Outline (Chapters 1 and 2)

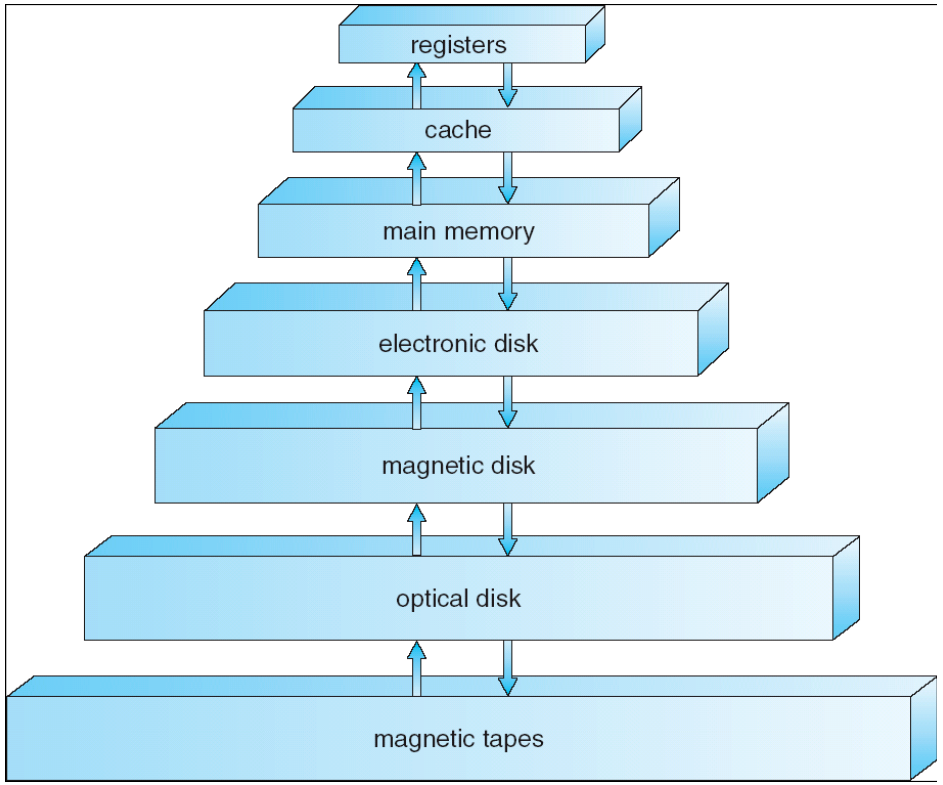
- Chapter 1
  - Introduce important concepts (caching)
- Chapter 2
  - Interacting with services provided by the OS
    - System calls - link between application programs and OS
    - System programs - users interact using programs
  - Installation, customization etc.
    - booting

Recap OS:allows for program execution

- load a program into memory and run that program, end execution, either normally or abnormally (indicating error)
  - I/O operations - A running program requires I/O, which may involve a file, an I/O device, shared with other programs or computers
  - Error detection – OS are constantly aware of errors
    - May occur in the CPU and memory hardware, in I/O devices and in user program
    - For each type of error, OS should take the appropriate action to ensure correct and consistent computing
    - Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

Storage structure

- Computer programs must be stored in main memory
  - Fast memory is expensive - we use hierarchy and move stuff around to achieve cost benefits and speed
  - Implicit or explicit



Hierarchy performance difference

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

Caching principle

- Caching is an important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information “in use” is copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is (cache hit), information used directly from the cache (fast)
  - If not (cache miss), data copied to cache and used there
    - May need to evict some other data (cache replacement)
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policies are important
  - Sometimes bring data before needed (pre-fetch)

Interfacing with OS

- User interface - Almost all operating systems have a user interface (UI). Varies between
  - **Command-Line (CLI)** (e.g., shells in UNIX, command.exe in Windows). The command line may itself perform functions or call other system programs to implement functions (e.g. in UNIX, /bin/rm to remove files) [more later]
  - **Graphics User Interface (GUI)** (e.g., MS windows, MAC OS X Aqua, Unix X & variants). point and click interface
  - **Batch**. Commands are given using a file/command script to the OS and are executed with little user interaction. Used in high performance computers. (e.g. .bat files in DOS, shell scripts, JCL interpreters for Main frames)

