LSN 12
Interprocess Communications

ECT362 Operating Systems
LSN 12 – Message Passing

- Utilize send/receive primitives
  - send(destination, message)
  - receive(source, message)

- Synchronization
  - Send
    - The sending process may be blocked until receipt of message occurs
  - Receive
    - Receives previously sent message
LSN 12 – Message Passing

• Common scenarios
  – Blocking send and receive
  – Non-blocking send with a blocking receive
  – Non-blocking send with a non-blocking receive

• Non-blocking send is most useful for concurrent programming
  – Susceptible to resource competition from erroneous messages

• Blocking receive is most useful for concurrent programming
  – Requested message is usually necessary to continue execution
LSN 12 – Message Addressing

• Direct
  – Send includes specific identifier of destination process

Explicit — Receive includes specific process ID to receive from
  or

Implicit — Receive includes an implicit address in which the source
  parameter of the receive primitive possesses a value
  returned when the receive operation has been performed

• Indirect
  – Messages are sent to a shared data structure consisting of
    queues that can temporarily hold messages (mailboxes)
  – Relationship between sender/receiver can be one-to-one,
    one-to-many, many-to-one, or many-to-many
LSN 12 – Message Format

- Can be fixed or variable length
LSN 12 – Mutual Exclusion Using Messages

- Blocking receive
- Non-blocking send
- Set of concurrent processes share a mailbox
  - Used by all processes to send and receive
- Mailbox is initialized with a single NULL message
- A process wishing to enter its critical section first attempts to receive a message
  - If mailbox is empty, the process is blocked
  - When the process has acquired the message, it performs its critical section and then places the message back into the mailbox
/* program mutualexclusion */
const int n = /* number of processes */;
void P(int i)
{
    message msg;
    while (true)
    {
        receive (box, msg);
        /* critical section */;
        send (box, msg);
        /* remainder */;
    }
}
void main()
{
    create_mailbox (box);
    send (box, null);
    parbegin (P(1), P(2), . . . , P(n));
}
LSN 12 – The Pipe Model

- Utilizes a FIFO buffer in the kernel memory space
- Has a read and a write end in user space
  - `read()` to extract data from buffer
  - `write()` to insert data into buffer
- Anonymous pipes
  - Child process automatically inherits parent process pipe-ends
- Named pipes
  - Process obtains a pipe-end by using a string that is analogous to a file name
LSN 12 – The Pipe Model

Address Space for $p_1$

- Info to be shared

System Call Interface

Pipe for $p_1$ and $p_2$

write function

Address Space for $p_2$

- Info copy

read function
LSN 12 – Homework

- Reading
  - Chapter 5.5