OPERATING SYSTEM: CSET209

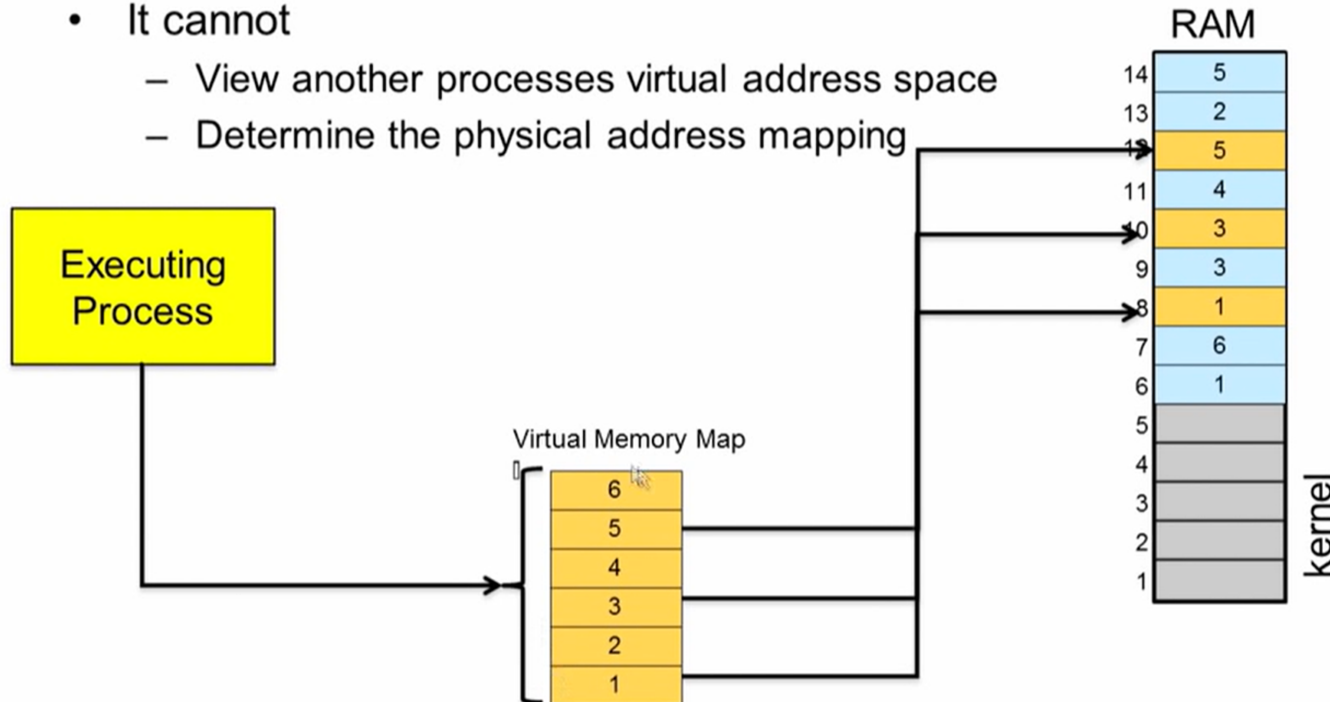
INTER PROCESS COMMUNICATION (IPC)

- Processes executing **concurrently** in the OS may be **independent or cooperating**.
- **Independent** process is a process that **doesn't share data** with any other process. Thus, Independent process cannot affect or be affected by the execution of another process
- Cooperating process is a process that **shares data** with other process(es). Thus, can affect or be affected by the execution of another process
- **IPC is a mechanism where on process communicates with another process.**
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity/Convenience

INTER PROCESS COMMUNICATION (IPC) : NEED

- A program may involve more than one flow of control in its execution.
  - Example: A program may execute in the form of multiple threads cooperating to achieve a common goal.
- To cooperate, processes somehow must communicate.
- But, Operating system does not allow a process to read memory locations of another process.