System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C, C++)
- Mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use
- 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)
- Why use APIs rather than system calls?
  - Underlying systems calls (error codes) can be more complicated. API gives a uniform, portable interface

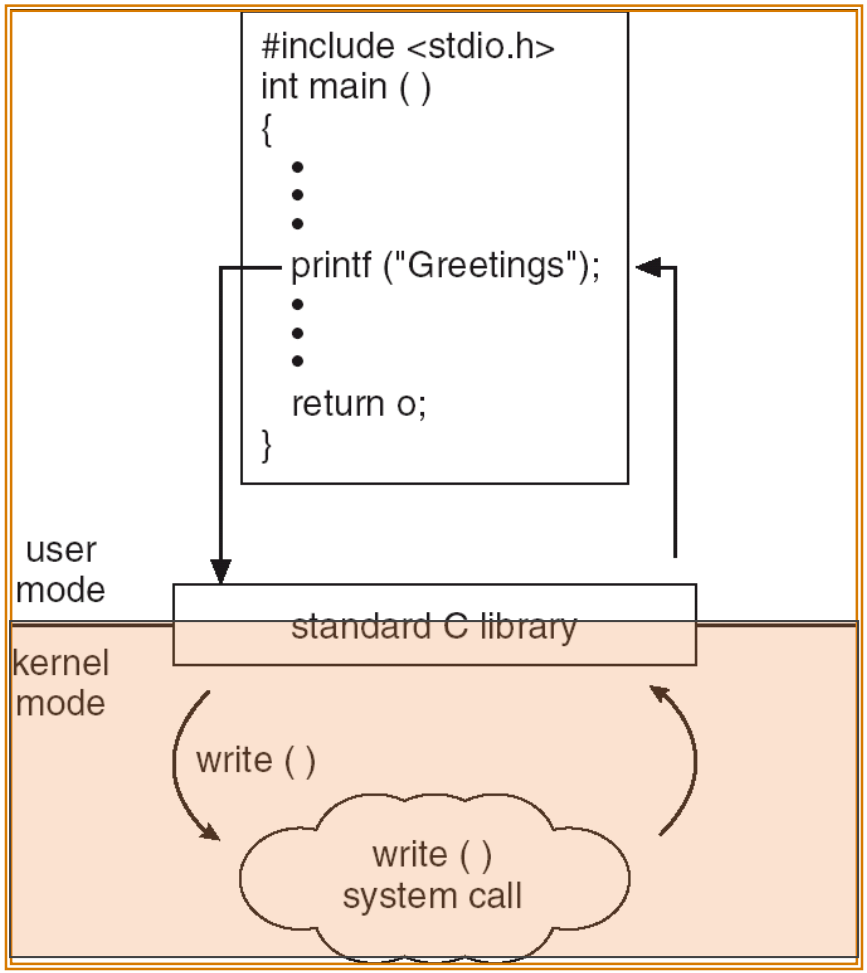
Example of System Calls

- System call sequence to copy the contents of one file to another file (POSIX like C pseudo code) (bold are API system calls)

```
write(1, “Input file\n”, 11);
read(0, &buffer, 100);
…..
fd = open(buffer, O_RDONLY);
outfd = open(buffer, O_WRONLY | O_CREAT | O_TRUNC,
    0666);
if (outfd < 0) abort(“File creation failed”);
…..
close(fd);
```

