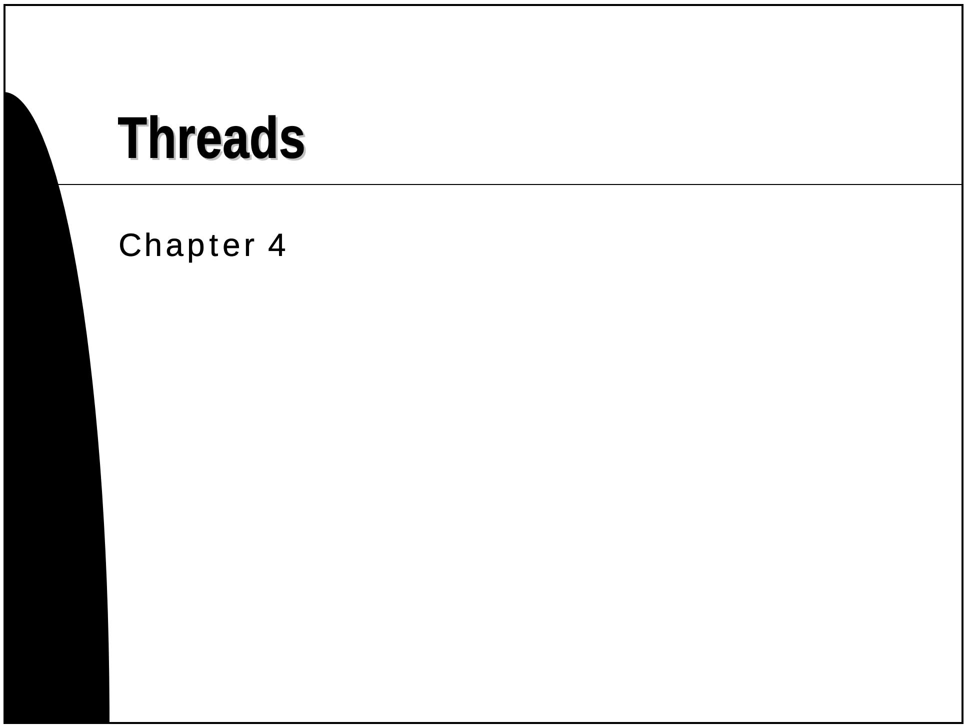
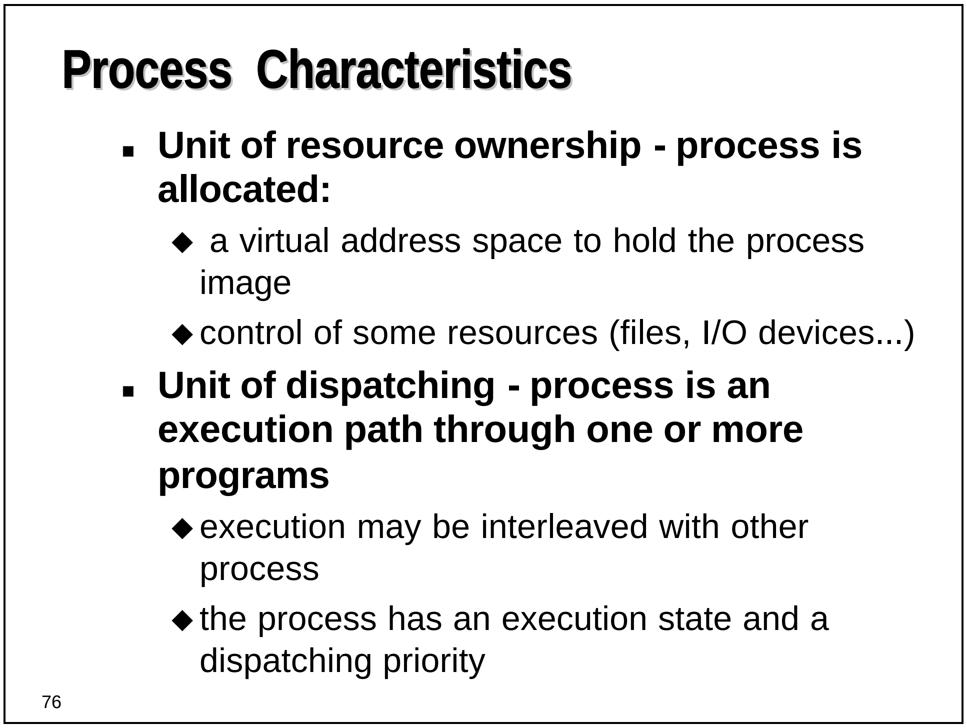


