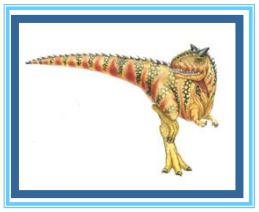
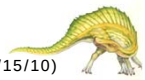
Chapter 4: Threads



(slides selected/reordered/modified by R. Doemer, 01/11/11)

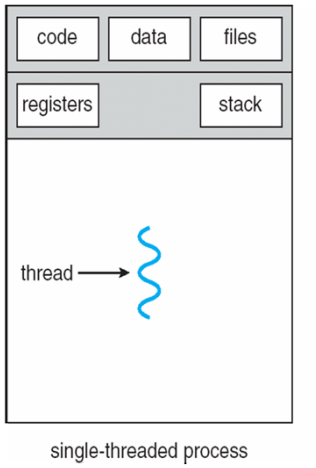
Chapter 4: Threads

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

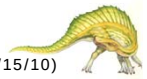


(slide modified by R. Doemer, 04/15/10)

Single and Multithreaded Processes

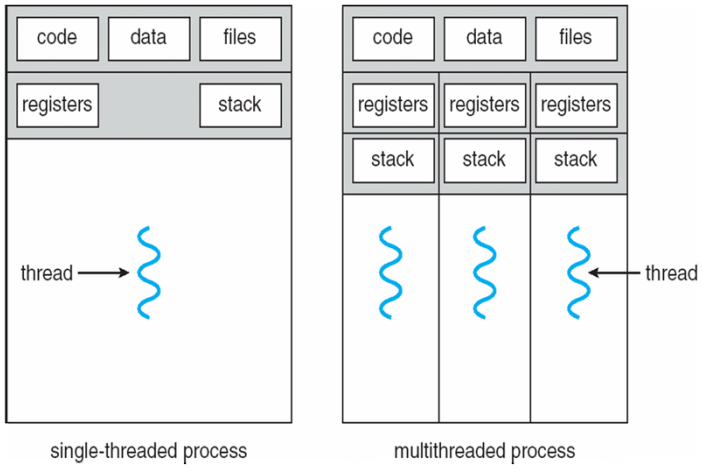


single-threaded process

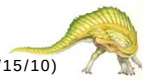


(slide modified by R. Doemer, 04/15/10)

Single and Multithreaded Processes



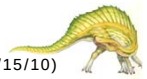
single-threaded process multithreaded process



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

Benefits of Multi-Threading

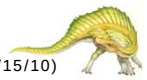
- Responsiveness
  - Application can still continue to “run” while some of its threads are “busy” (e.g. blocked in system-calls for I/O)
- Resource Sharing
  - Threads share most of the resources of their process
- Economy
  - Threads are “cheaper” to manage than processes
- Scalability
  - Threads can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



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

Multi-Core Programming

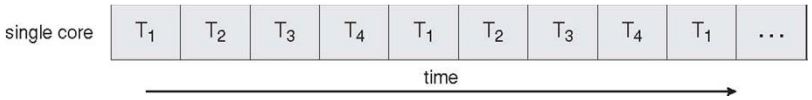
- Multi-core systems offer scalability, but at the same time, are putting pressure on programmers
- Challenges include
  - Dividing activities
  - Balancing
  - Data splitting
  - Data dependency
  - Testing and debugging
- We may need an entirely new approach to design *parallel* software!



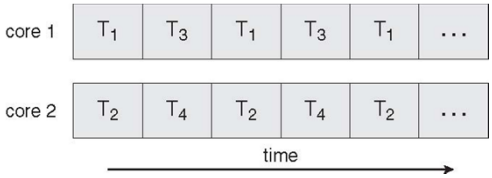
(slide modified by R. Doemer, 04/15/10)

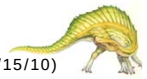
Multi-Core Programming

- Concurrent Execution on a Single-core System



- Parallel Execution on a Multi-core System

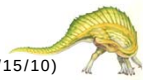




(slide modified by R. Doemer, 04/15/10)

Multithreading Models

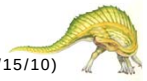
- User Threads
  - Thread management done by user-level threads library
  - OS kernel is un-aware of user-level threads
- Kernel Threads
  - Supported by the Kernel
  - Examples
    - Windows XP/2000
    - Solaris
    - Linux
    - Tru64 UNIX
    - Mac OS X



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

Multithreading Models

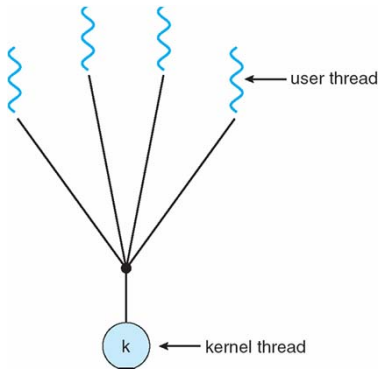
- User-level threads can be mapped to kernel threads in different ways:
  - Many-to-One Model
  - One-to-One Model
  - Many-to-Many Model



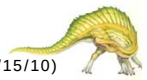
(slide modified by R. Doemer, 04/15/10)

Multithreading: Many-to-One Model

- Many user-level threads mapped to single kernel thread



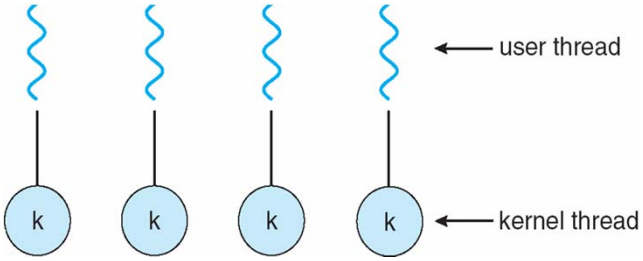
- Examples
  - Solaris Green Threads
  - GNU Portable Threads



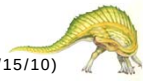
(slide modified by R. Doemer, 04/15/10)

Multithreading: One-to-One Model

- Each user-level thread maps to a kernel thread



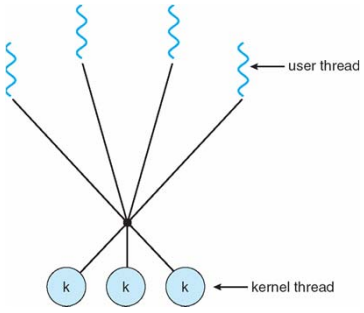
- Examples
  - Windows NT/XP/2000
  - Linux
  - Solaris 9 and later



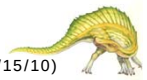
(slide modified by R. Doemer, 04/15/10)

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



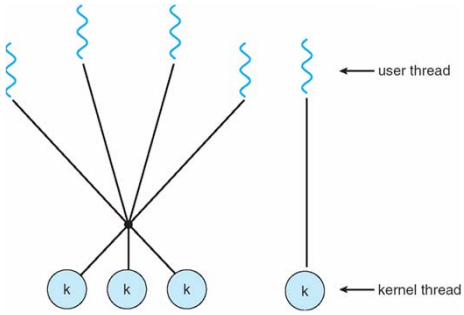
- Examples
  - Solaris prior to version 9
  - Windows NT/2000 with the ThreadFiber package



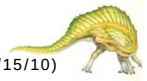
(slide modified by R. Doemer, 04/15/10)

Multithreading: Two-level Model

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



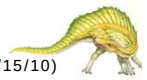
- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier



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

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)

End of Chapter 4