UNIVERSITY OF WISCONSIN-MADISON
Computer Sciences Department

Operating Systems **Spring 2010**

Instructions: There are *six* questions on this exam; you must answer *all* of the following questions.

Question 1: Virtual Machines

In the Disco paper, virtual machine technology is used to run operating systems on scalable shared-memory parallel machines. In this question, we explore the benefits and drawbacks of this approach.

1. One of the problems of running IRIX on Disco (instead of on the raw hardware) is the time overhead encountered, particularly in how Disco "virtualizes" memory underneath the OS. What are the core components of this time overhead? What does Disco do to lower these costs?

2. Another large potential overhead in Disco's virtualization is the space cost in memory. How do such space costs arise? What can Disco (or any virtual machine) do to lower these costs?

3. Finally, in some cases Disco was not able to properly virtualize the machine under IRIX, which prompted the authors to rewrite small portions of IRIX. In general, what makes a system difficult to virtualize? (examples are useful) Why does rewriting the OS help?

Question 2: Distributed Systems

The V kernel was an early distributed operating system supporting diskless workstations served by a common file server over Ethernet.

1. The V kernel provides two messaging primitives: (i) sending short, fixed length messages and (ii) copying a larger chunk of data from a remote address or to a remote address. Explain how and why these two primitives are sufficient and useful as a substrate for building a distributed file system such as AFS.

2. The Xerox RPC system (Birrell and Nelson) provides similar facilities to the V messaging primitives, but handled larger messages (larger than a packet) differently. The Xerox RPC mechanism also optimizes for small messages, and splits larger messages into packets that are individually acknowledged.

A. Would the V kernel approach be appropriate for the RPC system? Why or why not?

B. Compare the performance of the V system approach to the Xerox RPC approach. Is one fundamentally faster? If so, why?

3. The end-to-end argument for system design argues that applications are fundamentally responsible for reliability and shouldn't rely on intermediaries. Does either the V kernel or the RPC system apply the end-to-end argument? Explain your answer.

Question 3: Kernels

The Pilot operating system focused on providing service for a single-user workstation, and as such its protection mechanisms and scheduling policies focus on reliability more than on security.

1. Give two examples of design choices made in Pilot that reflect this focus, and explain why they make sense for single user systems on a local-area network.

2. Suppose Pilot had been successful, and you wanted to attach a Pilot system to the Internet, where it could be vulnerable to remote attacks. Also, you would like to be able to download programs from the Internet and run them, even though you don't fully trust the program authors.

A. List two important aspects of the Pilot design you would have to change to be secure in this circumstance,

B. Explain how you would change these two pieces.

3. The Pilot system performs inter-process communication through shared memory, as does Mach with its RPC and port mechanism. Which system's mechanism is more appropriate for a machine connected to the Internet? Why?

Question 4: Process Migration

1. Imagine you have the task of designing an operating system that supports either remote invocation, in which a process can be placed on a remote machine only when it is first started, or process migration, in which a process can be moved to a different machine at any point in its lifetime. What are the pros and cons of each approach? Why is process migration so much harder to implement?

2. Assume you have decided to implement process migration within the operating system. The most appropriate design is likely to depend upon the environment in which it will be used. What questions would you ask about about the target environment and the requirements of the resulting system?

3. Imagine your system must run in an environment in which administrators can reboot the original home machine of a migrated process. How does this target environment impact your implementation of process migration? What design options are no longer available to you? What design options can you still choose between?

Question 5: Specialized File Systems

Starting with a typical file system design, such as the Berkeley Fast File System, consider changes to this design that would enable it to have improved efficiencies in the following two cases:

1. *A small-file file system:* Describe a design for a file system design to more efficiently handle small files. By small files, we mean those no more than about 100 bytes. In what ways would you expect your design to operate more efficiently than the Berkeley FFS?

2. *A temporary-file file system:* Consider a file system that has no requirement for persistent storage between reboots. Files would typically be used for a short time and then removed and, in no case, be

needed when the operating system restarted. Describe the design for a file system intended for temporary files. In what ways would you expect your design to operate more efficiently than the Berkeley FFS?

Question 6: locks, races, hangs

1. Mutual exclusion can be implemented in many different ways. Please explain the pros and cons of non-blocking spin-locks and blocking mutex locks.

2. The following code snippet is taken from Apache web server. Its design philosophy is as follows:
* Apache has one worker thread that mainly works inside the *while(!stopped)* loop in function *run()*.
* When Apache needs to shut down, a second thread will execute *DoStop()*.
* *DoStop* forces the worker thread to exit the *sock.accept* function; it then sets *stopped* to be TRUE and makes itself wait for the worker thread's acknowledgement.
* When the worker thread reads a TRUE from *stopped*, it jumps out of the loop, and notifies the DoStop thread using a conditional variable.

```
/*These four are all global variables
   and are all correctly initialized */
pthread_mutex_t mutex;
pthread_cond_t cond;
ServerSocket sock;
Bool stopped=FALSE;

void run ( ){
 ...                                                    void DoStop ( ){
 while (!stopped) {
  sock.accept (…);                                       sock.cancel ( );
     /*sock.accept ( ) returns when
       (1) it receives a request from client
             or                                          stopped=TRUE;
       (2) a thread executes sock.cancel ( )
             during the sock.accept ( )
     */
  ...
 }                                                       pthread_mutex_lock(&mutex);
 pthread_mutex_lock(&mutex);                             pthread_cond_wait(&cond,&mutex);
 pthread_cond_signal(&cond);                             pthread_mutex_unlock(&mutex);
 pthread_mutex_unlock(&mutex);                           …
}                                                       }
```

Everything looked fine. The code was released. Soon, users began to complain that their Apache servers randomly hang during shut-down.
Please explain what execution order(s) caused the hang. (You do not need to fix the code.)

Note 1: There are two different hang scenarios. Please find both.
Note 2: Assume there are only 2 threads when the user begins to shut down Apache: one inside the sock.accept() and one calling DoStop().
Note 3: The semantics of pthread_cond_signal and pthread_cond_wait are the same as Mesa's notify/wait.