CS415 Module 1 Part A - The Relationship Between the OS, the Hardware, and the User

Athens State University

Contents

What Does the Operating System Actually Do?

- Depends upon you point of view
  - Users want convenience and *ease of use*
    - But don't care about *resource utilization*
  - Users of dedicated systems have dedicated local resources but frequently have need of remote shared resources from servers
  - Handheld, mobile, and embedded devices are resource poor and are optimized for usability and battery life
  - Many embedded computers have little or no user interface

The Two Main Functions of the Operating System

Anyone old enough to remember the Saturday Night Live bit about *Shimmer: the floor wax and desert topping*?

- The OS is an extended machine or virtual machine

    - Easier to program than the underlying hardware

- The OS is a resource manager

    - Shares resources in time and space

1 A Top-Level View of Hardware

A Top-Level View of Hardware

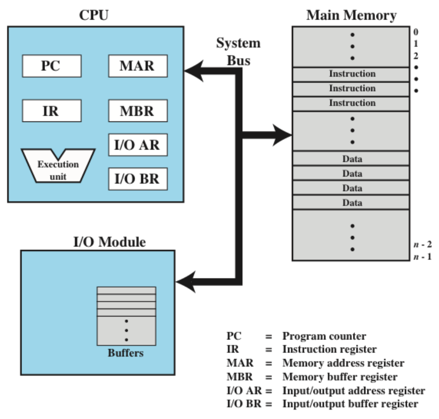


Figure 1.1 Computer Components: Top-Level View

- A CPU
    - Connected via a System Bus
        * To Main Memory (Instructions & Data)
        * To an I/O module (buffers data coming and going to devices)

Multiprocessor and Multicore

- Combines two or more processors

  Multiprocessor Multiple processors connected to the system bus

  Multicore Multiple processors (cores) on a single piece of silicon (die)

- In general, the OS treats a multicore machines as if it was an SMP multiprocessor

Symmetric Multiprocessor (SMP)

- A stand-alone computer system with the following characteristics
    - Two or more similar processors of comparable capability
    - Processors share the same main memory and have some form of connection (bus) between them

  - Processors share access to I/O devices
  - All processors can perform the same functions

- The system is controlled by an integrated operating system

Storage Structure

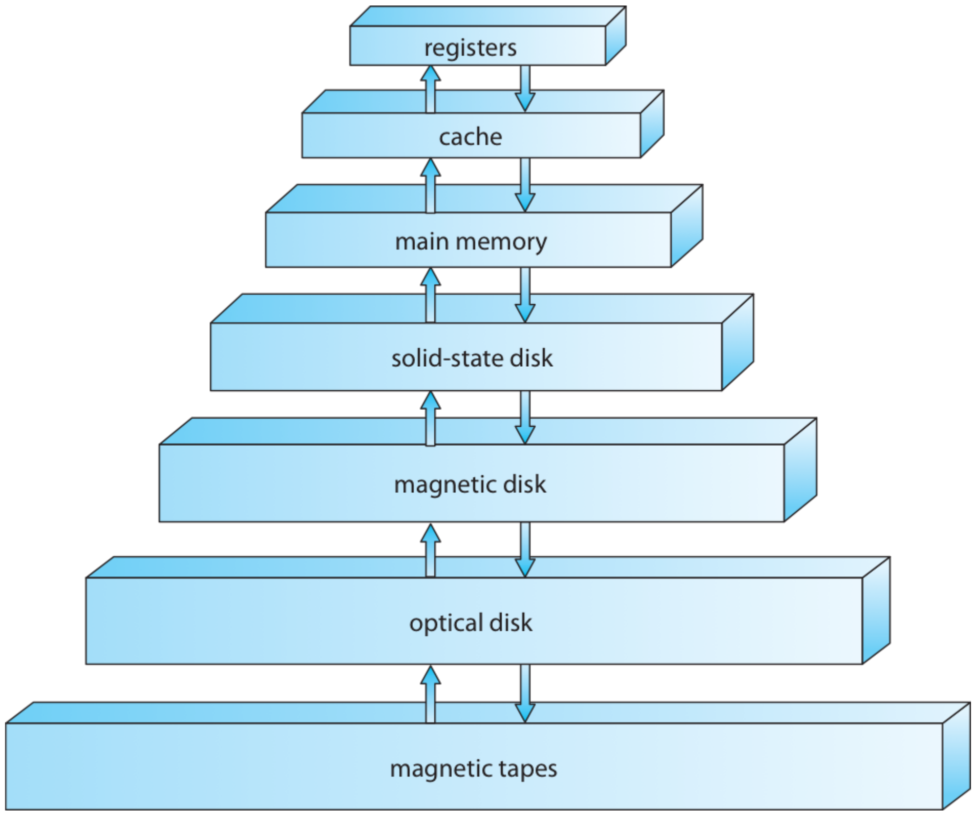
- Main memory
  - Only large storage media that the CPU can access directly
  - Random access
  - Typically volatile
- Secondary storage
  - Extension of main memory that provides large nonvolatile storage
  - Magnetic disks - rigid metal or glass platters covered with magnetic recording material
  - Solid state disks - faster than magnetic disks, nonvolatile

