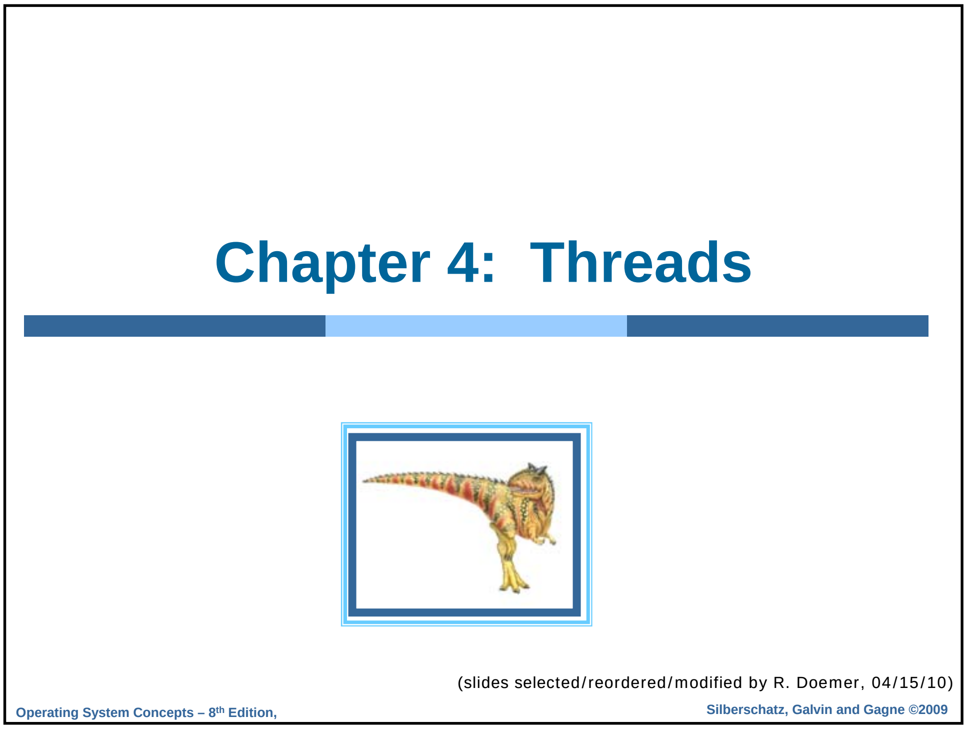
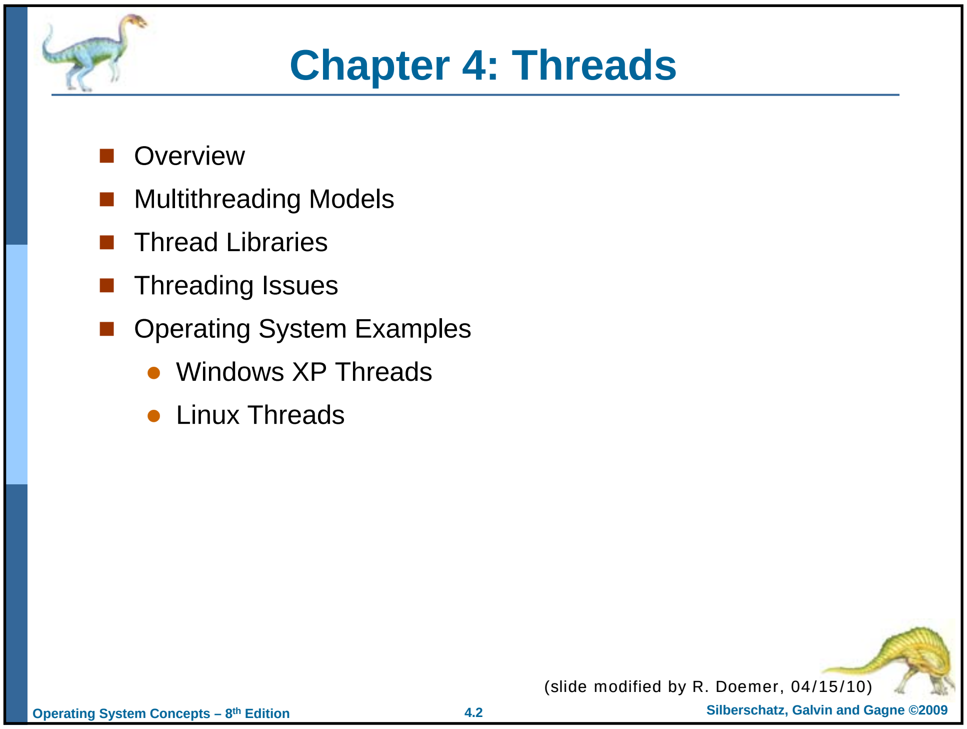


