Processes

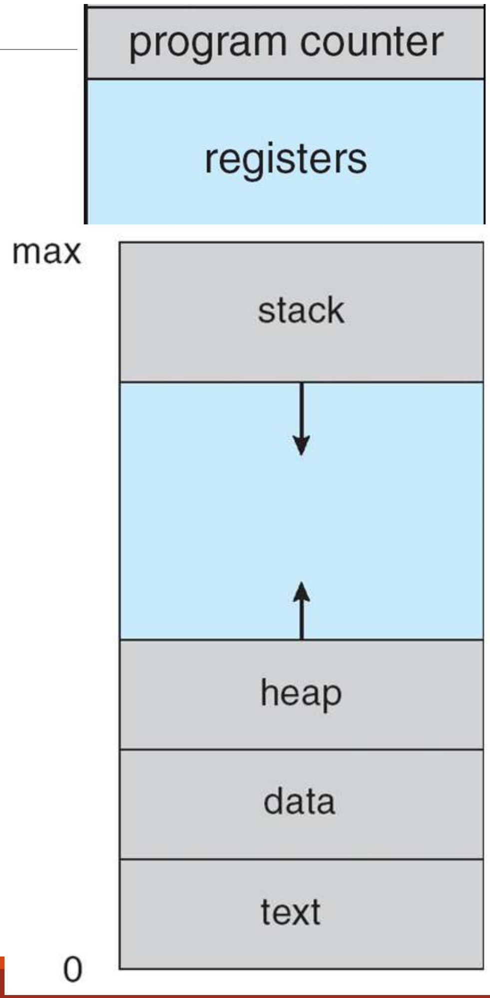
CHAPTER 3

Outline

- Introduction to process concept
- Process scheduling
- Operations on the processes
- Inter-process communication

Processes

- A Process is a program in execution; process execution must progress in a sequential fashion
- A Process has multiple parts
  - Current activity including program counter, processor registers
  - Stack containing temporary data
  - Function parameters, local variables
  - Heap containing memory dynamically allocated during run time
  - Data section containing global variables
  - The program code, also called the text section



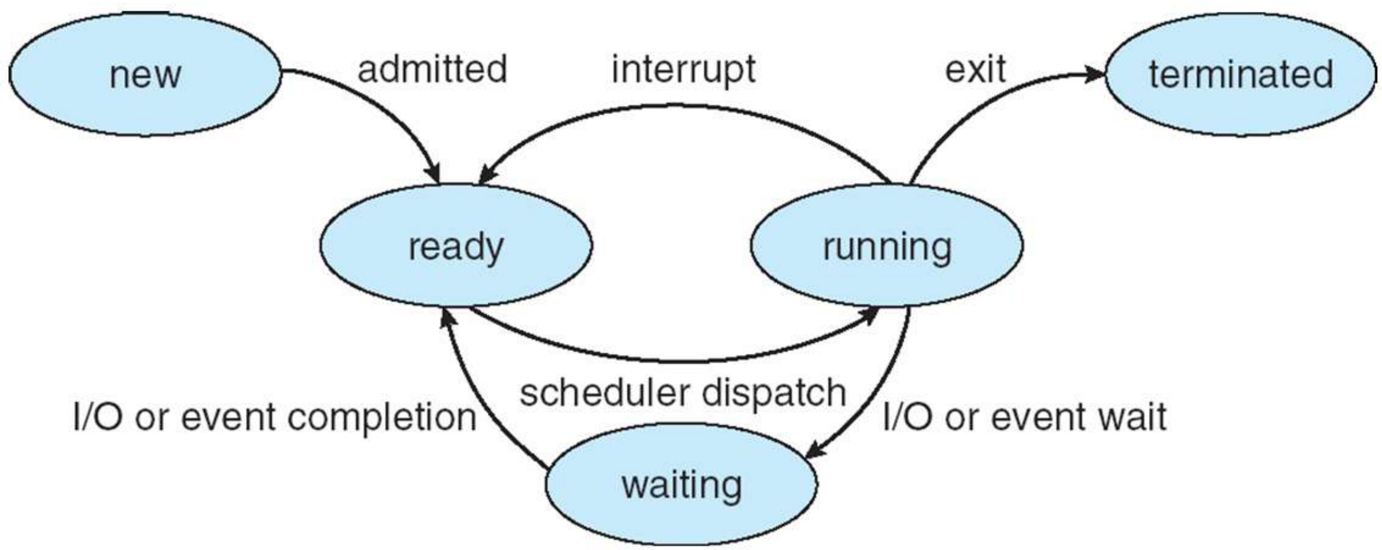
Processes

- Program is a passive entity stored on disk (executable file), the process is active
- Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc.
- One program can be several processes
  - Consider multiple users executing the same program
  - For example running multiple instances of Google Chrome

