Deadlocks, Message Passing Brief refresh from last week

Tore Larsen

Oct. 2010

Deadlocks

- Formal definition :

A set of processes is deadlocked
if each process in the set is waiting for an event
that only another process in the set can cause

- Usually the *event* is release of a currently held resource
- None of the processes can …
  - Run
  - Release resources
  - Be awakened

Four Conditions for Deadlock

1. **Mutual exclusion condition**
   - Each resource is either assigned to one process or it is available
2. **Hold and wait condition**
   - Process holding resources may request more resources
3. **No preemption condition**
   - Previously granted resources cannot be taken away by force
4. **Circular wait condition**
   - Must be at least one circular chain involving two or more processes
   - Each is waiting for resource held by next member of the chain

Deadlocks: Strategies

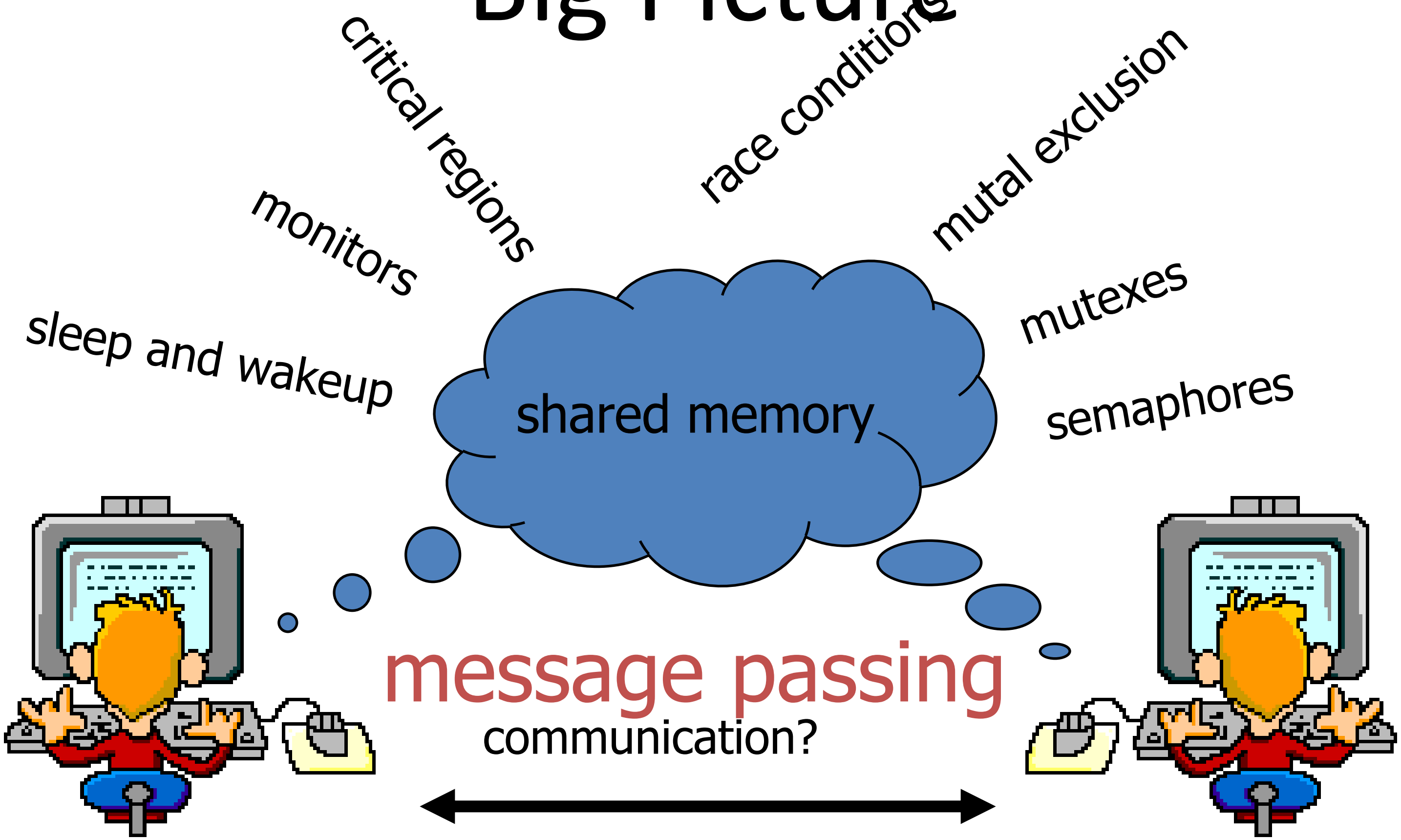
- Ignore the problem
  - It is user’s fault
- Detection and recovery
  - Fix the problem afterwards
- Dynamic avoidance
  - Careful allocation
- Prevention
  - Negate one of the four conditions

