Threads

- Suspending a process
  - suspends all threads of the process since all threads share the same address space
- Termination of a process
  - terminates all threads within the process

Thread States

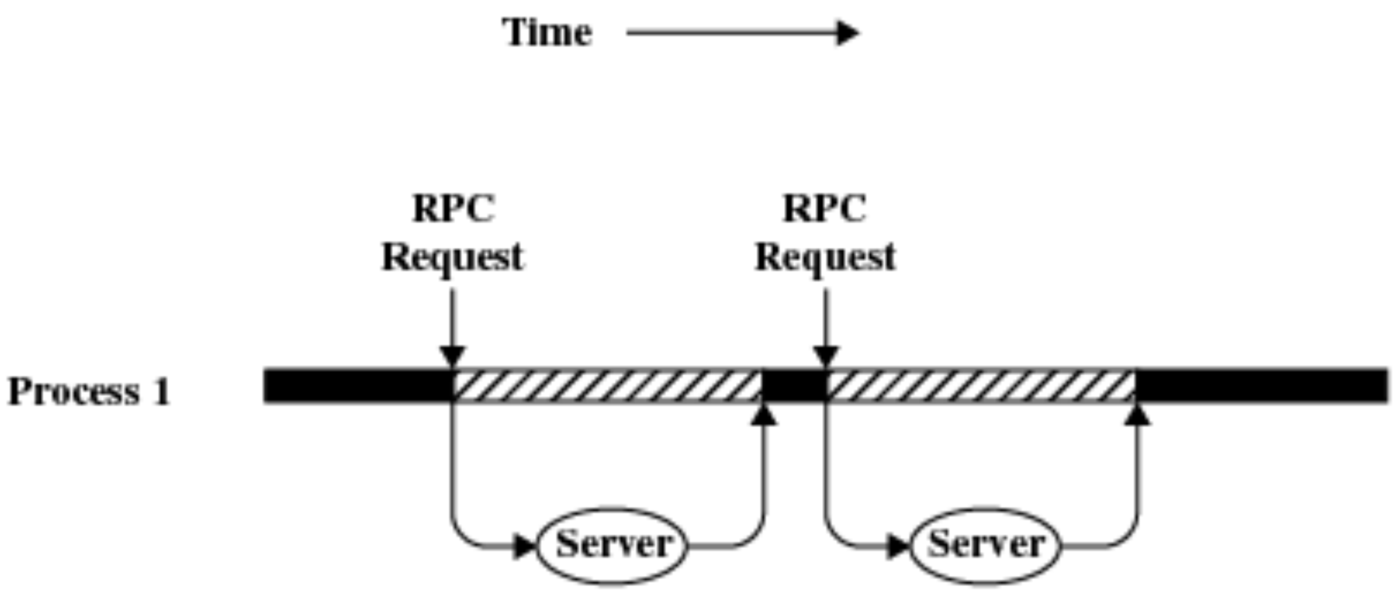
- States of a thread
  - Spawn
    - when process is spawned
    - thread may spawn other threads
    - each thread has its own:
      - register context, state space, and place in ready queue
  - Block
    - when thread waits for event
      - saves user registers, PC and stack pointer

Thread States

- States of a thread
  - Unblock
    - when blocking event occurs
    - thread is moved to ready queue
  - Finish
    - register context and stack is deallocated

Remote Procedure Call Using Single Thread

What is a RPC?