Process States

- As a process executes, it changes state
  - new: The process is being created
  - running: Instructions are being executed
  - waiting: The process is waiting for some event to occur
  - ready: The process is waiting to be assigned to a processor
  - terminated: The process has finished execution

Process States

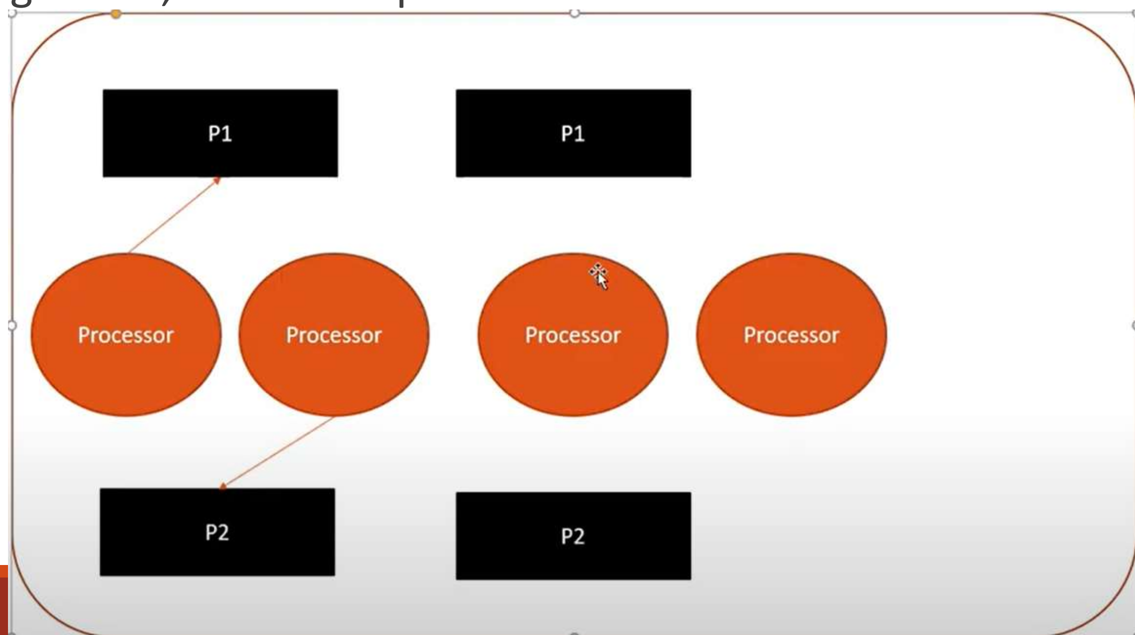


Process Control Block

- Information associated with each process also called task control block
  - Process state – running, waiting, etc.
  - Process number is its ID for the OS
  - Program counter – location of instruction to next execute
  - CPU registers – contents of all process registers
  - CPU scheduling info- priorities, scheduling queue pointers
  - Memory-management info – memory allocated to the process
  - Accounting info – CPU used, clock time elapsed since the start, etc.
  - I/O status information – I/O devices allocated to the process, list of open files

Process Scheduling

- In a single processor environment, managing the processing time of the processor is very important.
- Processes waiting for a long time to process using CPU is not a good idea.
- All processes should be allowed to use the processor to complete their tasks.
- Better process management, better the performance of the machine.