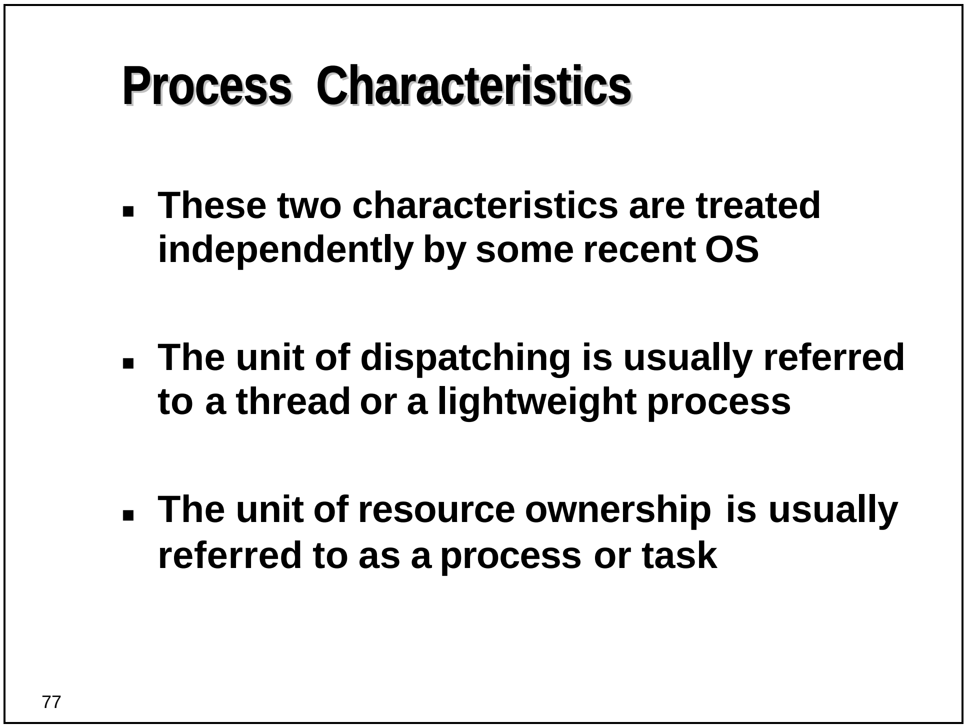
Threads

Chapter 4



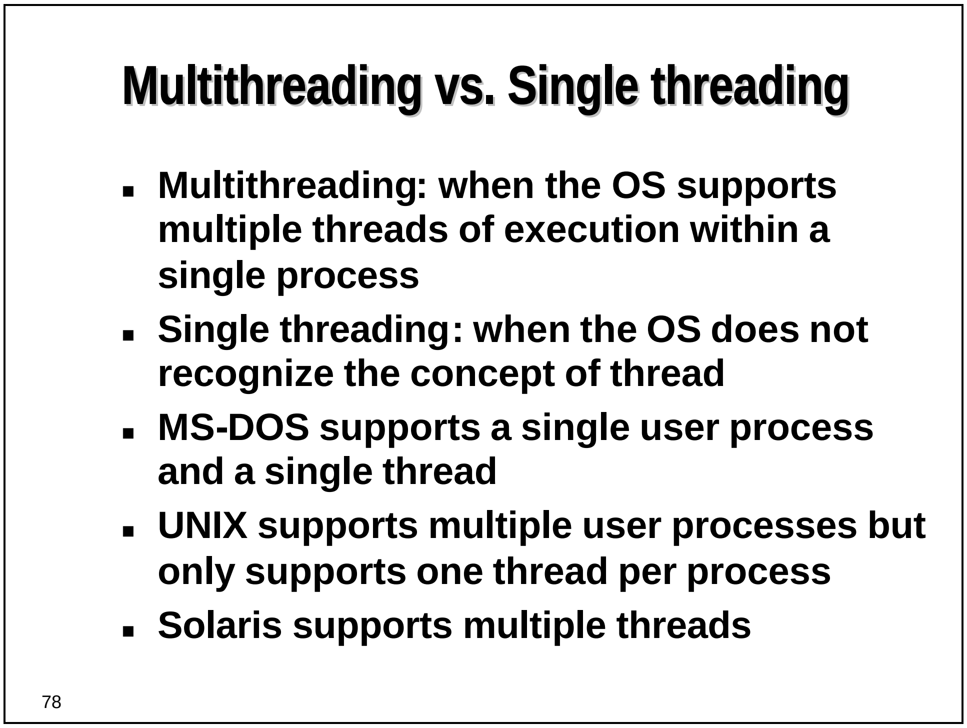
Process Characteristics

- **Unit of resource ownership - process is allocated:**
  - ◆ a virtual address space to hold the process image
  - ◆ control of some resources (files, I/O devices...)
- **Unit of dispatching - process is an execution path through one or more programs**
  - ◆ execution may be interleaved with other process
  - ◆ the process has an execution state and a dispatching priority



Process Characteristics

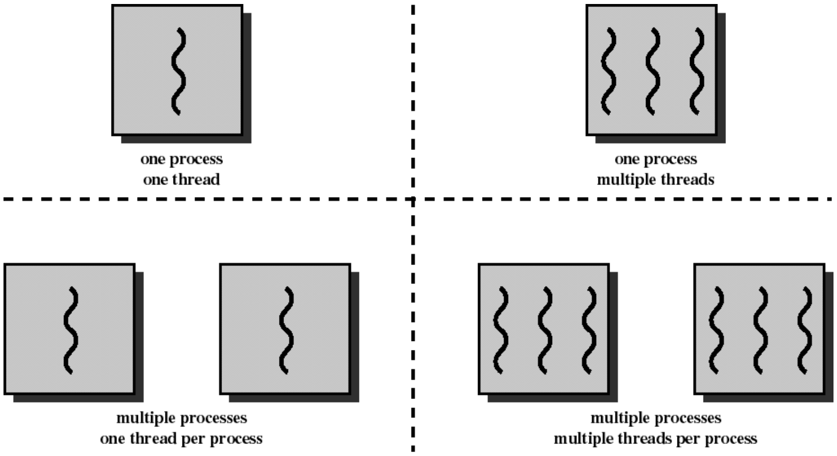
- These two characteristics are treated independently by some recent OS
- The unit of dispatching is usually referred to a thread or a lightweight process