The surface of magnetic disks is logically divided into *tracks*, which are divided into *sectors*. The logical interaction between the device and the computer is managed by the *disk controller*.

Solid state disks are built from various memory technologies and are starting to become the prevalent secondary storage technology; esp., in the mobile and laptop device spaces.

This causes an issue for the disk controller as that device is programmed thinking magnetic media: tracks and sectors. So, it is the responsibility of kernel module for solid date devices to address the resulting impedance mis-match.

The Storage-Device Hierarchy



As one travels up and down the storage access hierarchy, one notices two characteristics:

1. Faster access times lead to a greater cost per bit
2. Greater capacity means smaller cost per bit with the penalty of slower access speed

So, we use smaller and faster forms of memory higher in the hierarchy as a cache to store more frequently accessed memory.

Principle of Locality

- Memory references by the processor tend to cluser
- Data is organized so that the percentage of accesses to each successively lower level is substantially less than that of the level above
- Can be applied across more than two levels of memory

The basis for the validity of condition (d) is a principle known as locality of reference . During the course of execution of a program, memory references by the processor, for both instructions and data, tend to cluster. Programs typically contain a number of iterative loops and subroutines. Once a loop or subroutine is entered, there are repeated references to a small set of instructions. Similarly, operations on tables and arrays involve access to a clustered set of data bytes. Over a long period of time, the clusters in use change, but over a short period of time, the processor is primarily working with fixed clusters of memory references. Accordingly, it is possible to organize data across the hierarchy such that the percentage of accesses to each successively lower level is substantially less than that of the level above.

Consider a two-level cache hierarchy. Let level 2 memory contain all program instructions and data. The current clusters can be temporarily placed in level 1. From time to time, one of the clusters in level 1 will have to be swapped back to level 2 to make room for a new cluster coming in to level 1. On average, however, most references will be to instructions and data contained in level 1. This principle can be applied across more than two levels of memory. The fastest, smallest, and most expensive type of memory consists of the registers internal to the processor. Typically, a processor will contain a few dozen such registers, although some processors contain hundreds of registers. Skipping down two levels, main memory is the principal internal memory system of the computer. Each location in main memory has a unique address, and most machine instructions refer to one or more main memory addresses. Main memory is usually extended with a higher-speed, smaller cache. The cache is not usually visible to the programmer or, indeed, to the processor. It is a device for staging the movement of data between main memory and processor registers to improve performance.

2 So, What Does the Operating System Do?

Multiprogramming and Multitasking

- Multiprogramming needed for efficiency
- Single user cannot keep CPU and I/O devices busy at all times
- Multiprogramming organizes jobs (code and data) so the CPU always has something to do
- A subset of total jobs in system is kept in memory
- One job selected and run via *job scheduling*
- When the OS must wait (for I/O, for example), OS switches to another job