Chapter 4: Threads

(slides selected/reordered/modified by R. Doemer, 04/15/10)



Chapter 4: Threads

- Overview
- Multithreading Models
- Thread Libraries
- Threading Issues
- Operating System Examples
  - Windows XP Threads
  - Linux Threads

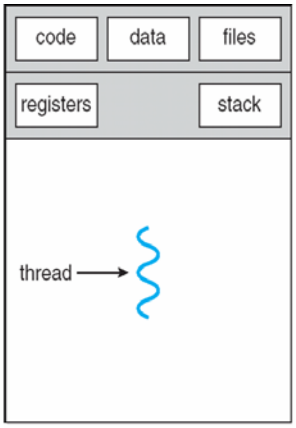
(slide modified by R. Doemer, 04/15/10)

Objectives

- To introduce the notion of a thread —
  a fundamental unit of CPU utilization
  that forms the basis of multithreaded computer systems
- To discuss the APIs for the Pthread thread library
  (for EECS111, we will skip Win32 and Java thread APIs)
- To examine issues related to multithreaded programming

(slide modified by R. Doemer, 04/15/10)

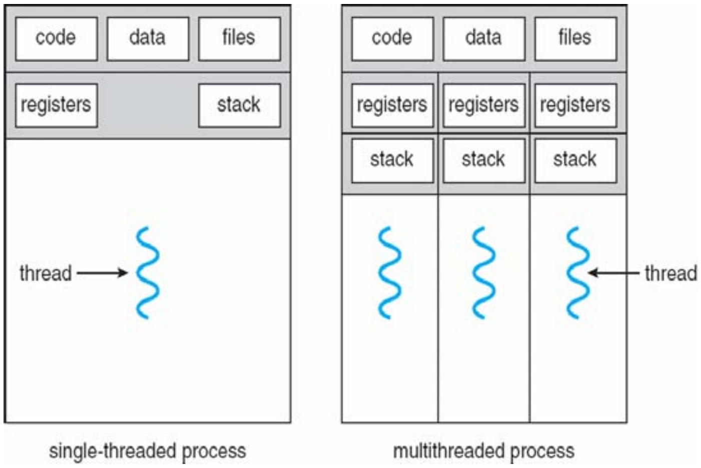
Single and Multithreaded Processes



single-threaded process

(slide modified by R. Doemer, 04/15/10)

Single and Multithreaded Processes



(slide modified by R. Doemer, 04/15/10)

Benefits of Multi-Threading

- Responsiveness
  - Application can still continue to “run”
    while some of its threads are “busy”
    (e.g. blocked in system-calls for I/O)
- Resource Sharing
  - Threads share most of the resources of their process
- Economy
  - Threads are “cheaper” to manage than processes
- Scalability
  - Threads can utilize available multi-core hardware
    (see next slide)

(slide modified by R. Doemer, 04/15/10)

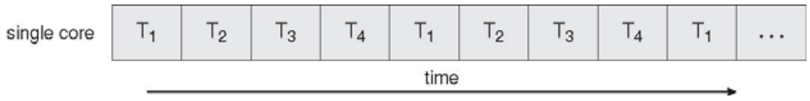
Multi-Core Programming

- Multi-core systems offer scalability, but at the same time, are putting pressure on programmers
- Challenges include
  - Dividing activities
  - Balancing
  - Data splitting
  - Data dependency
  - Testing and debugging
- We may need an entirely new approach to design *parallel* software!

