Chapter 2: Operating System Structures

Three views of an OS: each, respectively, focuses on

- the services it provides [**Ser**]
- programming interface it makes available to users and programmers [**Int**]
- its components and interconnections [**Com**]

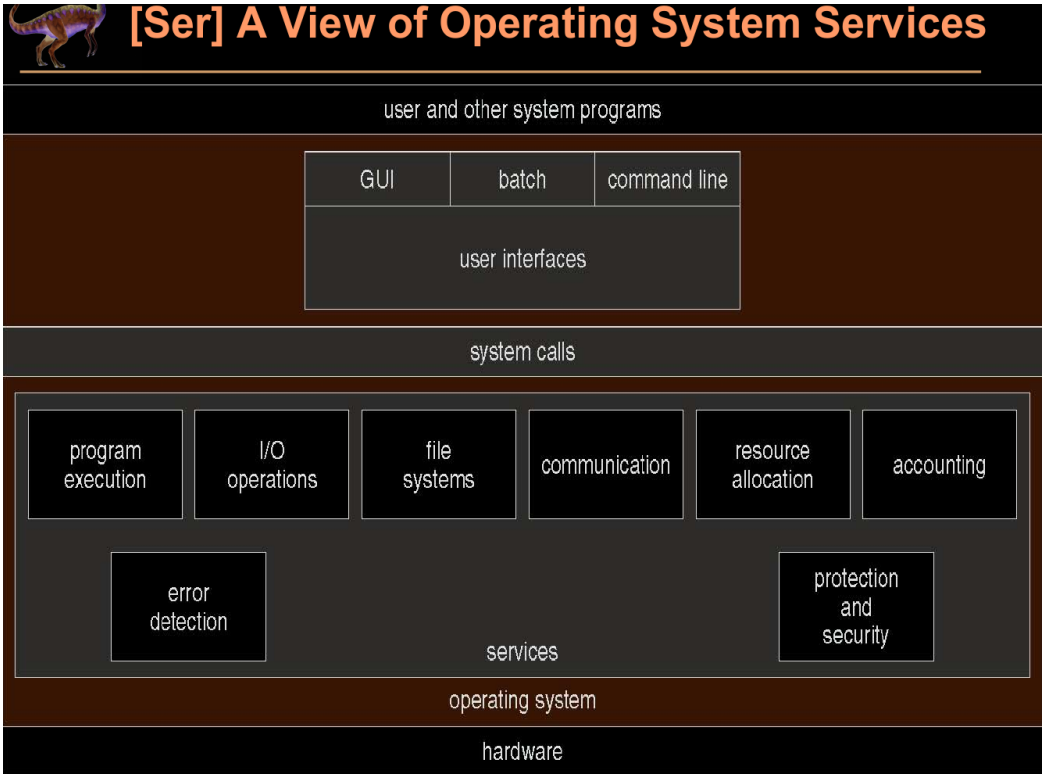
Operating System Services

OS services providing **functions** helpful to the **user**:

- **User interface**: CLI, GUI, Batch Interface
- **Program Execution**: system must be able to load a program into memory and to run that program, end execution, either normally or abnormally
- **I/O operations**: running a program may require I/O, which may involve a file or I/O device
- **File-system manipulation**: programs need to read/write files/directories, create/delete them, etc
- **Communications**: processes may exchange information, on the same computer or over a network
  - may be via shared memory or message passing
- **Error detection**: OS needs to be constantly aware of possible errors

OS **functions providing efficient operation** of the **system** via **resource sharing**

- **Resource allocation**: when multiple users/jobs running concurrently, resources must be allocated to each of them in an efficient and optimal manner
- **Accounting**: keep track of user resource usage
- **Protection and Security**: data control



System-Calls

- provide programming interface to services made available by the OS
- kernel functions
- accessed via high-level API rather than direct syscall use (win32, posix)
- just know how to use the api

