

CSE 421/521 - Operating Systems
Fall 2011

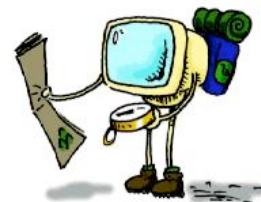
LECTURE - VI
CPU SCHEDULING - II

Tevfik Koşar

University at Buffalo
September 15th, 2011

Roadmap

- Multilevel Feedback Queues
- Estimating CPU bursts
- System Calls
- Virtual Machines

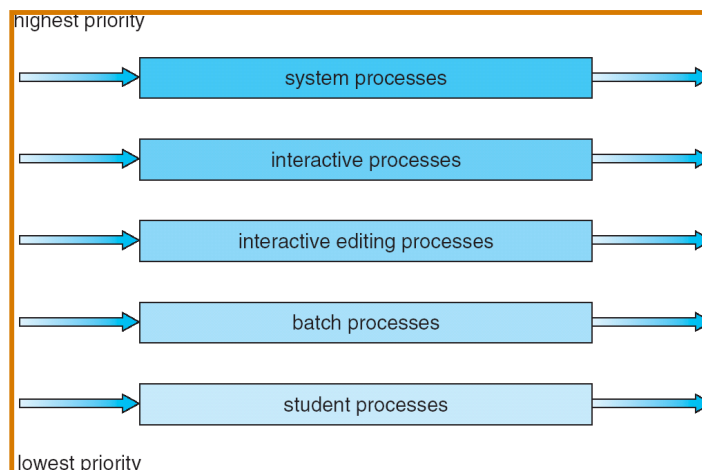


Multilevel Queue

- Ready queue is partitioned into separate queues:
foreground (interactive)
background (batch)
- Each queue has its own scheduling algorithm
 - foreground - RR
 - background - FCFS
- Scheduling must be done between the queues
 - **Fixed priority scheduling**; (i.e., serve all from foreground then from background). Possibility of starvation.
 - **Time slice** - each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR, 20% to background in FCFS

3

Multilevel Queue Scheduling



4

Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
 - number of queues
 - scheduling algorithms for each queue
 - method used to determine when to upgrade a process
 - method used to determine when to demote a process
 - method used to determine which queue a process will enter when that process needs service

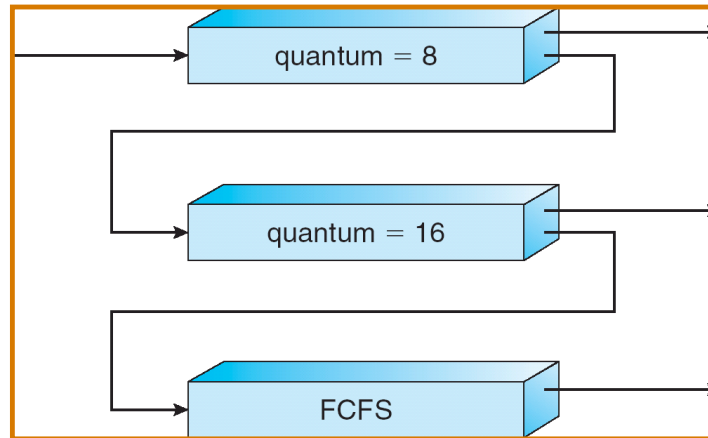
5

Example of Multilevel Feedback Queue

- Three queues:
 - Q_0 - RR with time quantum 8 milliseconds
 - Q_1 - RR time quantum 16 milliseconds
 - Q_2 - FCFS
- Scheduling
 - A new job enters queue Q_0 which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue Q_1 .
 - At Q_1 job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2 .

6

Multilevel Feedback Queues



7

Determining Length of Next CPU Burst

- Can only estimate the length
- Can be done by using the length of previous CPU bursts, using exponential averaging

$$\tau_{n+1} = \alpha t_n + (1 - \alpha) \tau_n.$$

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define :

8

Examples of Exponential Averaging

- $\alpha = 0$
 - $\tau_{n+1} = \tau_n$
 - Recent history does not count
- $\alpha = 1$
 - $\tau_{n+1} = \alpha t_n$
 - Only the actual last CPU burst counts
- If we expand the formula, we get:
$$\tau_{n+1} = \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \dots$$
$$+ (1 - \alpha)^j \alpha t_{n-j} + \dots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$
- Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor

9

Exercise

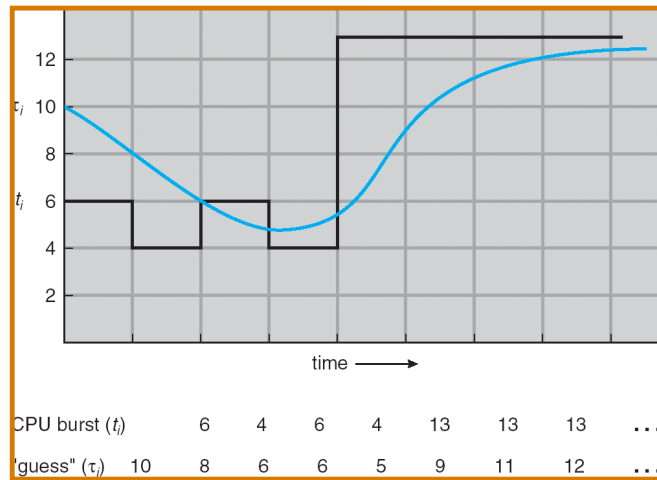
Consider the exponential average formula used to predict the length of the next CPU burst. What are the implications of assigning the following values to the parameters used by the algorithm?

- $\alpha = 0$ and $\tau_0 = 100\text{milliseconds}$
- $\alpha = 0.99$ and $\tau_0 = 10\text{milliseconds}$

Answer: When $\alpha = 0$ and $\tau_0 = 100\text{milliseconds}$, the formula always makes a prediction of 100 milliseconds for the next CPU burst. When $\alpha = 0.99$ and $\tau_0 = 10\text{milliseconds}$, the most recent behavior of the process is given much higher weight than the past history associated with the process. Consequently, the scheduling algorithm is almost memory-less, and simply predicts the length of the previous burst for the next quantum of CPU execution.

10

Prediction of the Length of the Next CPU Burst



Alpha = 1/2, T0 = 10

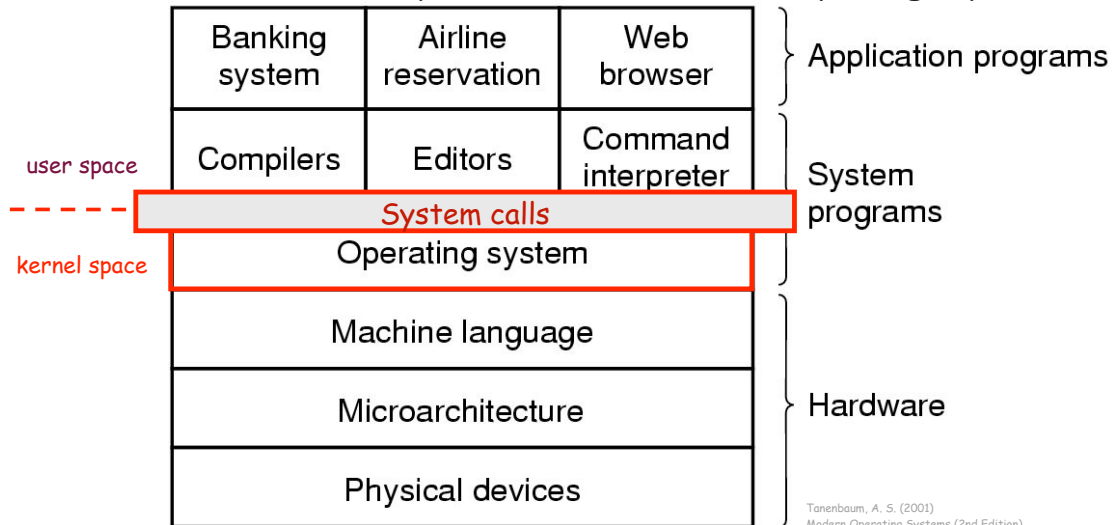
11

OS API: SYSTEM CALLS

12

System Calls

➤ Location of the system calls in the Computing System



The system calls are the mandatory interface between the user programs and the O/S

13

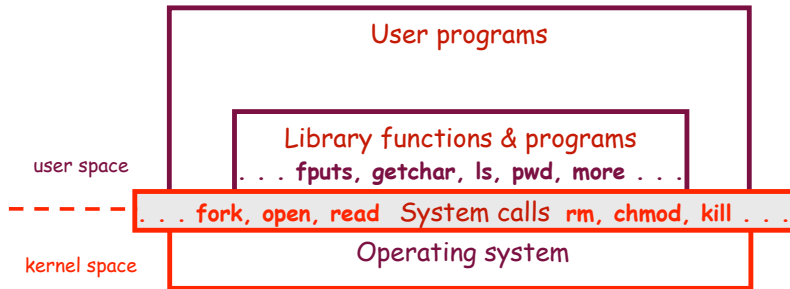
System Calls

- 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)

