Computer Science 15-410: Operating Systems
Mid-Term Exam, Fall 2003

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. You must complete the exam by the end of the class period.
5. Answer all questions. 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. Please don't ask us questions of the form "If I answered like this, would it be ok?" or "Are you looking for ...?"

Andrew Username	
Full Name	

Question	Max	Points	Grader
1.	10		
2.	10		
3.	10		
4.	20		
	100		

1. [10 points] Give a *brief* definition of each of the following terms as they apply to this course. You may add a second sentence providing an example or a clarification.

   (a) [2 points] Register

   (b) [2 points] Stack

   (c) [2 points] Program counter

(d) [2 points] System call

(e) [2 points] Preemption

2. [10 points] What will happen if the following code is executed in kernel mode?

```
void foo(void)
{
    while (1)
        (void) malloc(1);
}
```

If you feel that multiple scenarios are possible, please pick one and focus on its details. You may wish to break your answer into several parts, such as "At first, ...", "After some time..." and/or "Eventually...".

3. [10 points] ???

4. [20 points] Condition-variable implementation

Imagine your company ships arcade-style (coin-operated) video games which run your proprietary operating system kernel, library code, and applications. Your boss tells you that too much time is being spent in cond_wait(). Furthermore, it has been determined that the slowdown is due to a malloc() call and that they want to remove the necessity for a call to malloc() to obtain a queue entry. You suggest bounding the number of threads waiting on a condition variable and using an array-based queue, but your boss claims no reasonable bound exists. Instead, you are asked to investigate whether the system-call interface can be changed to turn the queueing job over to the kernel.

The system call interface you begin with is:

- `int deschedule( int *reject )` - Examines the integer pointed to by `reject`. If the integer is non-zero, the call returns immediately with return value zero. If the integer pointed to by `reject` is zero, then the calling process will be suspended (not run by the scheduler) until some other process makes a call to `make_runnable()` on the process that called `deschedule()`. An integer error code less than zero is returned if `reject` is not a valid pointer. This system call is *atomic* with respect to `make_runnable()`: the process of examining `reject` and suspending the process will not be interleaved with any execution of `make_runnable()` by another process.
- `int make_runnable( int pid )` - Makes the `deschedule()`d process with process ID `pid` runnable by the scheduler. On success, zero is returned. If `pid` is not the process ID of a process suspended due to having called `deschedule()`, then an integer error code less than zero is returned.

(a) [10 points] Briefly and clearly sketch out your new `deschedule()` and `make_runnable()` system calls. You may add, delete, or modify the existing parameters. Be sure to choose good names and brief descriptions for your new parameters. Make sure your proposed changes do not dramatically increase the cost (time or memory) of these system calls.

(b) [5 points] Show how cond_wait() and cond_signal() would use your modified system call interface. Remember that the goal is for a condition variable to be a small constant size with no variable-length memory use.

(c) [5 points] Briefly explain why it is that your system call changes will not require the kernel to allocate more memory than it plausibly was before (in particular, we are not trying to move the malloc() into the kernel!).