Operating-System Structures

Practice Exercises

2.1 What is the purpose of system calls?

Answer:
System calls allow user-level processes to request services of the operating system.

2.2 What is the purpose of the command interpreter? Why is it usually separate from the kernel?

Answer:
It reads commands from the user or from a file of commands and executes them, usually by turning them into one or more system calls. It is usually not part of the kernel because the command interpreter is subject to changes.

2.3 What system calls have to be executed by a command interpreter or shell in order to start a new process on a UNIX system?

Answer: A `fork()` system call and an `exec()` system call need to be performed to start a new process. The `fork()` call clones the currently executing process, while the `exec()` call overlays a new process based on a different executable over the calling process.

2.4 What is the purpose of system programs?

Answer:
System programs can be thought of as bundles of useful system calls. They provide basic functionality to users so that users do not need to write their own programs to solve common problems.

2.5 What is the main advantage of the layered approach to system design? What are the disadvantages of the layered approach?

Answer:
As in all cases of modular design, designing an operating system in a modular way has several advantages. The system is easier to debug and modify because changes affect only limited sections of the system rather

than touching all sections. Information is kept only where it is needed and is accessible only within a defined and restricted area, so any bugs affecting that data must be limited to a specific module or layer. The primary disadvantage to the layered approach is theapoor performance due to the overhead of traversing through the different layers to obtain a service provided by the operating system.

2.6 List five services provided by an operating system, and explain how each creates convenience for users. In which cases would it be impossible for user-level programs to provide these services? Explain your answer.

Answer:
The five services are:

a. **Program execution**. The operating system loads the contents (or sections) of a file into memory and begins its execution. A user-level program could not be trusted to properly allocate CPU time.

b. **I/O operations**. It is necessary to communicate with disks, tapes, and other devices at a very low level. The user need only specify the device and the operation to perform on it, and the system converts that request into device- or controller-specific commands. User-level programs cannot be trusted to access only devices they should have access to and to access them only when they are otherwise unused.

c. **File-system manipulation**. There are many details in file creation, deletion, allocation, and naming that users should not have to perform. Blocks of disk space are used by files and must be tracked. Deleting a file requires removing the name file information and freeing the allocated blocks. Protections must also be checked to assure proper file access. User programs could neither ensure adherence to protection methods nor be trusted to allocate only free blocks and deallocate blocks on file deletion.

d. **Communications**. Message passing between systems requires messages to be turned into packets of information, sent to the network controller, transmitted across a communications medium, and reassembled by the destination system. Packet ordering and data correction must take place. Again, user programs might not coordinate access to the network device, or they might receive packets destined for other processes.

e. **Error detection**. Error detection occurs at both the hardware and software levels. At the hardware level, all data transfers must be inspected to ensure that data have not been corrupted in transit. All data on media must be checked to be sure they have not changed since they were written to the media. At the software level, media must be checked for data consistency—for instance, whether the number of allocated and unallocated blocks of storage match the total number on the device. There, errors are frequently process-independent (for instance, the corruption of data on a disk), so there must be a global program (the operating system) that handles

all types of errors. Also, when errors are processed by the operating system, processes need not contain code to catch and correct all the errors possible on a system.

2.7 Why do some systems store the operating system in firmware, while others store it on disk?

Answer:
For certain devices, such as embedded systems, a disk with a file system may be not be available for the device. In this situation, the operating system must be stored in firmware.

2.8 How could a system be designed to allow a choice of operating systems from which to boot? What would the bootstrap program need to do?

Answer:
Consider a system that would like to run both Windows and three different distributions of Linux (for example, RedHat, Debian, and Ubuntu). Each operating system will be stored on disk. During system boot, a special program (which we will call the **boot manager**) will determine which operating system to boot into. This means that rather than initially booting to an operating system, the boot manager will first run during system startup. It is this boot manager that is responsible for determining which system to boot into. Typically, boot managers must be stored at certain locations on the hard disk to be recognized during system startup. Boot managers often provide the user with a selection of systems to boot into; boot managers are also typically designed to boot into a default operating system if no choice is selected by the user.

Exercises

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

Answer:

a. Parameters can be passed in registers.

b. Registers can 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

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

Answer:
An advantage of using the same system-call interface for manipulating both files and devices is that, since most of the kernel deals with devices through this interface, it is relatively easy to add a new device driver by implementing the appropriate hardware-specific code. 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 of 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, resulting in either a loss of functionality or a loss of performance. Some of this difficulty could be overcome by the use of the ioctl operation, which provides a general-purpose interface for processes to invoke operations on devices.

2.11 Describe why Android uses ahead-of-time (AOT) rather than just-in-time (JIT) compilation.

Answer: Ahead-of-time compilation compiles Java code when it is installed on a device. This is a time and energy-saving strategy which is important for mobile devices.

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

Answer:
The two models of interprocess communication are the message-passing model and the shared-memory model. Message passing is useful for exchanging smaller amounts of data, because no conflicts need be avoided. It is also easier to implement than is shared memory for intercomputer communication. Shared memory allows maximum speed and convenience of communication, since it can be done at memory transfer speeds when it takes place within a computer. However, this method involves problems in the areas of protection and synchronization between the processes sharing memory.

2.13 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 need to frequently use the operating system's messaging functions to enable the user process and the system service to interact with each other.

2.14 How are iOS and Android similar? How are they different?

Answer:
Similarities:

- Both are based on existing kernels (Linux and macOS).
- Both have architecture that uses software stacks.
- Both provide frameworks for developers.

Differences:

- iOS is closed source, and Android is open source.
- iOS applications are developed in Objective-C, and Android in Java.
- Android uses a virtual machine, and iOS executes code natively.

2.15 Explain why Java programs running on Android systems do not use the standard Java API and virtual machine.

Answer:
It is because the standard API and virtual machine are designed for desktop and server systems, not mobile devices. Google developed a separate API and virtual machine for mobile devices.