Operating Systems – IPC: Inter-Process Communication

Message Passing
Semaphores & Monitors

• Monitors are high-level programming language concepts
  – Make mutual exclusion of critical section “automatic” and therefore less error-prone
  – Require compiler support
  – Not widely available in modern languages (e.g., avail. In Modula 3)
  – Available as an “approximation” in Java

• Semaphores are more lower-level
  – Require OS support
  – Applications can be layered on top of them, i.e., no compiler support required

• Both concepts require some kind of shared memory

• How about synchronization problems in a distributed system (no shared memory)?
Conclusion About Monitors and Semaphores

• Monitors are high-level, but not widely supported; therefore not too useful in practise
• Semaphores are rather low-level
• None of the mechanisms discussed so far allow for information exchange between distributed processes

So we need something else!
Message Passing

- An **inter-process communication** mechanism
- Based on two primitives
  - `send (destination, &message)`
    - Sends a message to a destination
  - `receive (source, &message)`
    - Receives a message from a source (any source)
- **System calls**, not language constructs
- **Blocking** versions and **non-blocking** versions are available
Message Passing

send(…)  receive(…)  

- Naming of sender / receiver
- Number of interacting entities
- Reliability of link
- Capacity of link
Design Issues

Various design concerns not applicable in semaphore & monitor discussion:

- Networked system, i.e., messages maybe lost (network is unreliable)
- Sender/receiver agree to use acknowledgements and retransmission timers
- Message received, but acknowledgement lost
  - Receiver will get the same message twice
  - Use consecutive sequence numbers
    - Computer Networks
Design Issues

- **Naming** of processes for **unambiguous** send/receive
- **Authentication**, how can client know it is communicating with proper server
- **Tune for performance**, if sender & receiver are on the same machine (semaphore is usually cheaper)
Mutual Exclusion Based on Message Passing

1. Request to use resource
2. Resource free
3. Use of resource completed
Mutual Exclusion Based on Message Passing Code

P1:

```plaintext
send(Resource Contro, "Give me resource");
receive(Resource Controller, Message);
// block while waiting for a reply from RC
{
    ...Critical Section...
}
send(Resource Controller, "Done with resource");
```
Bounded Buffer Problem
(no shared memory)
void consumer(void)

int item;
message m;

// initialization loop
for (i=0; i<N; i++) send(producer, &m);  // N empty slots

while (TRUE) {
    receive(producer, &m);  // receive item
    item=extract_item(&m);
    send(producer, &m);      // send empty slot
    consume_item(item);
}
Consumer Initialization

Represents the empty slots.
void producer(void)

int item;
message m;
while (TRUE) {
    item Produce_Item();
    receive(consumer, &m);  // empty slot
    build_message(&m, item);
    send(consumer, &m);      // send item
}
Producer-Consumer

consumer

... 

producer

// N empty messages (i.e., slots)

- Consumer sends N empty messages (i.e., slots)
- Producer picks up an empty message/slot and send back an item produced
- Consumer receives an item and sends back an empty message (i.e., a slot)
- Messages sent, but not received are buffered by OS
Characteristics
Naming: Direct Communication

• Sender/receiver refer to each other, as seen before
• Properties of communication link
  – Link is *established automatically* between communicating processes
  – Link is associated with *exactly two processes*
  – **Exactly one link** for every pair of processes
• Communication is *symmetric* (above) or *asymmetric*
  – `send(P, m)` // send a message to P
  – `receive(&id, m)` // receive from any process, set id to sender
Direct Naming

send(j,m)  receive(i,m)

send(j,m)

receive(&from,m)

ECE 344 Operating Systems
Naming: Indirect Communication

• Communication via mailboxes (or ports)
• Processes communicate by putting and taking messages in/from mailboxes
  – \texttt{send(A, m)} and \texttt{receive(A,m)}
• Properties of communication link
  – A link is established between two processes, if they share a mailbox
  – Link maybe associated with more than two processes
  – A number of different links may exist between any pair of processes; each one a separate mailbox
Indirect Naming

Mailbox

Mailbox

Mailbox

Mailbox

ECE 344 Operating Systems
Mailboxes

Who should get the message?

- At most one process may execute a receive operation at a time
- The system arbitrarily selects one as the receiver
- Message is retrieved by both (a form of multicast)

Ownership
- Either process can own the mailbox
- OS can own it and/or creator owns it

ECE 344 Operating Systems
Sender/Receiver Synchronization

- Message passing maybe **blocking** or **non-blocking** (synchronous or asynchronous)
- **Blocking send** – sender blocked until message is received by receiver (or by mailbox)
- **Non-blocking send** – sending process resumes operation right after sending
- **Blocking receive** – receiver blocks until message is available
- **Non-blocking receive** – receiver retrieves a valid message or returns an error code
- Any combination of the above send/receive is possible
Blocking vs. non-blocking send

- blocking send (blocking)
- non-blocking send (non-blocking)

ECE 344 Operating Systems
Buffering

• The mechanism that buffers messages (a.k.a. queue) may have the following properties
  – **Zero capacity**: queue has length 0, no messages can be outstanding on link, sender blocks for message exchange
  – **Bounded capacity**: queue has length $N$, $N$ messages can be in queue at any point in time, sender blocks if queue is full, otherwise it may continue to execute
  – **Unbounded capacity**: queue has infinite length, sender never blocks
Message Passing in Practice

- Message Passing Interface (MPI) used in parallel programming systems
- Open MPI, a standard
- Unix’s IPC – Inter-process communication facilities
  - Semaphores
  - Pipes, FIFO
  - Message queues
  - Sockets
  - Shared memory, memory mapped files
  - …
Characteristics of Message Passing

• Addressing
  – Each process has a unique name
  – Introduce the concept of a mailbox
    • With sender, with receiver, with both, with OS

• Buffering
  – No buffering, send blocks until receive happens or vice versa, then copy message (called a rendezvous)
Copy-on-write

• Message passing for process-to-process communication can be quite inefficient due to context switching etc.

• Copy-on-write first sets a pointer in receivers address space pointing to received data in sender’s address space

• Only if sender / receiver start writing this data will the OS copy the actual data from sender to receiver

• Otherwise, no copying of the data is performed