CISC 7310X
C03: Threads

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Recap & issues
- Modeling revisited
- Thread
- Assignment
  - Team Project 2

Recap & Issues

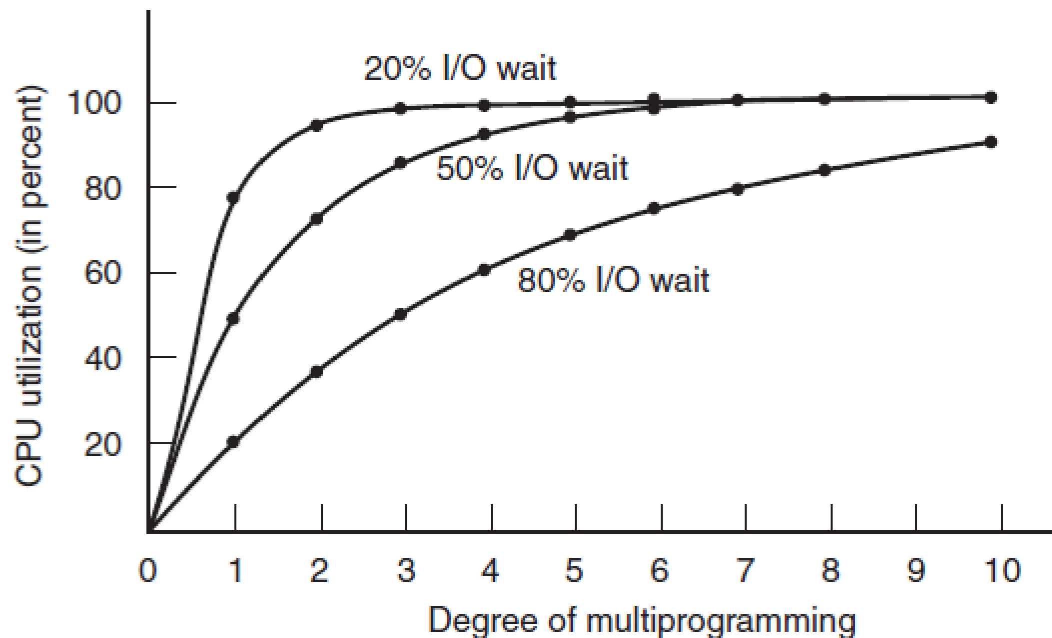
- Topics
  - Process and a simple model of multiprogramming
- Assignments
  - Practice and Project 1
    - Where are those fields in the process table entry?
    - Question received: should I trace to next level/data structure? How deep should I dig?

Simple Multiprogramming Model

- Assumptions
  - n processes in the main memory;
  - a process spends a fraction p of its waiting for I/O independent of the others
- Analysis
  - CPU is idle when all processes are waiting for I/O
  - The probability that all n processes waiting for I/O is p^n
  - CPU Utilization = $1 - p^n$