(slide modified by R. Doemer, 04/15/10)

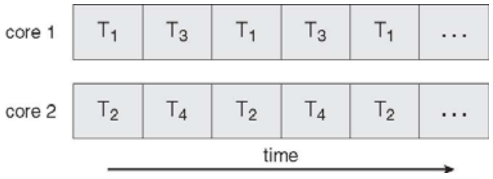
Multi-Core Programming

- Concurrent Execution on a Single-core System



single core	T_1	T_2	T_3	T_4	T_1	T_2	T_3	T_4	T_1	. . .

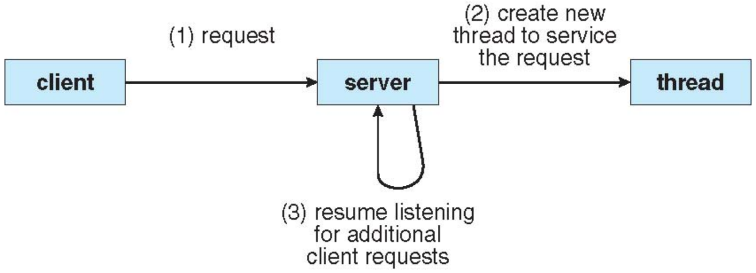
- Parallel Execution on a Multi-core System



core 1	T_1	T_3	T_1	T_3	T_1	. . .
core 2	T_2	T_4	T_2	T_4	T_2	. . .

(slide modified by R. Doemer, 04/15/10)

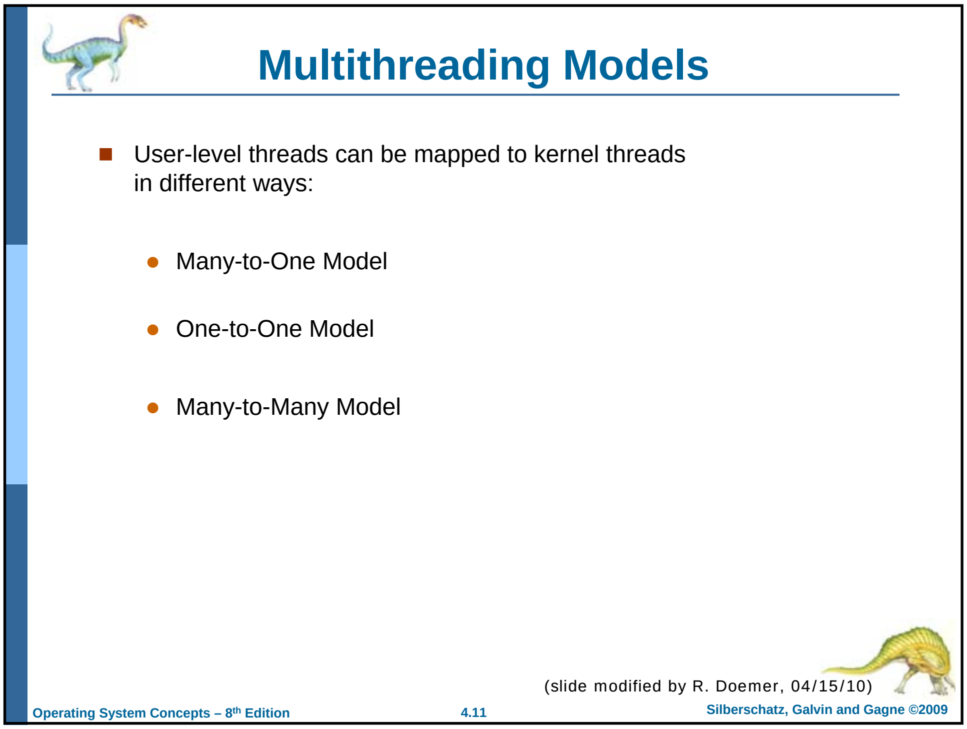
Multithreaded Server Architecture


(1) request
(2) create new thread to service the request
client
server
thread
(3) resume listening for additional client requests

