Name:

Operating Systems (COP 6611) Spring 2010 Midterm Exam

Closed books, notes, cell phones, PDAs, iPods, laptops, etc. No headphones, please. You do not need a calculator.

You have 75 minutes to solve 5 problems (first 4 worth 20 points each, the last 10 points). You receive a 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	
Total	

1. List three broad design goals of the ESX memory management system, and briefly outline how the design meets those goals.

2. In class we discussed several research operating systems that offer an alternative to the classical *monolithic kernel* structure, in which the OS kernel manages system resources according to predefined policies (e.g., Unix). Notable examples are: the microkernel, Exokernel, and virtual machines (Xen). Rank these alternatives with respect to their fulfillment of each of the goals listed below (best first). Your answer should consist of a rank list with three entries for each of the four goals and a brief explanation of your choice; you may group approaches that are equivalent with respect to some goal by joining them with a line or bracket.

   The goals are:
   a. Support for safe sharing of data and resources across processes;
   b. Support for multiple OS personalities or APIs (e.g., Unix and Windows syscall interfaces in the same system);
   c. Support for application control over physical resource management;
   d. Overhead of system extensions.

3. Consider a system implementing multilevel feedback queue scheduling. What strategy can a computer user employ to maximize the amount of CPU time allocated to the user's process?

4. Your OS has a set of queues, each of which is protected by a lock. To enqueue or dequeue an item, a thread must hold the lock associated to the queue.

   You need to implement an atomic transfer routine that dequeues an item from one queue and enqueues it on another. The transfer must appear to occur atomically.

   This is your first attempt:

```
void transfer(Queue *q1, Queue *q2)
{
     Item thing; /* the thing being transferred */

     q1->lock.Acquire();
     thing = q1->Dequeue();
     if (thing != NULL){
          q2->lock.Acquire();
          q2->Enqueue(thing);
          q2->lock.Release();
     }
     q1->lock.Release()
}
```

You may assume that `q1` and `q2` never refer to the same queue. Also, assume that you have a function `Queue::Address` that takes a queue and returns, as an unsigned integer, its address in memory.

a. Explain how using this implementation of `transfer` leads to deadlock;
b. Write a modified version of `transfer` that avoids deadlock and does the transfer atomically;
c. If the transfer does not need to be atomic, how might you change your solution to achieve a higher degree of concurrency? Justify how your modification increases concurrency.

5. How many outputs are possible when running the process below with argument 4? Explain your answer.

```
#include  <pthread.h>
#include  <stdio.h>

int sum;
void  *runner(void *param);

main(int argc, char  *argv[]) {
    pthread_t tidl,  tid;
    pthread_attr_t  attr;
    if (argc != 2)  {
      fprintf(stderr,"usage:  a.out <integer_value>\n");
      exit(0);
    }

    if  (atoi(argv[1]) < 0)  {
      fprintf(stderr, "%d must be >= 0\n",atoi(argv[1]));
      exit(0);
    }
    pthread_attr_init(&attr);

    /*  create the threads*/
    pthread_create(&tid,&attr,&runner,argv[1]);
    pthread_create(&tidl,&attr,&runner,argv[1]);
    pthread_join(tid,NULL);
    pthread_join(tidl,NULL);

    printf("sum =%d\n",sum);
  }

void *runner(void *param) {
  int upper = atoi(param);
  sum = 0;
  int  i;
  if (upper > 0) {
    for (i=1; i<=upper;i++)
     sum+=i;
  }
  pthread_exit(0);
}
```