Standard C Library Example

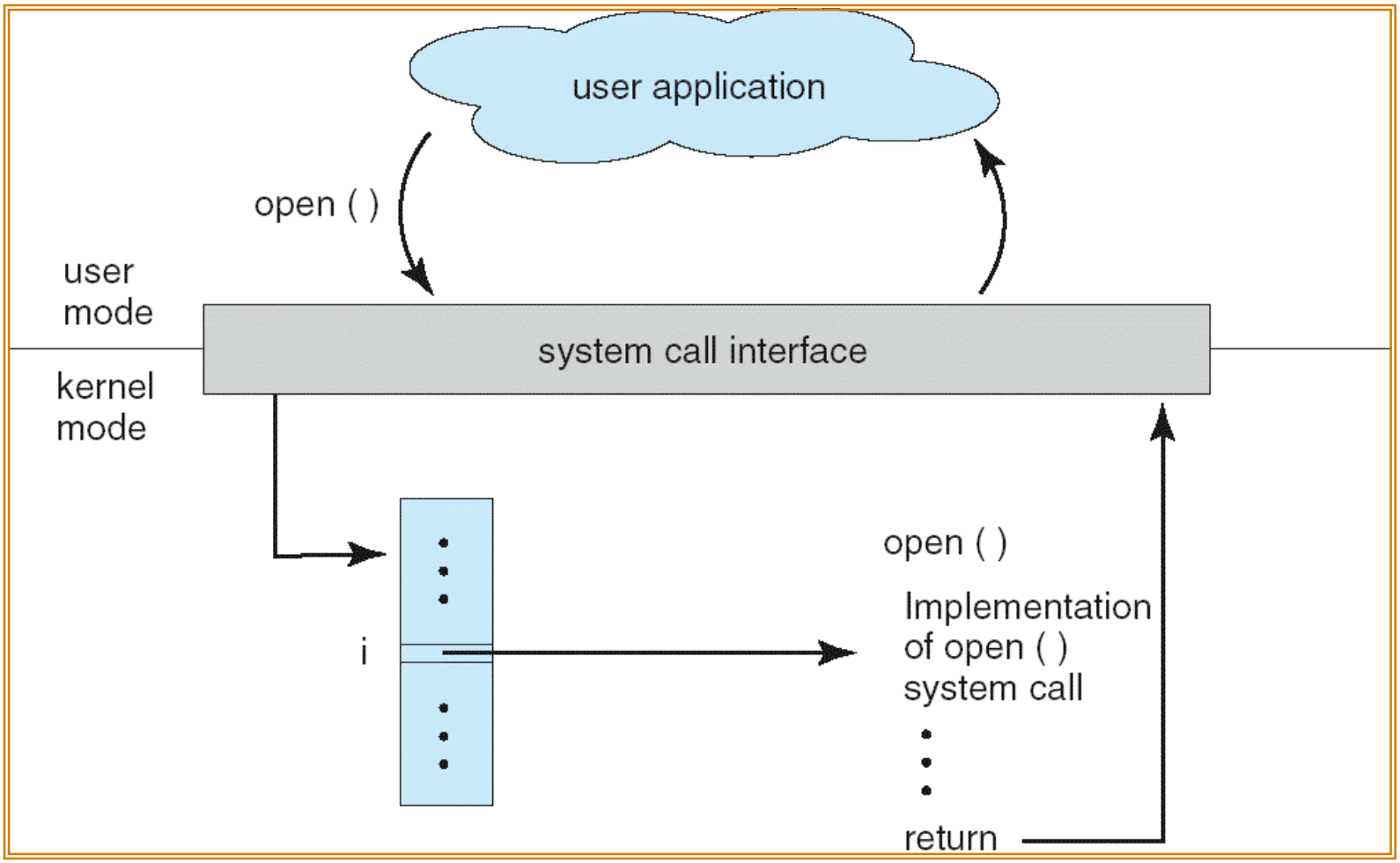
- C program invoking printf() library call, which calls write() system call



System Call Implementation

- A number associated with each system call
  - System-call interface maintains a table indexed according to these numbers
  - Additional info: check /usr/include/sys/syscall.h
- 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
  - Details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