Process Scheduling

- Maximize CPU use, quickly switch processes onto CPU for time-sharing
- Process scheduler selects among available processes for the next execution on CPU
- Maintains scheduling queues of processes
  - Job queue – set of all processes in the system
  - Ready queue – set of all processes residing in main memory, ready and waiting to execute
  - Device queues – a set of processes waiting for an I/O device
  - Processes migrate among the various queues

Process Scheduling

- In a single processor environment different processes might wait for their turn.
- Since one process is processed at a time.
- If there are several processes waiting for processing, a technique is required to process them.
- Processes are scheduled to be processed.
- The part of OS that schedules the processes is called a **Scheduler**.
- Algorithm followed by Scheduler is **Scheduling Algorithm**.

Schedulers

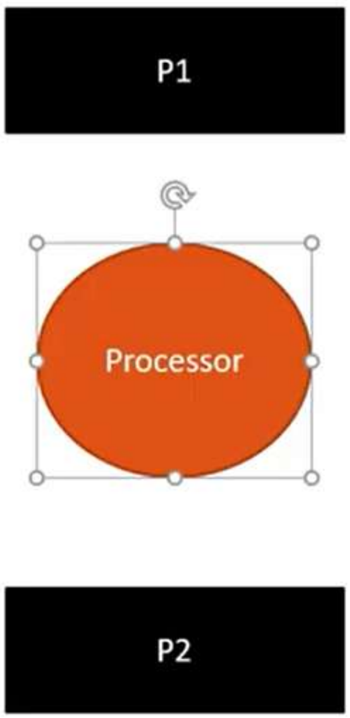
- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) and (must be fast)
- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) and (may be slow)

Schedulers

- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations
  - **CPU-bound process** – spends more time doing computations;

Context Switch

- When the CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB, the longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU, multiple contexts loaded at once

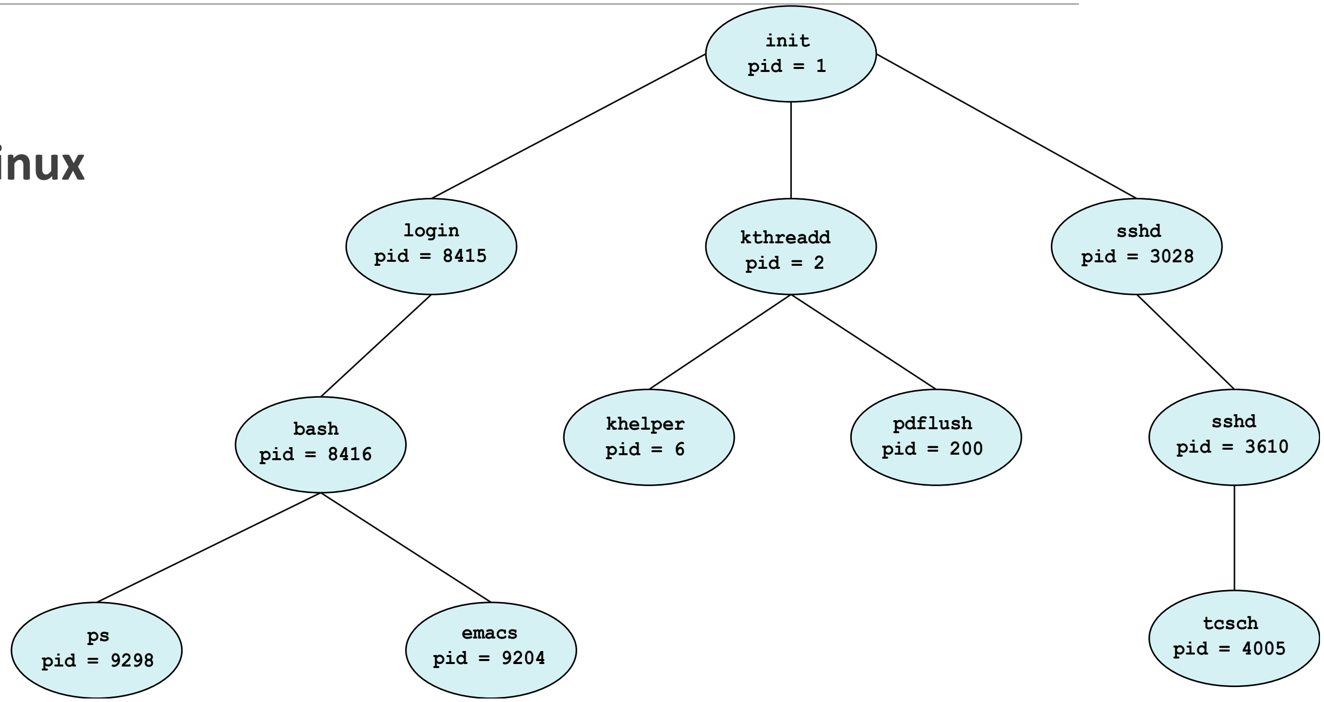


Process Operations

- There are several operations on the processes
- Most common ones are
  - Process Creation
  - Process Termination