Which is your favorite?

- Ignore the problem
  - It’s the user’s fault
- Detection and recovery
  - Fix the problem afterwards
- Dynamic avoidance
  - Careful allocation
- Prevention (Negate one of four conditions)

– Avoid mutual exclusion	Spool everything
– Avoid hold and wait	Request all resources initially
– No preemption	Forcefully reclaim resources
– No circular wait	Order resources numerically

Big Picture



critical regions

race conditions

mutal exclusion

monitors

mutexes

sleep and wakeup

semaphores

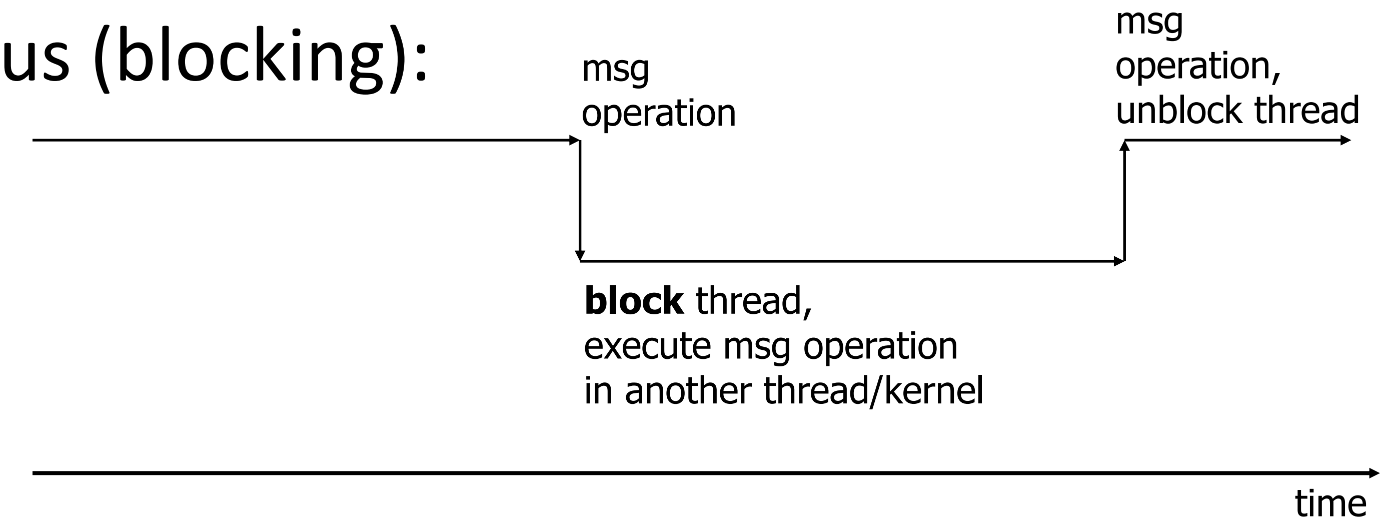
shared memory

message passing

communication?