- The unit of resource ownership is usually referred to as a process or task



Multithreading vs. Single threading

- Multithreading: when the OS supports multiple threads of execution within a single process
- Single threading: when the OS does not recognize the concept of thread
- MS-DOS supports a single user process and a single thread
- UNIX supports multiple user processes but only supports one thread per process
- Solaris supports multiple threads

Threads and Processes



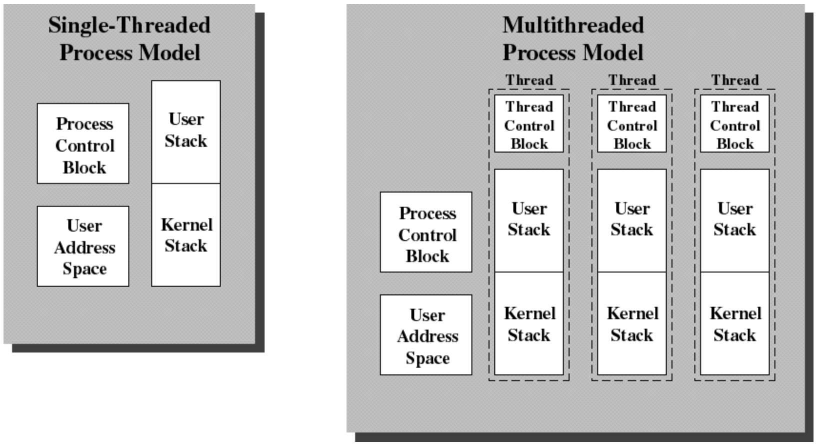
Processes

- Have a virtual address space which holds the process image
- Protected access to processors, other processes, files, and I/O resources

