Cheng | Midterm 1: Chapter 2 | Study Guide

Chapter 2: OS Structures

System Calls: System calls provide the means for a user program to ask the operating system to perform tasks reserved for the operating system on the user program's behalf.

System calls provide an interface to the service made available by an operating system.

Application Programming Interface (API):

A simple program can make heavy use of the operating system. Frequently, systems execute thousands of system calls per second. If you look at the example, just copying a source file to a desination file requires a lot.

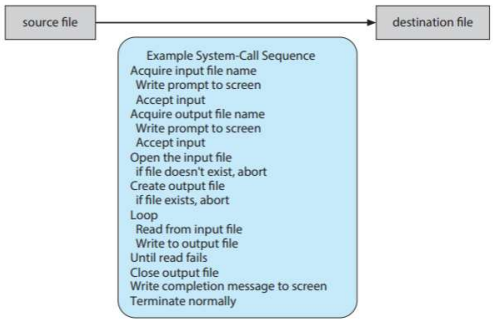

Figure 2.5 Example of how system calls are used.

The API specifies a set of functions that are available to an application programmer, including the parameters that are passed to each function and the return values the programmer can expect. A programmer can access an API via a library of code provided by the operating system.

How does an API and system calls interact?

Behind the scenes, the functions that make up an API typically invoke the actual system calls on behalf of the application programmer.

What are the benefits of an API?

1. Program portability, program can compile on any other system that all supports the same API.
2. System calls are more detailed and difficult to work with than the API available to an applciation programmer.

Run-Time Environment (RTE):

The RTE provides a system-call interface that serves as the link to system calls made available by the operating system. The system-call interface intercepts function calls in the API and invokes the necessary system calls within the operating system.

System-call interface maintains a table indexed according to these numbers. The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values. The caller need know nothing about how the system call is implemented.

Just needs to obey API and understand what OS will do as a result call. Most details of OS interface hidden from programmer by API. ABSTRACTION!

Parameters: In a system call, sometimes more information is required than simply the identity of the desired system call, like it needs to specify the file or device to use as a source.

Three general methods are used to pass parameters:

1. Pass parameters in registers, but what if there are more parameters then registers avaliable?
2. Parameters are stored in a block, or table in memory and the address of this block is passed as a parameter in a register.
3. Parameters can be placed, or pushed, onto a stack by the program and popped off the stack by the operating system.

Approaches 2 & 3 are perfered because they do not limit the number or length of paramets being passed.

Do we need to memorize the types of system calls?

OS Layered Approach (Pros and Cons):

There are two approaches to an operating system structure.

Tightly Coupled: Changes to one part of the system can have wide-ranging effects on other parts.

Loosely Coupled: A system is divided into separate, smaller components that have specific and limited functionality. All these components together comprise the kernel. Changhes to one part doesn't affect other components.

Layered Approach:

In which the operating system is broken into a number of layers or levels. The bottom layer is hardware and the highest layer is the user interface.

Advantage: Simplicity of contruction and debugging, The layers are selected so that each uses functions and services of only lower-level layers. This simplifies debugging and system verification. You can debug layer by layer, easy to see where the problem is. Each layer also provides abstraction that follows it, you don't need to know how these operations are implemented, just what they do.

Disadvantage: How do you appropriately define the functions of each layer? This is hard to do. Performance of system sinks, because the user program has to go through mulitple layers to obtain some service. More layer is equal to more traverse which takes longer.

Microkernels:

This method structures the operating system by removing all nonessential components from the kernel and implementing them as user-level programs that reside in separate address spaces. So a smaller kernel, that provides minimal proces and memory management and communication facility.

Microkernel's Communication:

Provides communication between the client program and the various services that are also running in the user space. Communiction is through message passing, see diagram below.

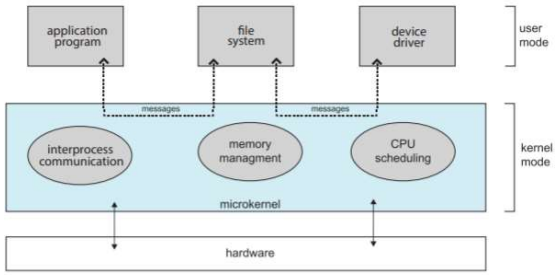

Figure 2.15 Architecture of a typical microkernel.

Advantages: Makes extending the OS easier, all new services are added to a user space and changes to kernel are small, because well the kernel is small. Smaller kernel a has higher portability to work another hardware design. More secure too because most services are running as user processes, not kernel, so if one thing fails the remainder is untouched.

Disadvantages: Overhead is garbage so performance suffers. When two user-level services communciate, messages must be copied between services, so separate address spaces. And the OS has to switch from one process to the next to exchange messages. Switching costs time.

Modules:

Current and best approach for structure is Loadable Kernel Modules (LKM). A kernel has a set of core components and can link in additional services via modules, either at boot time or run time.

In plain english, this means…

Major idea of this design is for the kernel to provide core services, while other services are implemented dynamically as the kernel is running. Linking ‘dynamically’ is better then just adding/changing directly the kernel because you don’t have to recompile the kernel every time a change is made.

Why Module is better than a Microkernel or the Layered Approach?

This much better than a layered system because it is like a layered system but more flexible, any module can call any other module, not just the one ‘beneath’ it. Similar to a microkernel because there is a primary module that has only core funtios but better because it does not need to invoke message passing to communicate.