## Today

- IPC
  - Tradeoffs



## Review

- What are some scheduling policies?
- How should we choose between scheduling policies?
- What are ways that processes communicate?

## Review: Scheduling Policies

- FCFS, RR, SFJ, MFQ, CFS
- Considerations
  - Preemptive or non-preemptive
  - Workload – CPU-bound or I/O-bound, job sizes
  - Your system goals
    - Throughput
    - Latency
    - Fairness
    - CPU Utilization
    - Other: Meet deadlines, energy/battery usage, …



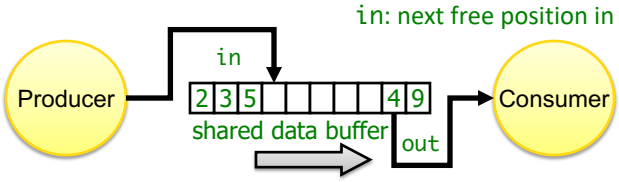
## Review: Interprocess Communication: Pipeline


Example: `$ ls | sort | grep py`

Unix philosophy: Do one thing, and do it well. (Modularity)
Result: lots of small utilities chained together at the command line.

## Review: Interprocess Communication: Shared Memory

One implementation: Circular buffer
`out`: first full position in the buffer
`in`: next free position in the buffer




- Processes use system calls to add shared memory to their address spaces

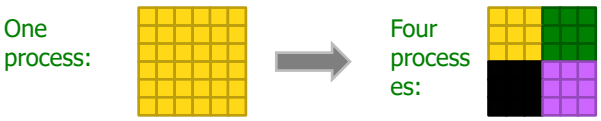
Each side can perform I/O and block independently of the other.



## Interprocess Communication Model

Divide and Conquer model

- Data parallel tasks




Example: Large scientific computing tasks

- Weather / earthquake / fluid dynamics simulations

Each process can execute concurrently.

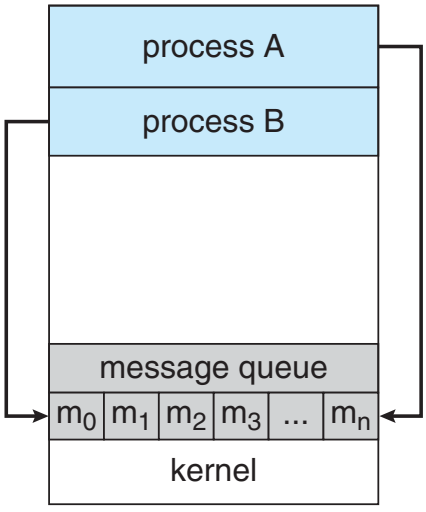
# Interprocess Communication: Message Passing

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - send(message)
  - receive(message)
- The message size is either fixed or variable



# IPC via Message Passing

1. Process A creates a message M.
2. Process A uses a *send* system call to send the message to process B.
3. The message is copied into memory in the kernel's address space.
4. Process B uses a *receive* system call to retrieve the message from A.
5. The message is copied into B's address space.



## Message Passing Interface (MPI)

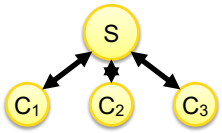
- Typically used in scientific computing tasks/super computers
- Each process is given a numeric rank ID by the MPI library
- Can send/recv by specifying rank of recipient
- Can also broadcast, reduce values, add barriers, and many other functions

Details of data transfer and synchronization vary according to MPI implementation, available hardware, and configuration.



## Interprocess Communication Model

Client / Server model:



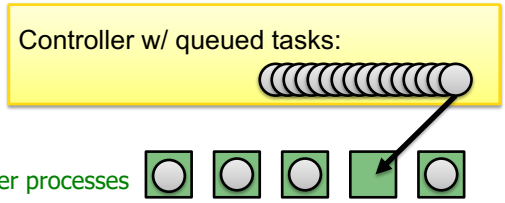

Example: Most Internet services

- Server: wait for clients to request service, satisfy requests when they do
- Client: connect to server, make request(s), disconnect when done

Each party (client and server) is specialized. (Modularity)
Server can do work for some clients while others are idle. (Independent I/O)

# Interprocess Communication Model

Work Queue model:




Examples:

- Folding@home: distributed biochemistry research
- Internet server (web, email, etc.): tasks are client requests

Each worker can perform I/O and block independently of the other. Each worker can fail independently without stopping the system.