Asynchronous vs. Synchronous

- Synchronous (blocking):



msg
operation

msg
operation,
unblock thread

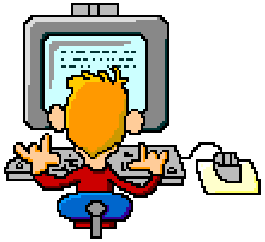
block thread,
execute msg operation
in another thread/kernel

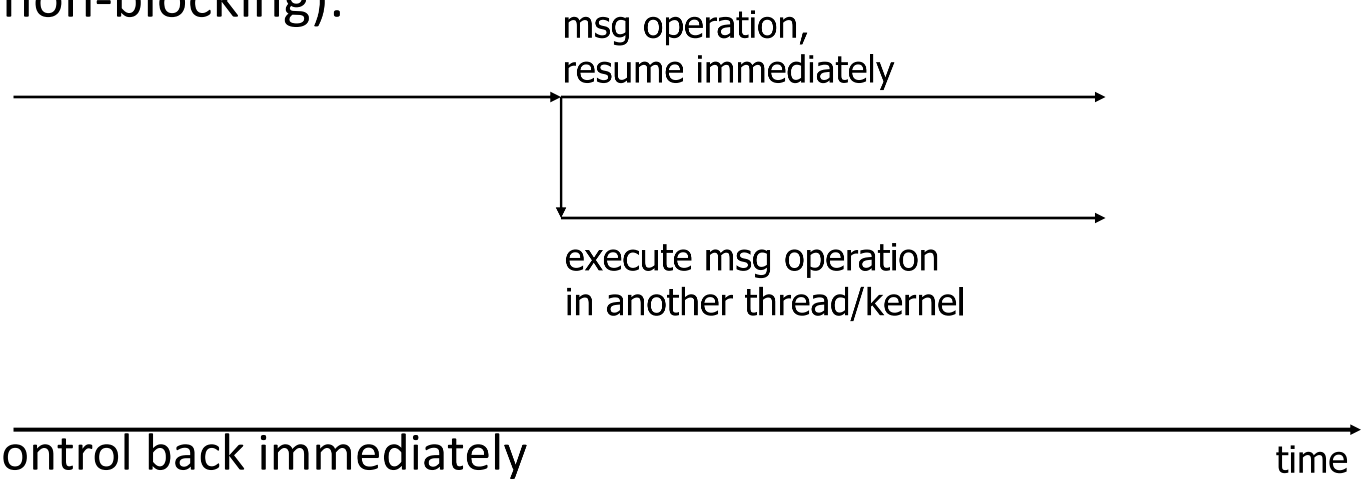
time

  - thread is blocked until message primitive has been performed
  - may be blocked for a very long time

Asynchronous vs. Synchronous

- Asynchronous (non-blocking):





  - thread gets control back immediately
  - thread can run in parallel other activities
  - thread cannot reuse buffer for message before message is received
  - how to know when to start if blocked on full/empty buffer?
    - poll
    - interrupts/signals
    - …

Asynchronous vs. Synchronous

- Send semantic:
  - Synchronous
    - Will not return until data is out of its source memory
    - Block on full buffer
  - Asynchronous
    - Return as soon as initiating its hardware
    - Completion
      - Require application to check status
      - Notify or signal the application
    - Block on full buffer

- Receive semantic:
  - Synchronous
    - Return data if there is a message
    - Block on empty buffer
  - Asynchronous
    - Return data if there is a message
    - Return null if there is no message

Buffering

- No buffering
  - synchronous
  - Sender must wait until the receiver receives the message
  - Rendezvous on each message

- Buffering
  - asynchronous or synchronous

  - Bounded buffer
    - Finite size
    - Sender blocks when the buffer is full
    - Use mesa-monitor to solve the problem?