Process Operations

Tree of Processes in Linux



bash: unix shell
sshd: provide secure encrypted communications
ps: process status
Emacs is the text editor that runs on the Linux operating system
…

Process Operations: Process Creation

- Parent process create children processes, which, in turn, create other processes, forming a tree of processes
- Processes are identified and managed via a process identifier (pid)
- Resource-sharing options
  - Parent and children share all resources
  - Children share a subset of parent's resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate
- Address space: How RAM is used?
  - Child duplicate of the parent
  - Child has a program loaded into it
- UNIX examples
  - fork() is a system call that creates a new process
  - exec() is a system call used to replace the current process image with a new process image

Process Operations

Process Termination

- Process executes the last statement and then asks the operating system to delete it using the exit() system call.
  - Returns status data from child to parent (via wait())
  - Process' resources are deallocated by the operating system
- Parent may terminate the execution of children's processes using the abort() system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to a child is no longer required
  - The parent is exiting and the operating system does not allow a child to continue if its parent terminates

Process Operations: Process Termination

- Some operating systems do not allow child to exists if its parent has terminated. If a process terminates, then all its children must also be terminated.
  - Cascading termination. All children, grandchildren, etc. are terminated.
  - The termination is initiated by the operating system.
- The parent process may wait for termination of a child process by using the wait() system call. The call returns status information and the pid of the terminated process
- If no parent waiting (did not invoke wait()) process is a **zombie**: it is dead process but no one is taking it. In general, it is the result of a bad program. Use the command '***top***' in Linux to see them.
- If parent terminated without invoking wait , process is an orphan. Use the following command to see them ***# ps -elf | head -1; ps -elf | awk '{if ($5 == 1 && $3 != "root") {print $0}}' | head***
- The UNIX **nohup** command allows a child to continue executing after its parent has exited.

Inter-Process Communication (IPC)

- Processes within a system may be **independent** or **cooperating**
- Cooperating process can affect or be affected by other processes
- Reasons for cooperating processes:
  - Information sharing
  - Computation speedup
  - Modularity
  - Convenience
- Cooperating processes need inter-process communication (IPC)
  - Shared memory
  - Message passing

Inter-Process Communication (IPC)

