

Operating System Structures



Chapter 2 is concerned with the operating-system interfaces that users (or at least programmers) actually see: system calls. The treatment is somewhat vague since more detail requires picking a specific system to discuss. This chapter is best supplemented with exactly this detail for the specific system the students have at hand. Ideally they should study the system calls and write some programs making system calls. This chapter also ties together several important concepts including layered design, virtual machines, Java and the Java virtual machine, system design and implementation, system generation, and the policy/mechanism difference.

Exercises

- 2.12 The services and functions provided by an operating system can be divided into two main categories. Briefly describe the two categories and discuss how they differ.

Answer: One class of services provided by an operating system is to enforce protection between different processes running concurrently in the system. Processes are allowed to access only those memory locations that are associated with their address spaces. Also, processes are not allowed to corrupt files associated with other users. A process is also not allowed to access devices directly without operating system intervention. The second class of services provided by an operating system is to provide new functionality that is not supported directly by the underlying hardware. Virtual memory and file systems are two such examples of new services provided by an operating system.

- 2.13 Describe three general methods for passing parameters to the operating system.

Answer:

- Pass parameters in registers
- Registers pass starting addresses of blocks of parameters

- c. Parameters can be placed, or *pushed*, onto the *stack* by the program, and *popped* off the stack by the operating system

2.14 Describe how you could obtain a statistical profile of the amount of time spent by a program executing different sections of its code. Discuss the importance of obtaining such a statistical profile.

Answer: One could issue periodic timer interrupts and monitor what instructions or what sections of code are currently executing when the interrupts are delivered. A statistical profile of which pieces of code were active should be consistent with the time spent by the program in different sections of its code. Once such a statistical profile has been obtained, the programmer could optimize those sections of code that are consuming more of the CPU resources.

2.15 What are the five major activities of an operating system in regard to file management?

Answer:

- The creation and deletion of files
- The creation and deletion of directories
- The support of primitives for manipulating files and directories
- The mapping of files onto secondary storage
- The backup of files on stable (nonvolatile) storage media

2.16 What are the advantages and disadvantages of using the same system-call interface for manipulating both files and devices?

Answer: Each device can be accessed as though it was a file in the file system. Since most of the kernel deals with devices through this file interface, it is relatively easy to add a new device driver by implementing the hardware-specific code to support this abstract file interface. Therefore, this benefits the development of both user program code, which can be written to access devices and files in the same manner, and device-driver code, which can be written to support a well-defined API. The disadvantage with using the same interface is that it might be difficult to capture the functionality of certain devices within the context of the file access API, thereby resulting in either a loss of functionality or a loss of performance. Some of this could be overcome by the use of the `ioctl` operation that provides a general-purpose interface for processes to invoke operations on devices.

2.17 Would it be possible for the user to develop a new command interpreter using the system-call interface provided by the operating system?

Answer: An user should be able to develop a new command interpreter using the system-call interface provided by the operating system. The command interpreter allows an user to create and manage processes and also determine ways by which they communicate (such as through pipes and files). As all of this functionality could be accessed by an user-level program using the system calls, it should be possible for the user to develop a new command-line interpreter.

- 2.18 What are the two models of interprocess communication? What are the strengths and weaknesses of the two approaches?

Answer: TBA

- 2.19 Why is the separation of mechanism and policy desirable?

Answer: Mechanism and policy must be separate to ensure that systems are easy to modify. No two system installations are the same, so each installation may want to tune the operating system to suit its needs. With mechanism and policy separate, the policy may be changed at will while the mechanism stays unchanged. This arrangement provides a more flexible system.

- 2.20 It is sometimes difficult to achieve a layered approach if two components of the operating system are dependent on each other. Identify a scenario in which it is unclear how to layer two system components that require tight coupling of their functionalities.

Answer: The virtual memory subsystem and the storage subsystem are typically tightly coupled and requires careful design in a layered system due to the following interactions. Many systems allow files to be mapped into the virtual memory space of an executing process. On the other hand, the virtual memory subsystem typically uses the storage system to provide the backing store for pages that do not currently reside in memory. Also, updates to the file system are sometimes buffered in physical memory before it is flushed to disk, thereby requiring careful coordination of the usage of memory between the virtual memory subsystem and the file system.

- 2.21 What is the main advantage of the microkernel approach to system design? How do user programs and system services interact in a microkernel architecture? What are the disadvantages of using the microkernel approach?

Answer: Benefits typically include the following: (a) adding a new service does not require modifying the kernel, (b) it is more secure as more operations are done in user mode than in kernel mode, and (c) a simpler kernel design and functionality typically results in a more reliable operating system. User programs and system services interact in a microkernel architecture by using interprocess communication mechanisms such as messaging. These messages are conveyed by the operating system. The primary disadvantages of the microkernel architecture are the overheads associated with interprocess communication and the frequent use of the operating system's messaging functions in order to enable the user process and the system service to interact with each other.

- 2.22 In what ways is the modular kernel approach similar to the layered approach? In what ways does it differ from the layered approach?

Answer: The modular kernel approach requires subsystems to interact with each other through carefully constructed interfaces that are typically narrow (in terms of the functionality that is exposed to external modules). The layered kernel approach is similar in that respect. However, the layered kernel imposes a strict ordering of subsystems such that subsystems at the lower layers are not allowed to invoke opera-

tions corresponding to the upper-layer subsystems. There are no such restrictions in the modular-kernel approach, wherein modules are free to invoke each other without any constraints.

- 2.23 What is the main advantage for an operating-system designer of using a virtual-machine architecture? What is the main advantage for a user?

Answer: The system is easy to debug, and security problems are easy to solve. Virtual machines also provide a good platform for operating system research since many different operating systems can run on one physical system.

- 2.24 Why is a just-in-time compiler useful for executing Java programs?

Answer: Java is an interpreted language. This means that the JVM interprets the bytecode instructions one at a time. Typically, most interpreted environments are slower than running native binaries, for the interpretation process requires converting each instruction into native machine code. A just-in-time (JIT) compiler compiles the bytecode for a method into native machine code the first time the method is encountered. This means that the Java program is essentially running as a native application (of course, the conversion process of the JIT takes time as well, but not as much as bytecode interpretation). Furthermore, the JIT caches compiled code so that it can be reused the next time the method is encountered. A Java program that is run by a JIT rather than a traditional interpreter typically runs much faster.

- 2.25 What is the relationship between a guest operating system and a host operating system in a system like VMware? What factors need to be considered in choosing the host operating system?

Answer: A guest operating system provides its services by mapping them onto the functionality provided by the host operating system. A key issue that needs to be considered in choosing the host operating system is whether it is sufficiently general in terms of its system-call interface in order to be able to support the functionality associated with the guest operating system.

- 2.26 The experimental Synthesis operating system has an assembler incorporated within the kernel. To optimize system-call performance, the kernel assembles routines within kernel space to minimize the path that the system call must take through the kernel. This approach is the antithesis of the layered approach, in which the path through the kernel is extended to make building the operating system easier. Discuss the pros and cons of the Synthesis approach to kernel design and to system-performance optimization.

Answer: Synthesis is impressive due to the performance it achieves through on-the-fly compilation. Unfortunately, it is difficult to debug problems within the kernel due to the fluidity of the code. Also, such compilation is system specific, making Synthesis difficult to port (a new compiler must be written for each architecture).