Multithreading Models

- User Threads
  - Thread management done by user-level threads library
  - OS kernel is un-aware of user-level threads
- Kernel Threads
  - Supported by the Kernel
  - Examples
    - Windows XP/2000
    - Solaris
    - Linux
    - Tru64 UNIX
    - Mac OS X

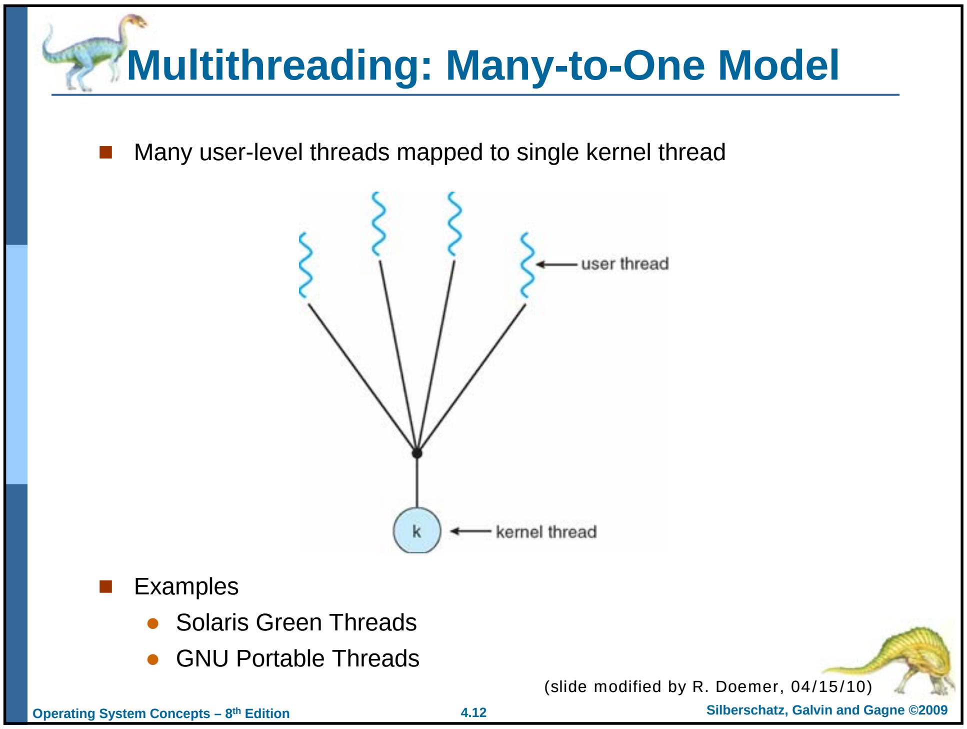
(slide modified by R. Doemer, 04/15/10)



Multithreading Models

- User-level threads can be mapped to kernel threads in different ways:
  - Many-to-One Model
  - One-to-One Model
  - Many-to-Many Model

(slide modified by R. Doemer, 04/15/10)



Multithreading: Many-to-One Model

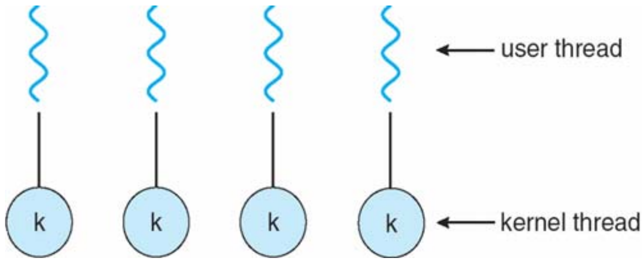
- Many user-level threads mapped to single kernel thread

- Examples
  - Solaris Green Threads
  - GNU Portable Threads

(slide modified by R. Doemer, 04/15/10)

Multithreading: One-to-One Model

- Each user-level thread maps to a kernel thread

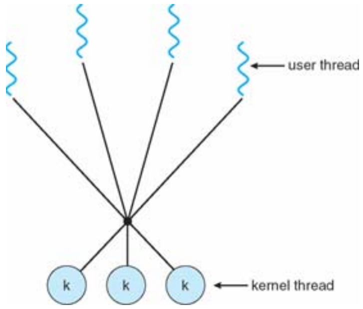


- Examples
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 and later

(slide modified by R. Doemer, 04/15/10)