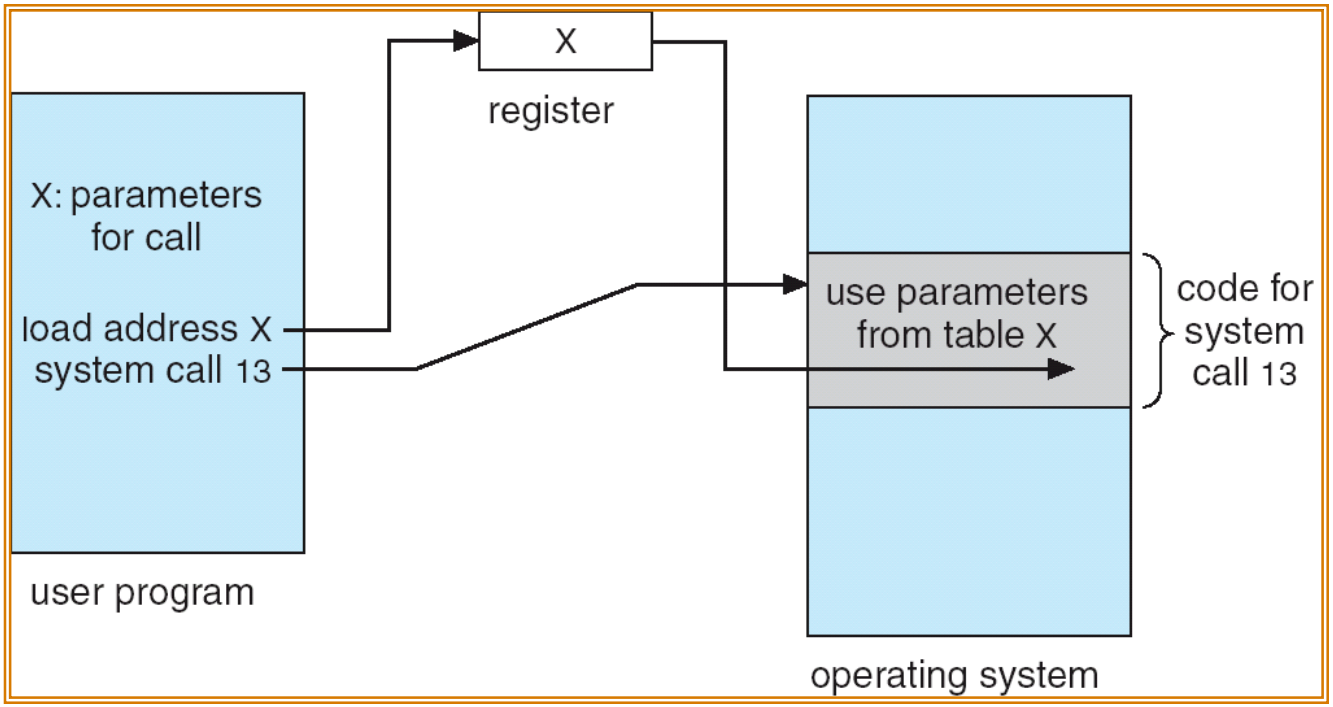
API – System Call – OS Relationship



System Call Parameter Passing

- More information is required than simply identity of desired system call
  - 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 hardware *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



Strace program to trace system calls

- Try a program called strace in Linux
- strace date
  - execve("/bin/date", ["date"], [/* 57 vars */]) = 0
  - uname({sys="Linux", node="sys.cse.nd.edu", ...}) = 0
  - brk(0) = 0x8621000
  - access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
  - open("/opt/intel_cc_80/lib/tls/i686/sse2/librt.so.1", O_RDONLY) = -1 ENOENT (No such file or directory)
- …

System 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, edit, rename, print, dump, list, and generally manipulate files and directories
  - Programming-language support - Compilers, assemblers, debuggers and interpreters sometimes provided
  - Program loading and execution- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
  - Communications - chat, web browsing, email, remote login, file transfers
  - Status information - system info such as date, time, amount of available memory, disk space, number of users

Operating System Generation

- 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
- *Booting* – starting a computer by loading the kernel
- *Bootstrap program* – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution

System Boot

- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loads bootstrap loader
  - When power initialized on system, execution starts at a fixed memory location
    - Firmware used to hold initial boot code

Wrapup

- System calls provide a mechanism for user programs to access OS services
  - System programs use system calls to provide functionality to users
- Other issues such as bootstrapping to initialize the OS

Operating System Design and Implementation

- Design and Implementation of OS affected by choice of hardware, type of system
- *User* goals and *System* goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and **maintain (portable?)**, as well as flexible, reliable, error-free, and efficient
- Important principle to separate

  Policy: What will be done?
  Mechanism: How to do it?
  - The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later