CS323 – Exercises
Week 1

22 Feb 2019

Exercise 1:

From the list below, say which items are part of the OS and which are not. For those that are, explain how each creates convenience for users and which hardware resources they are managing. Would it be possible for user-level programs to provide these services?

a) Scheduler	e) File management
b) Video player	f) Command interpreter
c) VPN client	g) Text editor
d) Device drivers	

Answer:

a) Scheduler

Part of kernel? Yes

Convenience for the user?
The scheduler's purpose is determining how to best use the CPU. CPU-scheduling routines take into account the speed of the CPU, the jobs that must be executed, the number of registers available, and other factors.

Would it be possible for user-level programs to provide these services?
It would be possible for user-level programs to provide a subset of the services a scheduler residing in the kernel could provide. For instance, it is possible to build a user level scheduler that does load balancing (i.e., decides which threads go first, and on which cores they would execute). However, it is not straightforward to have a user level scheduler that can multiplex time (i.e., making a thread run at a given moment). A user-level scheduler in Linux will itself be considered a process that is scheduled with other processes. So, the scheduler would have to wait to be scheduled, thus not being able to guarantee time multiplexing.

b) Video player

Part of kernel? No

Convenience for the user?
Allows users to play videos.

Would it be possible for user-level programs to provide these services?
Video players are generally user-level programs.

c) VPN client

Part of kernel? Not generally.

Sometimes, they can be implemented as kernel modules, for performance reasons.

Convenience for the user?
A VPN client is a piece of software designed to connect a user to a network by establishing connections using tunneling protocols. VPNs can create secure connections to private networks across public networks such as the Internet.

Would it be possible for user-level programs to provide these services?
VPN clients are typically user-level programs.

d) Device drivers

Part of kernel? Yes

Convenience for the user?
A device driver is a program that allows users to control a particular type of device that is attached to your computer (e.g., printers, displays).

Would it be possible for user-level programs to provide these services?
Yes, it would be possible to have user-level device drivers. Running device drivers as unprivileged user-level code has been proposed as a technique for increasing system robustness.

e) File management

Part of kernel? Yes

Convenience for the user?
To make the computer system convenient for users, the OS provides a uniform, logical view of information storage. The operating system abstracts from the physical properties of its storage devices to define a logical storage unit, the file. The operating system maps files onto physical media and accesses these files via the storage devices.

Would it be possible for user-level programs to provide these services?
Yes, it would be possible. For example, FUSE (Filesystem in Userspace) is a software interface for Unix-like computer operating systems that lets non-privileged users create their own file systems without editing kernel code. This is achieved by running file system code in user space while the FUSE module provides only a *bridge* to the actual kernel interfaces.

f) Command Interpreter

Part of kernel? Some operating systems include the command interpreter in the kernel. Others, such as Windows and UNIX, treat the command interpreter as a special program that is running when a job is initiated or when a user first logs on (on interactive systems).

Convenience for the user?
The command interpreter allows users to directly enter commands to be performed by the operating system.

Would it be possible for user-level programs to provide these services?
If the command interpreter is not included in the kernel, another approach is to implement the commands through system programs. In this case, the command interpreter does not understand the command in any way; it merely uses the command to identify a file to be loaded into memory and executed. In this way, programmers can add new commands to the system easily by creating new files with the proper names. The command-interpreter program, which can be small, does not have to be changed for new commands to be added.

g) Text editor

Part of kernel? No

Convenience for the user?
Allows users to edit text.

Would it be possible for user-level programs to provide these services?
Text editors are typically user-level programs.

Exercise 2:

What is the purpose of system calls? How do language libraries (such as libc) the kernel API and system calls interact? In the list below, which of the C Library functions make use of system calls and which don't? Why?

a) `malloc()`
b) `printf()`
c) `sqrt()`
d) `strcmp()`
e) *`strcat()`*
f) `getchar()`
g) `isdigit()`
h) `time()`
i) `difftime()`
j) `fseek()`

Answer:

A *system call* is the way in which a computer program requests a service from the kernel of the OS the program is executed on. When a function in libc needs to make a system call, it makes a call to the Kernel API, which in turn makes the system call.

a) `malloc()` -- occasional system call; if a call to malloc needs to alter the heap size it will issue the brk() system call to do so or, if the required memory allocation is above the threshold allowed by the kernel, it will call mmap() which calls a syscall. Otherwise there is no system call.
b) `printf()`-- does a system call; needs to write to a stream.
c) `sqrt()` -- no system call; this is a mathematical function.
d) `strcmp()` -- no system call; the two strings are compared character by character.
e) `strcat()` -- no system call; the second string is appended to the first string.
f) `getchar()`—no systematic system call; reads a char from stdin (buffered), when buffer is empty, performs a read() syscall to fill it.
h) `isdigit()` -- no system call; this is just a verification.

i) `time()` -- does a system call to get the time.
j) `difftime()` -- no system call; only compares times.
k) `fseek()` -- does a system call to position the cursor at a certain point in a file.

To find out whether the clib functions make use of a system call, you can use `man`. Next to the function name, you will see a number, in parentheses. The number interpretation is:

(1) General commands (will perform a system call)
(2) System calls
(3) C library functions (no system call)

You can also use `strace` to capture and record all system calls made by a process.

Exercise 3:

What is the purpose of interrupts?

Answer:
The occurrence of an event is usually signaled by an interrupt from either the hardware or the software. Hardware may trigger an interrupt at any time by sending a signal to the CPU, usually by way of the system bus.

How does an interrupt differ from a trap?

Answer:
Interrupts are usually orthogonal to a program whereas traps are exceptions that arise as part of the program. Traps are caused either by an error (for example, division by zero or invalid memory access) or by a specific request from a user program that an operating-system service be performed. A trap usually results in a switch to kernel mode, wherein the operating system performs some action before returning control to the originating process. Traps are handled immediately by the OS.

Can traps be generated intentionally by a user program? If so, for what purpose? (*Exercise 1.19 from Operating System Concepts*)

Answer:

Traps typically occur in case of errors, such as dividing by 0. However, they could be intentionally generated by a user program, for instance, for debugging purposes.