OPERATING SYSTEMS CPS 510, SPRING 2016

Out: Monday, January 6, 9:00
Due via Hardcopy: Monday, January 6, 12:00

This is a closed-book, no-computer exam. You must answer the questions on your own without any external assistance.

Read the entire exam through before you begin working. Read each question carefully, and note all that is required of you. Keep your answers *clear and concise*. Often when you write more than you need you end up saying things that are incorrect. I don't expect you to need more than a page for any question.

You are to abide by the Duke University honor code. Additionally, you may not consult with any other person about any part of this exam until the exam period ends at 12:00, Monday. This is true even if you have handed in your exam. Please sign this page to indicate that you have completed your exam within these rules. Exams without corresponding, signed cover pages will not be graded

Name:

Signature:

1. Airport taxi line

Consider an airport taxi line. Passengers wait in FIFO queue to take a taxi. Taxis wait in FIFO queue to pick up passengers. A dispatcher assigns the first passenger in the passenger queue to the first taxi in the taxi queue. After assigning a passenger to a taxi, the dispatcher checks to see if there are any more taxis or passengers. If no passengers are available or no taxis are available, the dispatcher must wait before assigning the next passenger to the next taxi.

Simulate a taxi line by writing pseudo-code for the passenger threads (of which there are an arbitrary number), the dispatcher thread (of which there is only one), and the taxi threads (of which there are an arbitrary number). All threads are initialized elsewhere; passenger and taxi threads are initialized with a unique identifier. Once a passenger has been assigned to a taxi, the taxi takes the passenger to her destination by calling the **gotoDest()** function. Only after **gotoDest** returns may the calling taxi thread and its assigned passenger thread exit.

You may use either monitors or semaphores to synchronize the activities of the threads.

```
// STRUCTS AND GLOBAL VARIABLES GO HERE
```

```
Passenger (int passID)
{









}
```

```
Dispatcher ()
{
	while (1) {









	}
}
```

```
Taxi (int taxiID)
{

  gotoDest ();

}
```

2. Atomicity

For each sub-question below, I will only count the first four sentences of your answer.

a) What does it mean for an instruction to be *atomic* with respect to another instruction?

b) Describe a special instruction that allows programmers to easily define regions of code that are atomic with respect to each other?

c) Why are locks sufficient for defining atomic regions of code, but not for defining atomic sets of updates to a storage system?

d) Describe an example file-system update that demonstrates why file systems must write blocks to disk in the correct order to prevent corruption.

e) Describe an example situation that demonstrates why carefully ordering a series of disk writes is not always sufficient to prevent corruption. Note that this need not be a file system operation.