14

System Calls

- All programs needing resources must use system calls

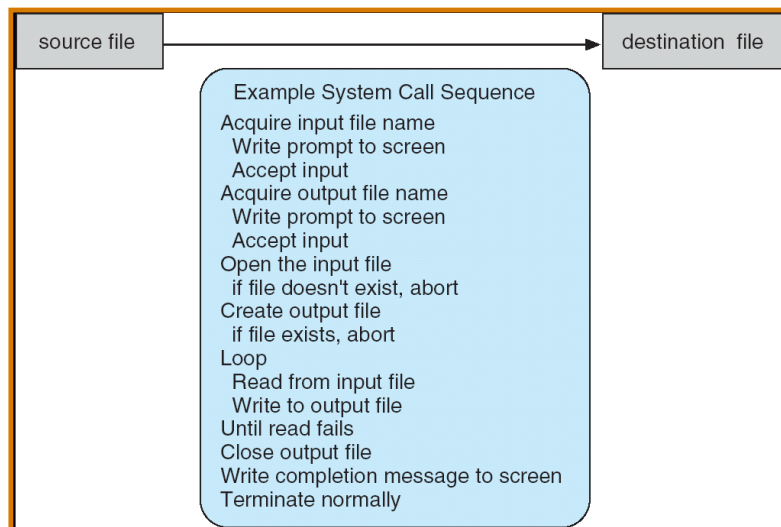


the "middleman's counter"

- ✓ system calls are the only **entry points** into the kernel and system
- ✓ most UNIX commands are actually library functions and utility programs (e.g., shell interpreter) built on top of the system calls
- ✓ however, the distinction between library functions and system calls is not critical to the programmer, only to the O/S designer

Example of System Calls

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



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)

17

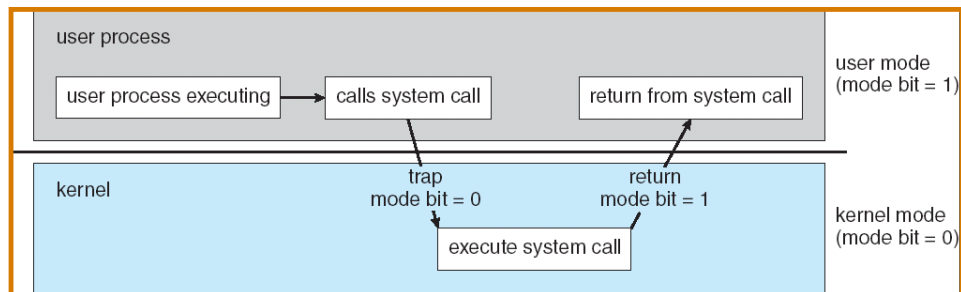
Dual-Mode Operation

- **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

18

Transition from User to Kernel Mode

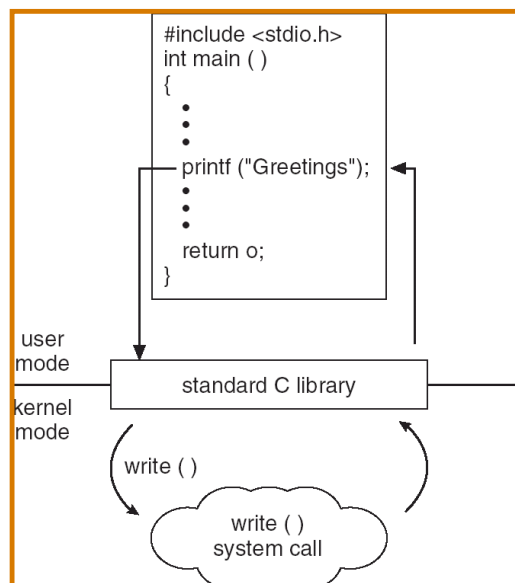
- How to prevent user program getting stuck in an infinite loop / process hogging resources
 - **Timer:** Set interrupt after specific period (1ms to 1sec)
 - Operating system decrements counter
 - When counter zero generate an interrupt
 - Set up before scheduling process to regain control or terminate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