Multithreading: Many-to-Many Model

- Many user level threads mapped to many kernel threads
  - Allows the OS to create a “sufficient” number of kernel threads

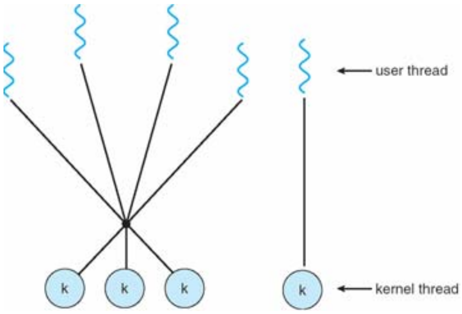


- Examples
  - Solaris prior to version 9
  - Windows NT/2000 with the ThreadFiber package

(slide modified by R. Doemer, 04/15/10)

Multithreading: Two-level Model

- Similar to Many-to-Many Model,
  except that it allows a user thread to be **bound** to kernel thread



- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier

(slide modified by R. Doemer, 04/15/10)

Thread Libraries

- Thread library provides programmer with API
  for creating and managing threads

- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

- Examples of primary thread libraries
  - POSIX Pthreads
  - Win32 threads
  - Java threads

(slide modified by R. Doemer, 04/15/10)

Pthreads

- May be provided either as user-level or kernel-level threads
- A POSIX standard (IEEE 1003.1c) API
  for thread creation and synchronization
- API specifies behavior of the thread library,
  implementation is up to development of the library
- Common in UNIX operating systems
  - Solaris
  - Linux
  - Mac OS X

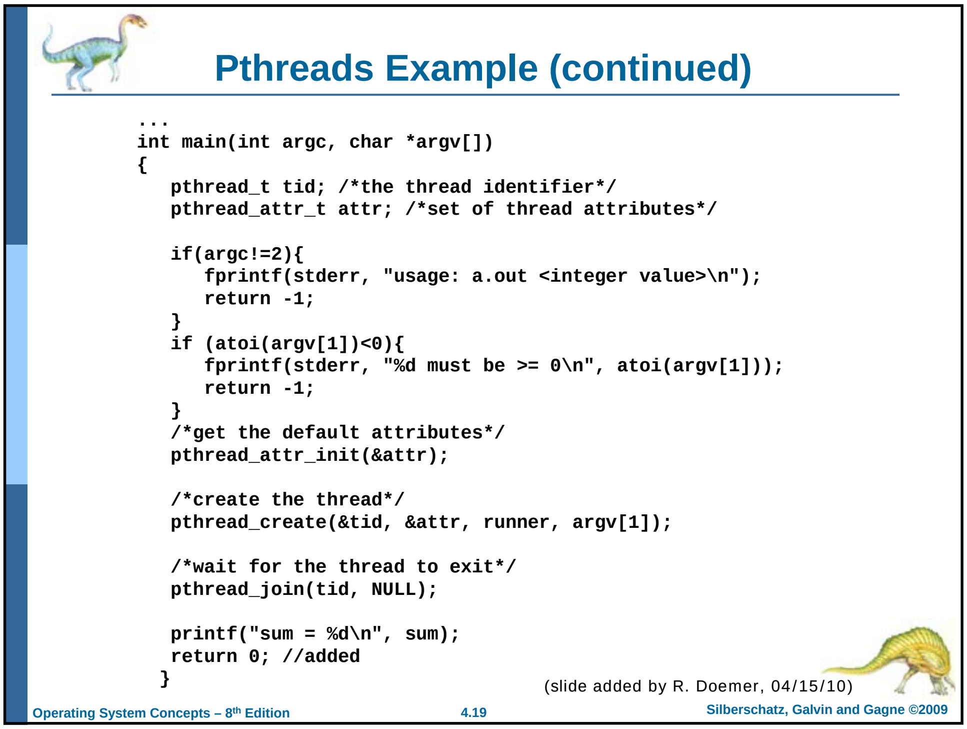
(slide modified by R. Doemer, 04/15/10)

