COP 4225 Advanced Unix Programming

Operating System Review

Chi Zhang
czhang@cs.fiu.edu
About the Course

- Prerequisite: COP 4610
- Concepts and Principles
- Programming
  - System Calls
- Advanced Topics
  - Internals, Structures, Details
  - Unix / Linux
What is an Operating System?

- A general purpose software that acts as an intermediary between users of a computer and the computer hardware.
  - Encapsulates hardware details.
  - Controls and coordinates the use of the hardware among the various application programs for the various users.
- Use the computer hardware in an efficient manner.
Abstract View of O.S.
CPU scheduling – the system must choose among several jobs ready to run.

Memory management – the system must allocate the memory to several jobs.

I/O routine supplied by the system.

Allocation of devices (e.g. Disk usage).
Parallel Systems

- Multiprocessor systems with more than one CPU in close communication.

- *Tightly coupled system* – processors share memory and a clock; communication usually takes place through the shared memory.

- Advantages of parallel system:
  - Increased *throughput*
  - Economical
  - Increased reliability
Parallel Systems (Cont.)

- **Symmetric multiprocessing** (SMP)
  - Each processor runs an identical copy of the operating system.
  - Many processes can run at once without performance deterioration.
  - Most modern operating systems support SMP
Computer-System Architecture
Computer-System Operation

- I/O devices and the CPU can execute concurrently, competing for memory accesses.
  - Memory controller synchronizes accesses.
- Each device controller has a local buffer.
- CPU moves data between main memory and local buffers of controllers.
- I/O is from the device to local buffer of controller.
  - The buffer size varies
- Device controller informs CPU that it has finished its operation by causing an *interrupt*. 
Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the *interrupt vector*, which contains the addresses of all the service routines.
- The operating system preserves the state of the CPU before the interrupt by storing registers and the program counter.
- Incoming interrupts are *disabled* while another interrupt is being processed to prevent a *lost interrupt*.
- A *trap* is a software-generated interrupt caused either by an error or a user request.
I/O Structure

- **Device-status table** contains entry for each I/O device indicating its type, address, and state.

- Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

- After I/O starts, control returns to user program only upon I/O completion (Synchronous I/O).
  - System call – request to the operating system to allow user to wait for I/O completion.
  - Wait instruction idles the CPU (could be used by other processes) until the next **interrupt**
  - **Wait loop** (contention for memory access and CPU).
    - Poll the device status if it does not support interrupt
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing.

- After I/O starts, control returns to user program without waiting for I/O completion (Asynchronous I/O).
Device-Status Table

- **device: card reader 1**
  - status: idle

- **device: line printer 3**
  - status: busy

- **device: disk unit 1**
  - status: idle

- **device: disk unit 2**
  - status: idle

- **device: disk unit 3**
  - status: busy

  - request for disk unit 3
    - file: xxx
    - operation: read
    - address: 43046
    - length: 20000

  - request for disk unit 3
    - file: yyy
    - operation: write
    - address: 03458
    - length: 500

- request for line printer
  - address: 38546
  - length: 1372
Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
  - Direct I/O: Device $\leftrightarrow$ CPU Register $\leftrightarrow$ Mem
- Only one interrupt is generated per block, rather than the one interrupt per byte.
Storage Hierarchy

- Storage systems organized in hierarchy.
  - Speed / Cost / Volatility
- Caching – copying information into faster storage system
  - Consistency and Coherency (Multiple CPUs): guaranteed by the hardware.
  - Main memory can be viewed as a last cache for secondary storage (e.g. Hard disk).

<table>
<thead>
<tr>
<th>magnetic disk</th>
<th>main memory</th>
<th>cache</th>
<th>hardware register</th>
</tr>
</thead>
</table>
Moving-Head Disk Mechanism

Disk surface is logically divided into *tracks*, which are subdivided into *sectors*.
The *disk controller* determines the logical interaction between the device and the computer.
Storage Structure

- Memory-mapped I/O
  - Physical memory is only part of the entire address space.
  - Each location on the screen is mapped to a memory location in the address space.

- Electronic Disk (Non-Volatile Memory)
  - DRAM array + battery-backed magnetic hard disk (small)
  - If external power is off, the data are copied from RAM to the disk
  - When the external power is restored, the data are copied back to the RAM.

- ROM
Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly.

- Dual-Mode Operation
- I/O Protection
- Memory Protection
- CPU Protection (Time-Sharing)
Dual-Mode Operation

- Provide hardware support for two modes of operations.
  1. *User mode* – execution done on behalf of a user.
  2. *Monitor mode* (also *kernel mode* or *system mode*) – execution done on behalf of operating system.
     - *Privileged instructions* can be issued only in monitor mode.
- *Mode bit* added to computer hardware to indicate the current mode: monitor (0) or user (1).
  - associated with each memory segment
Dual-Mode Operation (Cont.)

