CS420: Operating Systems

Interprocess Communication

James Moscola
Department of Engineering & Computer Science
York College of Pennsylvania
Interprocess Communication

- Processes within a system may be independent or cooperating

- Cooperating processes can affect or be affected by other processes, including sharing data

- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience

- Cooperating processes need interprocess communication (IPC)

- Two models of IPC
  - Message passing
  - Shared memory
Communication Models

(a) Message passing

(b) Shared Memory
Cooperating Processes

• An independent process cannot affect or be affected by the execution of another process

• A cooperating process can affect or be affected by the execution of another process

• Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience
Producer-Consumer Problem

- Producer-consumer problem is a common paradigm for cooperating processes

- Producer process produces information that is consumed by a consumer process
  - One solution is to use shared memory for the two processes to communicate
  - Useful to have a buffer that can be filled by the producer and emptied by the consumer if they are to run concurrently
    - **Unbounded-buffer** places no practical limit on the size of the buffer
    - **Bounded-buffer** assumes that there is a fixed buffer size
Bounded-Buffer – Shared-Memory Approach

• The following information is in shared memory and is available to both the producer and the consumer

```c
#define BUFFER_SIZE 10

typedef struct {
    /* info to be passed */
} item;

item buffer[BUFFER_SIZE]; /* circular buffer */

int in = 0;
int out = 0;
```

• This implementation can only use \texttt{BUFFER\_SIZE-1 elements}
Bounded-Buffer – Producer

while (true) {
    /* Produce an item */
    
    while (((in + 1) % BUFFER_SIZE) == out)
        ; /* do nothing -- no free buffers */
    
    buffer[in] = item; /* buffer not full, add item */
    
    in = (in + 1) % BUFFER_SIZE;
}
Bounded Buffer – Consumer

while (true) {
    while (in == out)
        ;  /* do nothing -- nothing to consume */

    // remove an item from the buffer
    item = buffer[out];

    out = (out + 1) % BUFFER_SIZE;

    return item;
}

Interprocess Communication – Message Passing

- **Message passing** – 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 two processes want to communicate, they need to:**
  - Establish a communication link between them
  - Exchange messages via send/receive
Interprocess Communication – Message Passing

• Communication link can be implemented in variety of ways (including shared memory)

• There are several choices when implementing the communication link
  - Direct or indirect communication
  - Synchronous or asynchronous communication
  - Automatic or explicit buffering
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 direct communication link**
  - Links are established automatically between the two processes
  - 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 indirect communication link**
  - Link established only if processes share a common mailbox
  - A link may be associated with more than two processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional
Indirect Communication (Cont.)

• **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 (Cont.)

• Mailbox sharing - consider the following ...
  - $P_1$, $P_2$, and $P_3$ share mailbox A
  - $P_1$, sends; $P_2$ and $P_3$ receive
  - Who gets the message?

• Possible solutions to avoid unpredictable behavior
  - 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 arbitrarily select 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

- Regardless of how messages are exchanged between processes, messages are queued

- Queueing can be implemented in one of three ways
  
  1. **Zero capacity** – queue has maximum length of 0
     Sender must wait (or block) until the receiver gets the message
  
  2. **Bounded capacity** – queue has finite length of \( n \) messages
     Sender must wait if link full
  
  3. **Unbounded capacity** – queue has ‘infinite’ length
     Sender never waits
Examples of IPC Systems - POSIX

- **POSIX Shared Memory**

  - Process first creates shared memory segment
    
    ```c
    segment_id = shmget(IPC_PRIVATE, size, S_IRUSR | S_IWUSR);
    ```

  - Any process wanting access to that shared memory must attach to it
    
    ```c
    shared_memory = (char *) shmat(segment_id, NULL, 0);
    ```

  - Now the process could write to the shared memory
    
    ```c
    sprintf(shared_memory, "Writing to shared memory");
    ```

  - When done a process can detach the shared memory from its address space
    
    ```c
    shmdt(shared_memory);
    ```

  - When the shared memory space is no longer needed, free it
    
    ```c
    shmctl(segment_id, IPC_RMID, NULL);
    ```
Examples of IPC Systems - Mach

• **Mach communication is message based**
  - Even system calls are messages
  - Each task gets two mailboxes at creation - *Kernel* and *Notify*
    • *Kernel* mailbox is used by the kernel to communicate with the process
    • *Notify* mailbox is used by the kernel to send notifications of events to the process
  - Only three system calls needed for message transfer
    `msg_send( )`, `msg_receive( )`, `msg_rpc( )`
  - Mailboxes needed for communication, created via:
    `port_allocate( )`