Operating Systems and Networks

Lecture 03:

Introduction to OS-part 2

Behzad Bordbar

Recap

Lecture 1:

- ❑CPU: how it works, user and kernel mode, use of register, cache, …
- ❑System call trap and interrup
- ❑What happens when computer starts?
- ❑Different types of Memory and their speed

Will continue with preliminaries of OS

Demo lecture

Contents

- ❑Service view (provider of services) of the OS
- ❑Shell
- ❑Everything a directory
- ❑mkdir, mv, cp,…
- ❑Access control
- ❑Find, grep
- ❑|, >, >>, ; and their differences
- ❑wget,…

Contents

- How does mouse and keyboard work?
- Device controller
- CPU multitasking
- Time sharing
- a short study of system calls
  - API

How does mouse, keyboard …work?

- We said

Hardware may trigger an interrupt at any time by sending a signal to the CPU.

- But how?
- **Devices** interact via device controller connected through a common bus to CPU. Draw picture!
- small computer-systems interface (SCSI) controller.
- SCSI controller: hardware (card or chip) that allows SCSI storage device to communicate with the operating system using a SCSI bus

Device controller

- Maintains some local buffer storage and a set of special-purpose registers.
- Device controller moves the data between the peripheral devices that it controls and its local buffer storage.
- Operating systems have a device driver for each device controller.
- you download drivers! Manually or automtaically
- Device driver understands the device controller and provides the rest of the OS with a uniform interface to the device: copy the same no matter what device

Back to, How does I/O (mouse..) work?

- Device driver loads the appropriate registers within the device controller.
- Device controller examines the contents of the registers to determine what action to take (read char from k/b)
- controller starts the transfer of data from the device to its local buffer.
- when transfer of data is complete, the device controller informs the device driver via an interrupt that it has finished its operation.
- Device driver then returns control to the operating system, possibly returning the data or a pointer to the data if the operation was a read or status (success…)
- Exercise: draw a sequence diagram for yourself!

How does I/O (mouse..) work? (cont..)

- This form of interrupt-driven I/O is fine for moving **small amounts** of data
- Not suitable for bulk data movement such as disk I/O.

Instead: Direct Memory Access (DMA) is used that takes CPU out of the loop.

- After setting up buffers, pointers, and counters for the I/O device, the device controller transfers an entire block of data directly to or from its own buffer storage to memory, with no intervention by the CPU.
- Hence: only one interrupt is generated per block, to tell the device driver that the operation has completed, rather than the one interrupt per byte generated for low-speed devices.
- CPU is made free to do other things

Multitasking in CPU

- OS picks and begins to execute one of the jobs in memory.
- Eventually job may have to wait for some task, such as an I/O operation, to complete.
- OS switches to, and executes, another job.
- When that job needs to wait, the CPU switches to another job, and so on.
- Eventually, the first job finishes waiting and gets the CPU back.

Time-sharing:

- CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.

Time sharing

requires CPU scheduling of user tasks. But how?

- ❑Each user has at least one separate program in memory.
- ❑A program loaded into memory and executing is called a **process**. We will study this in details!
- ❑When a process executes, it typically executes for only a short time before it either finishes or needs to perform I/O.
- ❑I/O takes long long long time compare to execution! (look at the speed of access slides!)

Time sharing

- Time sharing: several jobs be kept simultaneously in memory.
- CPU scheduling: process of deciding which job is brought to memory to be executed, when there are not enough room.

Reasonable response time must be ensured:

1. processes are swapped in and out of main memory to the disk
2. use virtual memory: a technique that allows the execution of a process that is not completely in memory

- virtual-memory scheme enables users to run programs that are larger than actual **physical memory**. Further, it abstracts main memory into a large, uniform array of storage, separating **logical memory** as viewed by the user from physical memory. This arrangement frees programmers from concern over memory-storage limitations.