Pthreads Example

- Textbook Figure 4.9 (page 161)

```
#include<pthread.h>
#include<stdio.h>
#include <stdlib.h>  //added

int sum; /* this data is shared by the threads */

/* the thread will begin control in this function */

void *runner(void *param)
{
   int i, upper = atoi(param);
   sum = 0;

   for (i=1; i<=upper; i++)
      sum += i;

   pthread_exit(0);
   return 0;
}

...
```

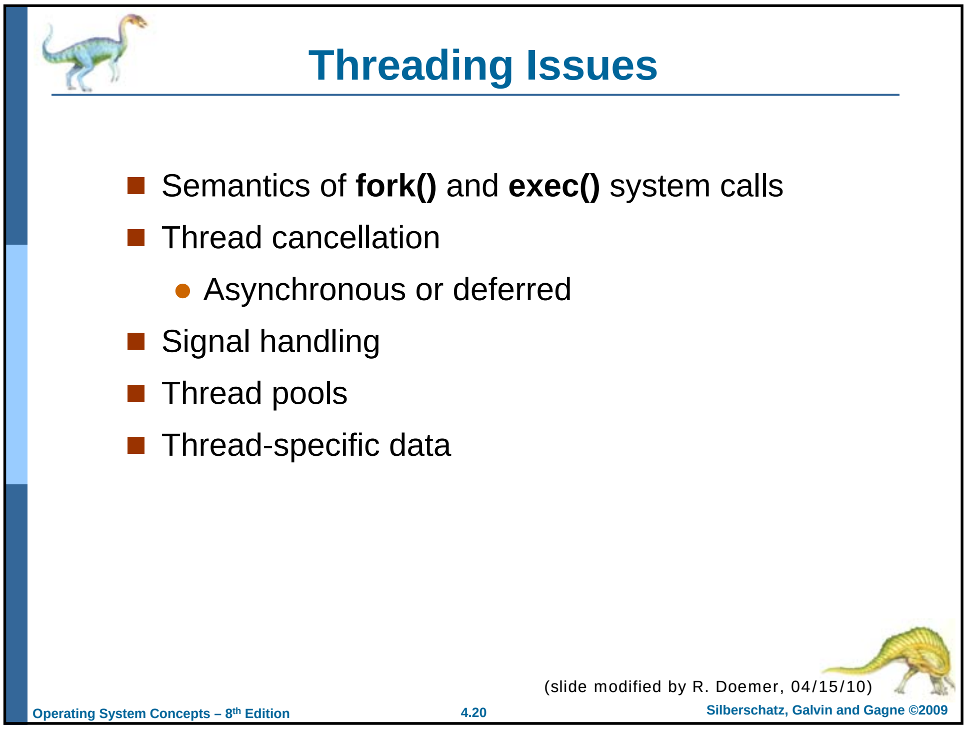
(slide added by R. Doemer, 04/15/10)



Pthreads Example (continued)

```
...
int main(int argc, char *argv[])
{
   pthread_t tid; /*the thread identifier*/
   pthread_attr_t attr; /*set of thread attributes*/

   if(argc!=2){
      fprintf(stderr, "usage: a.out <integer value>\n");
      return -1;
   }
   if (atoi(argv[1])<0){
      fprintf(stderr, "%d must be >= 0\n", atoi(argv[1]));
      return -1;
   }
   /*get the default attributes*/
   pthread_attr_init(&attr);

   /*create the thread*/
   pthread_create(&tid, &attr, runner, argv[1]);

   /*wait for the thread to exit*/
   pthread_join(tid, NULL);

   printf("sum = %d\n", sum);
   return 0; //added
  }
```

