

COS 318: Operating Systems

Message Passing

Kai Li
Computer Science Department
Princeton University

(<http://www.cs.princeton.edu/courses/cos318/>)

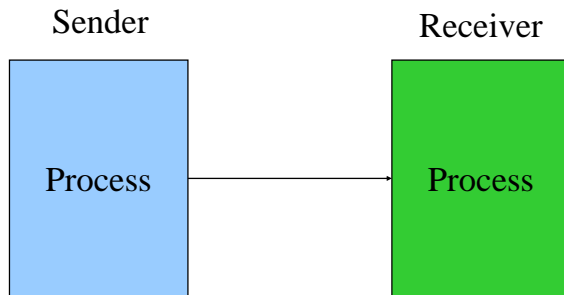


Today's Topics

- ◆ Message passing
 - Semantics
 - How to use
- ◆ Implementation issues
 - Synchronous vs. asynchronous
 - Buffering
 - Indirection
 - Exceptions



Big Picture



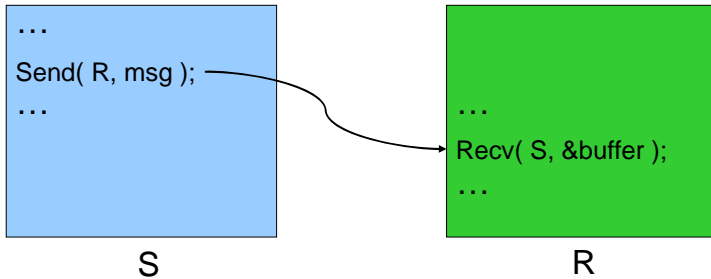
Message Passing API

- ◆ Generic API
 - `send(dest, msg)`
 - `recv(src, msg)`
- ◆ What should the “dest” be?
 - pid
 - file: e.g. a pipe
 - port: network address, pid, etc
- ◆ What should the “src” be?
 - Same as above
 - no src: receive any message
 - src combines both specific and any
- ◆ What should “msg” be?
 - Need buffer and size for a variable sized message



Using Message Passing (Synchronous)

- ◆ Move data between processes
 - Sender: when data is ready, send it to the receiver process
 - Receiver: when the data has arrived and when the receive process is ready to take the data, move the data
- ◆ Synchronization
 - Sender: signal the receiver process that a particular event happens
 - Receiver: return when the event has happened



Example: Producer-Consumer

```

Producer(){
    ...
    while (1) {
        produce item;
        recv(Consumer, &credit);
        send(Consumer, item);
    }
}

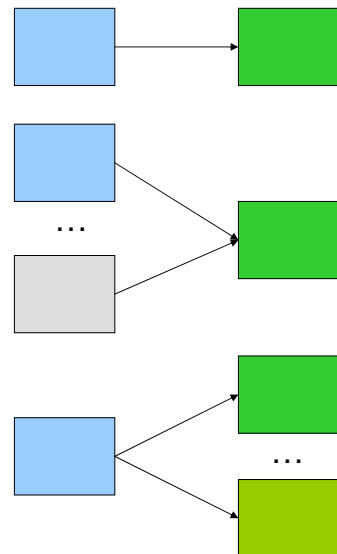
Consumer(){
    ...
    for (i=0; i<N; i++)
        send(Producer, credit);
    while (1) {
        recv(Producer, &item);
        send(Producer, credit);
        consume item;
    }
}
    
```

- ◆ Questions
 - Does this work?
 - Would it work with multiple producers and 1 consumer?
 - Would it work with 1 producer and multiple consumers?
 - What about multiple producers and multiple consumers?



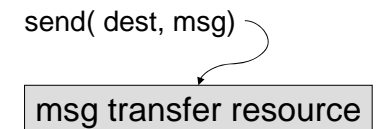
Implementation Issues

- ◆ Asynchronous vs. synchronous
- ◆ Event handler vs. receive
- ◆ How to buffer messages?
- ◆ Direct vs. indirect
- ◆ Matching options
 - 1-to-1
 - 1-to-many
 - many-to-one
 - many-to-many
- ◆ Unidirectional vs. bidirectional
- ◆ What is the size of a message?
- ◆ How to handle exceptions (when bad things happen)?