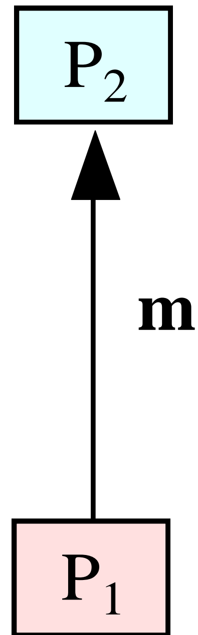
- During execution, each process can only view its virtual addresses,

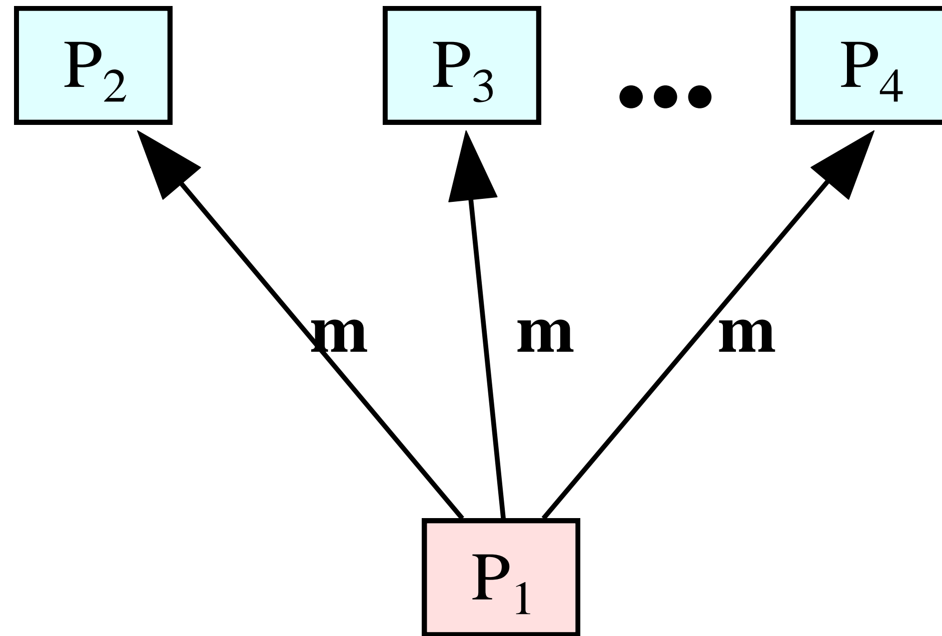


IPC – UNICAST AND MULTICAST

- In distributed computing, two or more processes engage in **IPC** using a **protocol** agreed upon by the **processes**. A process may be a sender at some points during a protocol, a receiver at other points.
- When communication is from one process to a single other process, the IPC is said to be a ***unicast***, e.g., **Socket communication**. When communication is from one process to a group of processes, the IPC is said to be a ***multicast***, e.g., **Publish/Subscribe Message model**, a topic that we will explore in a later chapter.

UNICAST VS. MULTICAST





unicast **multicast**

INTER PROCESS COMMUNICATION (IPC) MODELS

Common mechanisms of IPC are as follows:

- Shared files
- Shared memory
- Message passing

IPC: SHARED FILES

- Most basic form of IPC accomplished by using file operations to read from and write to a **shared file on disk**.
- Advantages:
  - **Easy to use** for developers who are familiar with file handling.
  - Shared files can **persist beyond the lifetime of the processes** that use them. Hence, useful for sharing data between processes that do not run simultaneously.
- Limitation:
  - **Slower** due to disk operations involved.
  - **Synchronization** among the processes is required to prevent race conditions.

