Interprocess communication

- Shared Memory
- Message Passing
  - Signals
Shard Memory

Process 1  Process 2  Process 3

Shared memory
Shared Memory in Solaris

- Processes can share the same segment of memory directly when it is mapped into the address space of each sharing process
- Faster communication
- System calls:
  - `int shmget(key_t key, size_t size, int shmflg)`: creates a new region of shared memory or returns an existing one
  - `void *shmat(int shmid, const void *shmaddr, int shmflg)`: attaches a shared memory region to the virtual address space of the process
  - `int shmdt(char *shmaddr)`: detaches a shared region
- Mutual exclusion must be provided by processes using the shared memory
Message Passing
Design Attributes

- **Naming**
  - Process id, mailbox

- **Buffering**
  - Size: zero, bounded, unbounded
  - Place: kernel space, user space

- **Send operation**
  - Synchronous vs. asynchronous

- **Receive operation**
  - Blocking vs. non-blocking
Interprocess Communication

Message Passing

Many possible naming schemes. One is direct naming:

\[
\begin{align*}
\text{send}(\text{process}\_\text{id}, \text{message}) \\
\text{receive}(\text{process}\_\text{id}, \text{buffer})
\end{align*}
\]

Example

process P1:
\[
\begin{align*}
declare x \text{ integer} \\
\text{send}(P2, x)
\end{align*}
\]
end process

process P2:
\[
\begin{align*}
declare y \text{ integer} \\
\text{receive}(P1, y)
\end{align*}
\]
end process

Effect of this communication is

\[
y := x
\]

local var of P2 \hspace{1cm} \text{local var of P1}
Buffering

- A buffer, with bounded-buffer synchronization, can be associated with each pair of communicating processes.
- A “zero-capacity” buffer means processes must “handshake” in order to communicate.

A buffer can reside in memory of receiving process or in OS address space.

Examples:

- **no buffer needed**
  
  \[
  \begin{align*}
  P1: & \text{ send}(P2, x) \quad P2: \text{ receive}(P1, x) \\
  & \text{ receive}(P2, y) \quad \text{ send}(P1, y)
  \end{align*}
  \]

- **buffer needed**
  
  \[
  \begin{align*}
  P1: & \text{ send}(P2, x) \quad P2: \text{ send}(P1, x) \\
  & \text{ receive}(P2, y) \quad \text{ receive}(P1, y)
  \end{align*}
  \]
Mailboxes

- Also known as message queues, ports
- The explicit and symmetric naming of processes in direct naming
- Limited modularity since changing the name of a process requires changes elsewhere, i.e., in definitions of other processes

P or Q call

\[ \text{send(mbox-id, message)} \]

R calls

\[ \text{receive(mbox-id, message)} \]
Mailbox Issues

• communication is no longer “point-to-point”; e.g., a message received by R may be from P or Q
• “fair merge” property --- do not starve Q from queuing messages by allowing continual queuing of messages only from P
• natural extension to multiple receivers. Possible semantics:
  • Multicast to all in the group gets the same message
  • The first receiver removes it
  • Bulletin board: each receiver decides