Modeling Multiprogramming

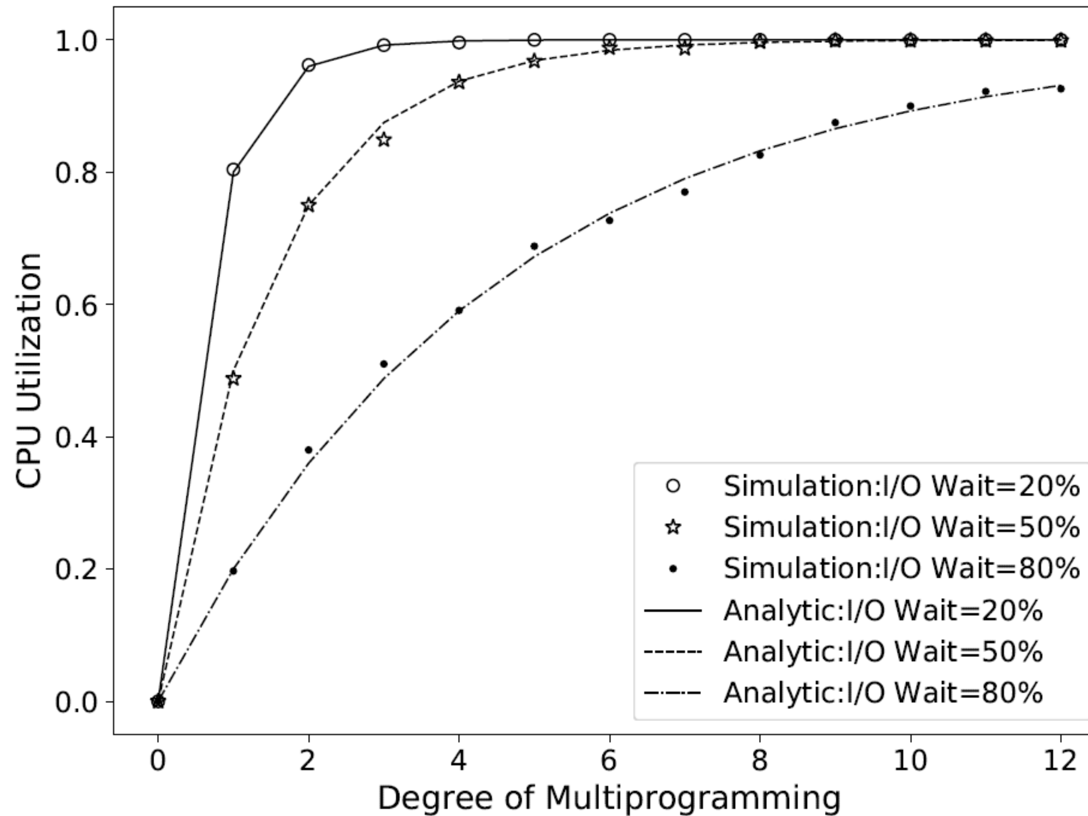
- CPU utilization



- CPU utilization [Figure 2-6 in Tanenbaum & Bos, 2014]

Simulation Model

- Analytic and simulation models

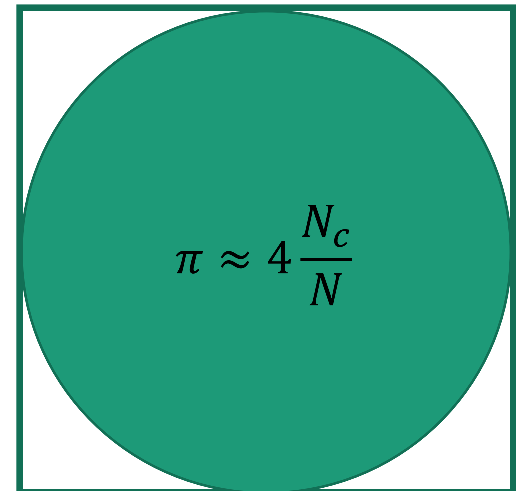


Monte Carlo Simulation

- Estimate the value of an unknown quantity
  - drawing random samples (from a population)
  - applying principles of inferential statistics

Estimate the Value of Unknown Quantity

- Estimate the value of π
  - If we are to toss a pebble in a square that bounds a circle, the chance that the pebble lands on inside the circle is proportional to its area.