Multiprogramming and Multitasking

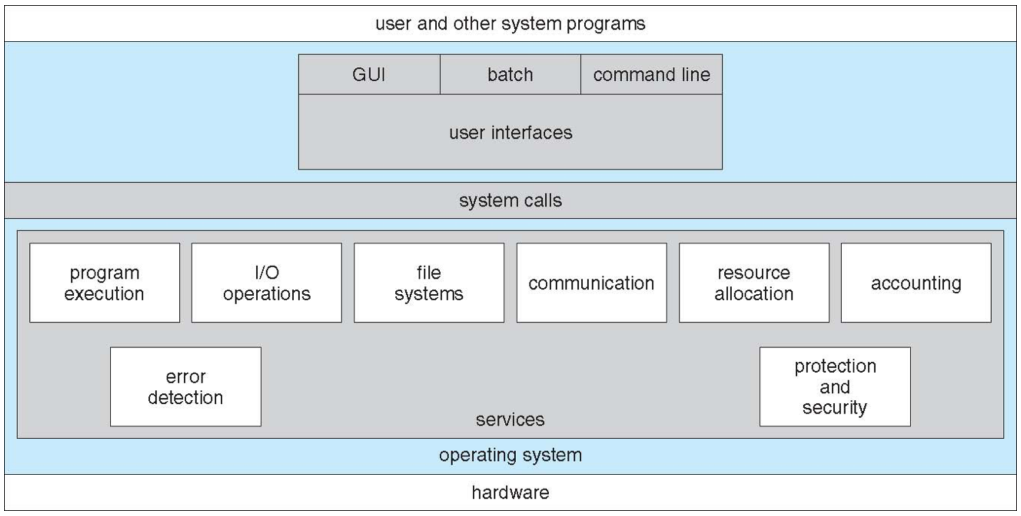
- *Multitasking* (also known as *timesharing*) is a logical extension to multiprogramming where the CPU switches between jobs so frequently that users can interact with each job while it's running
  - Enables *interactive computing*
  - *Response time* should be less than 1 second
  - Each user has at least one program executing in memory ⇒ *process*
  - If processes don't fit in memory, *swapping* moves them in and out to run
  - *Virtual memory* allows execution of processes not completely in memory

Operating System Operations

- The OS is a hardware *interrupt-driven* system
- Know the difference between *Interrupt* and *Exception*
- *Dual-mode* operation allows the OS to protect itself and other system components
  - *User mode* and *kernel mode*
  - Hardware provides a *mode flag* or *mode bit* that indicates the mode
  - Some instructions are *priveleged*, which means that they only execute in kernel mode
  - System call changes mode to kernel, return from the call returns the mode to user mode
- Recent CPUs support multi-mode operations, which are used to implement *virtual machine managers* for guest *virtual machines*

3 Operating System Services and The System Call Interface

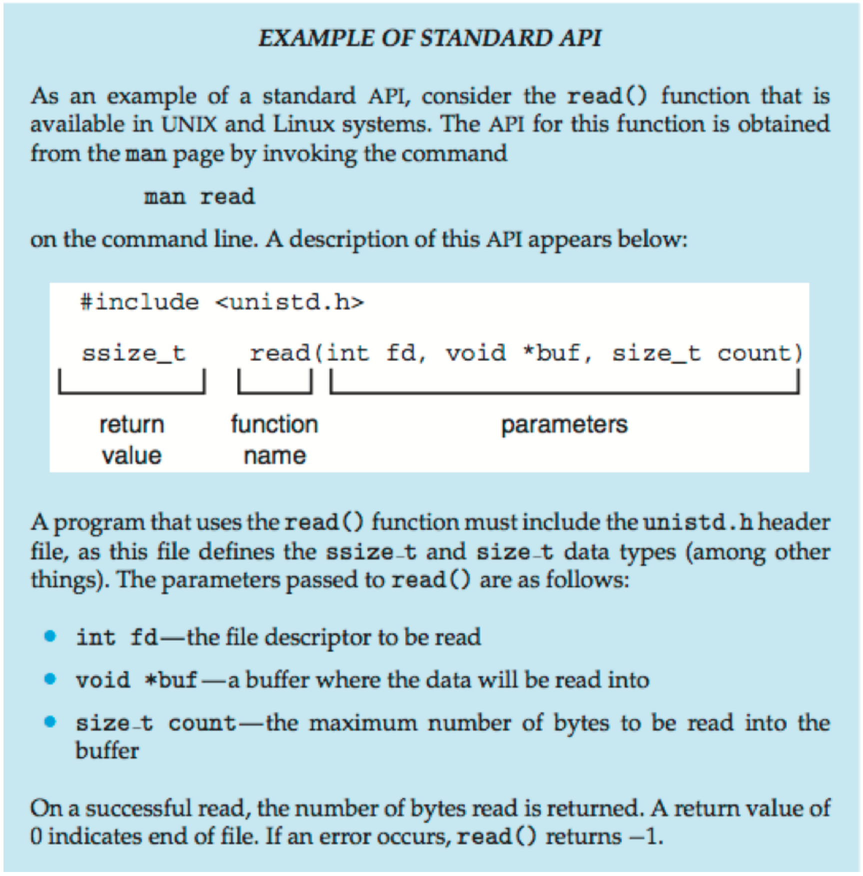
A View of Operating System Services



System Calls

- Programming interface to the services provided by the OS
  - Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level *Application Programming Interface* (API) rather than direct system call use
  - Most well-known APIs are the Win32 API for Windows and the POSIX API for POSIX-based system (including almost all versions of UNIX, Linux, and macOS)