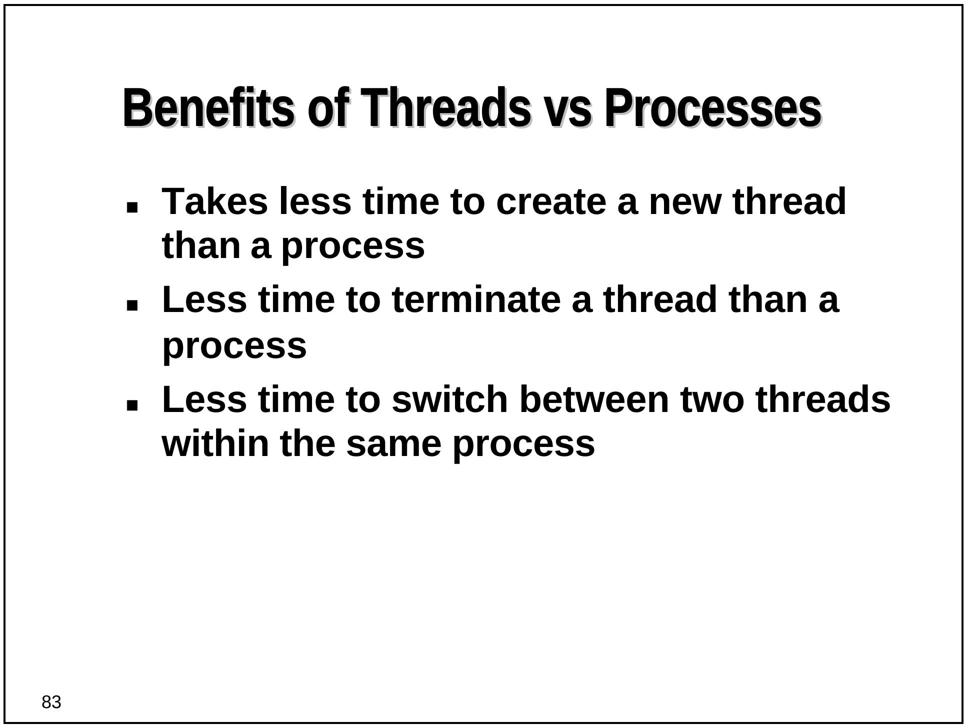
Threads

- Have an execution state (running, ready, etc.)
- Save thread context when not running
- Have an execution stack and some per-thread static storage for local variables
- Have access to the memory address space and resources of its process
  - all threads of a process share this
  - when one thread alters a (non-private) memory item, all other threads (of the process) sees that
  - a file open with one thread, is available to others

