UNIVERSITY OF WISCONSIN-MADISON
Computer Sciences Department

Operating Systems Depth Exam **Fall 2002**

Instructions: There are *seven* questions on this exam; answer *any six* of the following seven questions. This exam has three (3) pages.

Question 1. Exokernel:

This question considers the exokernel approach to operating system design.

1A. The exokernel introduces the concept of a *library operating system* (libOS). Discuss what a libOS is, and why it is important to the exokernel approach (a picture is worth a thousand words, it has been said). Which pieces of functionality found in a typical OS would be placed within a libOS, and which pieces within the exokernel itself?

1B. Consider CPU scheduling in the exokernel. Given mutually untrusting applications, how are they scheduled by the exokernel? Can you think of a workload where the exokernel scheduler would perform poorly, as compared to a typical scheduler found in a modern Unix-based system?

1C. Sometimes an exokernel has to use "downloaded code", i.e. code loaded into the exokernel from an untrusted application. What are the main issues that must be dealt with in order to allow downloaded code within the exokernel? Could downloaded code be useful in improving the performance of the exokernel CPU scheduler, as discussed in part (1B)?

Question 2. I/O Lite:

This question probes the principles behind I/O-Lite, a unified I/O buffering and caching system.

2A. In I/O-Lite, all I/O data buffers are said to be *immutable*. Discuss what this means, and why it is important to I/O-Lite.

2B. To provide more flexibility to applications, I/O-Lite introduces the concept of a *buffer aggregate*. What is a buffer aggregate, and how is it useful in the I/O-Lite context?

2C. Describe how data moves through a typical OS when running a web server that is serving static web documents. How many data copies might one typically expect to see? Now contrast this with a web-server built on top of I/O-Lite. What kinds of optimizations can the I/O-Lite web server implement to improve performance?

Question 3. DISCO:

Virtual machine monitors have traditionally suffered from time overheads, space overheads, and poor

resource management. Disco contains features that reduce or eliminate each of these problems.

3A. Describe the general reason why each of these three problems occur in a virtual machine monitor. Give short examples that illustrate each of the three problems.
3B. Discuss individual techniques in Disco that minimize time overheads, that minimize space overheads, and that enable better resource management decisions.

Question 4. Distributed Synchronization with Barriers:

A *barrier* is a global synchronization primitive that causes a process to block until all the other processes have reached the barrier call on the same barrier, b. Each process would have a call like

```
barrier(b);
```

and would block at the call until all the other processes reached their corresponding call.

For this question, assume that each process is running on its own host in a cluster of computers connected by a high-speed LAN. Assume that communication is performed with standard socket-based message operations and that the synchronization operations do not use any shared file systems.

4A. Describe an efficient algorithm for implementing the barrier protocol.
4B. What performance measures would you use to evaluate the barrier protocol? Explain why these measures are meaningful.
4C. Evaluate your protocol in terms of the measures you described in part 4B.

Question 5. Transport Layer Congestion:

Since the congestion collapse events of the late 1980's congestion control has been a central theme in transport layer protocol development. In fact, most of the versions of TCP running on the Internet today are "Jacobson derived".

5A. Describe the fundamental aspects of Jacobson derived transport protocols and their effects on network traffic in today's Internet.
5B. Are Jacobson derived transport protocols required for Internet stability?
5C. How does transport layer congestion control contrast and interact with packet congestion at other layers of the network protocol stack?

Question 6. Types in File Systems:

The original Unix file system recognized only four types of objects: directories, "plain" files, and two kinds of devices ("special" files). Modern versions of Unix have added a few others, such as symbolic

links, sockets, and "fifos", but the set of available types is still small and wired into the implementation.

6A. How does the operating system kernel use type information to protect against errors? Explain how the kernel uses types to protect itself from buggy or hostile processes, and how it protects processes from each other and themselves.
6B. Various application programs and subsystems have their own notion of file types, such as "C source" or "executable shell script". Give some other examples. How do programs use file system data and meta-data to keep track of types?
6C. Design an extension to the Unix file system to support user-defined file types. What should the API look like? How should the extension be implemented in the kernel?

Question 7. Nested Monitor Calls:

A monitor uses a "monitor lock" to enforce the invariant that at most one processes is active in any of its procedures at any one time.

7A. When a process executing in monitor `M1` calls a procedure in a different monitor `M2`, should the implementation release the monitor lock on `M1`? Give arguments for and against.
7B. What sorts of deadlocks may arise if the lock on `M1` is not released? What if it is released?
7C. How can an application programmer protect against deadlocks in each case? Consider general principles as well as features of specific applications that can be used to avoid deadlocks caused by nested monitor calls.