(a) RPC Using Single Thread

Remote Procedure Call Using Threads

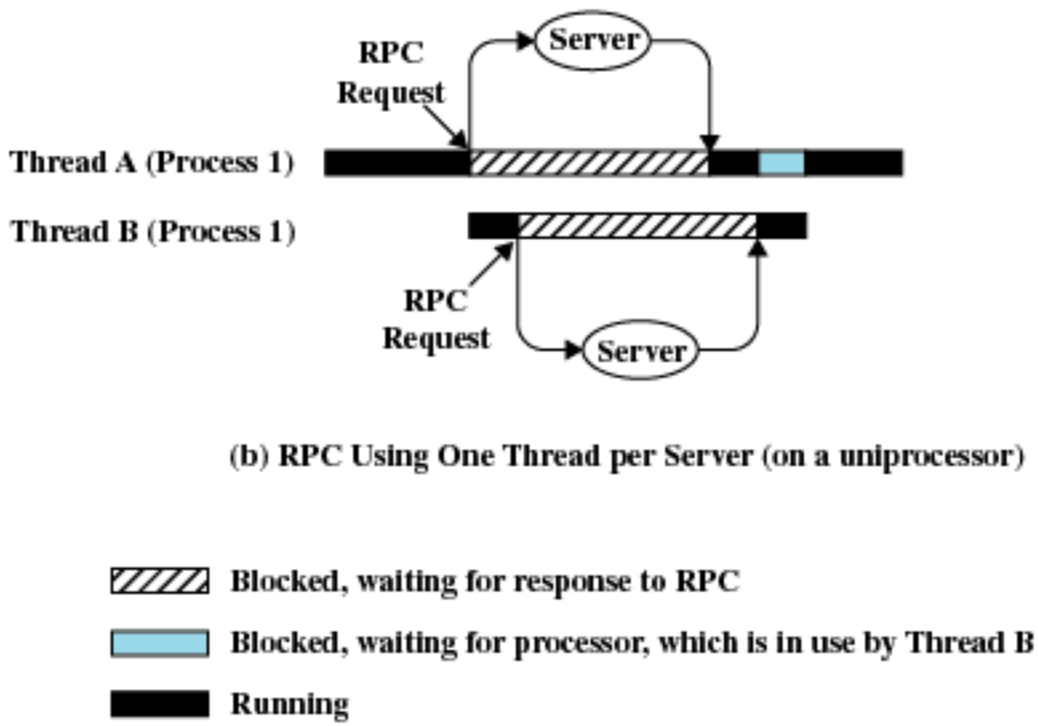


Figure 4.3 Remote Procedure Call (RPC) Using Threads

Multithreading

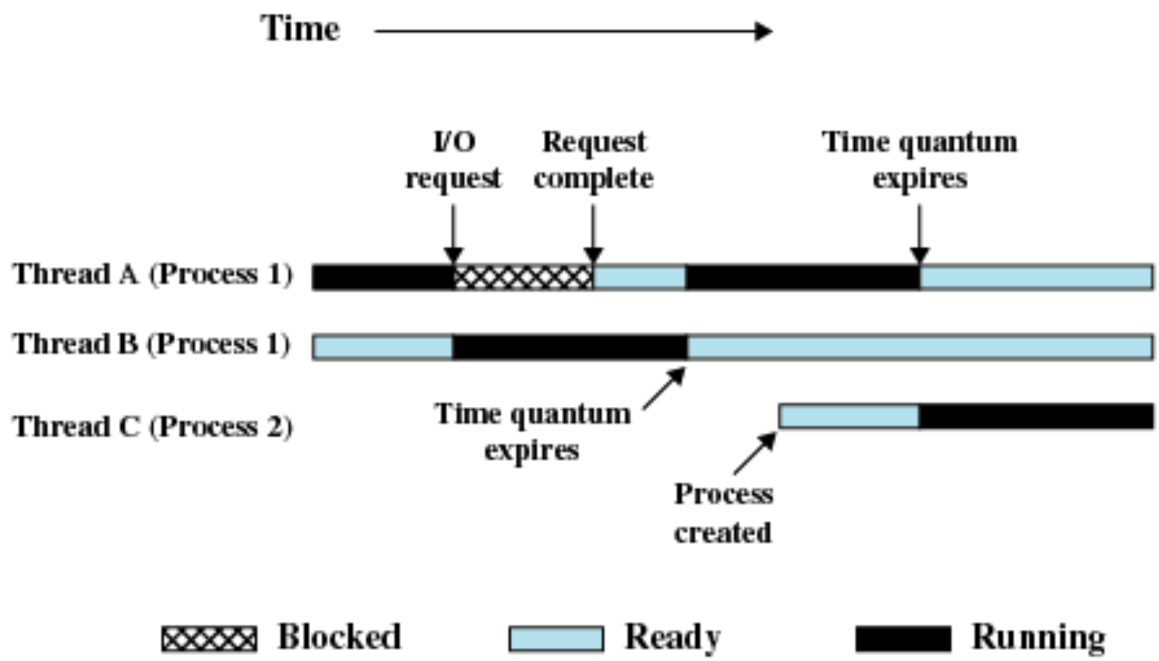


Figure 4.4 Multithreading Example on a Uniprocessor

Basic questions

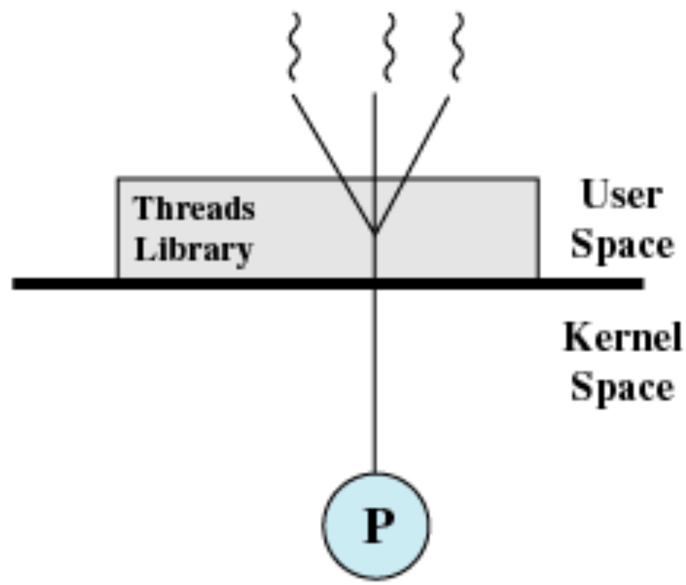
- What is the difference between this and multiprocessing?