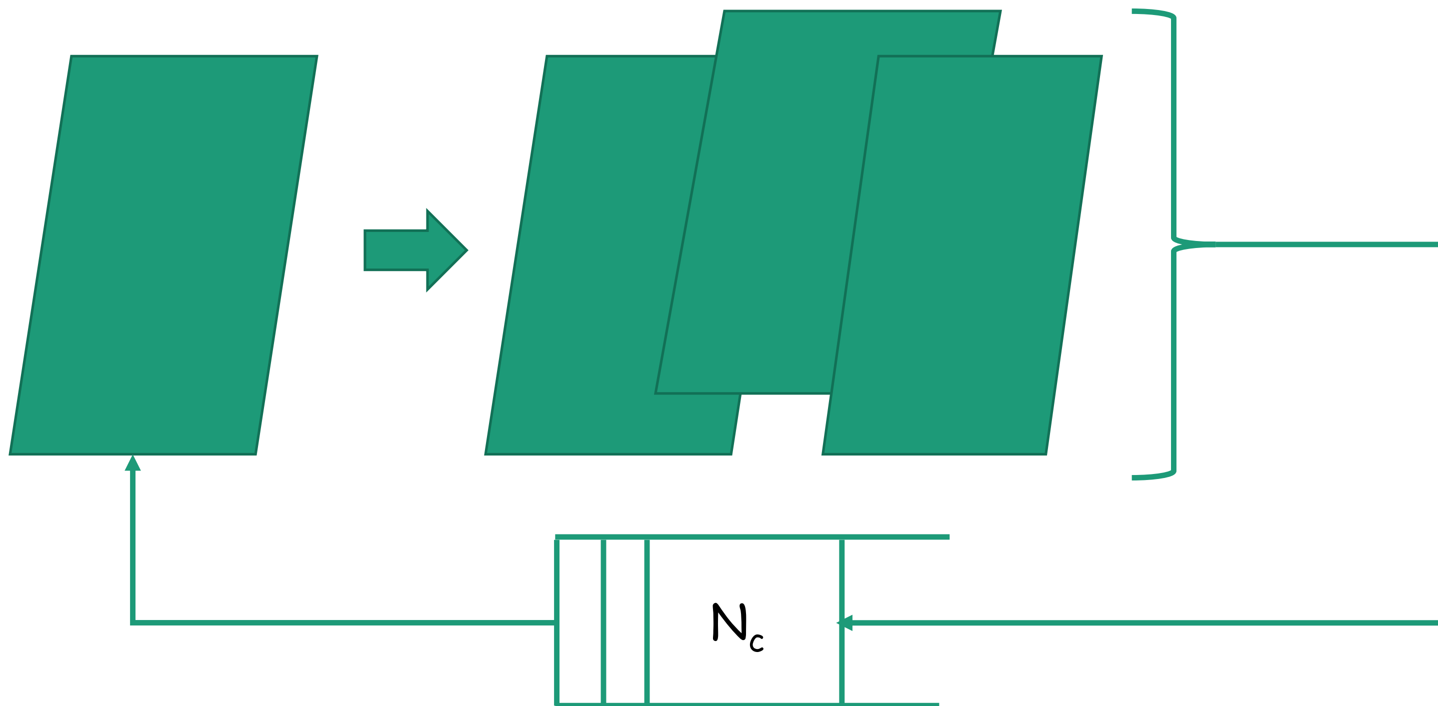
Law of Large Numbers

- Implication: it takes longer time to compute when N is increased.

$$\bar{X} \to E(X) \text{ as } N \to \infty$$

Multiple Processes

- Use multiple processes to estimate π



Data Sharing

- How are the data are shared?
- Examine the example implementation using an FIFO pipe
  - Sample Program Repository
    - W3_Process/simulation/multiprocess

Questions

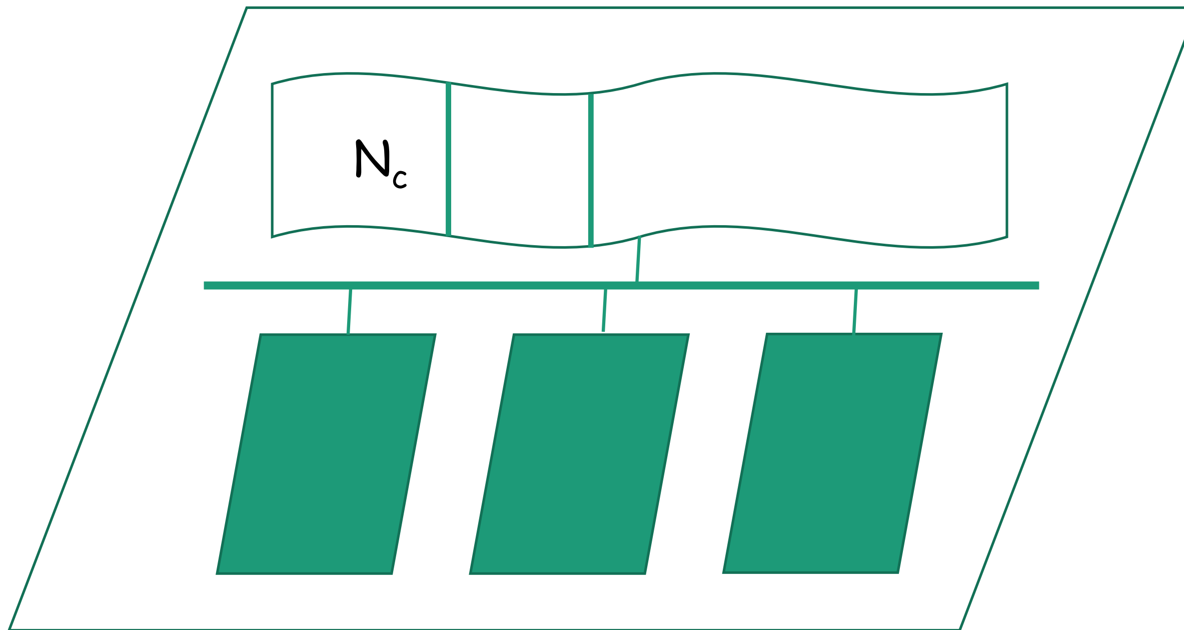
- Tools of the trade
  - Programming & graphing
- Monte Carlo simulation
- Application in estimation multiprogramming metrics
- Multiprocessed implementation of a Monte Carlo simuation

