Messages

Messages for Communication & Concurrency Control

- IPC
- Our Message Protocol
- Mutexes & Signaling

Messages

IPC

IPC

- **Interprocess communication** (IPC) refers to any of a number of systems which allow processes to talk to each other
  - Messages/mailboxes
  - Signals
  - Pipes, files
  - Semaphores
  - Shared memory

IPC

- IPC usually denotes communication between processes on the same computer

- However, most forms of IPC are also useful for **distributed processes**
  - Processes run on different computers
  - Connected through network
  - Sockets or messaging most common

IPC

- **Mailboxes** allow processes to send **messages** to other processes
  - Atomic messages, small payload
  - Buffering (often)
  - Blocking (often)
  - Structured vs. raw buffers?
  - Non-blocking versions?

IPC

- A **signal** is a software interrupt, sent from one process to another
  - OS must deliver, forces receiver into a **signal handler** function
  - Processes can block signals temporarily (same as disabling interrupts)
  - Various signals, but usually no payload
  - “Default” behavior if no handler
    - Ignore, terminate, core dump, etc.

IPC

- A **pipe** is a file-like mechanism which handles a stream of bytes, but is never stored on disk
  - `stdin,stdout,stderr`
  - `cmd1 | cmd2`
  - Usually only one reader, but many writers OK
  - Block writers when buffer fills

IPC

- **Semaphores** are integers used for counting events, block processes when insufficient resources
  - No payload
  - `P()` decrements, blocks if insufficient
  - `V()` increments, releases blocked procs
  - Useful for mutexes, resource control, buffering
  - Slide deck later!

IPC

- **Shared memory** is any time that two processes map the same writable virtual page
  - Probably map to different virtual addresses
    - No linked data structures (lists, trees)
  - Instantaneous data flow, but no way to block
  - Hard to dynamically resize

IPC

- **Non-blocking** functions allow a program to ask if something is available, but refuse to block if not
  - Mostly used for IPC & dist. communication
  - Great for handling multiple simultaneous connections
  - Often, confusing to use
  - Can hurt performance if you don't have some way to sleep (spinning or polling)

Messages

Our Messaging Protocol

Our Messaging Protocol

- In our simulation (Phase 2), we will implement mailboxes and messages

- Mailboxes are created with `MboxCreate()`, and identified by integer IDs
- Each mailbox will have two limits:
  - Max bytes per message (can be 0)
  - Max buffered messages (can be 0)

Our Messaging Protocol

- `MboxSend()` will "buffer" a message when there is no receiver ready to receive it
  - Each mailbox has a max (can be 0)
  - Sender will block if can't buffer
- Buffered messages must be delivered in order
- Blocked senders must be kept in order, too
  - Must buffer in same order they blocked
  - Beware priority problems!

Our Messaging Protocol

- We have a large pool of “message buffers”
  - Global pool, shared by all mailboxes
  - Allocate when you have a message
    - In `MboxSend()`, not `MboxCreate()`
    - Over-allocation is allowed
    - Buffering can fail if no free slots
  - Free in `MboxRecv()`

Our Messaging Protocol

- `MboxRecv()` will receive one message
  - Block if nothing buffered

- Messages must be received in order
- Blocked receivers must be kept in order
  - Beware priority problems here, too!

Our Messaging Protocol

- `MboxCondSend(), MboxCondSend()` are **non-blocking** versions
  - Send or recv if possible
  - Return special value if you would have blocked

Messages

Mutexes & Signaling

Mutexes & Signaling

- A mailbox can be used (abused?) as a mutex

- Version A:
  - Initialize with no messages
  - `Send()` to lock, `Recv()` to unlock
  - Class question: What blocks, and why?
- Version B:
  - Class question: Can you reverse it?

Mutexes & Signaling

- Soon, we'll be implementing syscalls
  - Need to block current process until something happens
- Class discussion: how to use mailboxes for sleep & wakeup
  - Nicer interface than calling Phase 1 code
  - How to do it?
  - When to allocate mailboxes?
  - What can be shared, or not?

Mutexes & Signaling

- Sometimes, we need a way to communicate that an event occurred to another process
  - Example: disk interrupt

- Class discussion: how to use mailboxes?
  - What if the receiver is busy?
  - What if nobody is listening?
  - Is it permissible to block in an interrupt handler?

Mutexes & Signaling

- Sometimes, we want to communicate a regular sequence, like a clock – perhaps to many processes

- Class discussion: how to wake up many procs?
  - What if nobody is listening?
  - Do we want to buffer or lose “missed” events?