Single Threaded and Multithreaded Process Models

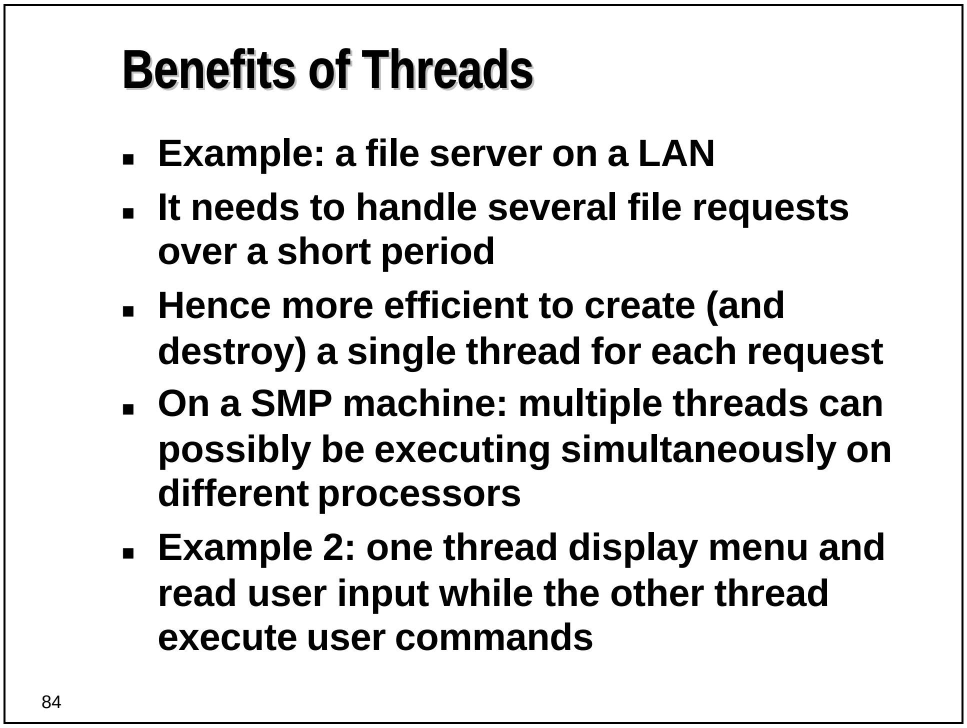


Thread Control Block contains a register image, thread priority and thread state information



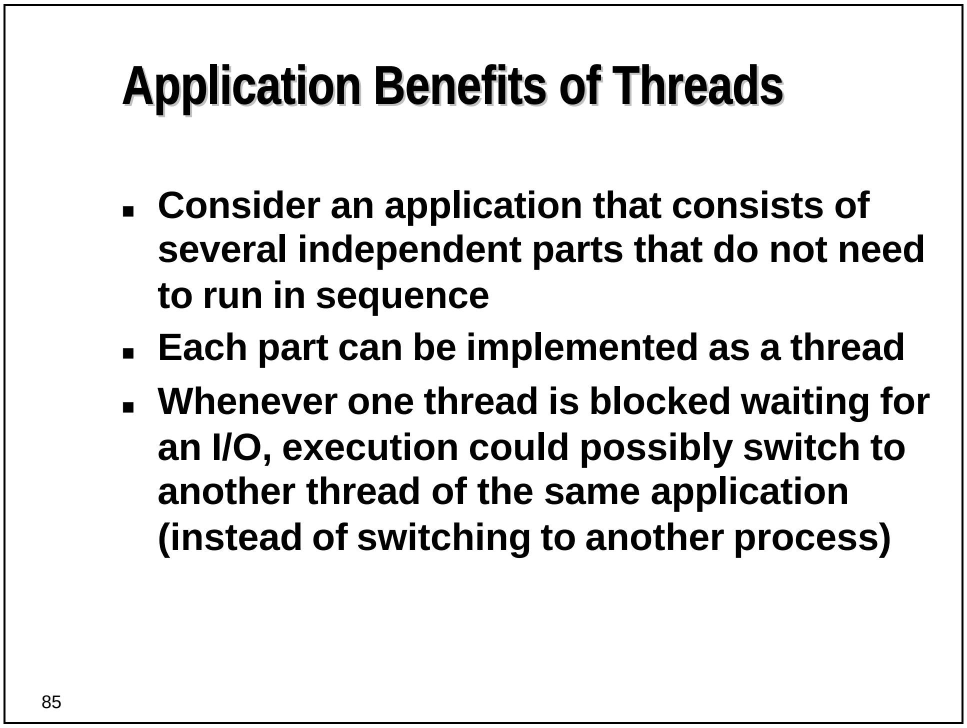
Benefits of Threads vs Processes

- Takes less time to create a new thread than a process
- Less time to terminate a thread than a process
- Less time to switch between two threads within the same process