Example of a Standard API



EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t    read(int fd, void *buf, size_t count)
```

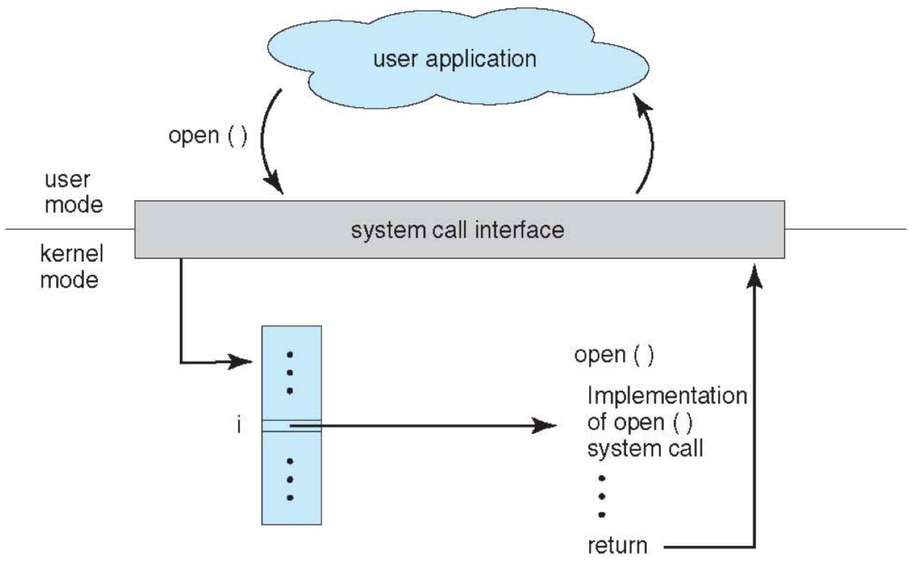
$\underbrace{\texttt{ssize\_t}}_{\text{return value}}\ \underbrace{\texttt{read}}_{\text{function name}}\underbrace{\texttt{(int fd, void *buf, size\_t count)}}_{\text{parameters}}$

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

The relationship between API, System Call, and OS Service



System Call Implementation

- Typically, each system call is identified by a unique number
- The system call interface invokes the intended system call in the OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the syscall is implemented
  - Most details of the OS interface hidden from programmer by the API
  - This process managed by the run-time support library in the OS

Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications
- Protection
- Process control
  - end, abort
  - load, execute
  - create process, terminate process
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal even

  - allocate and free memory
- File management
  - create file, delete file
  - open, close file
  - read, write re-position
  - get and set file attributes
- Device management
  - request and release a device
  - read, write, and re-position
  - get device attributes, set device attributes
  - logically attach or detach devices
- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, device attributes
- Communications
  - create and delete communication connections
  - send, receive messages from client of server
  - *shared memory model* create and gain access to memory regions
  - transfer status information
  - attach and detach remote devices
- Protection
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

System Programs

- Most users view of the OS is defined by system programs, not the actual system calls
- Provide the environment for program development and execution
- Categories
  - File manipulation
  - Status information
  - Programming language support (compilers, linkers)
  - Communications
  - Background services
  - Application programs

- Provide a convenient environment for program development and execution Some of them are simply user interfaces to system calls; others are considerably more complex
- File management - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories

Status information:

- Some ask the system for info - date, time, amount of available memory, disk space, number of users Others provide detailed performance, logging, and debugging information
- Typically, these programs format and print the output to the terminal or other output devices
- Some systems implement a registry - used to store and retrieve configuration information

4 Virtual Machines and Virtualization

Virtualization