  - kind of looks the same, or...?

- Is there a need to synchronize threads?
  - e.g. two threads insert an element into a linked structure

User-Level Threads (ULT)

- All thread management is done by the application
  - e.g. using threads library
- The kernel is not aware of the existence of threads

User-Level Threads



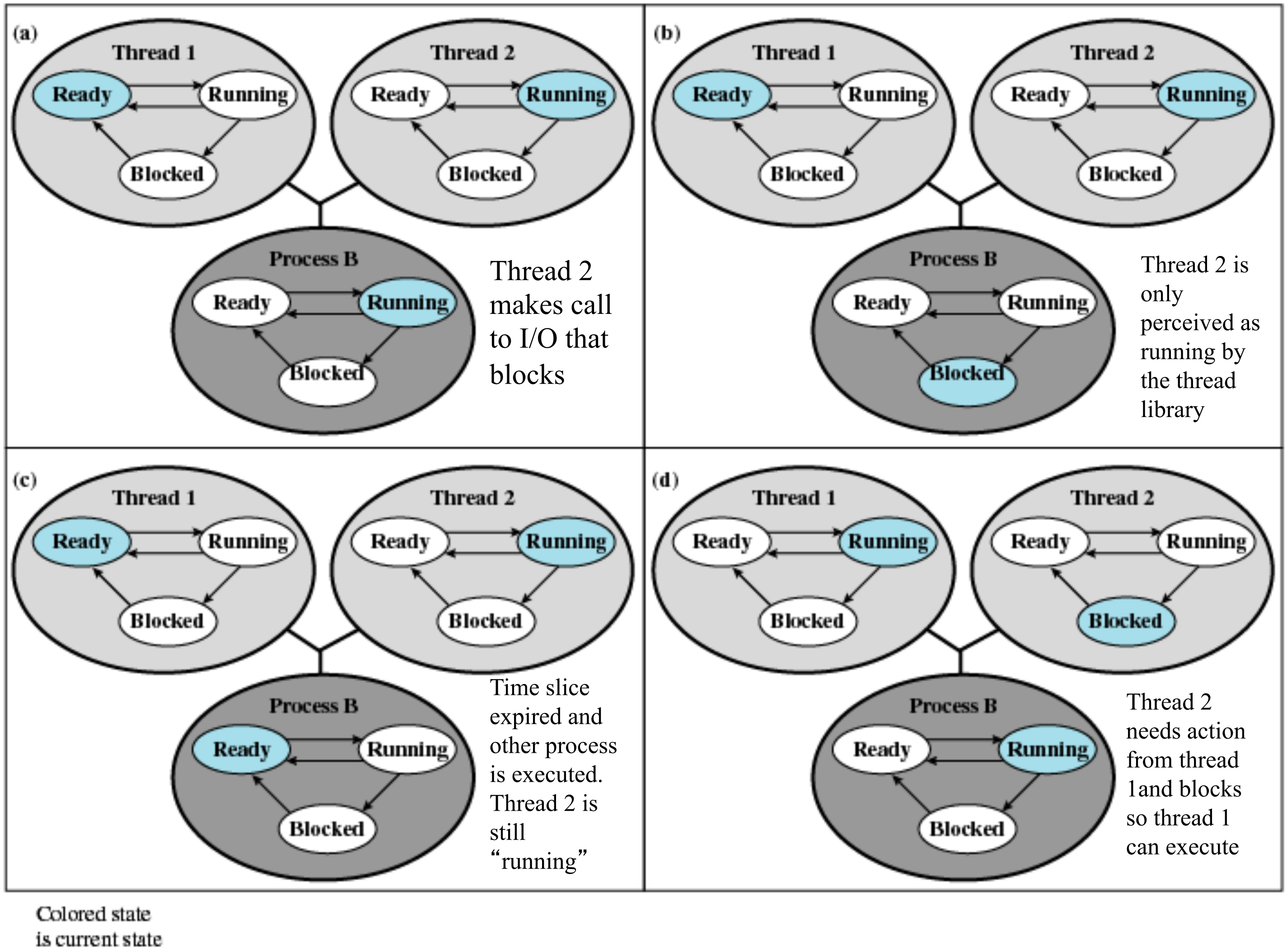
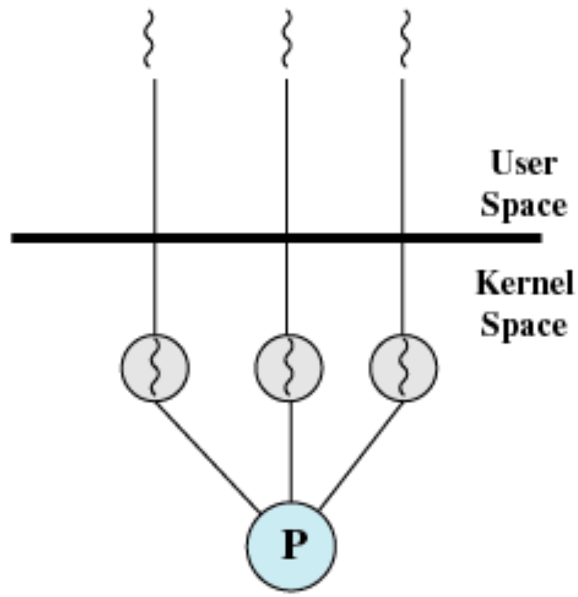


