CSC 453 Operating Systems

Lecture 3: Operating-System Structures

Operating System Components

- Operating systems are large and complex - the only way to manage such a project is to divide it into smaller subsystems.
- Operating systems have to provide:
  - Process management
  - Memory management
  - File Management
  - I/O Management
  - Disk Management
  - Networking
  - Protection
  - User Interface
- Not all operating systems have the same structure.

Process Management

- A ***process*** is a program in execution. Examples include:
  - Batch jobs
  - Time-shared user programs
  - System tasks (such as printer spoolers)
- If two users are each running the same program, that is two separate processes.
- Processes need resources (such as CPU time, memory, I/O devices) in order to run.

Process Management Activities

The operating system is responsible for:

- creating and deleting user and system processes
- suspending and resuming processes
- mechanisms for process synchronization
- mechanisms for process communication
- mechanisms for handling ***deadlocks***

Memory Management

- All processes need memory to store the instructions that comprise the program as well as its data.
- To execute a program, the computer must be able to translate its addresses into absolute addresses.
- When a process terminates, its memory must be freed and reallocated.
- We assume that multiple processes will be in memory concurrently.

Memory Management Activities

The operating system is responsible for:

- keeping track of the parts of memory in use and by which process
- deciding which process will be loaded into memory as it becomes available
- allocating and freeing memory as necessary

File Management

- Files are a logical organization of data.
- A ***file*** can be defined as a collection of related data defined by its creator.
- The operating system maps files onto the various input/output and secondary storage devices that the computer system uses.

File Management Activities

The operating system is responsible for:

- creating and deleting files and directories
- supporting primitives for manipulating files and directories
- mapping files onto I/O and storage devices
- backing up files onto ***non-volatile*** storage devices.

I/O System Management

- The operating system protects the user from having to deal with the idiosyncrasies of the computer's various I/O devices.
- The I/O subsystem includes:
  - memory management for buffers cache and spoolers.
  - A general device driver interface
  - Device-specific drivers

Secondary Storage Management

- Secondary storage is needed to hold all the computer's software and data because
  - it can't all fit in memory at the same time
  - computer memory is volatile.
- The operating system is responsible for:
  - free-space management
  - storage allocation
  - disk scheduling

Networking

- A distributed system has a collection of processors that do not share memory, peripheral devices or a clock.
- These processors need to communicate to be able to coordinate their work.
- Network access, facilitated by the operating system, allows the sharing of various resources.

Protection System

- A multi-user, multitasking system must protect its processes' resources from the activities of other processes. These resources include:
- memory
  - files
  - I/O devices
- Protection means providing a control mechanism for the resources of a computer system

Command-Interpreter

- The command interpreter is the most visible part of the operating system.
- Commands have traditionally been given control card or command line. More recently, ***command-line interpreters*** have been replaced and supplemented by ***graphical-user interfaces***.

Services of the Operating System

- The operating system provides a range of services for programs and their users, including
  - program execution
  - I/O operations
  - file-system manipulation
  - communications
  - error detection

Administrative Functions of the Operating System

- There are also services that exist for the benefit of system administration and not for user programs:
  - resource allocation
  - accounting
  - protection

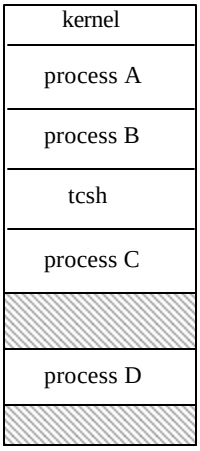
System Calls

- Systems calls are the way in which processes request services from the operating system.
  - These are generally written in assembly language.
  - Systems calls can also be written in some higher-level languages, such C, BLISS, BCPL, and PERL.
- Input/output statements in higher-level languages are compiled into procedures that make extensive use of system calls.

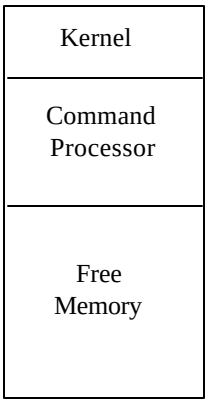
Process Control System Calls

- An operating system will include system calls for
  - creating and terminating processes
  - loading and executing processes
  - ending and aborting processing
  - getting and setting process attributes
  - sending a signal or waiting for a signal