- Enables a single PC or server to run multiple operating system or multiple sessions of a single OS
- A machine can host numerous applications, including those that run on different operating systems, on a single platform
- Host operating system can support a number of *virtual machines* (VM)
  - Each VM has the characteristics of a particular OS and, in some types of virtualization software, the characteristics of a particular hardware platform

Traditionally, applications have run directly on an OS on a PC or a server. Each PC or server would run only one OS at a time. Thus, the vendor had to rewrite parts of its applications for each OS/platform they would run on. An effective strategy for dealing with this problem is known as virtualization . Virtualization technology enables a single PC or server to simultaneously run multiple operating systems or multiple sessions of a single OS. A machine with virtualization can host numerous applications, including those that run on different operating systems, on a single platform. In essence, the host operating system can support a number of virtual machines (VM) , each of which has the characteristics of a particular OS and, in some versions of virtualization, the characteristics of a particular hardware platform.

The VM approach is becoming a common way for businesses and individuals to deal with legacy applications and to optimize their hardware usage by maximizing the number of kinds of applications that a single computer can handle. Commercial VM offerings by companies such as VMware and Microsoft are widely used, with millions of copies having been sold. In addition to their use in server environments, these VM technologies also are used in desktop environments to run multiple operating systems, typically Windows and Linux.

Virtual Machine Concept

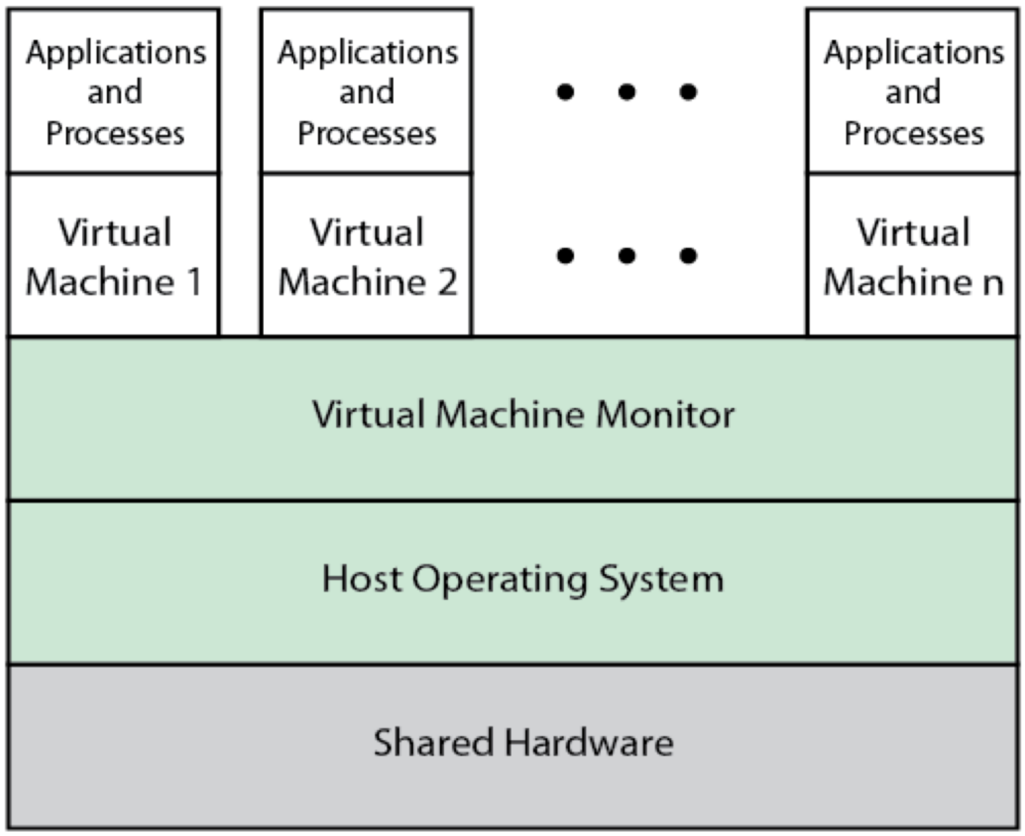


Figure 2.13 Virtual Memory Concept

The specific architecture of the VM approach varies among vendors. Figure 2.13 in the Stallings textbook shows a typical arrangement. The virtual machine monitor (VMM) , or hypervisor , runs on top of (or is incorporated into) the host OS. The VMM supports VMs, which are emulated hardware devices. Each VM runs a separate OS. The VMM handles each operating system's communications with the processor, the storage medium, and the network. To execute programs, the VMM hands off the processor control to a virtual OS on a VM. Most VMs use virtualized network connections to communicate with one another, when such communication is needed. Key to the success of this approach is that the VMM provides a layer between software environments and the underlying hardware and host OS that is programmable, transparent to the software above it, and makes efficient use of the hardware below it.

5 Key Points

Key Points

Contents