Multithread

- How are the data are shared?
- Examine the example implementation using POSIX threads
  - Sample Program Repository
    - W4_Thread/simulation/multithread

Multithread

- Use multiple threads to estimate π



Thread Library: POSIX Threads

- Example the simulation program

Thread call	Description
Pthread_create	Create a new thread
Pthread_exit	Terminate the calling thread
Pthread_join	Wait for a specific thread to exit
Pthread_yield	Release the CPU to let another thread run
Pthread_attr_init	Create and initialize a thread's attribute structure
Pthread_attr_destroy	Remove a thread's attribute structure

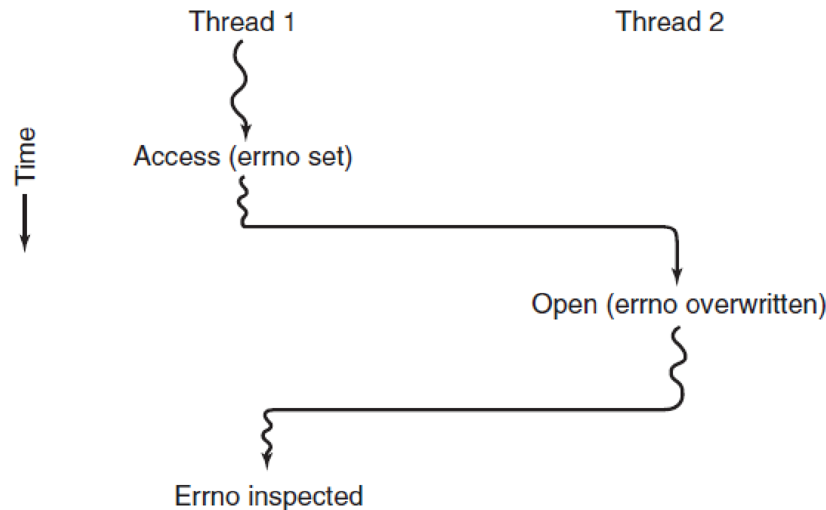
Pthread function calls [Figure 2-14 in Tanenbaum & Bos, 2014]

User's Perspective: Writing Multithreaded Code

- Multiprocessed and multithreaded code are more difficult to write
- Multithreaded code
  - Examples:
    - Global variables shared by multiple threads
    - Thread local variables

Global Variables

- Easier to share data among multiple threads than among multiple processes



Conflict may happen [Figure 2-19 in Tanenbaum & Bos, 2014]