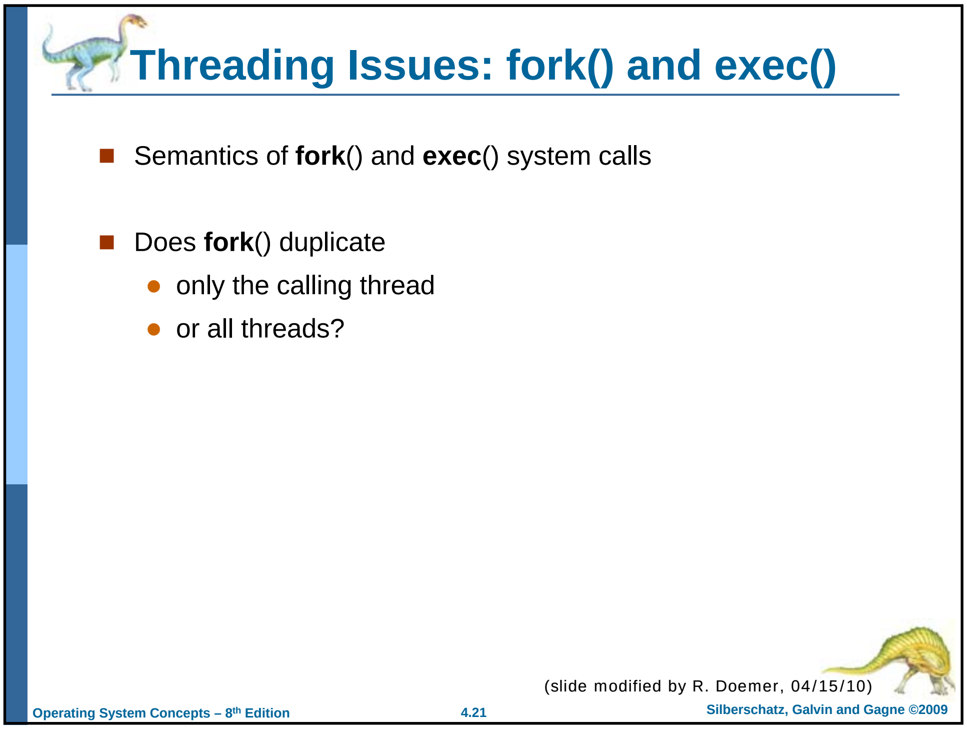
(slide added by R. Doemer, 04/15/10)



Threading Issues

- Semantics of **fork()** and **exec()** system calls
- Thread cancellation
  - Asynchronous or deferred
- Signal handling
- Thread pools
- Thread-specific data

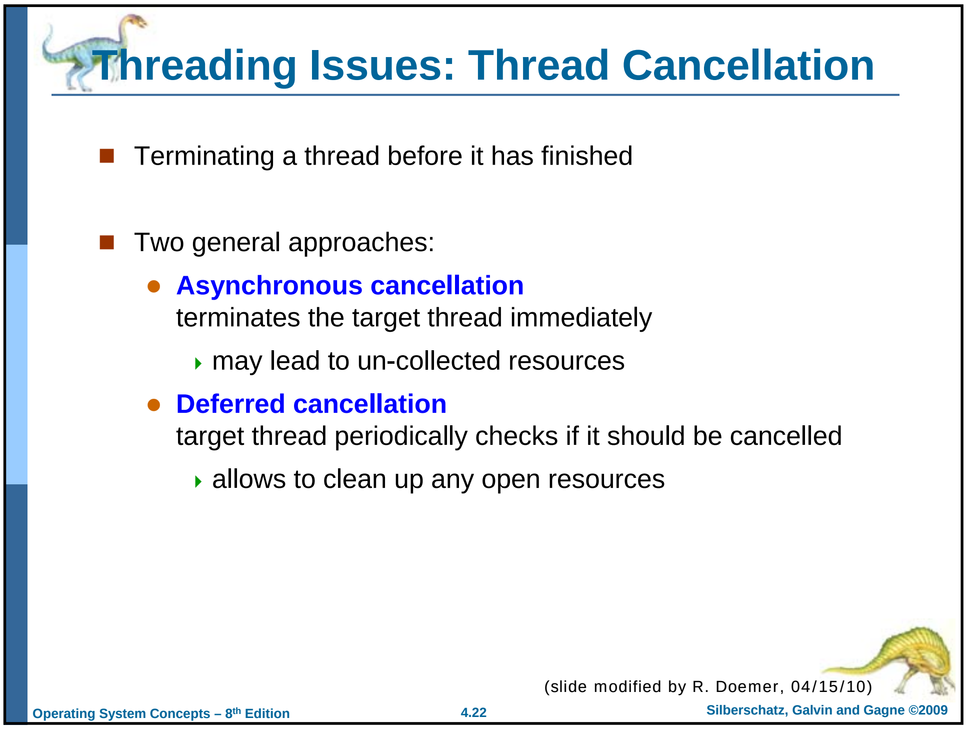
(slide modified by R. Doemer, 04/15/10)



