Student ID: _____

CS-736 Midterm: Live and Let Die
(Fall 2003)

Please Read All Questions Carefully!

There are twelve (12) total numbered pages

Please put your NAME on this page, and your STUDENT ID on this and all other pages

Name: _____

Student ID: _____

This is the grading page.

	Points	Total Possible
Q1		20
Q2		20
Q3		20
Q4		20
Q5		20

Student ID: _____

1. **Virtual Machines**

   Disco is an example of a virtual machine monitor (VMM), which runs underneath of an unsuspecting operating system (SGI's IRIX) in order to enhance service or provide some kind of functionality that would otherwise be unavailable. In this question, we will explore VMM technology.

   1: First, describe the flow of control when an application running on a typical operating system (IRIX) running on a VMM (Disco) issues a system call (a trap). What pieces of code get invoked? (a picture is worth a thousand words, so save some ink and draw one)

   2: Now let's imagine we are considering a TLB miss. What typically happens on a TLB miss? (again, consider a picture of control flow).

3: What does Disco do to speed up the operation of TLB misses? Why can't the OS itself do the same optimization? (or can it?)

4: Finally, assume that the operating system is no longer "unsuspecting" of the change – rather, assume we are building the OS with the knowledge that it will be run upon a VMM, and that the VMM can be changed in small ways as well. Describe *at least two* changes that could be made to increase the overall efficiency of such a system.

2. **Mirrored File Systems (FFS, LFS, RAID)**

   In this question, we explore the utilization of a hybrid file system that is built on top of two existing file systems, namely FFS and LFS.

   1: Before getting into details of design, describe a workload in which FFS would do much better than LFS.

   2: Now, describe a workload in which LFS would do much better than FFS.

Now assume we are building a hybrid file system on top of FFS and LFS (call it FLFS). Assume we have a two-disk system (for simplicity), and that conceptually we are running FFS on one disk and LFS on the other. Your job is to design a file system that directs reads and writes to one or the other or both of these file systems.

3: How might we direct reads and writes in the system so as to get the best *performance* out of the system? What issues arise, and how might you address them?

4: How might we direct reads and writes in the system so as to get the best *reliability* out of the system? What issues arise, and how might you address them?

3. **RAID-6**

In this question, we consider a new RAID level, called RAID Level 6, and we compare it to RAID Level 5. RAID-6 works as follows: instead of having just one check disk, RAID-6 has *two* check disks. Without getting into the details of how Reed-Solomon coding works, assume that if the data in a stripe changes, *both* check blocks will have to be updated too. Also, assume that the basic design allows the loss of any *two* disks without a perceived loss of data.

1: First, let's make sure we understand RAID Level 5. How does RAID-5 work (pictures are useful)?

2: What is the main performance problem with RAID-5, as related to "small" writes?

3: Now, let's understand the same issue for RAID-6. How would a "small" write work on RAID-6?

4: What is the performance impact of RAID-6, as compared to RAID-5? When should a storage administrator decide to use it instead of RAID-5?

4. **Gray boxes**

In this question, we explore the concept of the "gray box" approach to building systems. Assume we wish to build a gray-box layer on top of the file system that gives us more control over where files get allocated on disk. Assume further that we are running on top of FFS.

1: How does FFS decide where to place files on disk?

2: How could we exploit knowledge of this algorithm in order to give applications the ability to better control which files end up near one another on disk?

Now let's think about a gray-box approach to a lower-level of the disk system. Assume we are running on top of a RAID system, but we don't know how it's configured. Specifically, it could be RAID Level 0 (striping), RAID Level 1 (Mirroring), or RAID Level 4 (Parity-based).

3: Design a benchmark that figures out which of these three RAID levels you are running on. Assume you can do reads and writes to the disk's address space directly (these are called "raw" reads/writes), and also assume that you know how many total disks are in the system. Remember that you can use *timing* to elicit information about what is going on.

5. **Memory Activations**

In this question, we explore the generalization of the concept of scheduler activations to other resources in the system, specifically memory. Your job is to design this *memory activations* system.

1: Before getting into memory activations, let's discuss some of the key points of scheduler activations. One of the keys to scheduler activations is the notification to the user-level thread library when something changes in the system. When do these kernel-to-library notifications occur, and why is it useful for the user-level library to know of such changes?

2: Now imagine that we wanted to apply the activation concept to memory. What kind of information should be *passed up* from the kernel to the user-level library/application?

3: Similarly, in our memory activation system, what type of information should be *passed down* to the kernel from the application?

4: Why is applying the activation concept different for memory than it was for the CPU? What works better, what doesn't work as well? Discuss.