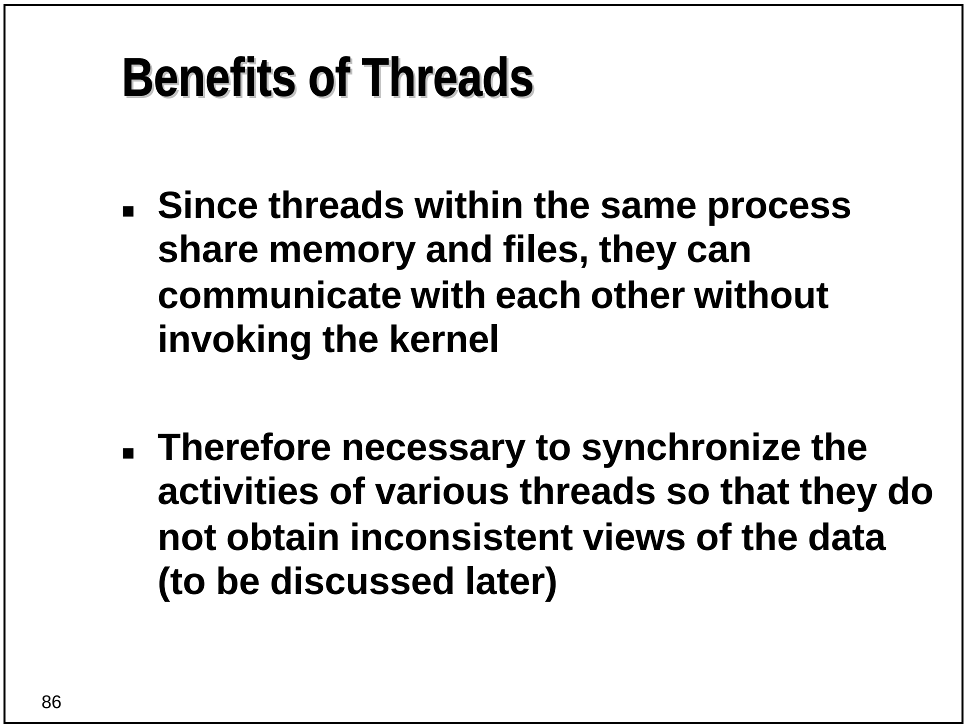
Benefits of Threads

- Example: a file server on a LAN
- It needs to handle several file requests over a short period
- Hence more efficient to create (and destroy) a single thread for each request
- On a SMP machine: multiple threads can possibly be executing simultaneously on different processors
- Example 2: one thread display menu and read user input while the other thread execute user commands



Application Benefits of Threads

- Consider an application that consists of several independent parts that do not need to run in sequence
- Each part can be implemented as a thread
- Whenever one thread is blocked waiting for an I/O, execution could possibly switch to another thread of the same application (instead of switching to another process)



Benefits of Threads

- Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel

- Therefore necessary to synchronize the activities of various threads so that they do not obtain inconsistent views of the data (to be discussed later)

Example of inconsistent view

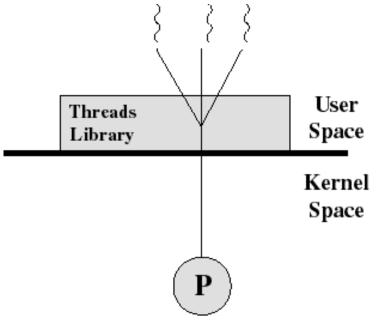
- 3 variables: A, B, C which are shared by thread T1 and thread T2
- T1 computes C = A+B
- T2 transfers amount X from A to B
  - T2 must do: A = A -X and B = B+X (so that A+B is unchanged)
- But if T1 computes A+B after T2 has done A = A-X but before B = B+X
- then T1 will not obtain the correct result for C = A + B

Threads States

- Three key states: running, ready, blocked
- They have no suspend state because all threads within the same process share the same address space
- Indeed: suspending (ie: swapping) a single thread involves suspending all threads of the same process
- Termination of a process, terminates all threads within the process

User-Level Threads (ULT)

- The kernel is not aware of the existence of threads
- All thread management is done by the application by using a thread library
- Thread switching does not require kernel mode privileges (no mode switch)
- Scheduling is application specific



Threads library

- Contains code for:
  - creating and destroying threads
  - passing messages and data between threads
  - scheduling thread execution
  - saving and restoring thread contexts