Use Thread

Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls, interrupts

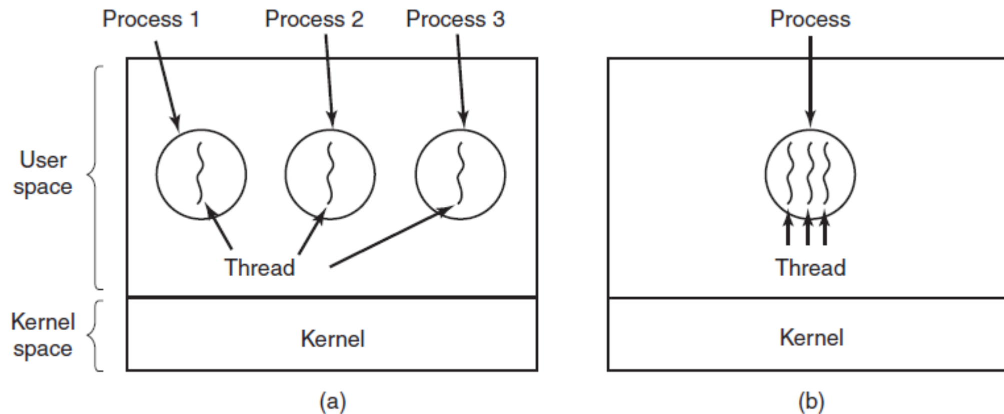
Three ways to construct a server. [Figure 2-10 in Tanenbaum & Bos, 2014]

Questions

- User's perspective on thread
- Issues when writing multithreaded code
  - Sharing data: variables
  - Synchronization issues (to be discussed in future lessons)

Thread Model

- User threads
  - Threads provided at the user level
- Kernel threads
  - Threads provided by the kernel



Process and threads [Figure 2-11 in Tanenbaum & Bos, 2014]

Process and Thread

- Many-to-one model
- One-to-one model
- Many-to-many model

Many-to-one model

- Map many user level threads to one kernel thread
  - Thread management done by the thread library at in user space
- Efficient, however, the entire process may be blocked
  - Examples: Solaris, GNU portal threads

One-to-One Model

- Maps each user thread to a kernel thread
- High concurrency
- Creating kernel threads is more costly
- Examples: Linux and Windows

Many-to-Many Model

- Multiplexes many user-level threads to a smaller or equal number of kernel threads
- More complex to design and build
- Enjoy benefits both one-to-one model and many-to-one model
- Examples: IRIX, HP-UX

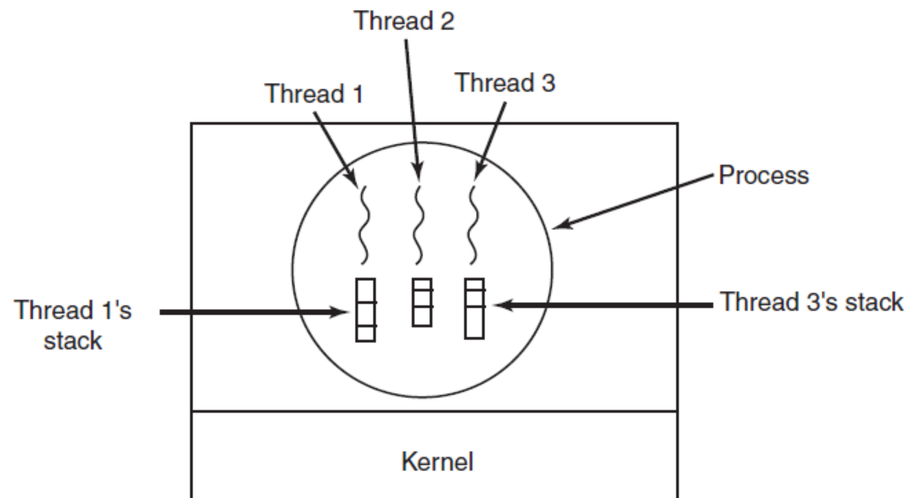
Process and Thread Properties

Per process items	Per thread items
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

Process and threads properties [Figure 2-12 in Tanenbaum & Bos, 2014]

