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:

```plaintext
send(process_id, message)
receive(process_id, buffer)
```

Example

```plaintext
process P1:          process P2:
   declare x integer    declare y integer
    send(P2, x)          receive(P1, y)
end process          end process
```

Effect of this communication is

```
y := x
```

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
  ```plaintext
  P1: send(P2, x)          P2: receive(P1, x)
    receive(P2, y)        send(P1, y)
  ```
- buffer needed
  ```plaintext
  P1: send(P2, x)          P2: send(P1, x)
    receive(P2, y)        receive(P1, y)
  ```
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
- send(mbox-id, message)
R calls
- 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