# Interprocess Communication: Sockets

- Message passing across machines
- Connect to a machine and specific port
  - Well-known ports?
- Use some *protocol* for communication

## Sockets

- Connect to a machine and specific port
  - Well-known ports?
    - 22: ssh
    - 80: http
    - …
- Use some *protocol* for communication
  - HTTP
  - FTP
  - SOAP
  - …



## Sockets

- Abstraction: `socket()` descriptor
  - write data to one side of socket
  - read it at other
- Low-level data transfer interface
  - `send(fd, …)` and `recv(fd, …)`
- Most commonly over TCP
  - one side `listen()`s for incoming connection
  - other side `connect()`s to listener

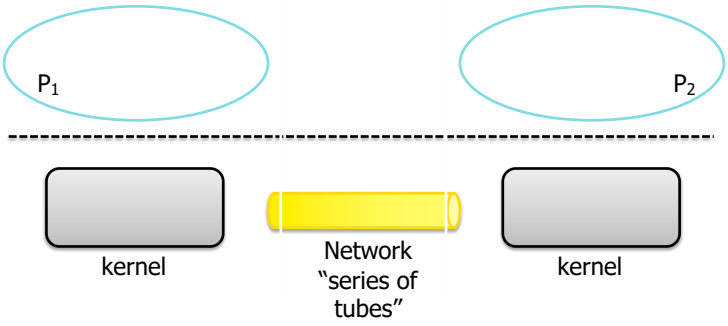
# Buffering



Network
"series of
tubes"

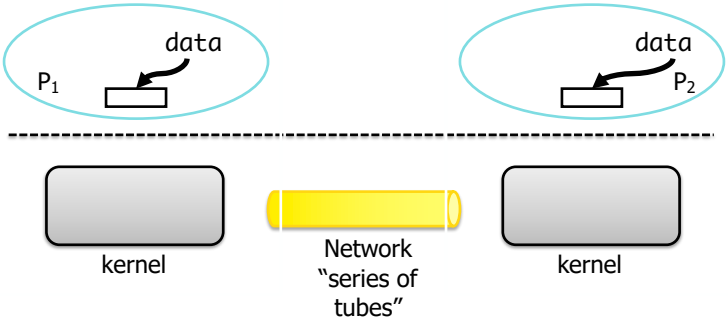


# Buffering



# Buffering

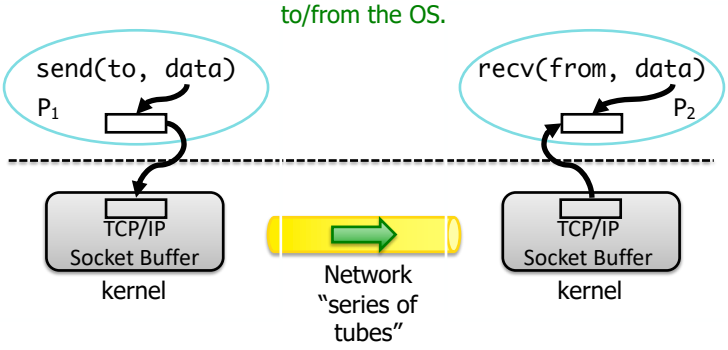
P1 wants to send data to P2.






# Buffering

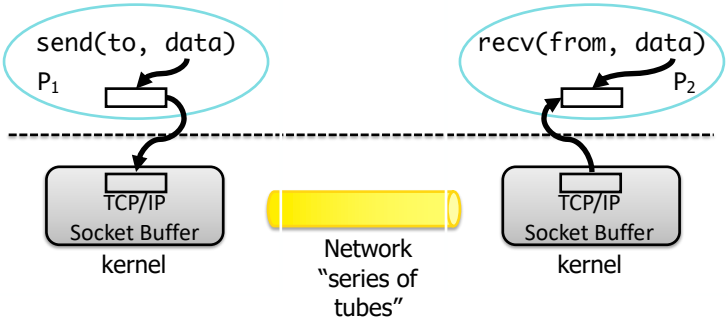
All send() and recv() do is copy data to/from the OS.




OS will format data and send it using the network device when it can.

# Buffering

Kernel's buffers have finite storage space!




If sender fills buffer, OS will mark the process as blocked – can't be scheduled until space is free.

If the buffer is empty, OS will mark the receiver process as blocked – can't read data before it arrives!



# Discussion: IPC Mechanism Tradeoffs

- Recap: What are the IPC Mechanisms we discussed?
  - What are the goals of the mechanisms?
- What are some of the relative advantages and disadvantages of these approaches to IPC?
  - What (qualitative or quantitative) measures should we use to measure "goodness"?
    - What concerns about communication do we have?
  - How do we choose which mechanism to use?

## Tradeoffs Summary

- Communication overhead
  - Higher with message passing, sockets
- Amount of cooperation/collaboration
  - Higher with shared memory, pipes
- Amount of protection
  - Higher with sockets → no direct access



## Looking Ahead

- Project 2 – due Monday
- Threads!