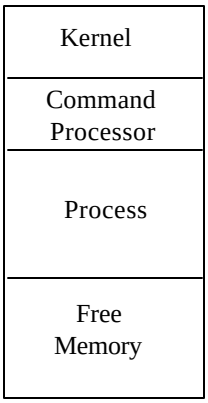
UNIX Program Execution



MS-DOS Program Execution

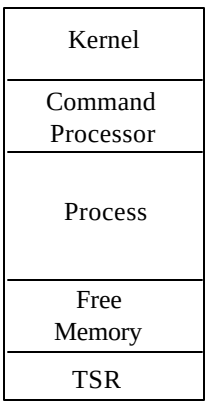


At Startup



Running A Program

MS-DOS System With TSR



File Manipulation System Calls

- An operating system will include system calls for:
  - opening and closing files
  - read and writing files
  - repositioning the file pointer
  - getting and setting file attributes
  - creating and deleting files

Device Management System Calls

- An operating system will include system calls for:
  - requesting and releasing devices
  - reading, writing and repositioning the device
- These system calls will be analogous to those for files

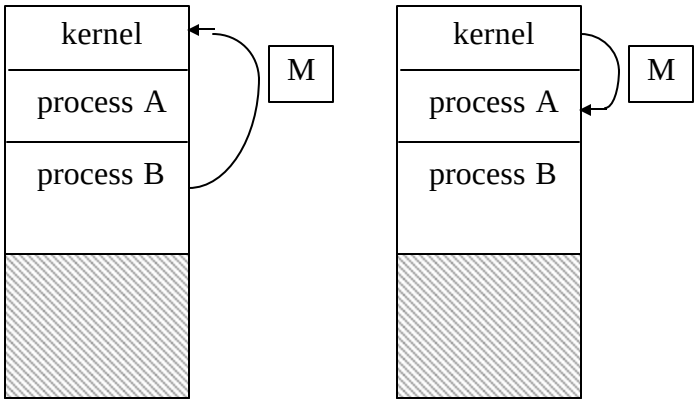
Information Maintenance System Calls

- Information maintenance calls include information such as:
  - date and time
  - number of current users
  - current operating system version number
  - amount of free disk space

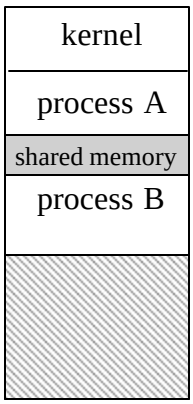
Communications

- There are two common models for communications:
  - message-passing model
  - shared memory model

Message-Passing Model



Shared Memory Model



Communications System Calls

- An operating system will include system calls for:
  - creating and deleting communication channels
  - sending and receiving messages
  - transferring status information
  - attaching and detaching remote devices

System Programs

- Systems programs do some of the systems-related work for users and serve as an additional interface layer between the user and the hardware.

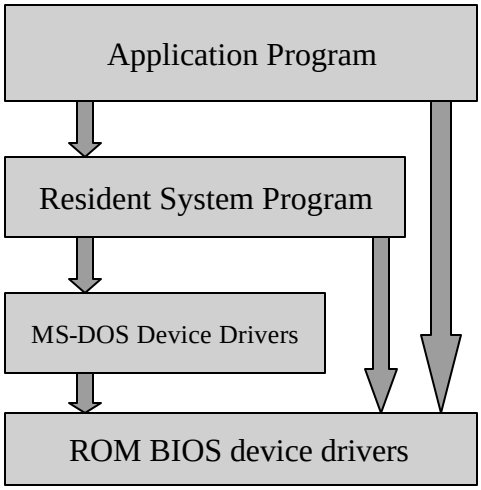
What Do System Programs Do?

- System programs:
  - perform file manipulation
  - provide status information
  - perform file modification
  - support programming languages
  - load and execute programs
  - perform communications

System Structure

- Given the complexity of an operating system, a structured approach is badly needed.
- There are five different ways in which the internal structure of an operating system may be structured:
  - Simple Structure
  - Monolithic Structure
  - Layered Structure
  - Virtual Machines
  - Client-Server Model

Simple Structure



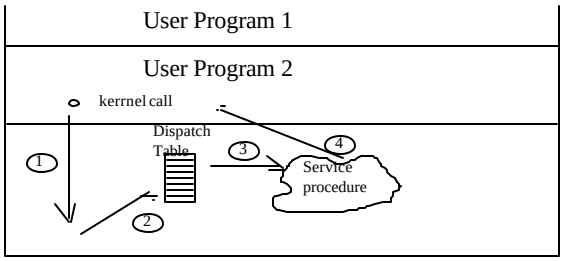
Limited Structure

users		
shells	compilers & interpreters	systems libraries
system-call interface to kernel		
signals terminal handling character I/O terminal drivers	file system swapping block I/O disk & tape drivers	CPU scheduling page replacement demand paging virtual memory
kernel interface to kernel		
terminal controllers terminals	device controllers disks and tapes	memory controllers physical memory