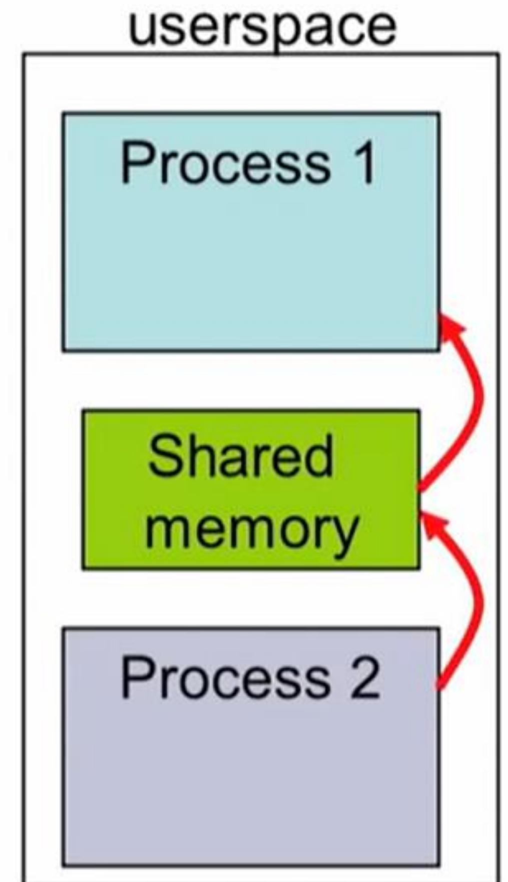
IPC : SHARED MEMORY

- One process will create a **shared memory segment in RAM** which the

  other process can also access.
- Advantages:
  - Access to this shared memory region is like a **regular array like** access.
  - Communication **is extremely fast as no system calls** are involved.
- Limitation:
  - **Needs synchronization** between the processes. Absence of synchronization

    leads to errors.

Example: Producer- consumer



IPC : SHARED MEMORY

Example: Producer- consumer

Two types of buffers can be used. The places no practical limit on the size of the buffer. The consumer may have to wait for new items, but the producer can always produce new items. The assumes a fixed buffer size. In this case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full. Let's look more closely at how the bounded buffer can be used to enable processes to share memory.

```
#define BUFFER_SIZE 10
typedef struct { }item;
item buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
```

IPC : SHARED MEMORY

//The buffer is empty when in== out; the buffer is full when ((in+ 1)% BUFFER_SIZE) == out.

```
item nextProduced;
while (true) {
/* produce an item in nextProduced */
 while ( ((in + 1) % BUFFER_SIZE) == out) ;
/* do nothing */
 buffer[in] = nextProduced;
in = (in + 1) % BUFFER_SIZE; }
```

The producer process.

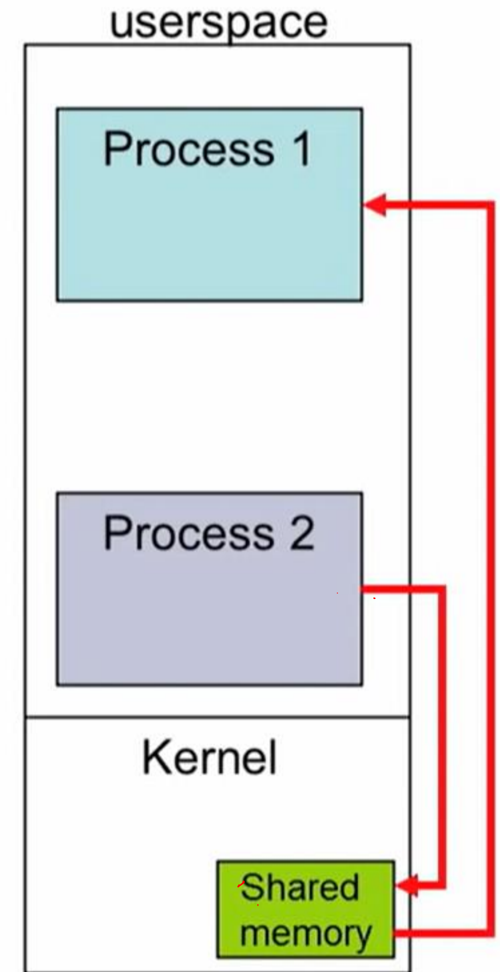
```
item nextConsumed;
while (true) {
while (in == out) ; // do nothing
nextConsumed = buffer[out];
out = (out + 1) % BUFFER_SIZE;
/* consume the item in nextConsumed */
 }
```

//The consumer process.