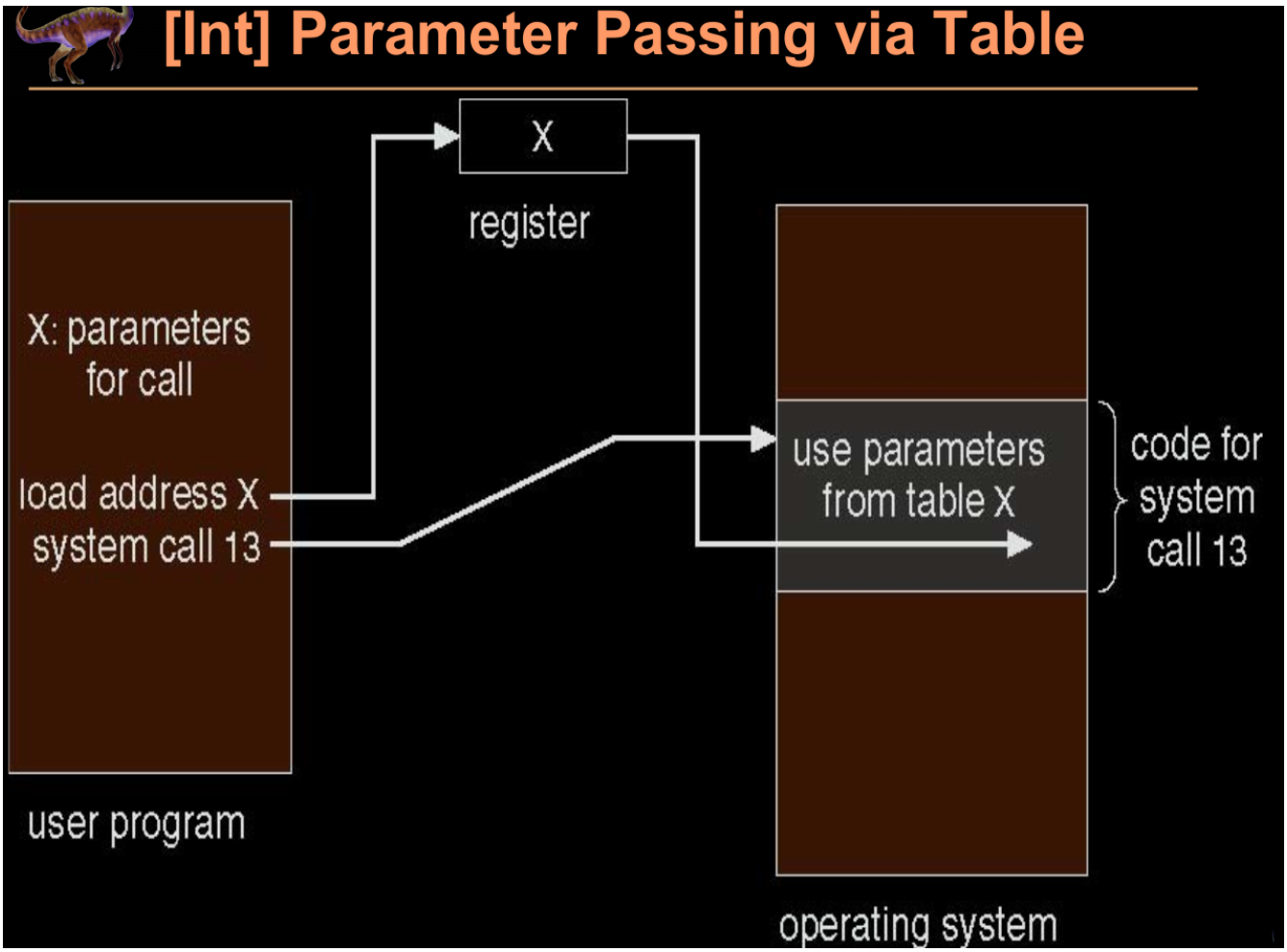
System-Call Implementation

Typically, a number is associated with each system-call. System-call interface maintains a table indexed according to these numbers

System-Call Parameter Passing

- simplest: pass parameters in registers: EAX, EDX, etc
- parameters stored in a block or table in memory, and *offset* address of block, table, or memory passed as a parameter in a register
- params placed, or *pushed*, onto **runtime stack** by the program, and popped off stack by OS when returning from sys-call

