CSC 4103 - Operating Systems
Spring 2007

Lecture - II

OS Structures

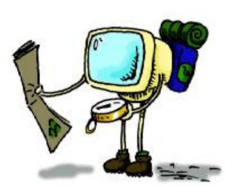
Tevfik Koşar

Louisiana State University

January 18th, 2007

Roadmap

- OS Operations
  - Kernel vs User Mode
- OS Structures
  - Multiprogramming and Multitasking
  - Storage Structure
  - System Calls



2

Operating System Operations

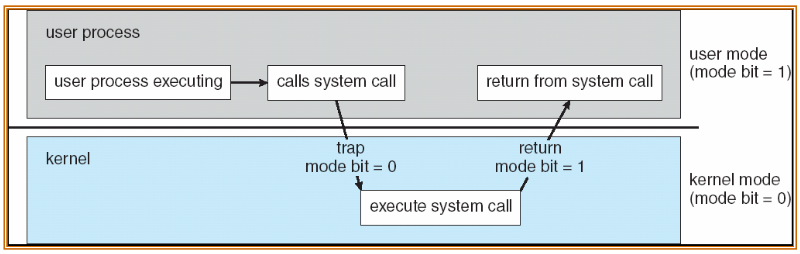
- Interrupt driven by hardware
- Unexpected errors can happen anytime
  - Software error or request creates **exception** or **trap**
    - eg. division by zero, invalid memory access
  - Other process problems include infinite loop, processes modifying each other or the operating system
- OS needs to protects itself
  - ➔ Dual-mode operation

Dual-Mode Operation

- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Protects OS from errant users, and errant users from each other
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user

Transition from User to Kernel Mode

- How to prevent user program getting stuck in an infinite loop / process hogging resources
  ➔ Timer: Set interrupt after specific period (1ms to 1sec)
  - Operating system decrements counter
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time



5

Operating System Structure

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

6

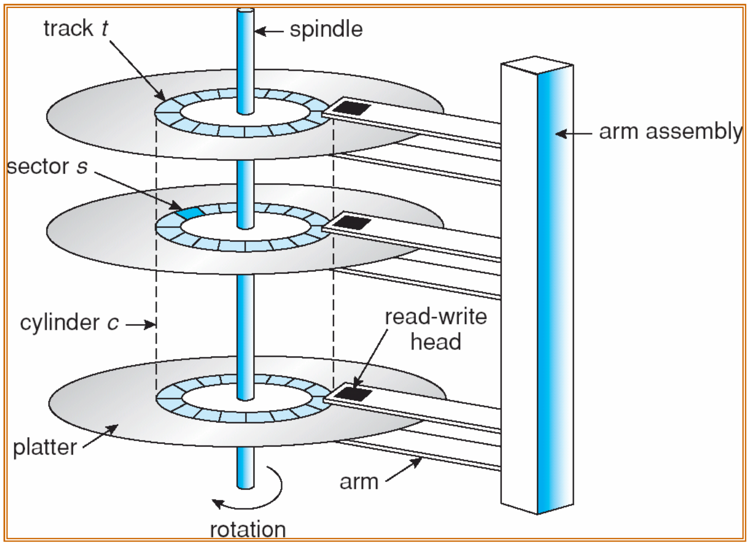
Operating System Structure

- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program loaded in memory and executing ⇨ **process**
  - If several jobs ready to be brought into memory ⇨ **job scheduling**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run

Storage Structure

- Main memory – only large storage media that the CPU can access directly.
- Secondary storage – extension of main memory that provides large nonvolatile storage capacity.
- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - 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.

Disk Architecture



9

Storage Structure

- Tertiary Storage: low cost, high capacity stroge
  - eg. tape libraries, CD, DVD, floppy disks
- Tape is an economical medium for purposes that do not require fast random access, e.g., backup copies of disk data, holding huge volumes of data.
- Large tape installations typically use robotic tape changers that move tapes between tape drives and storage slots in a tape library.
  - stacker – library that holds a few tapes
  - silo – library that holds thousands of tapes