  - Unbounded buffer
    - “Infinite” size
    - Sender never blocks

Direct Communication

- Must explicitly name the sender/receiver (“`dest`” and “`src`”) processes
- A buffer at the receiver
  - More than one process may send messages to the receiver
  - To receive from a specific sender, it requires searching through the whole buffer
- A buffer at each sender
  - A sender may send messages to multiple receivers

Indirect Communication

- “`dest`” and “`src`” are a shared (unique) mailbox

- Use a mailbox to allow many-to-many communication
  - Requires open/close a mailbox before using it

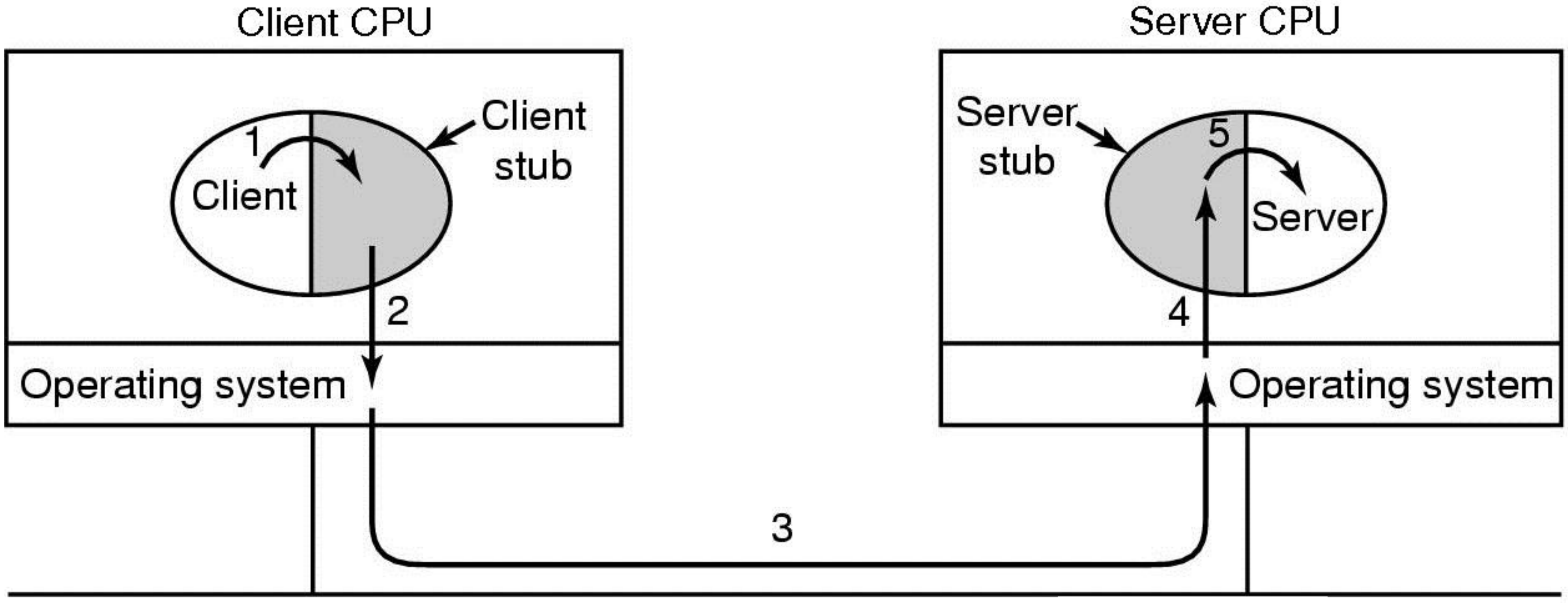
- Where should the buffer be?
  - A buffer and its mutex and conditions should be at the mailbox

Linux: Mailboxes vs. Pipes

- Are there any differences between a mailbox and a pipe?
  - Message types
    - mailboxes may have messages of different types
    - pipes do not have different types
  - Buffer
    - pipes – one or more pages storing messages contiguously
    - mailboxes – linked list of messages of different types
  - Termination
    - pipes exists only as long as some have open the file descriptors
    - mailboxes must often be closed
  - More than two processes
    - a pipe **often** (not in Linux) implies one sender and one receiver
    - many can use a mailbox

Remote Procedure Call

- Message passing uses I/O
- Idea of RPC is to make function calls
- Small libraries (stubs) and OS take care of communication



Publish – Subscribe

Subject for later course

- Decoupled
- Asynchronous
- Anonymous
- Filtering