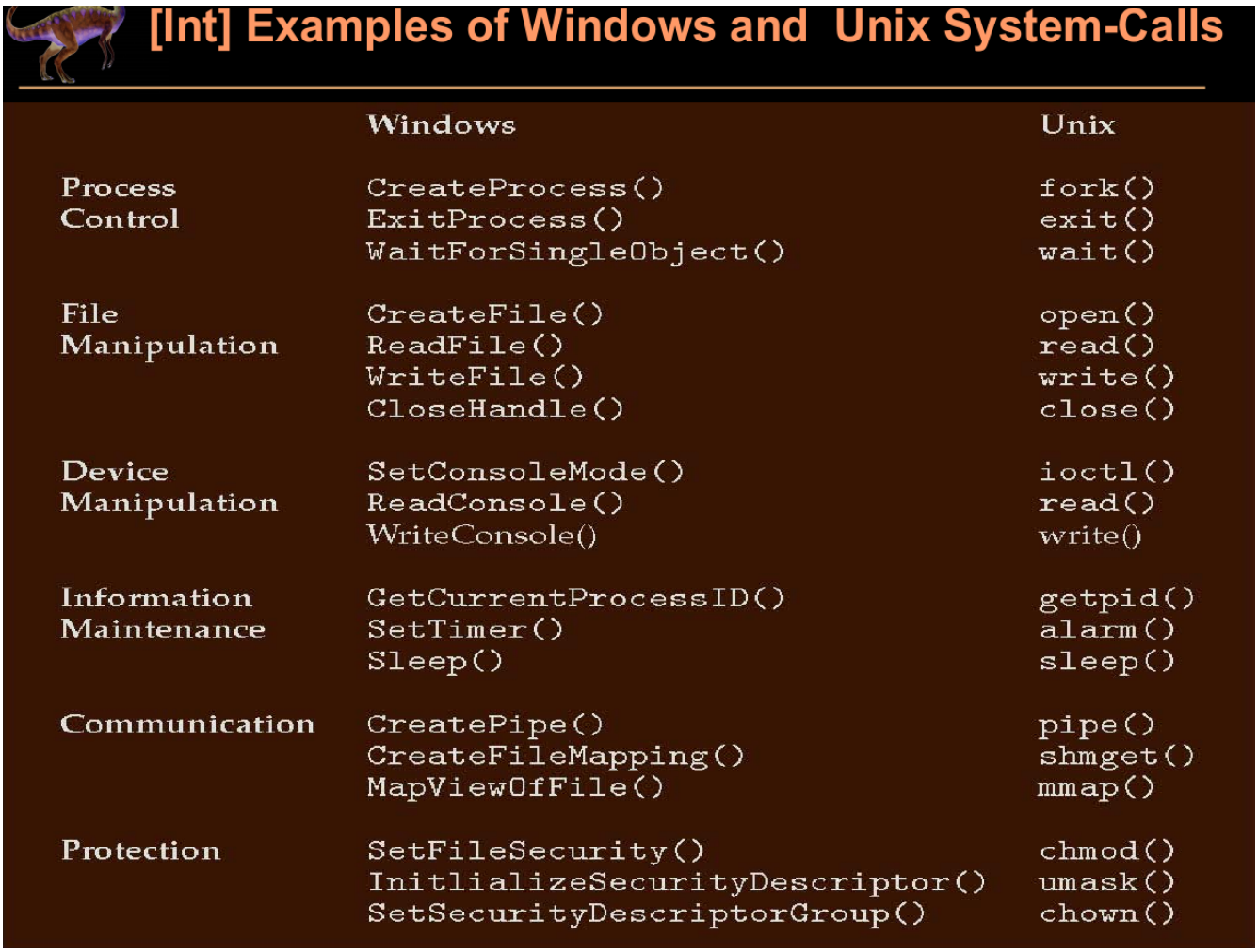
Block and stack methods are preferred. They do not limit the number of length of paramters being passed



Types of Sys-Calls

- file management
- device management

- information maintenance
  - all kinds of stats and data can be requested: # of users, free mem, OS version, disk space, etc
- communications
  - create, delete communication connection
  - *shared-memory* model create and gain access to memory regions
  - attacah, detach remote devices
  - transfer status info
- protection: machanism for controlling access to resources
- security



Protection & Security

- mechanism for controlling access to resources
- get and set permissions
- allow and deny user access

[Int] System Programs

Some programs (or system utilities) provide a convenient environment for program development and execution

- file manipulation, program language support, communication, etc

[Com] Operating System Structure

- OS design: partition into modules and define interconnections

- **Simple structure** - MS-DOS: monolithic, small kernel, not well separated modules, no protection, limited by 8088 hardware
  - not divided into modules
  - interfaces and levels of functionality not separated well
- **More complex** - original UNIX: monolithic, large kernel, two-layered UNIX (separates kernel and system programs), initially limited by hardware
  -
- **Layered** - an abstraction: modular OS, freedom to change/add modules
  - OS divided into a number of layers, each built on top of lower layers
  - with modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers
  - Pros: abstraction, simple to construct, easy to debug
  - Cons: defining various layers, less efficient than non-layered OS
- **Microkernel** - Mach: Modularized the expanded but large UNIX, keeps only essential component as system-level or user-level programs, smaller kernel, and easy to extend
  - moves as much from the kernel into user space
  - kernel provides: process and memory management and inter-process communication
  - comms take place between user modules using *message passing*. Function of microkernel: comm between client programs and services
  - Pros: easier to extend a microkernel, easier to port the OS to new hardware architectures, more reliable/secure (less code is running in kernel mode)
  - Cons: perf overhead of user space to kernel space communication

[Com] Modules

- many OSs implement *loadable kernel modules*
  - kernel provides core services, similar to microkernel
  - uses object-oriented approach
  - each core component is separate
  - each talks to the otherse over known interfaces
  - each is loadable as needed within the kernel

OS Design Goals

User Goals: OS should be conveient, easy to learn, reliable, safe, fast
System Goals: OS should be easy to design, implement, maintain, flexible, reliable, error-free, and efficient

OS System Generation

SYSGEN program obtains info concerning the specific configuration of the hardware system. It's used to build system-specific compiled kernel or system-tuned. In general it is more efficient code than one general kernel.

System Boot

1. When power initialized on system, execution starts at fixed mem location (firmware ROM used to hold initial boot code)
2. OS must be made available to hardware so hardware can start it
   - **Bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
   - **GRUB**, a boostrap loader, allows selection of kernel from multiple disks, versions, kernel options
3. Kernel loads and system is then **running**