Threading Issues: fork() and exec()

- Semantics of **fork**() and **exec**() system calls
- Does **fork**() duplicate
  - only the calling thread
  - or all threads?

(slide modified by R. Doemer, 04/15/10)



Threading Issues: Thread Cancellation

- Terminating a thread before it has finished
- Two general approaches:
  - **Asynchronous cancellation**
    terminates the target thread immediately
    - may lead to un-collected resources
  - **Deferred cancellation**
    target thread periodically checks if it should be cancelled
    - allows to clean up any open resources

(slide modified by R. Doemer, 04/15/10)

Threading Issues: Signal Handling

- **Signals** are used in UNIX systems to notify a process that a particular event has occurred
- A **signal handler** is used to process signals
  1. Signal is generated by particular event
  2. Signal is delivered to a process
  3. Signal is handled
- Options:
  - Deliver the signal to the thread to which the signal applies
  - Deliver the signal to every thread in the process
  - Deliver the signal to certain threads in the process
  - Assign a specific thread to receive all signals for the process

(slide modified by R. Doemer, 04/15/10)

Threading Issues: Thread Pools

- Create a number of threads in a pool where they await work
- Advantages:
  - Usually slightly faster to service a request with an existing thread than create a new thread
  - Allows the number of threads in the application to be bound to the size of the pool

(slide modified by R. Doemer, 04/15/10)

Threading Issues: Thread-Specific Data

- Allows each thread to have its own copy of data
  - Remember, all variables are shared in the process!
- Useful when a thread processes unique data
  - Example: transaction-processing system

(slide modified by R. Doemer, 04/15/10)

Operating System Examples

- Linux Threads
- Windows XP Threads

(slide modified by R. Doemer, 04/15/10)

Linux Threads

- Linux refers to them as *tasks* rather than *threads*
- Thread creation is done through **clone()** system call
- **clone()** allows a child task
  to share the address space of the parent task (process)

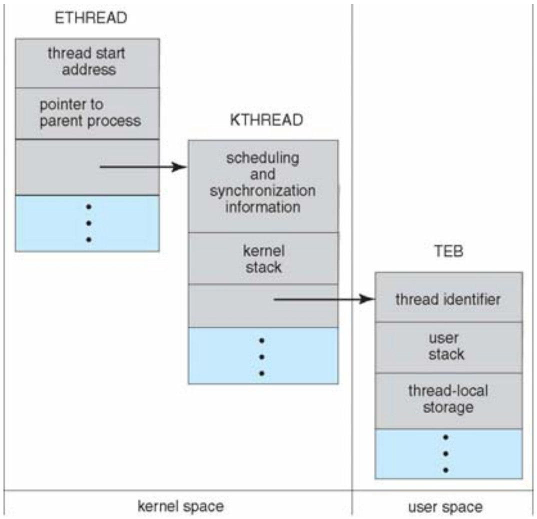
flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

(slide modified by R. Doemer, 04/15/10)

Windows XP Threads

- Implements the one-to-one mapping, kernel-level
- Each thread contains
  - A thread id
  - Register set
  - Separate user and kernel stacks
  - Private data storage area
- The register set, stacks, and private storage area are known as the context of the threads
- The primary data structures of a thread include:
  - ETHREAD (executive thread block)
  - KTHREAD (kernel thread block)
  - TEB (thread environment block)

Windows XP Threads



End of Chapter 4