Name:

Operating Systems (COP 6611)
Spring 2012
Midterm Exam

Closed books, notes, cell phones, PDAs, iPods, laptops, etc. No headphones, please. You do not need a calculator. You may use scratch paper that we provide.

You have 75 minutes to solve 6 problems, each worth 15 points. You receive a *too-long-until-spring-break* gift of 10 points just by signing your name on your exam paper. Total points: 100.

As with any exam, you should read through the questions first and start with those that you are most comfortable with. Answers will be graded for substance and clarity. Redundancy and vague generalities are considered bad form. Partial credit will be offered for meaningful progress towards solving the problems.

1	
2	
3	
4	
5	
6	
Total	

1. Explain the difference between monolithic (traditional) kernels and microkernels by presenting some concrete examples of OS functionality.

2. List at least 5 resource management challenges introduced by virtualization and discuss them in the context of Xen and VMWare ESX papers. For each, please state clearly the challenge and describe the solutions proposed in the paper to address it.

3. Threads:
    a. List two main differences between user-level threads and processes.
    b. Typical implementation of a thread package works as follows. The package provides user-level thread abstractions to the application, allowing cheap creation and destruction of threads. The package asks the kernel for a number of light-weight processes (also known as kernel-level threads). Typically each light-weight process corresponds to one processor in the system. The threads package performs thread scheduling by binding user-level threads to light-weight processes. Explain why such a scheme is unsatisfactory from a user application's point of view, particularly for those applications that wish to precisely control the schedule of threads to processors.
    c. Suggest some enhancements to kernel mechanisms (including new system calls or upcalls) that correct this problem.

4. Below is the solution to the producer/consumer problem using semaphores with one modification: the down operations in the Producer are inversed. That is, down(&mutex) is now before down(&empty)). What will happen in this case?

```
#define N 100                           /* number of slots in the buffer
*/
typedef int semaphore;                  /* semaphores are a special kind
of int */
semaphore mutex = 1;                    /* controls access to critical
region */
semaphore empty = N;                    /* counts empty buffer slots */
semaphore full = 0;                     /* counts full buffer slots */

void producer(void)
{
     int item;

     while (TRUE){                      /* TRUE is the constant 1 */
          item = produce_item();        /* generate something to put in
buffer */

          down(&mutex);                 /* enter critical region */
          down(&empty);                 /* decrement empty count */
          insert_item(item);            /* put new item in buffer */
          up(&mutex);                   /* leave critical region */
          up(&full);                    /* increment count of full slots
*/
     }
}

void consumer(void)
{
     int item;

     while (TRUE){                      /* infinite loop */
          down(&full);                  /* decrement full count */
          down(&mutex);                 /* enter critical region */
          item = remove_item();         /* take item from buffer */
          up(&mutex);                   /* leave critical region */
          up(&empty);                   /* increment count of empty
slots */
          consume_item(item);           /* do something with the item */
     }
}
```

5. Deadlock is a potentially nasty problem one encounters in multi-threaded programs.
    a. Please list the necessary conditions for a deadlock to occur.
    b. Can the following approaches guarantee that deadlock will be avoided or recovered from? Please explain, using examples to support your answers.
        i. Never do context switch inside a critical section.
        ii. Whenever a deadlock occurs, roll back the program to an earlier checkpoint outside any critical section and re-execute the program.

6. Memory:
    a. At one time, virtual memory system designers were advised to bias page replacement algorithms against modified pages in favor of those that have not been modified. The result of this suggestion was the unfortunate behavior of program code pages (which tend to be some of the only pages that are not modified) being kicked out of memory before other pages. Describe the rationale for the original suggestion of paging unmodified pages first.
    b. Would it ever make sense for two processes to share the same page table? Explain your answer.
    c. How many page tables are used by a process with multiple threads? Justify your answer.