CHAPTER 1

Introduction



Practice Exercises

1.1 What are the three main purposes of an operating system?

Answer:
The three main purposes are:

- To provide an environment for a computer user to execute programs on computer hardware in a convenient and efficient manner.
- To allocate the separate resources of the computer as needed to perform the required tasks. The allocation process should be as fair and efficient as possible.
- As a control program, it serves two major functions: (1) supervision of the execution of user programs to prevent errors and improper use of the computer, and (2) management of the operation and control of I/O devices.

1.2 We have stressed the need for an operating system to make efficient use of the computing hardware. When is it appropriate for the operating system to forsake this principle and to "waste" resources? Why is such a system not really wasteful?

Answer:
Single-user systems should maximize use of the system for the user. A GUI might "waste" CPU cycles, but it optimizes the user's interaction with the system.

1.3 What is the main difficulty that a programmer must overcome in writing an operating system for a real-time environment?

Answer:
The main difficulty is keeping the operating system within the fixed time constraints of a real-time system. If the system does not complete a task in a certain time frame, it may cause a breakdown of the entire system. Therefore, when writing an operating system for a real-time system, the

Troy University Resources

writer must be sure that his scheduling schemes don't allow response time to exceed the time constraint.

1.4 Keeping in mind the various definitions of ***operating system***, consider whether the operating system should include applications such as web browsers and mail programs. Argue both that it should and that it should not, and support your answers.

Answer:
An argument in favor of including popular applications in the operating system is that if the application is embedded within the operating system, it is likely to be better able to take advantage of features in the kernel and therefore have performance advantages over an application that runs outside of the kernel. Arguments against embedding applications within the operating system typically dominate, however: (1) the applications are applications—not part of an operating system, (2) any performance benefits of running within the kernel are offset by security vulnerabilities, and (3) inclusion of applications leads to a bloated operating system.

1.5 How does the distinction between kernel mode and user mode function as a rudimentary form of protection (security)?

Answer:
The distinction between kernel mode and user mode provides a rudimentary form of protection in the following manner. Certain instructions can be executed only when the CPU is in kernel mode. Similarly, hardware devices can be accessed only when the program is in kernel mode, and interrupts can be enabled or disabled only when the CPU is in kernel mode. Consequently, the CPU has very limited capability when executing in user mode, thereby enforcing protection of critical resources.

1.6 Which of the following instructions should be privileged?

a. Set value of timer.
b. Read the clock.
c. Clear memory.
d. Issue a trap instruction.
e. Turn off interrupts.
f. Modify entries in device-status table.
g. Switch from user to kernel mode.
h. Access I/O device.

Answer:
The following operations need to be privileged: set value of timer, clear memory, turn off interrupts, modify entries in device-status table, access I/O device. The rest can be performed in user mode.

1.7 Some early computers protected the operating system by placing it in a memory partition that could not be modified by either the user job or the operating system itself. Describe two difficulties that you think could arise with such a scheme.

Answer:
The data required by the operating system (passwords, access controls, accounting information, and so on) would have to be stored in or passed through unprotected memory and thus be accessible to unauthorized users.

1.8 Some CPUs provide for more than two modes of operation. What are two possible uses of these multiple modes?

Answer:
Although most systems only distinguish between user and kernel modes, some CPUs have supported multiple modes. Multiple modes could be used to provide a finer-grained security policy. For example, rather than distinguishing between just user and kernel mode, you could distinguish between different types of user mode. Perhaps users belonging to the same group could execute each other's code. The machine would go into a specified mode when one of these users was running code. When the machine was in this mode, a member of the group could run code belonging to anyone else in the group.

Another possibility would be to provide different distinctions within kernel code. For example, a specific mode could allow USB device drivers to run. This would mean that USB devices could be serviced without having to switch to kernel mode, thereby essentially allowing USB device drivers to run in a quasi-user/kernel mode.

1.9 Timers could be used to compute the current time. Provide a short description of how this could be accomplished.

Answer:
A program could use the following approach to compute the current time using timer interrupts. The program could set a timer for some time in the future and go to sleep. When awakened by the interrupt, it could update its local state, which it uses to keep track of the number of interrupts it has received thus far. It could then repeat this process of continually setting timer interrupts and updating its local state when the interrupts are actually raised.

1.10 Give two reasons why caches are useful. What problems do they solve? What problems do they cause? If a cache can be made as large as the device for which it is caching (for instance, a cache as large as a disk), why not make it that large and eliminate the device?

Answer:
Caches are useful when two or more components need to exchange data, and the components perform transfers at differing speeds. Caches solve the transfer problem by providing a buffer of intermediate speed between the components. If the fast device finds the data it needs in the cache, it need not wait for the slower device. The data in the cache must be kept consistent with the data in the components. If a component has

Troy University Resources

a data value change, and the datum is also in the cache, the cache must also be updated. This is especially a problem on multiprocessor systems, where more than one process may be accessing a datum. A component may be eliminated by an equal-sized cache, but only if: (a) the cache and the component have equivalent state-saving capacity (that is, if the component retains its data when electricity is removed, the cache must retain data as well), and (b) the cache is affordable, because faster storage tends to be more expensive.

1.11 Distinguish between the client–server and peer-to-peer models of distributed systems.

Answer:

The client-server model firmly distinguishes the roles of the client and server. Under this model, the client requests services that are provided by the server. The peer-to-peer model doesn't have such strict roles. In fact, all nodes in the system are considered peers and thus may act as *either* clients or servers—or both. A node may request a service from another peer, or the node may in fact provide such a service to other peers in the system.

For example, let's consider a system of nodes that share cooking recipes. Under the client-server model, all recipes are stored with the server. If a client wishes to access a recipe, it must request the recipe from the specified server. Using the peer-to-peer model, a peer node could ask other peer nodes for the specified recipe. The node (or perhaps nodes) with the requested recipe could provide it to the requesting node. Notice how each peer may act as both a client (it may request recipes) and as a server (it may provide recipes).