Address Space & Stack

- Threads shares the process's address space
- Each thread has its own stack



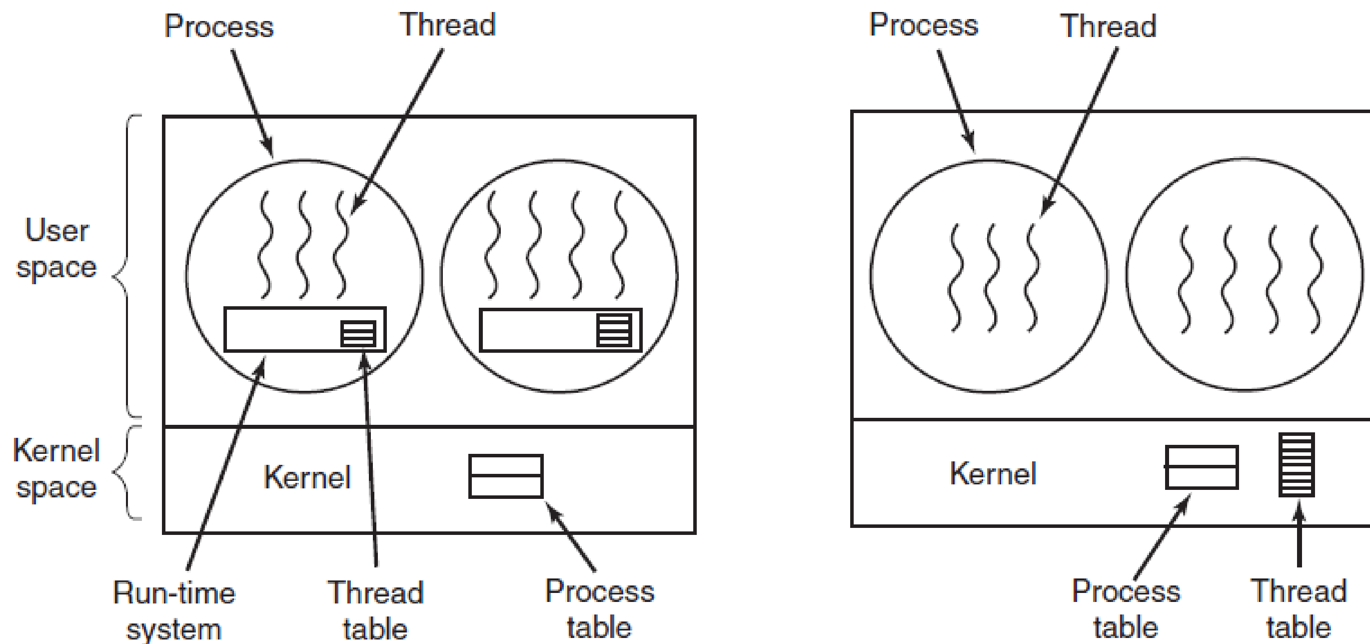
Each thread has its own stack [Figure 2-13 in Tanenbaum & Bos, 2014]

System Perspective: Implementing Threads

- User-level (many-to-one)
- The kernel (one-to-one)
- Hybrid of the two (many-to-many)

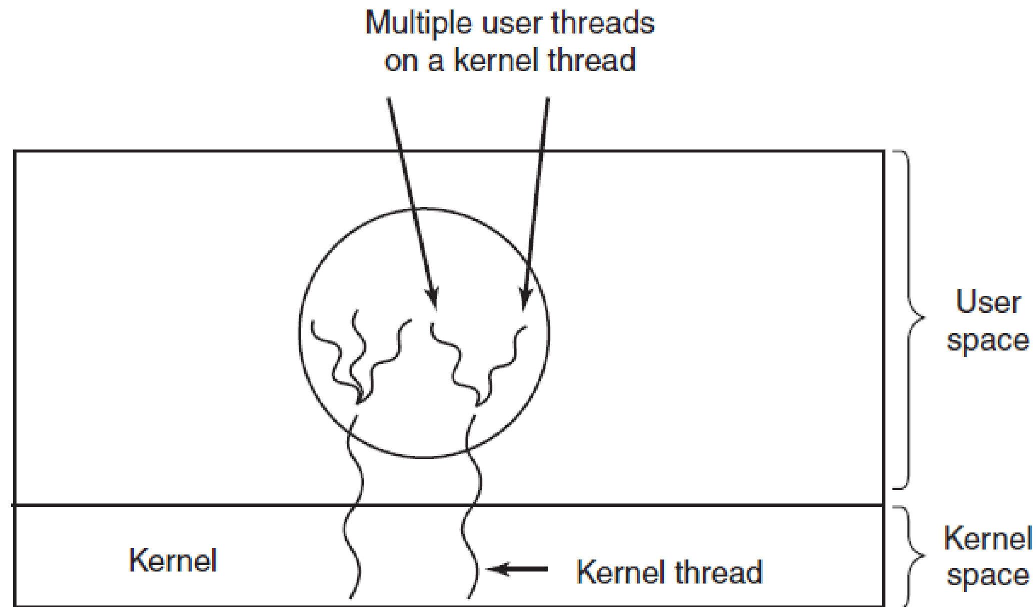
User Space or the Kernel

- In the user space, or in the kernel



User- and kernel-level threads [Figure 2-16 in Tanenbaum & Bos, 2014]

Hybrid

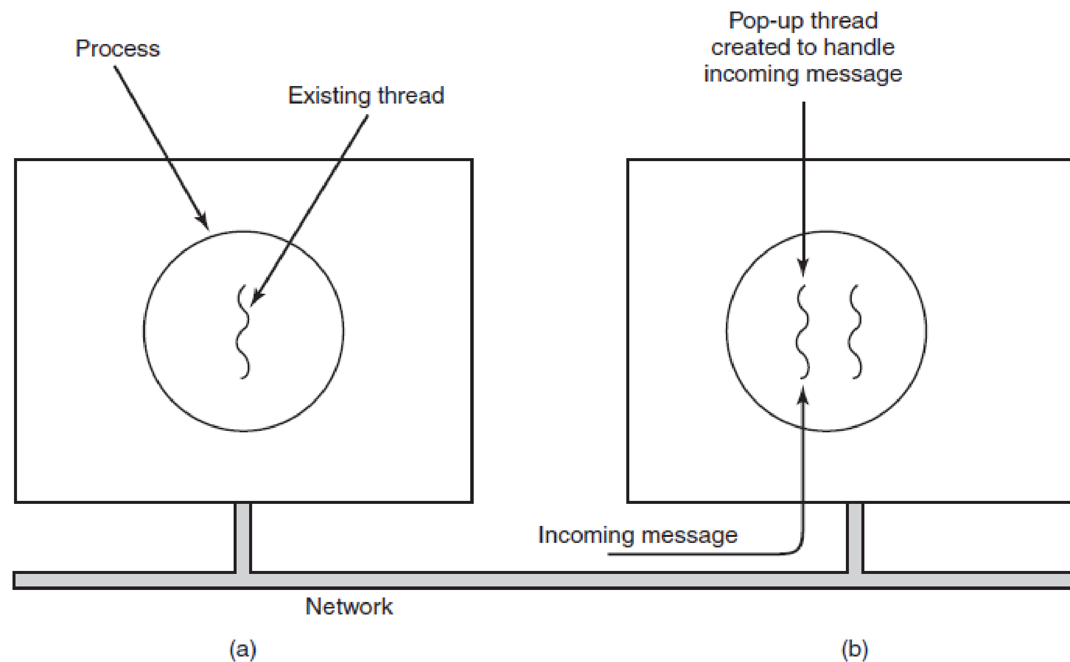


Multiplexing user-level onto kernel-level threads [Figure 2-17 in Tanenbaum & Bos, 2014]

Efficiency and Concurrency

- Kernel threads are more expensive to create
  - Can support multiple processors
- User-level threads can be blocked by the process
  - Less concurrency, in particular, on multiprocessor systems
- Scheduler activation
- Pop-up threads

Pop-up Threads



Creation of a new thread when a message arrives [Figure 2-18 in Tanenbaum & Bos, 2014]

Questions

- Threads and multithreads
- Why threads?
  - Shared address space
  - Light weight
  - More efficient use of CPUs
    - Similar to process (multiprogramming)
    - Multiple processors
- Modeling

Project 2

- Objectives
    - First hand experience with process and thread
    - Experimental approach to evaluate a system