Synchronous vs. Asynchronous: Send

- ◆ Synchronous
 - Block on if resource is busy
 - Initiate data transfer
 - Block until data is out of its source memory
- ◆ Asynchronous
 - Block if resource is busy
 - Initiate data transfer and return
 - Completion
 - Require applications to check status
 - Notify or signal the application



```

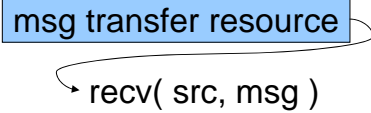
status = async_send( dest, msg )
...
if !send_complete( status )
    wait for completion;
...
use msg data structure;
...
    
```



Synchronous vs. Asynchronous: Receive

◆ Synchronous

- Return data if there is a message



◆ Asynchronous

- Return data if there is a message
- Return status if there is no message (probe)

```
status = async_recv( src, msg );
if ( status == SUCCESS )
    consume msg;

while ( probe(src) != HaveMSG )
    wait for msg arrival
recv( src, msg );
consume msg;
```



Event Handler vs. Receive

◆ hrecv(src, msg, func)

- msg is an arg of func
- Execute "func" on a message arrival

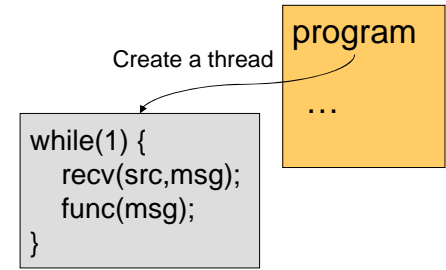
◆ Which one is more powerful?

- Recv with a thread can emulate a Handler
- Handler can be used to emulate recv by using Monitor

◆ Pros and Cons

```
void func( char * msg ) {
    ...
}

...
hrecv( src, msg, func)
...
```



Buffering

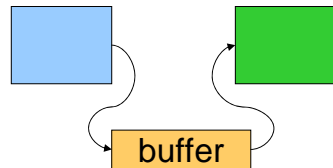
◆ No buffering

- Sender must wait until the receiver receives the message
- Rendezvous on each message



◆ Bounded buffer

- Finite size
- Sender blocks on buffer full
- Use mesa-monitor to solve the problem



◆ Unbounded buffer

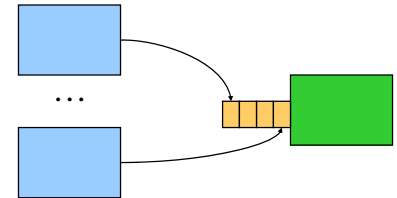
- "Infinite" size
- Sender never blocks



Direct Communication

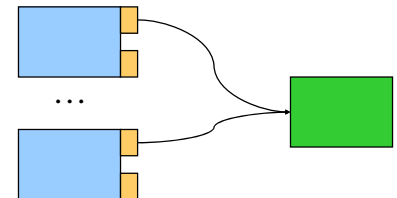
◆ A single buffer at the receiver

- More than one process may send messages to the receiver
- To receive from a specific sender, it requires searching through the whole buffer



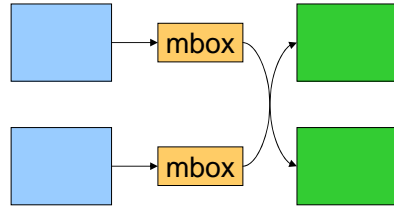
◆ A buffer at each sender

- A sender may send messages to multiple receivers
- To get a message, it also requires searching through the whole buffer



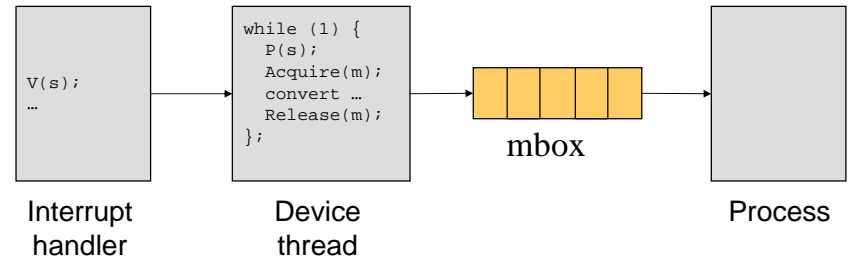
Indirect Communication

- ◆ Use a "mailbox" to allow many-to-many communication
 - Requires open/close a mailbox before using it
- ◆ Where should the buffer be?
 - A buffer, its mutex and condition variables should be at the mailbox
- ◆ Fixed sized messages?
 - Not necessarily. One can break a large message into packets
- ◆ Are there any differences between a mailbox and a pipe?
 - A mailbox allows many to many communication
 - A pipe implies one sender and one receiver