Kernel activity for ULTs

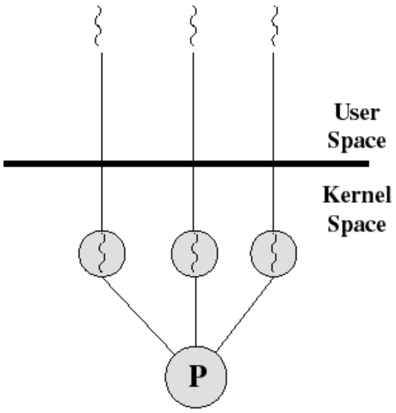
- The kernel is not aware of thread activity but it is still managing process activity
- When a thread makes a system call, the whole process will be blocked
- but for the thread library that thread is still in the running state
- So thread states are independent of process states

Advantages and inconveniences of ULT

- **Advantages**
  - Thread switching does not involve the kernel: no mode switching
  - Scheduling can be application specific: choose the best algorithm.
  - ULTs can run on any OS. Only needs a thread library
- **Inconveniences**
  - Most system calls are blocking and the kernel blocks processes. So all threads within the process will be blocked
  - The kernel can only assign processes to processors. Two threads within the same process cannot run simultaneously on two processors

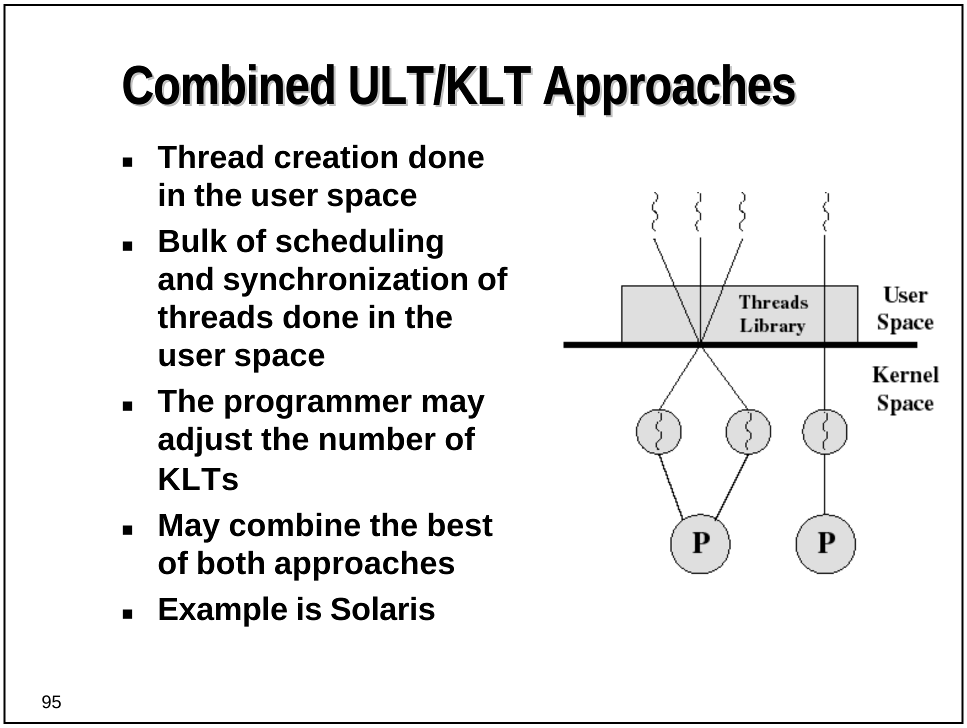
Kernel-Level Threads (KLT)

- **All thread management is done by kernel**
- **No thread library but an API to the kernel thread facility**
- **Kernel maintains context information for the process and the threads**
- **Switching between threads requires the kernel**
- **Scheduling on a thread basis**
- **Ex: Windows NT and OS/2**



Advantages and inconveniences of KLT

- **Advantages**
  - the kernel can simultaneously schedule many threads of the same process on many processors
  - blocking is done on a thread level
  - kernel routines can be multithreaded
- **Inconveniences**
  - thread switching within the same process involves the kernel. We have 2 mode switches per thread switch
  - this results in a significant slow down



Combined ULT/KLT Approaches

- Thread creation done in the user space
- Bulk of scheduling and synchronization of threads done in the user space
- The programmer may adjust the number of KLTs
- May combine the best of both approaches
- Example is Solaris

Threads Library
User Space
Kernel Space
P
P