InterProcess Communication

Inter-Process Communication

- A process has access to the memory which constitutes its own address space.
- When a child process is created, the only way to communicate between a parent and a child process is:
  - The variables are replicas
  - The parent receives the exit status of the child
- So far, we've discussed communication mechanisms only during process creation/termination
- Processes may need to communicate during their life time.

Cooperating Processes

- Independent process cannot affect or be affected by the execution of another process.
- Cooperating process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up:
    - make use of multiple processing elements
  - Modularity
  - Convenience:
    - editing, printing, compiling in parallel
- Dangers of process cooperation
  - Data corruption, deadlocks, increased complexity
  - Requires processes to synchronize their processing

Purposes for IPC

- IPC allows processes to communicate and synchronize their actions without sharing the same address space
  - Data Transfer
  - Sharing Data
  - Event notification
  - Resource Sharing and Synchronization

IPC Mechanisms

- Mechanisms used for communication and synchronization
  - Message Passing
    - message passing interfaces, mailboxes and message queues
    - sockets, pipes
  - Shared Memory: Non-message passing systems
  - Synchronization – primitives such as semaphores to higher level mechanisms such as monitors
  - Event Notification - UNIX signals
- We will defer a detailed discussion of synchronization mechanisms and concurrency until a later class
- Here we want to focus on some common (and fundamental) IPC mechanisms

Message Passing

- In a Message system there are no shared variables.
- IPC facility provides two operations for fixed or variable sized message:
  - send(message)
  - receive(message)
- If processes P and Q wish to communicate, they need to:
  - establish a communication link
  - exchange messages via send and receive
- Implementation of communication link
  - physical (e.g., memory, network etc)
  - logical (e.g., syntax and semantics, abstractions)

Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How are links made known to 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?

Message Passing Systems

- Exchange messages over a communication link
- Methods for implementing the communication link and primitives (*send*/*receive*):
  - Direct or Indirect communications (*Naming*)
  - Symmetric or Asymmetric communications (blocking versus non-blocking)
  - Buffering
  - Send-by-copy or send-by-reference
  - fixed or variable sized messages

Direct Communication – Internet and Sockets

- Processes must name each other explicitly:
  - Symmetric Addressing
    - send (**P**, message) – send to process P
    - receive(**Q**, message) – receive from Q
  - Asymmetric Addressing
    - send (**P**, message) – send to process P
    - receive(**id**, message) – rx from any; system sets id = sender
- Properties of communication link
  - Links established automatically between pairs
  - processes must know each others ID
  - Exactly one link per pair of communicating processes
- Disadvantage: a process must know the name or ID of the process(es) it wishes to communicate with

Indirect Communication

- Messages are sent to or received from mailboxes (also referred to as ports).
  - Each mailbox has a unique id.
  - Processes can communicate only if they share a mailbox.
- Primitives:
  - send(A, message) – send a message to mailbox A
  - receive(A, message) – receive a message from mailbox A
- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with more than 2 processes.
  - Each pair of processes may share several communication links.

Indirect Communication-Ownership

- process owns (i.e. mailbox is implemented in user space):
  - only the owner may receive messages through this mailbox.
  - Other processes may only send.
  - When process terminates any “owned” mailboxes are destroyed.
- kernel owns
  - then mechanisms provided to create, delete, send and receive through mailboxes.
  - Process that creates mailbox owns it (and so may receive through it)
  - but may transfer ownership to another process.

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.
  - OR allow only one process at a time to execute a receive operation.
  - OR 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 send:
    - sender blocked until message received by mailbox or process
  - nonblocking send:
    - sender resumes operation immediately after sending
  - blocking receive:
    - receiver blocks until a message is available
  - nonblocking receive:
    - receiver returns immediately with either a valid or null message.

Buffering

- All messaging system require framework to temporarily buffer messages.
- These queues are implemented in one of three ways:
  - Zero capacity
    - No messages may be queued within the link, requires sender to block until receiver retrieves message.
  - Bounded capacity
    - Link has finite number of message buffers. If no buffers are available then sender must block until one is freed up.
  - Unbounded capacity
    - Link has unlimited buffer space, consequently send never needs to block.