Interprocess Communication

- Processes within a system may be independent or cooperating.
- Cooperating process can affect or be affected by other processes, including sharing data.
- Reasons for cooperating processes (instead of single process):
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need interprocess communication (IPC).
- Two models of IPC:
  - Shared memory
  - Message passing

Communications Models
Producer-Consumer Problem

- Paradigm for cooperating processes, producer process produces information (repeatedly) that is consumed by a consumer process
  - unbounded-buffer places no practical limit on the size of the buffer
  - bounded-buffer assumes that there is a fixed buffer size

- How can we implement a producer and consumer using shared memory?

Bounded-Buffer – Shared-Memory Solution

- Shared data
  ```
  #define BUFFER_SIZE 10
  typedef struct {
      item[ BUFFER_SIZE ];
  } item;
  int produced = 0;
  int consumed = 0;
  ```

- How to ensure that producer and consumer don’t overwrite each other’s updates?
  - Following solution is correct, but can only have BUFFER_SIZE-1 elements waiting to be consumed

Bounded-Buffer – Producer

```
Producer:
while (true) {
    /* do nothing -- no free buffers */
    while (produced - consumed == BUFFER_SIZE) {} 
    buffer[produced % BUFFER_SIZE] = produceItem();
    produced++;
}
```

Bounded-Buffer – Consumer

```
Consumer:
while (true) {
    while (produced - consumed == 0) {} 
    consumeItem(buffer[consumed % BUFFER_SIZE]);
    consumed++;
}
```
Interprocess Communication – Message Passing

• Mechanism for processes to communicate and synchronize actions
• Message system – processes communicate with each other without resorting to shared variables
• IPC facility provides two operations:
  • send (message) – message size fixed or variable
  • receive (message)
• If P and Q wish to communicate, they need to:
  • establish a communication link between them
  • exchange messages via send/receive
• Implementation of communication link
  • physical (e.g., shared memory, hardware bus)
  • logical (e.g., logical properties)

Implementation Questions

• How are links established?
• Can a link be associated with more than two processes?
• How many links can there be between every pair of communicating processes?
• What is the capacity of a link?
• Is the size of a message that the link can accommodate fixed or variable?
• Is a link unidirectional or bi-directional?

Direct Communication

• Processes must name each other explicitly:
  • send (P, message) – send a message to process P
  • receive (Q, message) – receive a message from process Q
• Properties of communication link
  • Links are established automatically
  • A link is associated with exactly one pair of communicating processes
  • Between each pair there exists exactly one link
  • The link may be unidirectional, but is usually bi-directional
Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports)
  - Each mailbox has a unique id
  - Processes can communicate only if they share a mailbox

- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional

Indirect Communication

- Operations
  - create a new mailbox
  - send and receive messages through mailbox
  - destroy a mailbox

- Primitives are defined as:
  - send(A, message) – send a message to mailbox A
  - receive(A, message) – receive a message from mailbox A

Indirect Communication

- Mailbox sharing
  - \( P_1, P_2, \) and \( P_3 \) share mailbox A
  - \( P_1 \) sends; \( P_2 \) and \( P_3 \) receive
  - Who gets the message?

- Solutions
  - Allow a link to be associated with at most two processes
  - Allow only one process at a time to execute a receive operation
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.
Synchronization

- Message passing may be either blocking or non-blocking

  - **Blocking** is considered **synchronous**
    - **Blocking send** has the sender block until the message is received
    - **Blocking receive** has the receiver block until a message is available

  - **Non-blocking** is considered **asynchronous**
    - **Non-blocking send** has the sender send the message and continue
    - **Non-blocking receive** has the receiver receive a valid message or null

Buffering

- Queue of messages attached to the link; implemented in one of three ways
  1. Zero capacity – 0 messages
     - Sender must wait for receiver (rendezvous)
  2. Bounded capacity – finite length of \( n \) messages
     - Sender must wait if link full
  3. Unbounded capacity – infinite length
     - Sender never waits

Sockets

- **A socket** is defined as an endpoint for communication

  - Concatenation of IP address and port
  - The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
  - Communication consists between a pair of sockets
  - Will talk more about the network later in the course
Pipes

- Acts as a conduit allowing two processes to communicate

- **Issues**
  - Is communication unidirectional or bidirectional?
  - In the case of two-way communication, is it half or full-duplex?
  - Must there exist a relationship (i.e. parent-child) between the communicating processes?
  - Can the pipes be used over a network?

Ordinary Pipes

- **Ordinary Pipes** allow communication in standard producer-consumer style

  - Producer writes to one end (the write-end of the pipe)
  - Consumer reads from the other end (the read-end of the pipe)
  
  Ordinary pipes are therefore unidirectional

  - Require parent-child relationship between communicating processes
Ordinary Pipes

Named Pipes

- Named Pipes are more powerful than ordinary pipes
- Communication is bidirectional
- No parent-child relationship is necessary between the communicating processes
- Several processes can use the named pipe for communication
- Provided on both UNIX and Windows systems