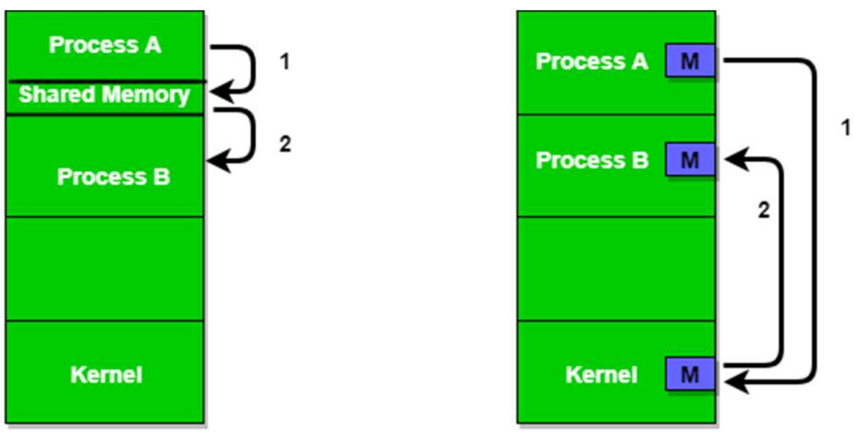
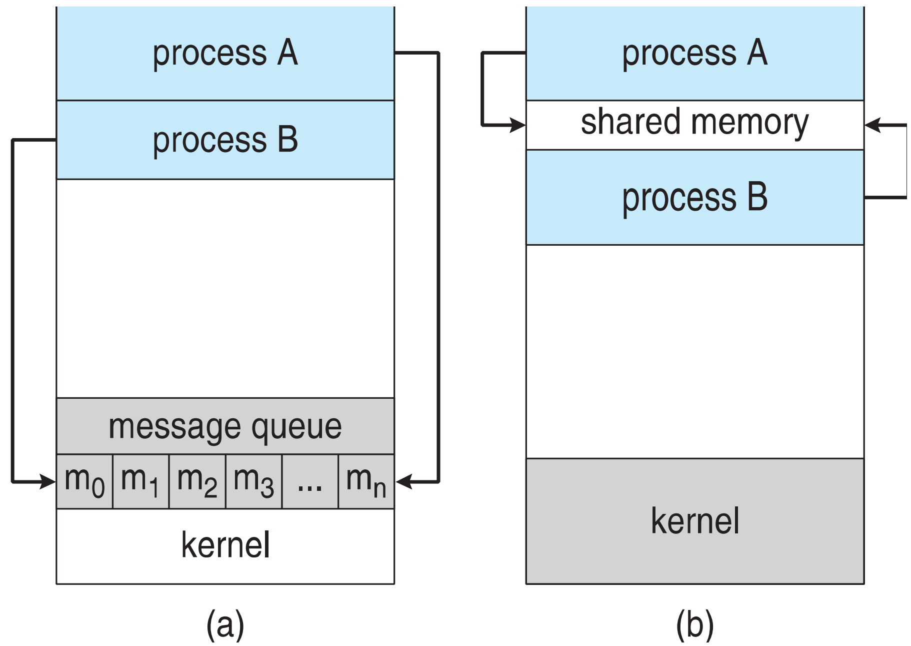


Figure 1 - Shared Memory and Message Passing



Message passing shared memory

Inter-Process Communication (IPC)

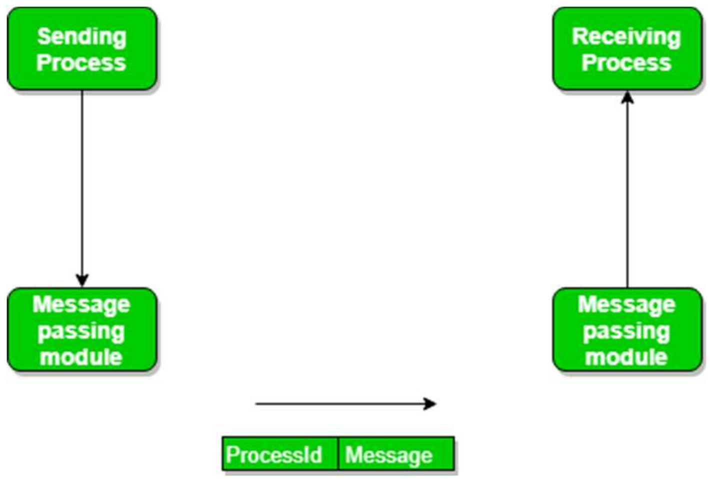
Shared Memory

- An area of memory shared among the processes that wish to communicate
- The communication is under the control of the users processes not the operating system.
- Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.

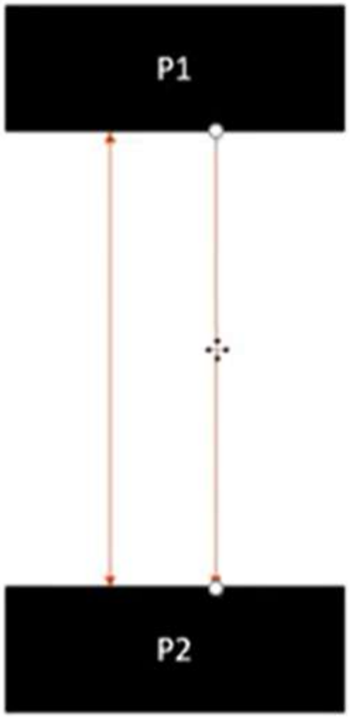
Inter-Process Communication (IPC)

