University of Wisconsin: Department of Computer Sciences

Operating Systems Qualifying Exam: Fall 2012

Instructions: There are six questions on this exam; answer all six questions.

Question 1. Local File Systems

Local file systems are the underlying store for data on desktops, mobile phones, and even serve as backend storage for clustered distributed file systems. In this question, we explore some aspects of local file systems and their implications.

(a) Most local file systems implement some form of crash consistency, such that they can quickly recover to after an unexpected software or hardware failure. What exactly is crash consistency?

(b) One common technique used to implement crash consistency is journaling. Describe how journaling works, and how it helps a file system retain consistency in the face of failure.

(c) Another common approach to crash consistency is copy-on-write (COW), similar to the log-structured file system (LFS). How does the "never overwrite" approach of LFS make implementing crash consistency simpler?

Question 2. Data Replication and Consistency

Distributed systems usually replicate their data to improve availability and performance.

(a) How can inconsistency arise among the replicas in a distributed system?

(b) How does GrapeVine handle inconsistency among its replicas?

(c) How does Dynamo handle inconsistency among its replicas?

(d) What is the difference between the GrapeVine approach and the Dynamo approach?

Question 3. Virtual Machine Monitors

Virtual Machine Monitors (VMMs), or hypervisors, are quite popular in today's datacenters. In this question, we'll explore some of the underlying technology.

(a) One classic way to construct a VMM is to use what is known as "trap and emulate" (T&E). The idea is simple: when a guest operating system tries to perform a privileged operation, a trap occurs and is vectored to the VMM to be handled. The VMM then emulates the operation, thus providing the illusion to the guest that the operation worked as desired. Give two examples of where trap-and-emulate is used in a system you are familiar with (e.g., Disco).

(b) Trap-and-emulate style virtualization is not without its flaws. One potentially large problem is with performance; as a result, guests running in a virtualized environment may run more slowly than non-virtualized guest OSes. Describe how performance is affected by T&E virtualization, using concrete examples. How much affect will this slowdown have on applications? Which types of applications suffer the most?

(c) Sometimes details of how the hardware handles privileged operations can lead to difficulties for trap-and-emulate style virtualization. For example, on x86, one can issue an instruction (popf) to change how interrupts are delivered. When an unprivileged app/guest attempts this instruction, it does not generate a trap; rather, the instruction executes but does not change the state of the system as it would if it were in privileged mode. Why is this a problem for T&E?

(d) Given that a guest OS might issue an instruction such as popf, what approaches could a VMM take in order to still properly virtualize the system?

Question 4. Proportional-Share Scheduling for I/O

(a) Algorithms for scheduling I/O requests for disks have existed for many years. Briefly describe a few traditional disk scheduling algorithms and their goals.

(b) Proportional-share scheduling (e.g., lottery scheduling) has been proposed and applied to CPUs. To apply proportional-share scheduling to a disk, what additional issues would need to be considered?

(c) Assume you would like to use proportional scheduling for a disk, while still delivering some of the goals of a traditional disk scheduler. What would be your basic approach?

Question 5. Inferring Resources and OS Properties

You have been given a user account on a remote machine to which you are porting a multi-threaded, memory and I/O-intensive application that requires high performance. All you know about this machine is that it supports the POSIX interface; you have no useful documentation, no one is answering your e-mail about the machine, and any interfaces you know of for acquiring information about the machine appear to be broken. The only information you can acquire is by timing different operations.

To obtain high performance for your application, you believe that it would be useful to know the following properties about the machine or OS:

1) Number of CPUs
2) Page size (in bytes)
3) Number of TLB entries
4) Amount of physical memory (in bytes)

Consider **each** of these four properties in turn and answer the following three questions **for each**:

(a) Give a brief but precise description of a benchmark program you develop that allows you to infer each property.

(b) What assumptions does your benchmark make about the machine or the operating system? (Hint: think about any assumptions you are making about the scheduler, replacement policies, minimum sizes, or relative speeds).

(c) To sanity check your benchmark results, what are reasonable numbers for a modern machine for each of these properties?

Question 6. Virtual Memory

Processors support for multiple page sizes. For example, Intel’s current x86 processors support 4KB, 2MB, and 1GB pages.

(a) When and how do larger pages sizes benefit applications?

(b) How can an operating system make large pages available to applications?

(c) What are the high-level features an operating system, built for a single page size, must add to support multiple page sizes at once?

Mach made great strides in operating-system portability by providing a virtual memory system that could be used across processors with different virtual-memory hardware.

(d) Explain the basic technique Mach uses to abstract virtual memory mapping from hardware-dependent translation mechanisms, such as hardware page tables (x86) or software-filled TLBs.

(e) Mach may face additional challenges in using multiple page sizes because it provides machine independence. If there are additional challenges, please explain. If not, explain why not.