Processes

AE4B33OSS

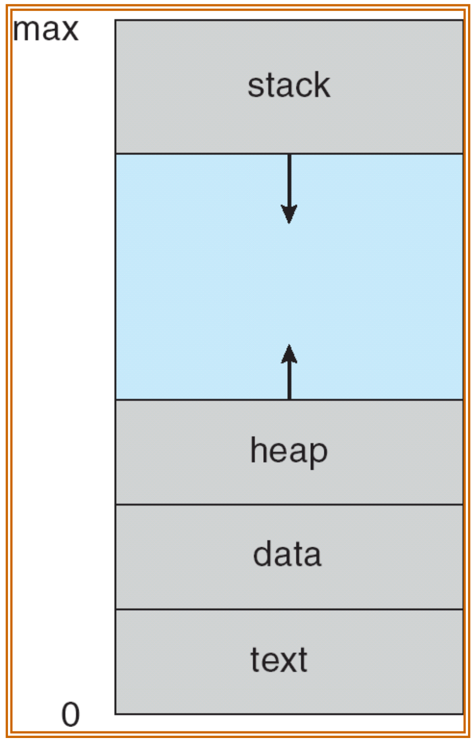
Processes

- Process Concept
- Process Scheduling
- Operations on Processes
- Cooperating Processes
- Inter-process Communication

Process Concept

- Process – a program in execution; active entity
- Program – passive entity
- A process includes:
  - Text section
  - Data section
  - Stack
  - Heap

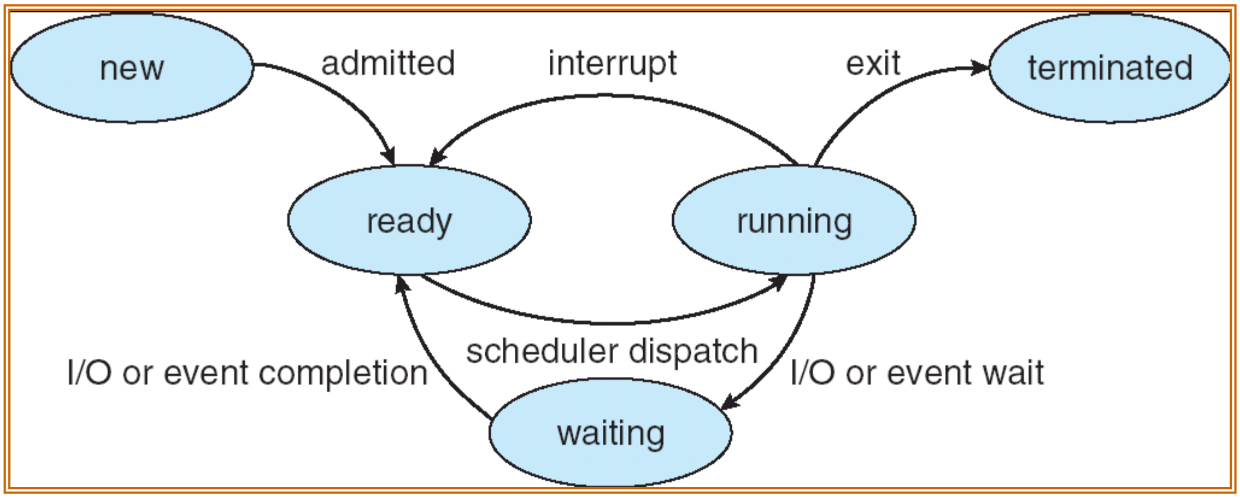
Process in Memory



Process State

- 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 the CPU
  - **terminated**: The process has finished execution