10

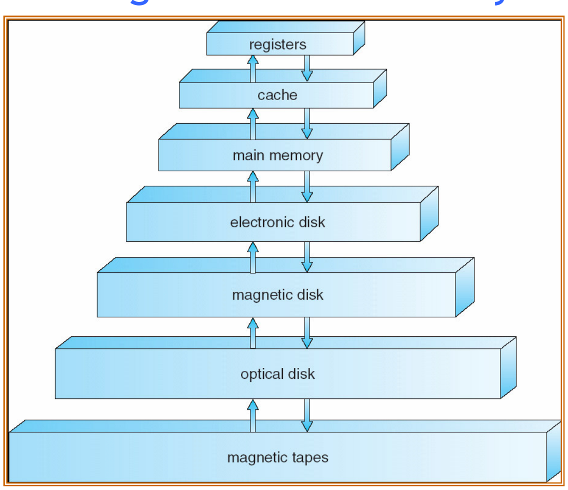
Storage Hierarchy

- Storage systems organized in hierarchy.
  - Speed
  - Cost
  - Volatility*
- *Caching* – copying information into faster storage system; main memory can be viewed as a last *cache* for secondary storage.

*volatile: loses its content when the power is off.

11

Storage-Device Hierarchy



12

Performance of Various Levels of Storage

- Movement between levels of storage hierarchy can be explicit or implicit

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

13

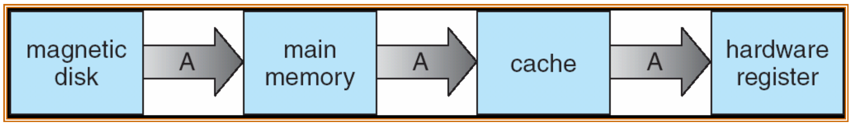
Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy

14

Migration of Integer A from Disk to Register

- Multitasking environments must be careful to use most recent value, not matter where it is stored in the storage hierarchy



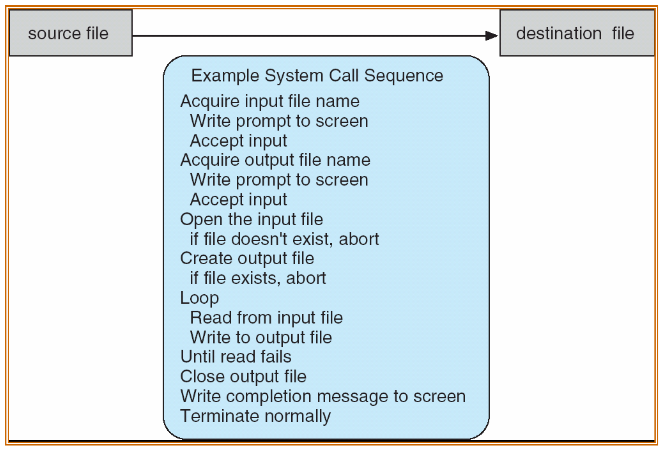
- Multiprocessor environment must provide cache coherency in hardware such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex
  - Several copies of a datum can exist
  - Various solutions covered in Chapter 17

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 Program Interface (API)** rather than direct system call use
  - Ease of programming
  - portability
- Three most common APIs are Win32 API for Windows, POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and Java API for the Java virtual machine (JVM)

Example of System Calls

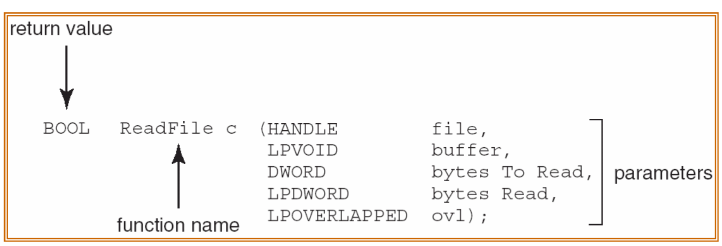
- System call sequence to copy the contents of one file to another file



17

Example of Standard API

- Consider the ReadFile() function in the
- Win32 API—a function for reading from a file



- A description of the parameters passed to ReadFile()
  - HANDLE file—the file to be read
  - LPVOID buffer—a buffer where the data will be read into and written from
  - DWORD bytesToRead—the number of bytes to be read into the buffer
  - LPDWORD bytesRead—the number of bytes read during the last read
  - LPOVERLAPPED ovl—indicates if overlapped I/O is being used