Issue: Synchronization

IPC : MESSAGE PASSING

- **Shared memory segment is created in the Kernel space.**
- Hence, system calls such as **send** and **receive** are required to communicate.
- Advantages:
  - Less error prone as the **synchronization is taken care of by the kernel**.
- Limitation:
  - **Slower** in comparison to shared memory due to the use of system calls.



IPC : MESSAGE PASSING

- Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space and is particularly useful in a distributed environment, where the communicating processes may reside on different computers connected by a network. For example, a chat program used on the World Wide Web could be designed so that chat participants communicate with one another by exchanging messages.
- A message-passing facility provides at least two operations: send(message) and receive(message). Messages sent by a process can be of either fixed or variable size.
- If processes P and Q want to communicate, they must send messages to and receive messages from each other; a communication link must exist between them. Here are several methods for logically implementing a link and the send() and receive() operations:
    - Direct or indirect communication
    - Synchronous or asynchronous communication
    - Automatic or explicit buffering

IPC : MESSAGE PASSING

- Direct communication

Each process that wants to communicate must explicitly name the recipient or sender of the communication. In this scheme, the send() and receive() primitives are defined as:

send(P, message) -Send a message to process P.

receive (Q, message)-Receive a message from process Q.

- A communication link in this scheme has the following properties:
  - A link is established automatically between every pair of processes that want to communicate. The processes need to know only each other's identity to communicate.
  - A link is associated with exactly two processes.
  - Between each pair of processes, there exists exactly one link.

This scheme exhibits symmetry in addressing; that is, both the sender process and the receiver process must name the other to communicate.

IPC : MESSAGE PASSING

- Direct communication

A variant of this scheme employs asymmetry in addressing. Here, only the sender names the recipient; the recipient is not required to name the sender. In this scheme, the send() and receive() primitives are defined as follows:
send(P, message) -Send a message to process P.
receive (id, message) -Receive a message from any process; the variable id is set to the name of the process with which communication has taken place.

IPC : MESSAGE PASSING

- Indirect communication

Messages are sent to and received from mailboxes, or ports. A mailbox can be viewed abstractly as an object into which messages can be placed by processes and from which messages can be removed. Each mailbox has a w1.ique identification. For example, POSIX message queues use an integer value to identify a mailbox. In this scheme, a process can communicate with some other process via a number of different mailboxes. Two processes can communicate only if the processes have a shared mailbox, however. The send() and receive 0 primitives are defined as follows:
send (A, message) -Send a message to mailbox A.
receive (A, message)-Receive a message from mailbox A.
In this scheme, a communication link has the following properties:
A link is established between a pair of processes only if both members of the pair have a shared mailbox.
A link may be associated with more than two processes.
Between each pair of communicating processes, there may be a number of different links, with each link corresponding to one mailbox.

IPC : MESSAGE PASSING

- Synchronization

Communication between processes takes place through calls to send() and receive () primitives. There are different design options for implementing each primitive. Message passing may be either blocking or nonblocking also known as synchronous and asynchronous.

  - Blocking send. The sending process is blocked until the message is received by the receiving process or by the mailbox.
  - Nonblocking send. The sending process sends the message and resumes operation.
  - Blocking receive. The receiver blocks until a message is available.
  - Nonblocking receive. The receiver retrieves either a valid message or a null.

IPC : MESSAGE PASSING

- Buffering
  - Zero capacity
  - Bounded capacity
  - Unbounded capacity

Thank you !