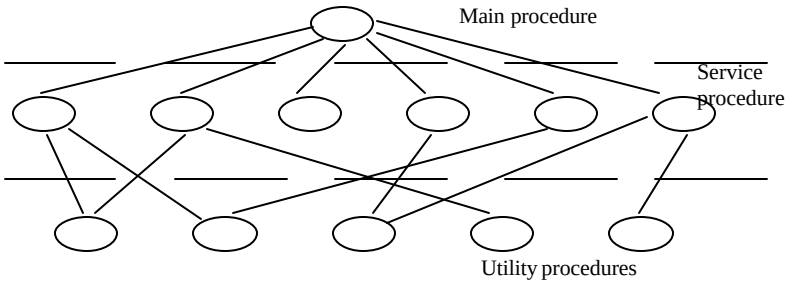
What Are Monolithic Systems?

- Monolithic systems could be called “the big mess”, because there is no real structure to the operating system’s kernel. Each procedure has its own parameters and results, but any procedure can call any other procedure.
- Such systems may actually have some structure, provided by having the operating system place parameter is well-defined areas such as registers or the stack, and then executing a ***supervisor call*** or ***kernel call***.

Monolithic Systems and Systems Calls



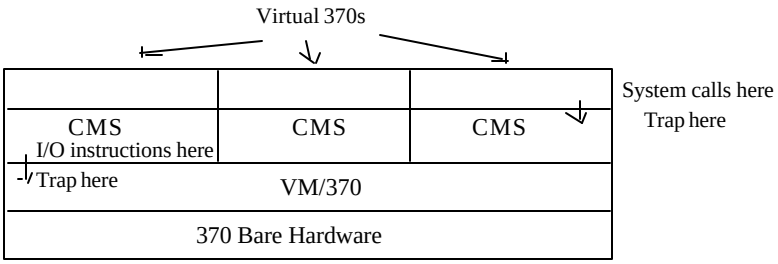
Structure In Monolithic Systems



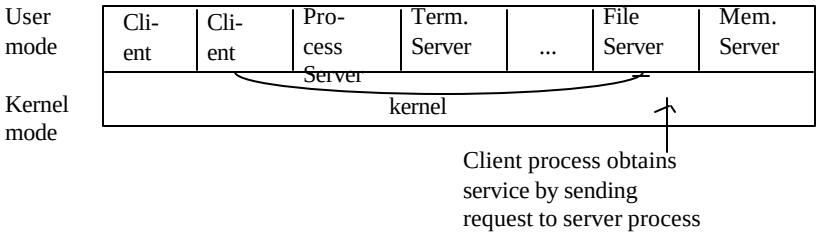
Layered Structure - THE

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator - process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

Virtual Machine



Operating System Structure: Client-Server Model



System Design Parameters

- Design will depend very much on hardware and type of system desire:
  - multi-user vs. single user
  - batch vs. interactive
  - distributive vs. standalone
  - real-time vs. general purpose

Defining System Design Goals

- Requirements are divided into user goals and systems goals.
- User goals usually include convenience, ease of use, speed and reliability.
- System goals usually include being easy to design, implement and maintain; flexibility; reliability and efficiency.
- These are all too vague to be useful without greater specificity.

Mechanisms vs. Policies

- Mechanisms determine ***how*** to do something.
  - Using a timer to provide CPU protection is an example of a mechanism.
- Policies determines ***what*** will be done.
  - The size of a time slice is an example of a policy decision.

Mechanisms and Policies

- Separating mechanisms and policies gives the design greater flexibility.
  - General mechanisms are good in that changes in policy do not require design changes.
  - Microkernel operating systems use a set of basic building-block to implement mechanisms that are completely policy-free.
  - The Apple MacIntosh has a tight connection between mechanism and policy to give a look and feel that is completely consistent within the system.

Implementation Languages

- Most operating systems are currently implemented in higher-languages.
- Higher-level languages allows for faster, more efficient and better-designed code that is easier to ***port***.
- Disadvantages purportedly include reduced speed, greater storage requirements.

System Generation

- The operating system is typically distributed on tape or disk and has to be ***configured*** for a particular system.
- Configuration specifications include:
  - Type of CPU(s)
  - Amount of Memory
  - Available Devices (Type and other characteristics)
  - Desired options (such as maximum number of processes)