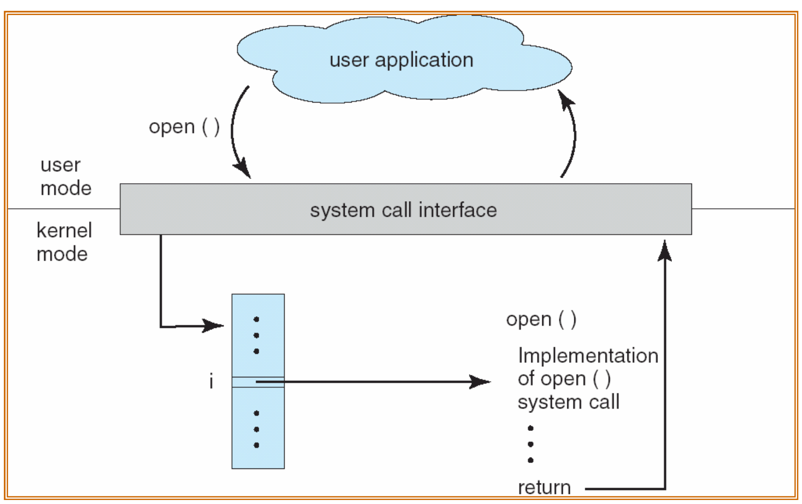
18

System Call Implementation

- Typically, a number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
- The system call interface invokes 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
    - Managed by run-time support library (set of functions built into libraries included with compiler)

19

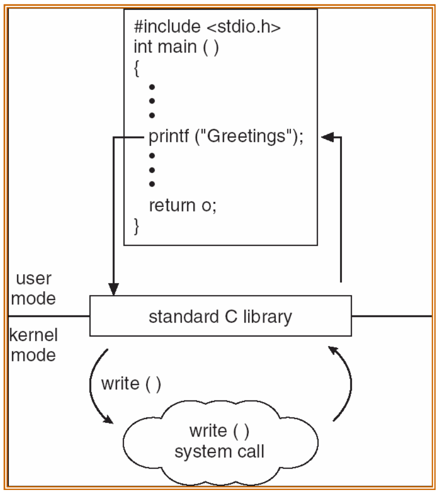
API – System Call – OS Relationship



20

Standard C Library Example

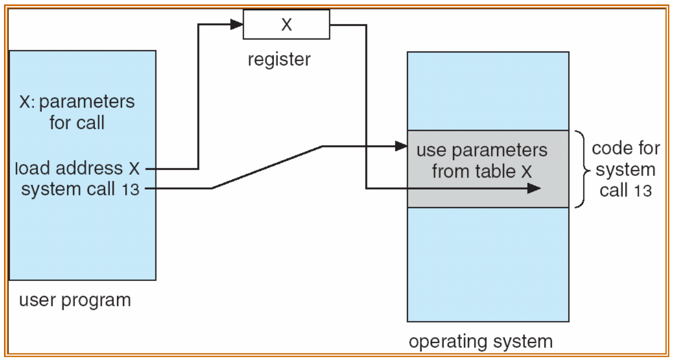
- C program invoking printf() library call, which calls write() system call



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - eg. source device, address and length of memory buffer
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in ***registers***
    - In some cases, may be more parameters than registers
  - Parameters stored in a ***block***, or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or *pushed*, onto the ***stack*** by the program and *popped* off the stack by the operating system
  - Block and stack methods do not limit the number or length of parameters being passed

Parameter Passing via Table



Solaris System Call Tracing

```
# ./all.d 'pgrep xclock' XEventsQueued
dtrace: script './all.d' matched 52377 probes
CPU FUNCTION
  0 -> XEventsQueued                     U
  0   -> _XEventsQueued                  U
  0     -> _X11TransBytesReadable        U
  0     <- _X11TransBytesReadable        U
  0     -> _X11TransSocketBytesReadable U
  0     <- _X11TransSocketBytesreadable U
  0     -> ioctl                         U
  0       -> ioctl                       K
  0         -> getf                      K
  0           -> set_active_fd           K
  0           <- set_active_fd           K
  0         <- getf                      K
  0         -> get_udatamodel            K
  0         <- get_udatamodel            K
...
  0         -> releasef                  K
  0           -> clear_active_fd         K
  0           <- clear_active_fd         K
  0           -> cv_broadcast            K
  0           <- cv_broadcast            K
  0         <- releasef                  K
  0       <- ioctl                       K
  0     <- ioctl                         U
  0   <- _XEventsQueued                  U
  0 <- XEventsQueued                     U
```

Any Questions?

Reading Assignment

- Read chapter 2 from Silberschatz.

Acknowledgements

- “Operating Systems Concepts” book and supplementary material by Silberschatz, Galvin and Gagne.

27