:

1. N1 asks for a 7Gb/s link to N3.
   (a) First, a circuit C1 of 7Gb/s is established from N1 to N2.
   (b) Second, circuit C1 is extended from N2 to N3.
2. N1 asks for a 3 Gb/s link to N2 (circuit C2). This is granted as before, as there is an appropriate amount of available capacity in the network.
3. N4 requests a 6 Gb/s link to N2 (circuit C3). This request is denied as N3 is already switching 7Gb/s and is incapable of adding another network flow of 6 Gb/s. Thus, circuit C3 cannot be established until C1 is released.
4. N1 releases C1, freeing 7Gb/s of switching capacity at N1, N2 and N3.
5. N4 may now establish a circuit to N2 via N3.

It is observed that the ClusterF system frequently deadlocks, and your manager asks you to investigate the feasibility of fixing the problem using *deadlock prevention*.

(a) $\boxed{\text{4 points}}$ What are the four *deadlock prevention* methods?

(b) $\boxed{\text{16 points}}$ For each of the four methods, describe whether it can feasibly be used to prevent deadlocks in the ClusterF system as it is described above. You may outline changes to the circuit-setup protocol but you may not change the definition of the ClusterF API (for example, you may not allow the system to reduce bandwidth of connections or break guaranteed connections).

For some of the methods you need provide only one to three sentences of explanation; for others a paragraph or two should suffice (and it should be clear to you which are which).

You may use this page as extra space for the `ClusterF` question if you wish.

3. $\boxed{\text{10 points}}$ Interrupt handling.

The new iSpend portable music player (an expensive device that forms the core of Professor Maggs' next startup) requires a small, Unix-like embedded operating system running on a low-power x86 processor. While the price of this music player is stunningly high, the cost of the components must be minimized so that everyone involved with the startup can get very rich. As part of this cost-minimization, all memory sizes are to be kept to a minimum. The kernel for the music player allocates just 4 Kbytes for thread stacks. The lead engineer on the project justifies this as follows:

> I have examined the function call graph starting from each system call, and the largest possible stack depth ignoring interrupts is exactly 3 Kbytes.
>
> The interrupt descriptor table contains 4 trap gates for handling interrupt-driven tasks such as refilling the audio output buffer, time and alarm clock management, user button presses, and power management. The largest of the 4 consumes 512 bytes of stack space and each of the other three consumes 256 bytes of stack space.
>
> Thus the maximum amount of kernel stack memory required is
>
> $$3K + \max(512, 256, 256, 256) = 3.5K$$

Do you agree with the lead engineer? Be sure to justify your judgement.

4. $\boxed{\text{15 points}}$ Process model

As you were working away in the cluster on your Project 3 implementation yesterday, you overheard two other 15-410 students discussing the exploits of the renowned hacker Claire Tokar. Apparently in her spare time she "ported" her Project 0 `traceback()` code into the Project 1 run-time environment just for fun. As your classmates tell the story, while Claire needed to make some unauthorized changes to the P1 Makefile to link in the traceback library and invoke `symtabgen.pl` on the kernel executable, the bottom line is that one ELF binary is much like another.

Later in the day, you find it increasingly difficult to concentrate on P3 and start to wonder if you might be able to justify "investing" a little time to get `traceback()` running in the P3 environment, or at least trying to add it to your P1 assignment as a stepping-stone toward a possible deployment later in your kernel project. Just before the urge to start typing sweeps you away, you realize that there's a problem: a critical part of your `traceback()` code *won't* just drop into your P1 code base.

(a) $\boxed{\textit{5 points}}$ *Briefly* summarize which critical part of a working Linux-process `traceback()` implementation would be challenging to port to the P1 run-time environment. If you can think of multiple parts which would be difficult, choose the most important one or two (no more).

(b) $\boxed{\text{10 points}}$ Explain how you would achieve or approximate the same result in the P1 run-time environment. If you can't provide an exactly correct solution, be sure to identify where you are making an approximation (and strive to make a good one).

You may use this page as extra space for the `traceback()` question if you wish.

5. $\boxed{\text{20 points}}$ Consider the following critical-section protocol:

```
boolean waiting[2] = { false, false };
int turn = 0;

do {
    waiting[i] = true;
    while (waiting[j]) {
        waiting[i] = false;
        while (turn == j)
           continue;
        waiting[i] = true;
    }
   ...begin critical section...
   ...end critical section...
   turn = j;
   waiting[i] = false;
   ...begin remainder section...
   ...end remainder section...
} while (1);
```

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

There is a problem with this protocol. That is, it does not ensure that all three requirements (mutual exclusion, progress, and bounded waiting) are always met. Identify a requirement which is not met and lay out a scenario which demonstrates your claim. Use the format presented in class, i.e.,

P0	P1
waiting[0] = false;	
	turn = 0;

You may use this page as extra space for the critical-section protocol question if you wish.