19

Standard C Library Example

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



20

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
```

21

VIRTUAL MACHINES

22

Virtual Machines

- A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware
- A virtual machine provides an interface *identical* to the underlying bare hardware
- The virtual machine creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory

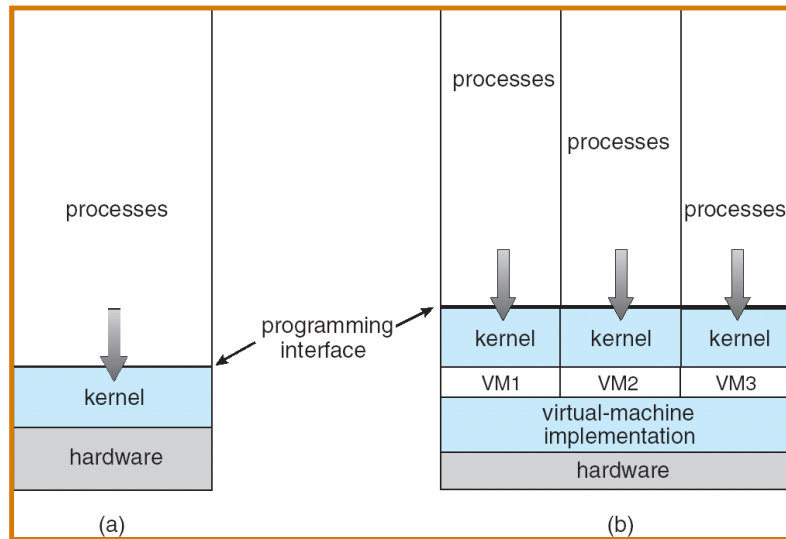
23

Virtual Machines (Cont.)

- The resources of the physical computer are shared to create the virtual machines
 - CPU scheduling can create the appearance that users have their own processor
 - Spooling and a file system can provide virtual card readers and virtual line printers
 - A normal user time-sharing terminal serves as the virtual machine operator's console

24

Virtual Machines (Cont.)



(a) Nonvirtual machine

(b) Virtual machine

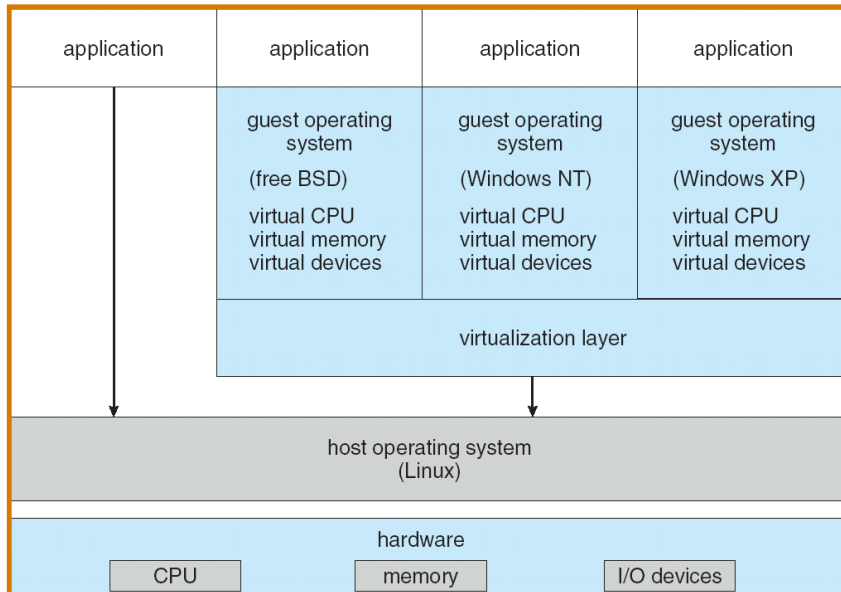
25

Virtual Machines (Cont.)

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- The virtual machine concept is difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine

26

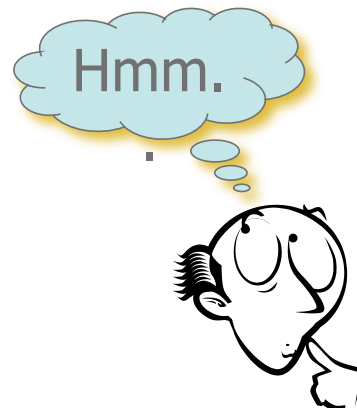
VMware Architecture



27

Summary

- Multilevel Feedback Queues
- Estimating CPU bursts
- System Calls
- Virtual Machines



- Next Lecture: Project-1 Discussion

28

Acknowledgements

- “Operating Systems Concepts” book and supplementary material by A. Silberschatz, P. Galvin and G. Gagne
- “Operating Systems: Internals and Design Principles” book and supplementary material by W. Stallings
- “Modern Operating Systems” book and supplementary material by A. Tanenbaum
- R. Doursat and M. Yuksel from UNR