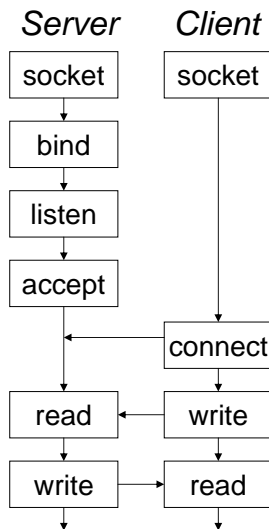
Example: Keyboard Input

- ◆ How do you implement keyboard input?
 - Need an interrupt handler
 - Generate a mbox message from the interrupt handler
- ◆ Suppose a keyboard device thread converts input characters into an mbox message
 - How would you synchronize between the keyboard interrupt handler and device thread?
 - How can a device thread convert input into mbox messages?



Example: Sockets API

- ◆ Abstraction for TCP and UDP
 - Learn more about Internetworking from a later lecture
- ◆ Addressing
 - IP address and port number (2^{16} ports available for users)
- ◆ Create and close a socket
 - `sockid = socket(af, type, protocol);`
 - `Sockerr = close(sockid);`
- ◆ Bind a socket to a local address
 - `sockerr = bind(sockid, localaddr, addrlen);`
- ◆ Negotiate the connection
 - `listen(sockid, length);`
 - `accept(sockid, addr, length);`
- ◆ Connect a socket to destination
 - `connect(sockid, destaddr, addrlen);`



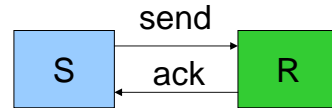
Exception: Process Termination

- ◆ R waits for a message from S, but S has terminated
 - Problem: R may be blocked forever
 - Solutions?
- ◆ S sends a message to R, but R has terminated
 - Problem: S has no buffer and will be blocked forever
 - Solutions?



Exception: Losing Messages

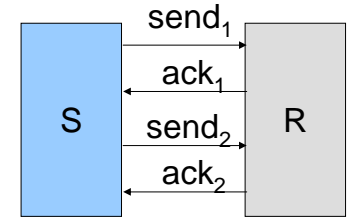
- ◆ Use ack and timeout to detect and retransmit a lost message
 - Require the receiver to send an ack message for each message
 - Sender blocks until an ack message is back or timeout
`status = send(dest, msg, timeout);`
 - If timeout happens and no ack, then retransmit the message
- ◆ Issues
 - Duplicates
 - Losing ack messages



17

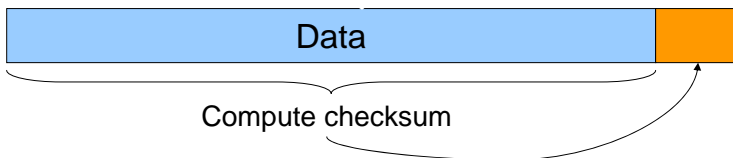
Exception: Losing Message, cont'd

- ◆ Retransmission must handle
 - Duplicate messages on receiver side
 - Out-of-sequence ack messages on sender side
- ◆ Retransmission
 - Use sequence number for each message to identify duplicates
 - Remove duplicates on receiver side
 - Sender retransmits on an out-of-sequence ack
- ◆ Reduce ack messages
 - Bundle ack messages
 - Receiver sends noack messages: can be complex
 - Piggy-back acks in send messages



18

Exception: Message Corruption



- ◆ Detection
 - Compute a checksum over the entire message and send the checksum (e.g. CRC code) as part of the message
 - Recompute a checksum on receive and compare with the checksum in the message
- ◆ Correction
 - Trigger retransmission
 - Use correction codes to recover

19

Summary

- ◆ Message passing
 - Move data between processes
 - Implicit synchronization
- ◆ Implementation issues
 - Synchronous method is most common
 - Asynchronous method provides overlapping but requires careful design considerations
 - Indirection makes implementation flexible
 - Exception needs to be carefully handled

20