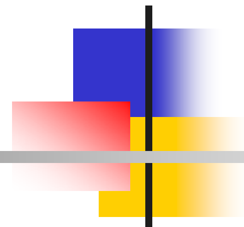
CSE398: Network Systems Design



Instructor: Dr. Liang Cheng
Department of Computer Science and Engineering
P.C. Rossin College of Engineering & Applied Science
Lehigh University

February 23, 2005

Outline

- Recap
  - Packet processing functions
- Protocol software
- Summary and homework

Outline

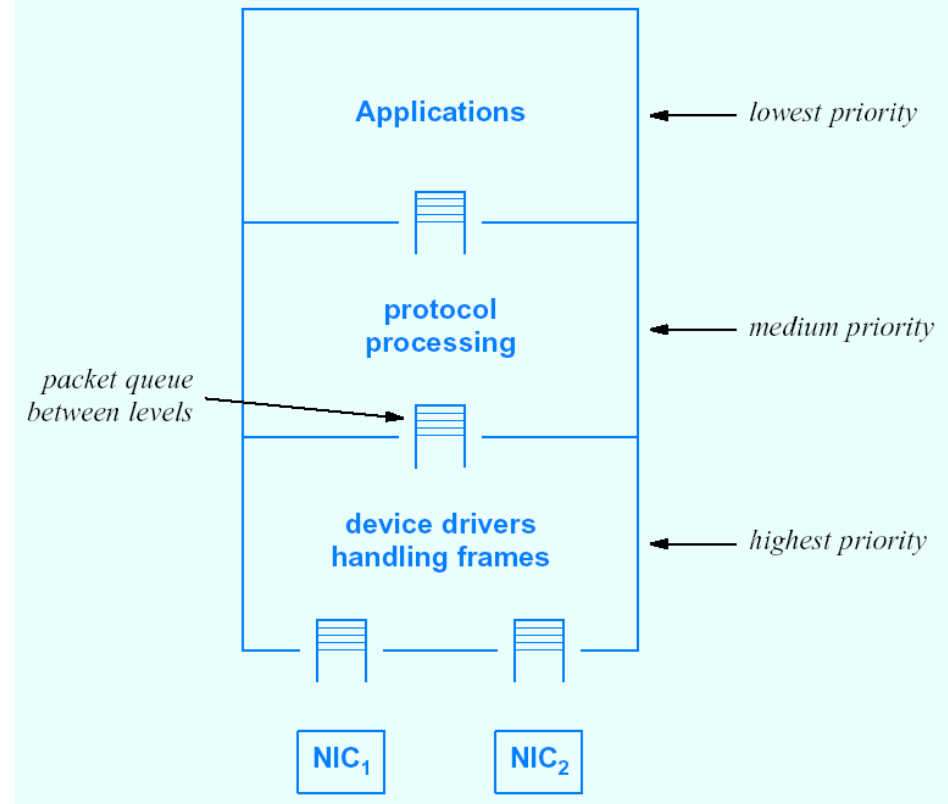
- Recap
- Protocol software on a conventional processor
- Summary and homework

Possible Implementations of Protocol Software

- In an application program
  - Easy to program
  - Runs as user-level process
  - No direct access to network devices
  - High cost to copy data from kernel address space
  - Cannot run at wire speed
- In an embedded system
  - Special-purpose hardware device
  - Dedicated to specific task
  - Ideal for stand-alone system
  - Software has full control
- In an operating system kernel
  - More difficult to program than application
  - Runs with kernel privilege
  - Direct access to network devices

Processing Priorities

- Determine which code CPU runs at any time
  - Hardware devices need highest priority
  - Protocol software has medium priority
  - Application programs have lowest priority
- Queues provide buffering across priorities
- Why hardware device-related processing has higher priority?



OS Implementation of Priority (1)

- Interrupt mechanism
  - Operates asynchronously
  - Saves current processing state
  - Changes processor status
  - Branches to specified location
- Hardware interrupt
  - Caused by device and must be serviced quickly
  - Livelock
- Software interrupt
  - Caused by executing program
  - Other OS code < Priority < hardware interrupt
  - Protocol stack operates as software interrupt
    - When packet arrives, hardware interrupts and device driver raises software interrupt
    - When device driver finishes, hardware interrupt clears and protocol code is invoked

OS Implementation of Priority (2)

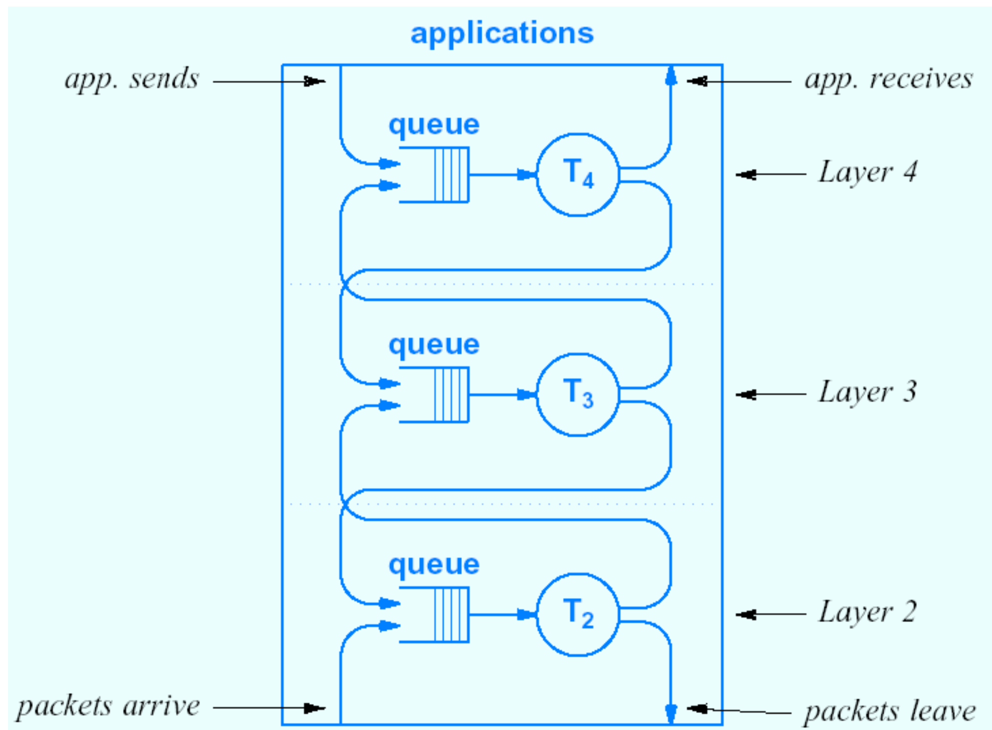
- Kernel threads
  - Finer-grain control than software interrupts
  - Can be assigned arbitrary range of priorities
- Packet passes among multiple threads of control
- A queue of packets between each pair of threads
- Threads synchronize to access queues

Possible Organization of Kernel Threads for Layered Protocols

- One thread per layer
- One thread per protocol
- Multiple threads per protocol
- Multiple threads per protocol plus timer management thread(s)
- One thread per packet

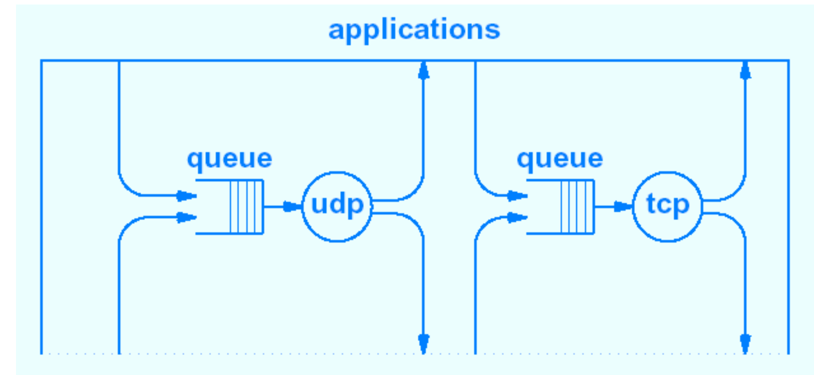
One Thread per Layer

- Easy for programmer to understand
- Implementation matches concept
- Allows priority to be assigned to each layer
- A packet is enqueued once per layer



One Thread per Protocol

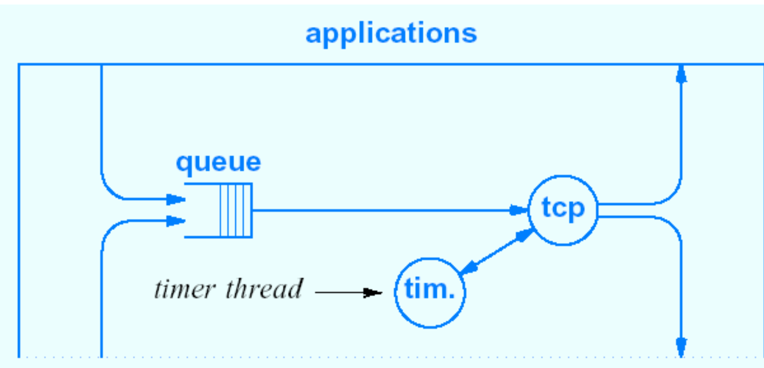
- Like one thread per layer
  - Implementation matches concept
  - A packet is enqueued once per layer
- Advantages over one thread per layer
  - Easier for programmer to understand
  - Finer-grain control
  - Allows priority to be assigned to each protocol



- TCP and UDP reside at same layer
- Separation allows different priorities

Multiple Threads per Protocol

- Further division of duties
- Simplifies programming
- More control than single thread
- Typical division
  - Thread for incoming packets
  - Thread for outgoing packets
  - Thread for management/timing



- Separate timer makes programming easier

Is One Timer Thread Sufficient?

- In theory
  - Yes
- In practice
  - Large range of timeouts (microseconds to tens of seconds)
  - May want to give priority to some timeouts
- Solution: two or more timer threads

Timers and Protocols

- TCP
  - Retransmission timeout
- ARP
  - Cache entry timeout
- IP
  - Reassembly timeout
- Observations
  - Many protocols implement timeouts
  - Each timer thread incurs overhead
  - Consolidate timers for multiple protocols

Multiple Timer Threads

- Two threads usually suffice
- Large-granularity timer
  - Values specified in seconds
  - Operates at lower priority
- Small-granularity timer
  - Values specified in microseconds
  - Operates at higher priority

- Thread for layer *i*
  - Needs to pass a packet to layer *i*+ 1
  - Enqueues the packet
- Thread for layer *i*+ 1
  - Retrieves packet from the queue
- Context switch required!
  - OS function
  - CPU passes from current thread to a waiting thread
  - High cost ⇒ Must be minimized

One Thread per Packet

- Preallocate set of threads
- Thread operation
  - Waits for packet to arrive
  - Moves through protocol stack
  - Returns to wait for next packet
- Minimizes context switches

Asynchronous & Synchronous Programming

- Software interrupts and threads lead to different styles of programming
  - Interrupts => asynchronous
  - Threads => synchronous

Asynchronous API

- Asynchronous: event-driven
  - Programmer writes set of functions and specifies which function to invoke for each event type
  - Programmer has no control over function invocation
  - Functions keep state in shared memory
  - Difficult to program

Synchronous API

- Using blocking
  - Writes main flow-of-control
  - Explicitly invokes functions as needed
  - Built-in functions block until request satisfied
  - Easy to program

Synchronous API Using Polling

- Nonblocking form of synchronous API
- Each function call returns immediately
  - Performs operation if available
  - Returns error code otherwise

Typical Implementations

- Application program
  - Synchronous API using blocking (e.g., socket API)
  - Another application thread runs while an application blocks
- Embedded systems
  - Synchronous API using polling
  - CPU dedicated to one task
- Operating systems
  - Asynchronous API
  - Built on interrupt mechanism

Outline

- Recap
- Protocol software
- Summary and homework

Homework (due on 02/28)

- 7.2. Problem 7 of Chapter 7 (Page 97).
- A question that does not need to be handed in: check whether link state routing requires a timer?