Dual mode

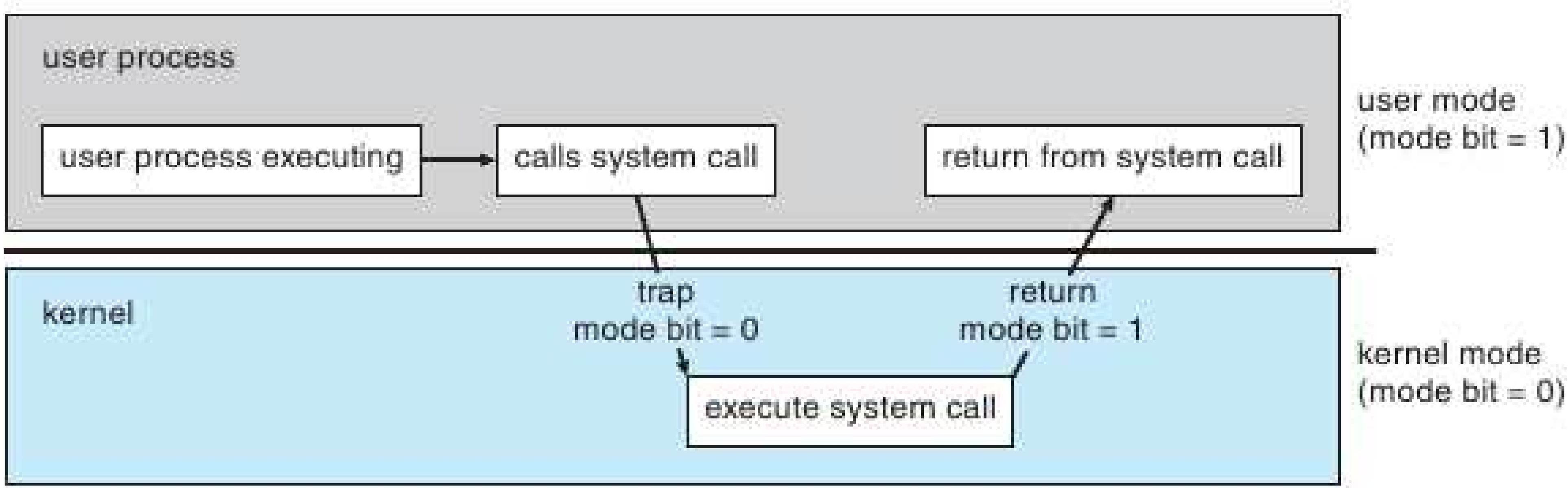

Figure from Dragon book.

- ❑Dual mode OS protect from harm caused by privileged instructions
- ❑Extended to multi mode by domains: Dom0, DomU

Why do we need hardware support?

- MS-DOS: Intel 8088 architecture, which has no mode bit
- user program can wipe out the whole OS
- programs are able to write to a device …

In dual mode:

- hardware detects errors that violate modes and handle them by Os
- stops user program attempts to execute an illegal instruction or to access memory of other users

When error detected

- OS must terminate the program
- OS gives error message
- produces memory dumps by writing to a file (users can check or OS vendors can check (Sun)).

system calls

provide an interface to the services made available by an operating system.

What language: are System-call written in?

- typically C and C++ and sometimes assembly-language involved
- explain the system calls for reading data from one file and writing to another file:

$cp file1 file2

open file1, possible error(print, abort), create file2 (file2 exists, rewrite/rename…), start read and write (errors:disk space, memory stick unplugged…), all read and written, close files, ack

Do I access system call directly?

system calls: API to wrap system calls

Application Programming Interface (API)

- specifies a set of functions that are available to an application programmer, including the parameters that are passed to each function and the return values the programmer can expect.
- programmer accesses an API via a library of code provided by the operating system.

Example of APIs:

1. Windows API for Windows systems

Example: CreateProcess() which invokes the NTCreateProcess() system call in the Windows kernel return value 0 or 1 (error)

system calls: API (continue)

Example of APIs:

2. POSIX API for POSIX-based systems (UNIX, Linux, and Mac OS X)

- programmer accesses an API via a library of code provided by the operating system.

Example: read

input:

- int fd: file descriptor to be read
- void *buf: pointer into buffer to be read into
- size_t count: maximum number of bytes to read

output:

- number of bytes read (if success)
- -1 if fail
- UNIX and Linux for programs written in the C language, the library is called libc.

system calls: API (continue)

Example of APIs:

3. Java API for programs that run on the Java virtual machine.

getParentFile()

invoked on a file object.

output:

Returns the abstract pathname of this abstract pathname's parent, or null if this pathname does not name a parent directory.

JVM uses the OS system calls.

why do we use API?

Why not invoking actual system calls directly?

- ❑Program portability: program can compile and run on any system that supports the API
- ❑system calls can often be more detailed and difficult to work with
- ❑give access to high level objects (java API)

Do you know interfaces in java? using system calls is like implementing an (or many) interfaces

What happens when a user prog. makes a system call

- caller only needs to know the signature!
- method call and parameters are passed into a registers
- values saved in memory for example on table or stacks but addresses in registers
- Stack is preferred because do not put limit on the number of parameters stored.

summary

- Study of I/O devices
- Device Controller
- How OS manages multiple tasks
- System call and their use in user programs
- API and examples of API
- Similarity between API and interface