- OS boots in monitor mode.
- OS starts user processes in user mode.
- When an interrupt or fault occurs hardware switches to monitor mode.
  - *trap* for system calls

*Privileged instructions* can be issued only in monitor mode.
I/O Protection

- All I/O instructions are privileged instructions.
- Must ensure that a user program could never gain control of the computer in monitor mode (loaded by OS).
Memory Protection

- In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
  - **Base register** – holds the smallest legal physical memory address.
  - **Limit register** – contains the size of the range

- In user mode, memory outside the defined range is protected.
  - Attempts trap to error
Hardware Protection

- When executing in monitor mode, the operating system has unrestricted access to both monitor and user’s memory.
  - The system call implementation can write back to buffers in user processes.
- The load instructions for the base and limit registers are privileged instructions.
CPU Protection

- *Timer* – interrupts computer after specified period to ensure operating system maintains control.
  - Timer is decremented every clock tick.
  - When timer reaches the value 0, an interrupt occurs and control transfers to OS
- OS performs various housekeeping tasks and switch context if necessary.
- Load-timer is a privileged instruction.
Common System Components

- Process Management
- Main Memory Management
- File Management
- I/O System Management
- Secondary Management
- Networking
- Protection System
- Command-Interpreter System
Process Management

- A process is a program in execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
  - program counter: the next instruction to execute.
- OS is responsible for the following activities in connection with process management.
  - Process creation and deletion.
  - process suspension and resumption.
Main-Memory Management

- Memory is shared by the CPU and I/O devices.
- Main memory is a volatile storage device.
- The operating system is responsible for the following activities in connection with memory management:
  - Keep track of which parts of memory are currently being used and by whom.
  - Decide which processes to load when memory space becomes available.
  - Allocate and deallocate memory space as needed.
The operating system is responsible for the following activities in file management:

- File creation and deletion.
- Directory creation and deletion.
- Mapping files onto nonvolatile storage.
- File backup on stable (nonvolatile) storage media.
Secondary-Storage Management

The operating system is responsible for the following activities in disk management:

- Free space management
- Storage allocation
- Disk scheduling
I/O System Management

- The I/O system consists of:
  - A buffer-caching system
  - A general device-driver interface
  - Drivers for specific hardware devices
The program that reads and interprets control statements is called variously:
- command-line interpreter
- shell (in UNIX)

Its function is to get and execute the next command statement.
- process creation and management, I/O handling, secondary-storage management, main-memory management, file-system access, protection, networking
Additional Operating System Functions

Additional functions exist for ensuring efficient system operations.

• Resource allocation – allocating resources to multiple users or multiple jobs running at the same time.

• Accounting – keep track of and record which users use how much and what kinds of computer resources for account billing or for accumulating usage statistics.

• Protection – ensuring that all access to system resources is controlled.
System Calls

- System calls provide the interface between a running program and the operating system.

- Three general methods are used to pass parameters between a running program and the operating system.
  - Pass parameters in *registers*.
  - Store the parameters in a table in memory, and the table address is passed as a parameter in a register (Linux).
  - *Push* (store) the parameters onto the *stack* by the program, and *pop* off the stack by operating system.
Passing of Parameters As A Table

X: parameters for call
load address X
system call 13

X
register

use parameters from table X

code for system call 13

user program

operating system
UNIX Running Multiple Programs

- `fork()`
- `exec()`
- `wait() / waitpid()`

- Foreground or Background execution.
- When a process is running in background, it cannot receive input directly from the keyboard.
Communication Models

Communication may take place using either message passing (e.g. socket) or shared memory.
System Programs

- System programs provide a convenient environment for program development and execution. The can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs

- The view of O.S. seen by users is defined by the system programs, rather than by system calls.
UNIX System Structure

The original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts.

○ Systems programs

○ The kernel
  ● everything below the system-call interface and above the physical hardware
  ● Provides the file system, CPU scheduling, memory management, and other operating-system functions
### UNIX System Structure

<table>
<thead>
<tr>
<th>(the users)</th>
</tr>
</thead>
<tbody>
<tr>
<td>shells and commands</td>
</tr>
<tr>
<td>compilers and interpreters</td>
</tr>
<tr>
<td>system libraries</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>system-call interface to the kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>signals terminal handling</td>
</tr>
<tr>
<td>character I/O system</td>
</tr>
<tr>
<td>terminal drivers</td>
</tr>
<tr>
<td>file system</td>
</tr>
<tr>
<td>swapping block I/O system</td>
</tr>
<tr>
<td>disk and tape drivers</td>
</tr>
<tr>
<td>CPU scheduling</td>
</tr>
<tr>
<td>page replacement</td>
</tr>
<tr>
<td>demand paging</td>
</tr>
<tr>
<td>virtual memory</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>kernel interface to the hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>terminal controllers terminals</td>
</tr>
<tr>
<td>device controllers disks and tapes</td>
</tr>
<tr>
<td>memory controllers physical memory</td>
</tr>
</tbody>
</table>
Mechanisms and Policies

- Mechanisms determine how to do something, policies decide what will be done.
- The separation of policy from mechanism allows maximum flexibility if policy decisions are to be changed later.
  - Timer is a mechanism for CPU protection, but deciding how long the timer is to be set for a particular user is a policy decision.
Traditionally written in assembly language, operating systems can now be written in higher-level languages.

Code written in a high-level language:
- can be written faster.
- is more compact.
- is easier to understand and debug.
- easier to port (move to some other hardware) if it is written in a high-level language.
Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site.

SYSGEN program obtains information concerning the specific configuration of the hardware system.

Bootstrap program – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution.