Winter 2025

EECS 482: Introduction to Operating Systems

Lecture 21: Networks

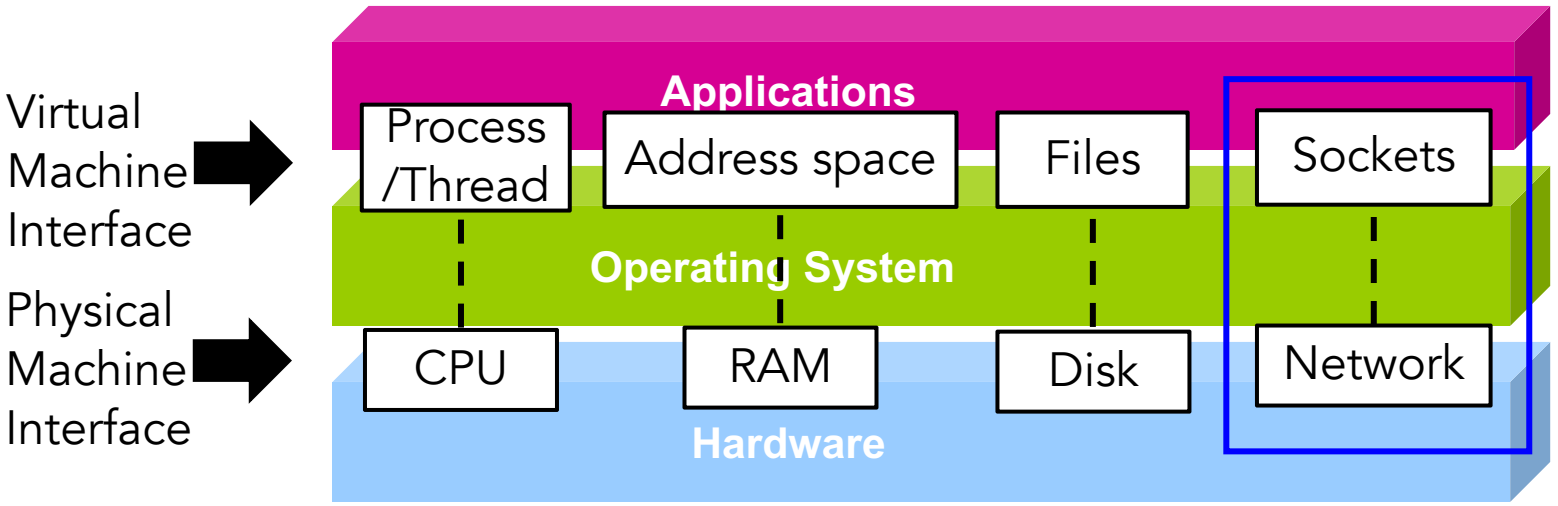
Prof. Ryan Huang

Administration

Student + staff social #2

- Thursday, April 3rd, from 19:00 to 21:00 in GGBL 2147
- Board games and card gams
- RSVP by like the Piazza post @1265

OS abstractions

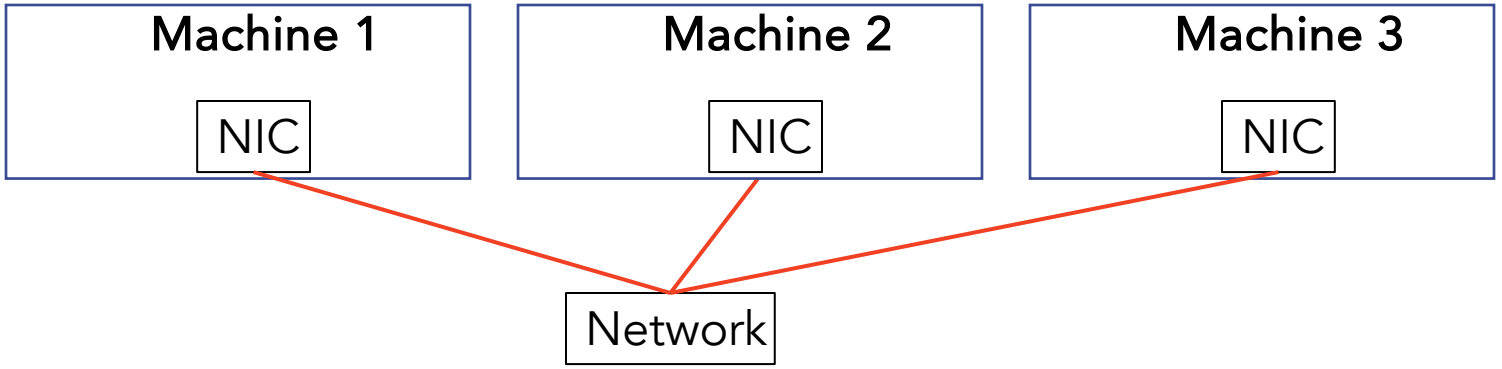


Network abstractions

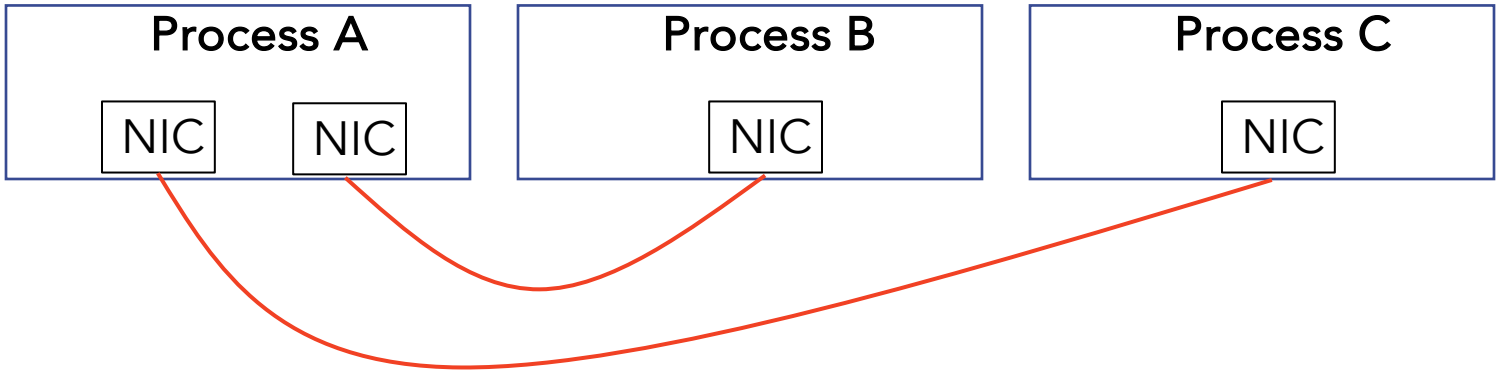
Hardware reality	**OS abstraction**
Destination named by MAC address of NIC	Destination named by symbolic hostname
Deliver to computer on LAN	Route to final destination across multiple networks
Machine-to-machine	Process-to-process
Unordered, unreliable	Ordered, reliable
Finite-size messages	Byte streams
Multiple distinct computers, each with own resources	Single computer with combined resources

OS abstraction of network

Hardware reality



OS abstraction



Machine-to-machine communication ➔ Process-to-process communication

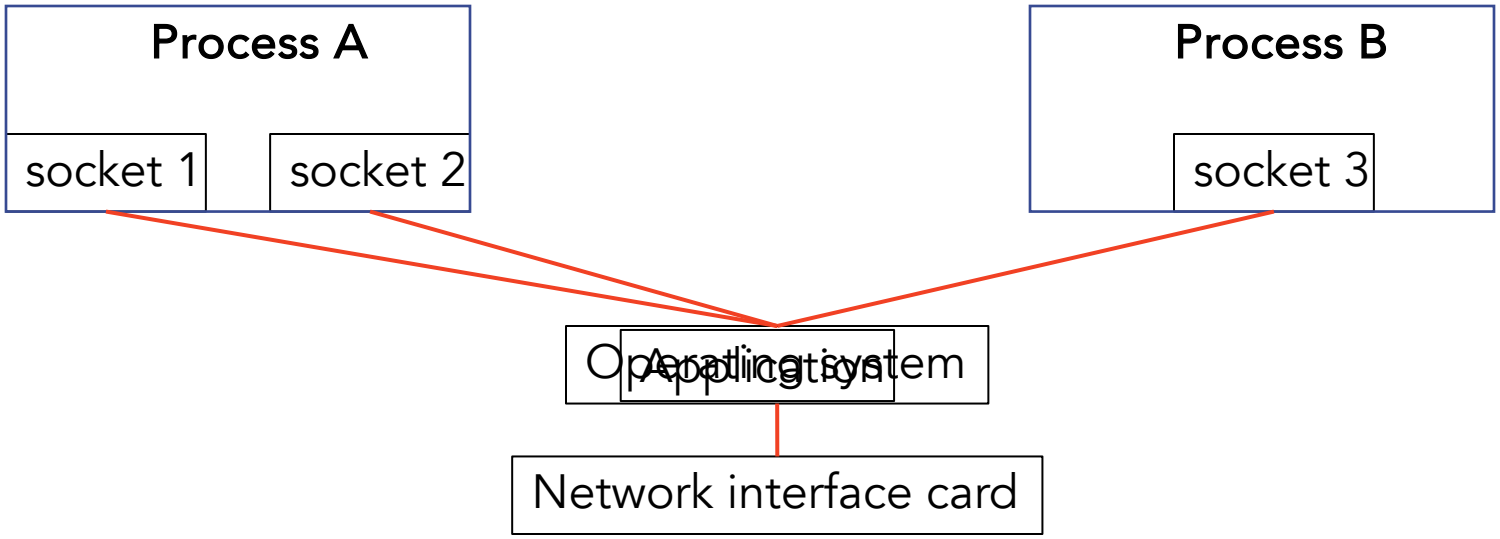
Machine ➔ process

- Processors ➔ threads
- DRAM ➔ address spaces
- Disks ➔ files and directories
- Network interface card ➔ sockets

Socket

- Virtual network interface card
- Endpoint for communication
- NIC named by MAC/IP address; socket named by "port number" (via bind)
- Programming interface: BSD sockets

Socket abstraction



Apartment analogy

Apartment building	Computer

Types of sockets

UDP (user datagram protocol): IP + sockets

TCP (transmission control protocol): IP + sockets + reliable, ordered byte streams

How to provide reliable, ordered byte streams on top of unreliable, unordered packets?

Unordered ➔ ordered messages

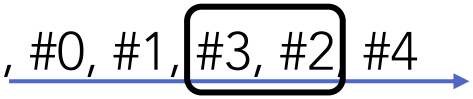
Hardware reality: messages can be re-ordered

- Sender: msg A, msg B, msg C, msg D, msg E
- Receiver: msg A, msg B, msg D, msg C, msg E

OS abstraction: messages received in order sent

How to provide this abstraction?

- Assign sequence numbers to successive messages
- What to do once an arrived message is out of order?
  - E.g., #0, #1, #3, #2, #4



a) Drop #3?
b) Drop #2?
c) Save #3, deliver after #2 is delivered?

Unordered ➔ ordered messages

Message ordering requires a notion of network "connection"

- Should not enforce ordering for unrelated messages

TCP: connects two sockets with a point-to-point, bidirectional, ordered channel

- Server calls `accept` to accept incoming connection
- Client calls `connect` to establish connection with server

Sequence # is specific to a socket-to-socket connection

Unreliable ➔ reliable messages

Hardware interface: messages can be

- Dropped
- Duplicated
- Corrupted

OS abstraction: each message is delivered exactly once (and in the right order)

Unreliable ➔ reliable messages

How to detect and fix a dropped message?

How to fix (once you've detected that a message was dropped)?

How does sender know the message was dropped?

Extra communication needed

Is the sender's conclusion always correct?

Unreliable → reliable messages

How to deal with duplicate messages?

How to deal with corrupted messages?

Solving problems via transformation

- Corrupted messages → dropped messages
- Potential dropped messages → potential duplicates
- Solve duplicates by adding sequence # and dropping duplicates

Byte streams

Hardware interface: Send/receive distinct (finite-sized) messages

OS abstraction: Send/receive infinite stream of bytes

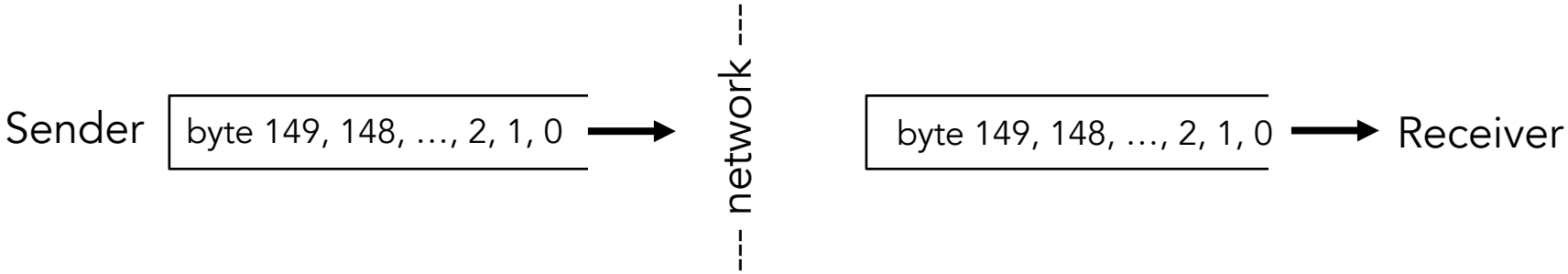
How to implement the byte stream abstraction?

- Sender:
- Receiver:

Using byte streams

TCP has no message boundaries (unlike UDP)

- Sender sends bytes; receiver receives bytes




```
size = recv(sock, buf, 150, 0);
```

- Blocks until can receive > 0 bytes (or error)
- May return [1, 150] bytes
- To receive > 1 byte, must call recv in loop
  - or use MSG_WAITALL to receive a known number of bytes

How to send message over byte streams?

How to know # of bytes to receive?

- Convention (size specified by protocol)
- Size specified in header
- End-of-message delimiter
- Sender closes connection

Project 4

Read manual pages

Understand C strings

- What are you passing when you call `fs_writeblock(…, "hello world", …)` ?

What data should be a C string, and what data need not be a C string?

- File data?
- `str` functions assume data is a C string. Caller must guarantee this pre-condition.

What data is trusted, and what is not trusted?

- Data sent by client?
- Initial file system image?