Message Passing

- Mechanism for processes to communicate and to synchronize their actions
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - send(message)
  - receive(message)
- The message size is either fixed or variable



Inter-Process Communication (IPC)



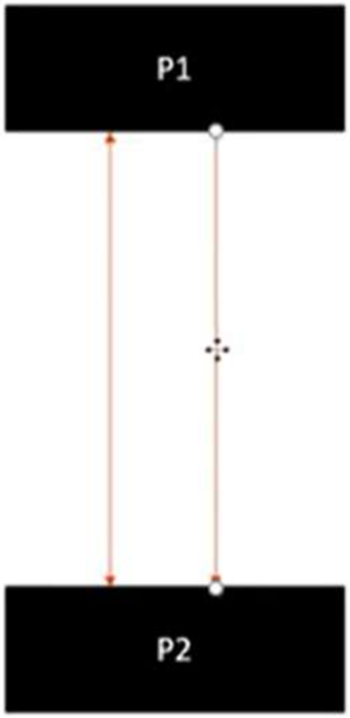
Message Passing

- If processes P and Q wish to communicate, they need to:
  - Establish a communication link between them
  - Exchange messages via send/receive
- Implementation issues:
  - How are links established?
  - Can a link be associated with more than two 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?

Inter-Process Communication (IPC)

Message Passing

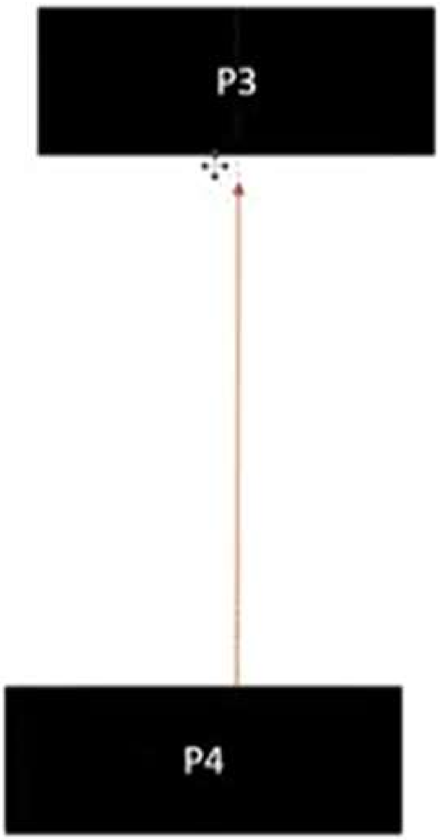
- Implementation of communication link
  - Physical:
    - Shared memory
    - Hardware bus
    - Network
  - Logical:
    - Direct or indirect
    - Synchronous or asynchronous
    - Automatic or explicit buffering



Inter-Process Communication (IPC)

Message Passing (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 communication link
  - Links are established automatically
  - 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



Inter-Process Communication (IPC)

Message Passing (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 communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes
  - Each pair of processes may share several communication links
  - Link may be unidirectional or bi-directional

Inter-Process Communication (IPC)

Message Passing (Indirect Communication)

- Operations
  - create a new mailbox (port)
  - 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

Inter-Process Communication (IPC)

Message Passing (Indirect Communication)

- Mailbox sharing
  - P1, P2, and P3 share mailbox A
  - P1, sends; P2 and P3 receive
  - Who gets the message?
- Solutions
  - Allow a link to be associated with at most two processes
  - Only one process is allowed to execute the receive operation at a given time
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

Inter-Process Communication (IPC)

Message Passing (Synchronization)

- Message passing may be either blocking or non-blocking
- Blocking is considered synchronous
  - Blocking send -- the sender is blocked until the message is received
  - Blocking receive -- the receiver is blocked until a message is available
- Non-blocking is considered asynchronous
  - Non-blocking send -- the sender sends the message and continue
  - Non-blocking receive -- the receiver receives:
- Different combinations possible

Inter-Process Communication (IPC)

Message Passing (Buffering)

- Queue of messages attached to the link.
- implemented in one of three ways
  1. Zero capacity – no messages are queued on a link. Sender must wait for receiver
  2. Bounded capacity – finite length of n messages. Sender must wait if link full
  3. Unbounded capacity – infinite length. Sender never waits