Diagram of Process State

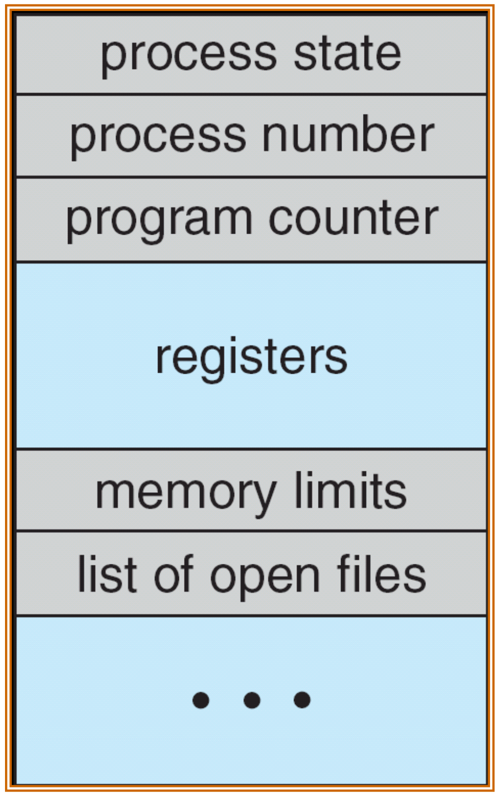


Process Control Block (PCB)

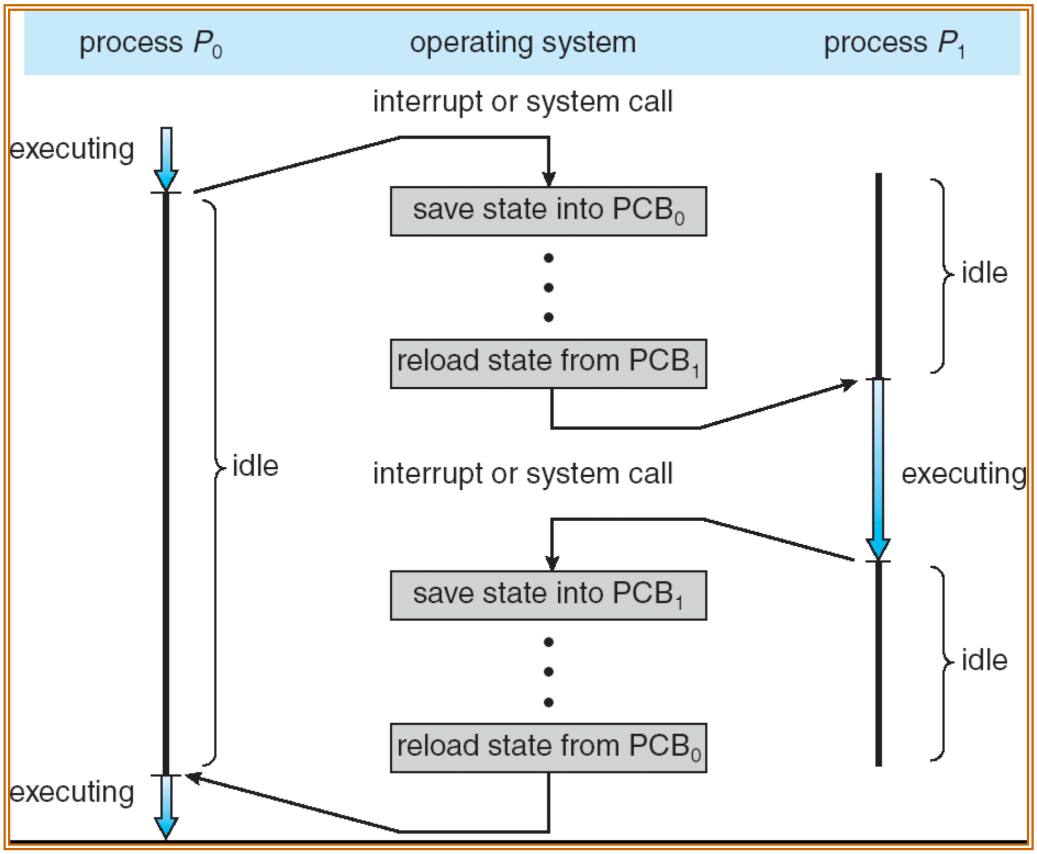
Information associated with each process:

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Process Control Block (PCB)



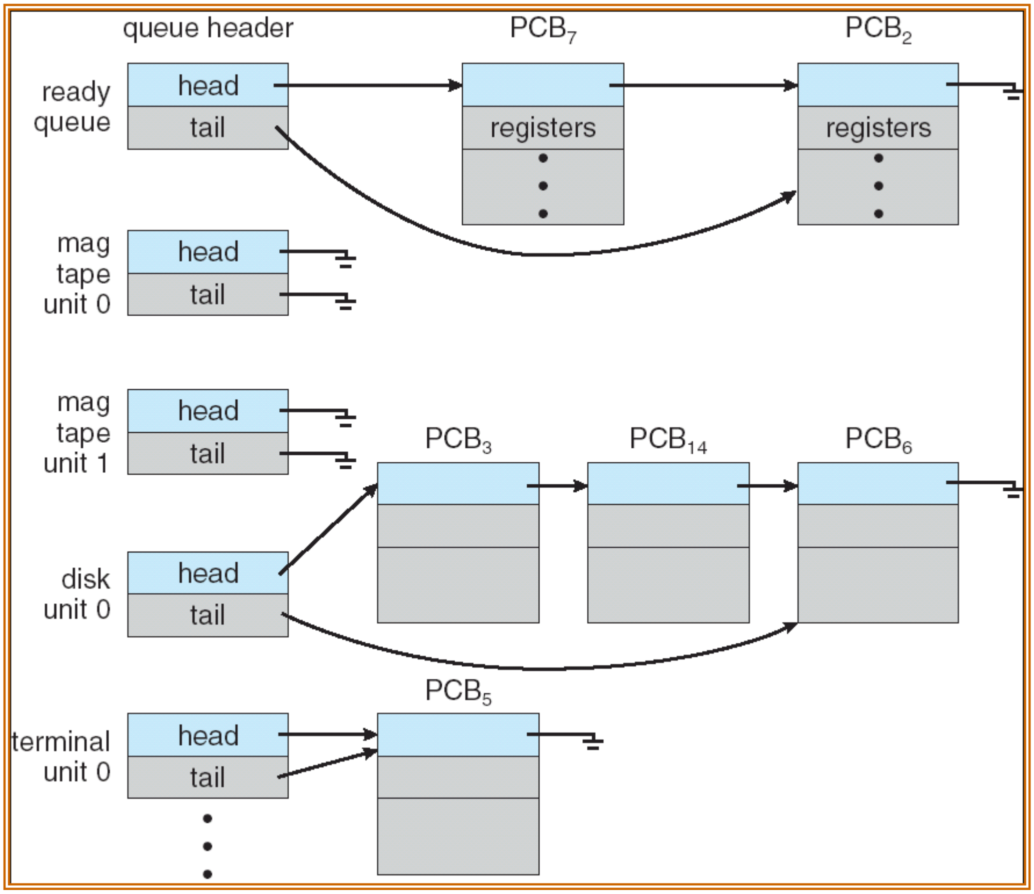
CPU Switch From Process to Process



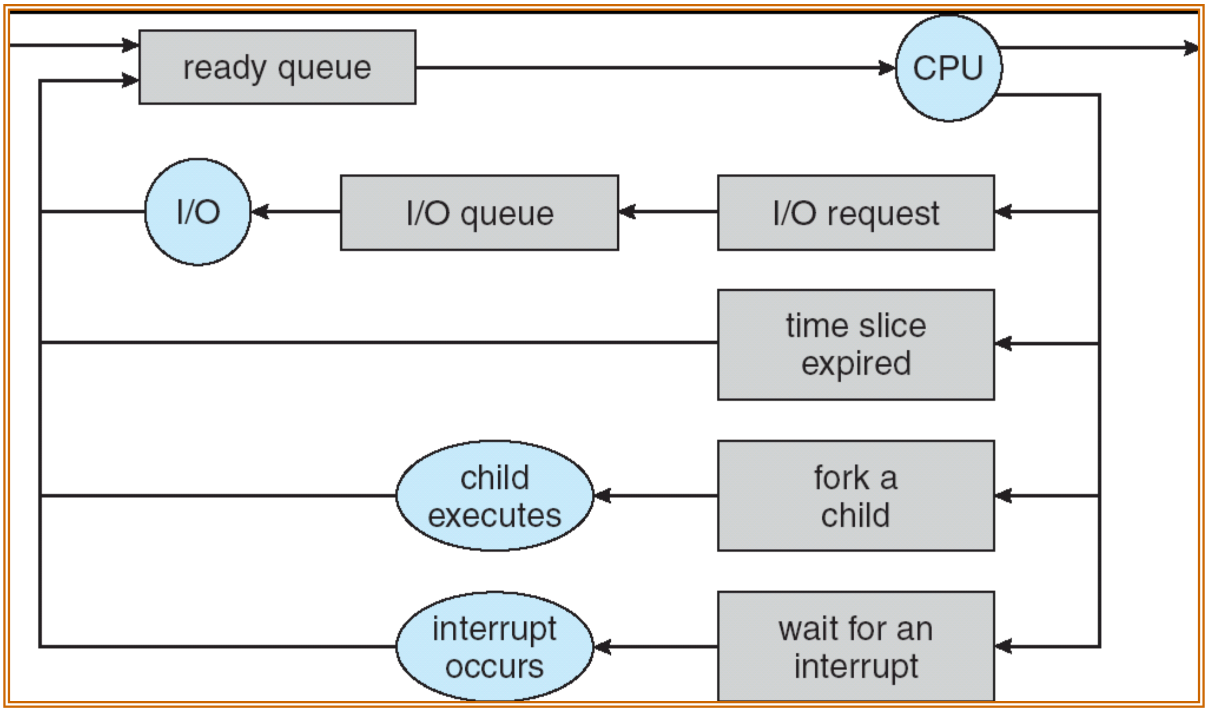
Process Scheduling Queues

- **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** – set of processes waiting for an I/O device
- Processes migrate among the various queues