Figure 4.7 Examples of the Relationships Between User-Level Thread States and Process States

Kernel-Level Threads (KLT)

- Often called *lightweight processes*
- Windows is an example of this approach
- Kernel maintains context information for the process and the threads
- Scheduling is done on a thread basis

Kernel-Level Threads



(b) Pure kernel-level

VAX Running UNIX-Like Operating System

Table 4.1 Thread and Process Operation Latencies (μs) [ANDE92]

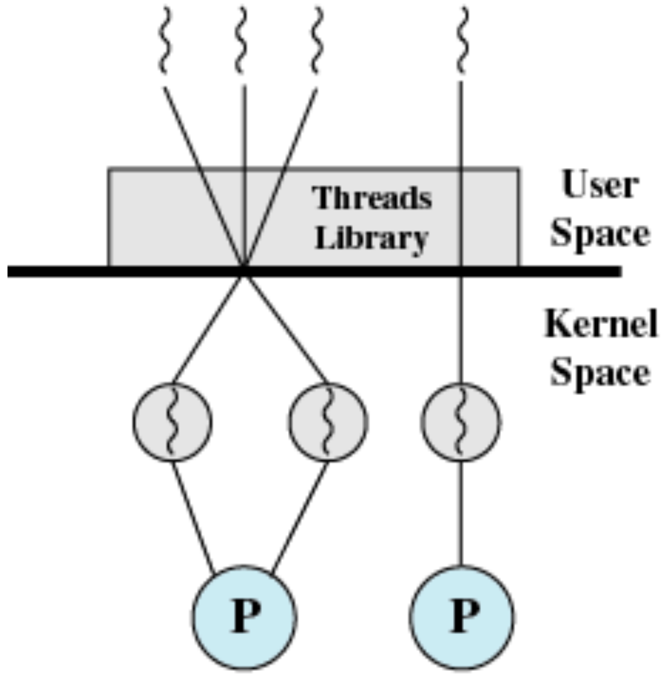
Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

Combined Approaches

- Thread creation is done in user space
- Bulk of scheduling and synchronization of threads done within application

- Example is Solaris

Combined Approaches



(c) Combined

Relationship Between Threads and Processes

Table 4.2 Relationship Between Threads and Processes

Threads:Processes	Description	Example Systems
1:1	Each thread of execution is a unique process with its own address space and resources.	Traditional UNIX implementations
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.	Windows NT, Solaris, Linux OS/2, OS/390, MACH
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.	Ra (Clouds), Emerald
M:N	Combines attributes of M:1 and 1:M cases.	TRIX

Advantages of ULT over KLT

- thread switching does not require kernel mode privileges
  - saves two mode switches (user-to-kernel and kernel-to-user)
- application specific scheduling
  - applications may prefer their own specific scheduling algorithm
- ULT can run on any OS

Disadvant. of ULT vs KLT

- Many OS system calls are blocking.
  - so if ULT executes such call all threads within its process are blocked
- In pure ULT strategy a multithreaded application cannot take advantage of multiprocessing
  - no concurrency