OS Components and Structure

H. Levy

OS Structure

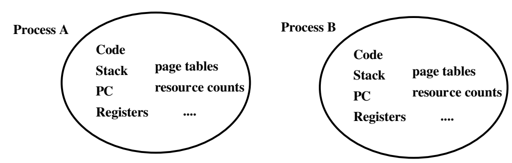
- **To understand an OS, let's first look at its components and then how they're composed or organized.**
- **We'll come back and look at each of these in detail as the course progresses.**
- **Realize that it's never as simple as it looks. These basic concepts exist in some form in all systems, however each system implements them in a slightly different way.**
- **Also, the divisions between components may not be as clean in the real world as in the model**

Process Management

- **An operating system executes many kinds of activities:**
  - user programs
  - batch jobs or command scripts
  - system programs: print spoolers, name servers, file servers, network listeners, etc...
- **Each of these "execution entities" is encapsulated in a <u>process</u>.**
- **The process includes the execution context (pc, registers, vm, resources, etc) and all info the activity (program) needs to run.**
- **The OS schedules processes to run.**

Processes

- **A <u>program</u> is a passive thing -- just a file on the disk with code that is *potentially* runnable.**
- **A <u>process</u> is one instance of a program *in execution*; at any instant, there may be many processes running copies of a single program (e.g., an editor): each is a *separate, independent* process.**



Process Operations

- **Processes are fundamental OS-provided objects.**
- **The OS supports operations on processes, e.g.:**
  - create a process
  - delete a process
  - suspend a process
  - resume a process
  - inter-process communication
  - inter-process synchronization
  - create/delete a subprocess

Memory Management

- **Primary memory is the direct access storage for CPU.**
- **Programs must be stored in memory to execute.**
- **OS must:**
  - allocate memory space for programs (both explicitly and implicitly)
  - deallocate memory space when needed
  - maintain the mappings from virtual to physical memory (page tables)
  - decide how much memory to allocate to each process, and when a process should be removed from memory (policies)

I/O Management

- Much of the OS kernel is concerned with I/O.
- The OS provides a standard interface between programs (user or sytem) and devices.
- Device drivers are the processes responsible for each device type. A driver encapsulates device-specific knowledge, e.g., for device initiation and control, interrupt handling, and errors.
- There may be a process for each device, or even for each I/O request, depending on the particular OS.

Secondary Storage Management

- Secondary storage (disk) is the *persistent* memory, i.e., it endures system failures (we hope).
- Low-level OS routines are typically responsible for low-level disk function, such as scheduling of disk operations, head movement, error handling, etc.
- These routines may also be responsible for managing space on the disk....
- BUT, the line between this and the file system is very fuzzy...space management functions may belong in the file system.

File Management

- Secondary storage devices are too crude to use directly for long-term storage.
- The file system provides logical objects and logical operations on those objects.
- A file is the basic long-term storage entity: a file is a *named* collection of *persistent* information that can be read or written.
- The file system supports directories -- special files that contain names of other files and associated file information.

File Management

- File system provides standard file operations, e.g.:
  - file creation and deletion
  - directory creation and deletion
  - manipulation of files and directories: read, write, extend, rename, protect....
  - file copy
- The file system also provides general services, e.g.:
  - backup
  - maintaining mapping information
  - accounting and quotas

Protection System

- protection is a general mechanism throughout the OS
- all resource objects need protection
  - memory
  - processes
  - files
  - devices
- protection mechanisms help to detect errors as well as to prevent malicious destruction

Command Interpreter

- process that handles interpretation of user input commands from keyboard (or script files)
- on some systems, command interpreter is a standard part of the OS
- on others, it's simply a non-privileged process that interfaces to the user, permitting replacement of interpreter with others
- on others, there's not really a command language (e.g., the MacIntosh has no commands in the conventional sense)

Accounting System

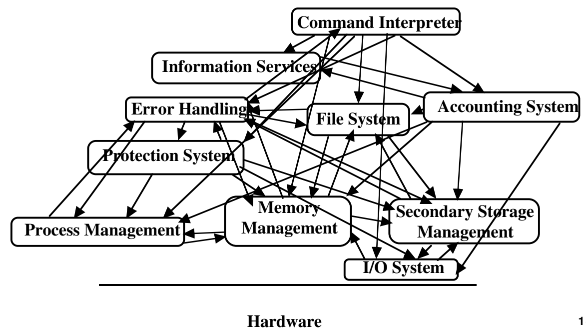
- General facility for keeping track of resource usage for all system objects
- May be used to enforce quotas, or to produce bill$.

Error Handling System

- Must catch errors and reflect them back to user with proper information
- Should provide a way to recover and restart
- Standard way to report errors and print error messages

OS Structure

The OS (a *simplified* view)

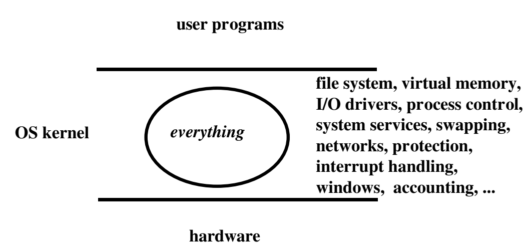


OS Structure

- An OS consists of all of these components, plus lots of others, plus system service routines, plus system programs (privileged and non-privileged), plus ....
- The big issue:
  - how do we organize all of this?
  - what are the entities and where do they exist?
  - how does these entities cooperate?
- Basically, how do we build a complex system that's:
  - performant
  - reliable
  - extensible

Structure

- Traditionally, systems such as Unix were built as a *monolithic* kernel:

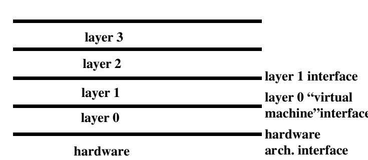


Structure

- Problems with monolithic kernels:
  - hard to understand
  - hard to modify
  - unreliable: a bug *anywhere* causes a system crash
  - hard to maintain
- Since the beginnings of OS design, people have sought ways to organize the OS to simplify its design and construction.

Structuring

- Traditional approach is layering: implement system as a set of layers, where each layer is a *virtual machine* to the layer above.
- That is, each layer provides a "machine" that has higher level features.



Layering in THE

- The first description of this approach was Dijkstra's THE system.

user programs
I/O device buffering
console device (commands)
memory management
CPU scheduling (processes)
hardware

THE System

- System was composed as a set of sequential processes.
- Each peforms a sequential computation.
- Processes communicate through explicit synchronization statements.
- Each process could be tested and verified independently.
- Each level sees a logical machine provided by lower levels.
  - level 2 sees virtual processors
  - level 3 sees VM (really segments)
  - level 4 sees a "virtual console"
  - level 5 sees "virtual" I/O drivers

Problems with Layering

- Systems must be hierarchical, but real systems are more complex than that, e.g.,
  - file system would like to be a process layered on VM
  - VM would like to use files for its backing store I/O
- Approach is not flexible.
- Often has poor performance due to layer crossings.
- Systems are often modelled as layered structures but not built that way (for better or worse).

Microkernel Approach

- The organizing structure currently in vogue is the *microkernel* OS.
- Goal is minimize what goes in the kernel, and implement much of the OS as user-level processes. This results in:
  - better reliability
  - ease of extension and customization
  - mediocre performance (unfortunately)
- First microkernel system was Hydra (CMU, 1970)
- Examples of microkernel systems are the CMU Mach system, Chorus (French Unix-like system), and in some ways Microsoft NT/Windows.

Microkernel System Structure