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling



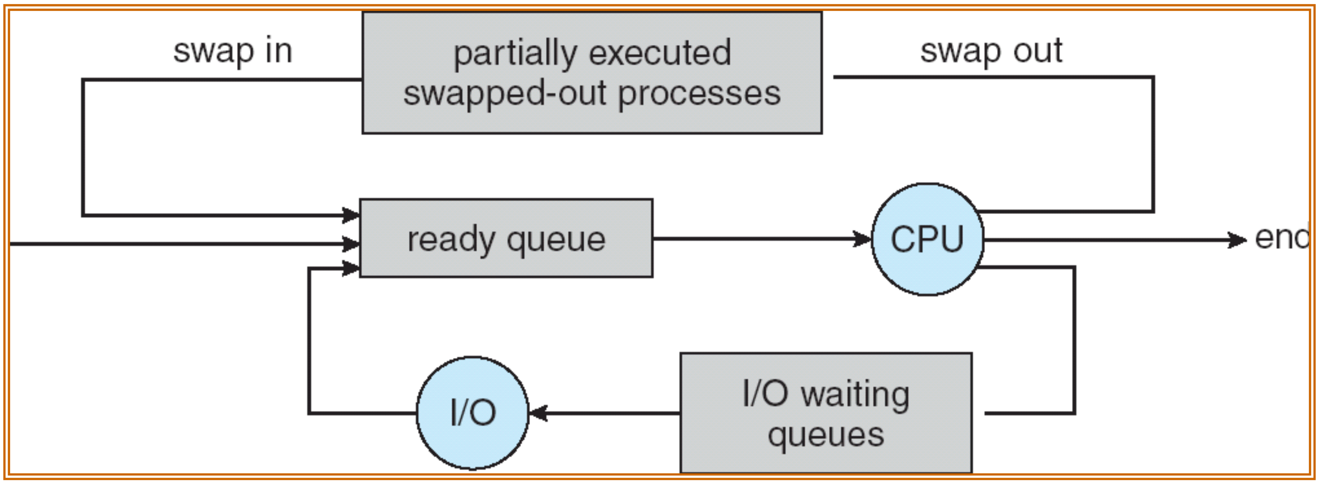
1) INT force

2) Willing

Schedulers

- **Long-term scheduler** (or job scheduler) – selects which processes should be brought into the ready queue
- **Short-term scheduler** (or CPU scheduler) – selects which process should be executed next and allocates CPU

Addition of Medium Term Scheduling



Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds) $\Rightarrow$ (must be fast)
- Long-term scheduler is invoked very infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
- The long-term scheduler controls the *degree of multiprogramming*
- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts

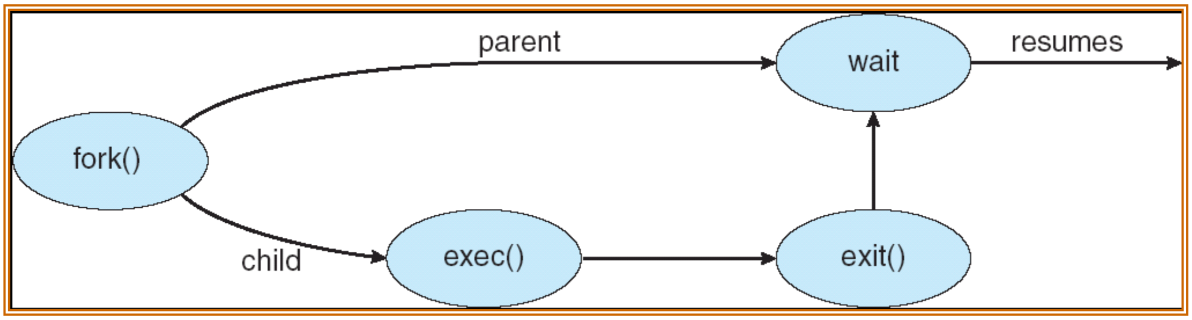
Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process
- Context-switch time is overhead; the system does no useful work while switching
- Time dependent on hardware support

Process Creation

- Parent process creates children processes, which, in turn create other processes, forming a tree of processes
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate

Process Creation



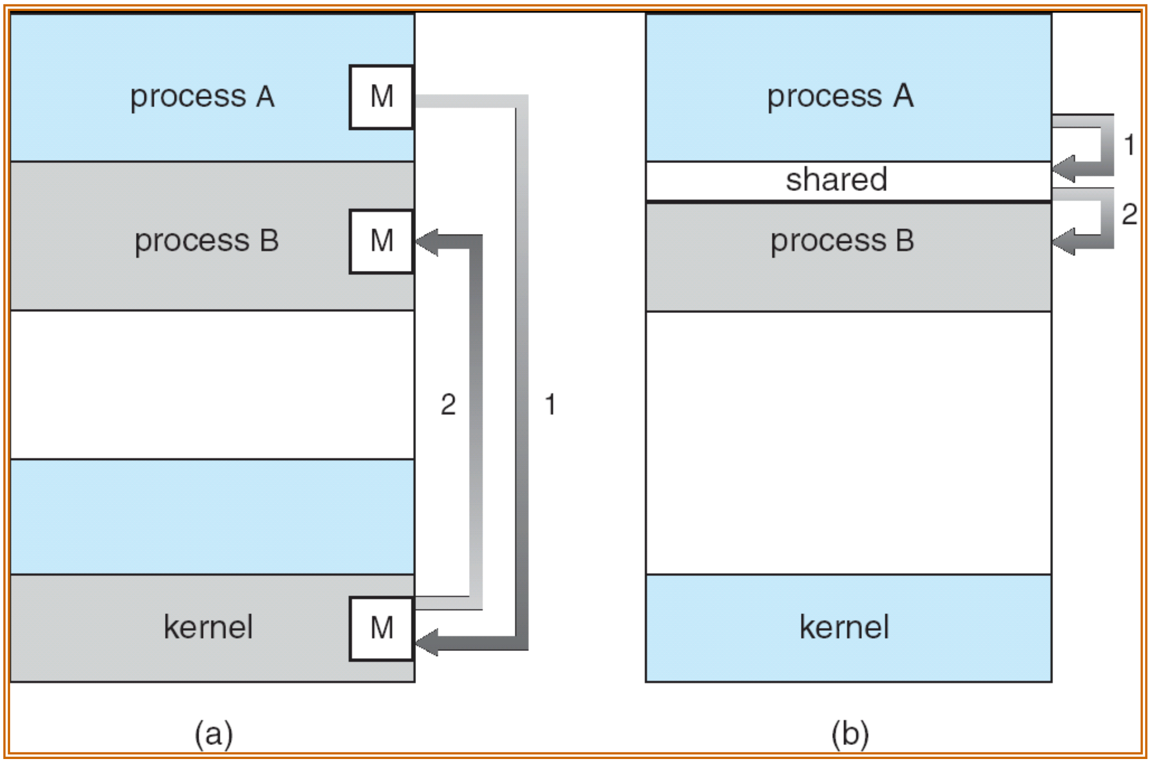
Process Termination

- Process executes last statement and asks the operating system to delete it (**exit**)
  - Output data from child to parent (via **wait**)
  - Process' resources are deallocated by operating system
- Parent may terminate execution of children processes (**abort**)
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - If parent is exiting
    - Some operating system do not allow child to continue if its parent terminates
      - All children terminated - *cascading termination*

Cooperating Processes

- **Independent** process cannot affect or be affected by the execution of another process
- **Cooperating** process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience
- Cooperating processes need **inter-process communication** (**IPC**)
- Two models of IPC
  - Shared memory
  - Message passing

Communications Models



Shared Memory Systems

Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process
  - *unbounded-buffer* places no practical limit on the size of the buffer
  - *bounded-buffer* assumes that there is a fixed buffer size
    - If buffer full, producer has to wait
    - If buffer empty, consumer has to wait

Bounded-Buffer – Shared-Memory Solution

- Shared data

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

- Shared buffer- circular array with 2 integer variables in & out.
- in - next free position in buffer
- out – first full position in buffer
- Buffer empty: in==out
- Buffer full: ((in+1)%BUFFER_SIZE)==out
- Atmost BUFFER_SIZE-1 elements in buffer

Bounded-Buffer – Producer process

```
item nextp;
    while (true) {
      /* Produce an item in nextp */
     while (((in + 1) % BUFFER SIZE)  == out)
       ;   /* do nothing -- no free buffers */
      buffer[in] = nextp;
      in = (in + 1) % BUFFER SIZE;
   }
```

Bounded Buffer – Consumer process

```
item nextc;
while (true) {
        while (in == out)
             ; // do nothing -- nothing to consume

         // remove an item from the buffer
         nextc = buffer[out];
         out = (out + 1) % BUFFER SIZE;
     // consume the item in nextc
    }
```

Message passing systems

- Message system – 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 *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive

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
  - 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 communication link
  - Link established only if processes share a common mailbox
  - Link may be unidirectional or bi-directional

Indirect Communication

- 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

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

- Queue of messages attached to the link; implemented in one of three ways
  1. Zero capacity – 0 messages or no messages waiting in the queue;
     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