Processes

Process Concept

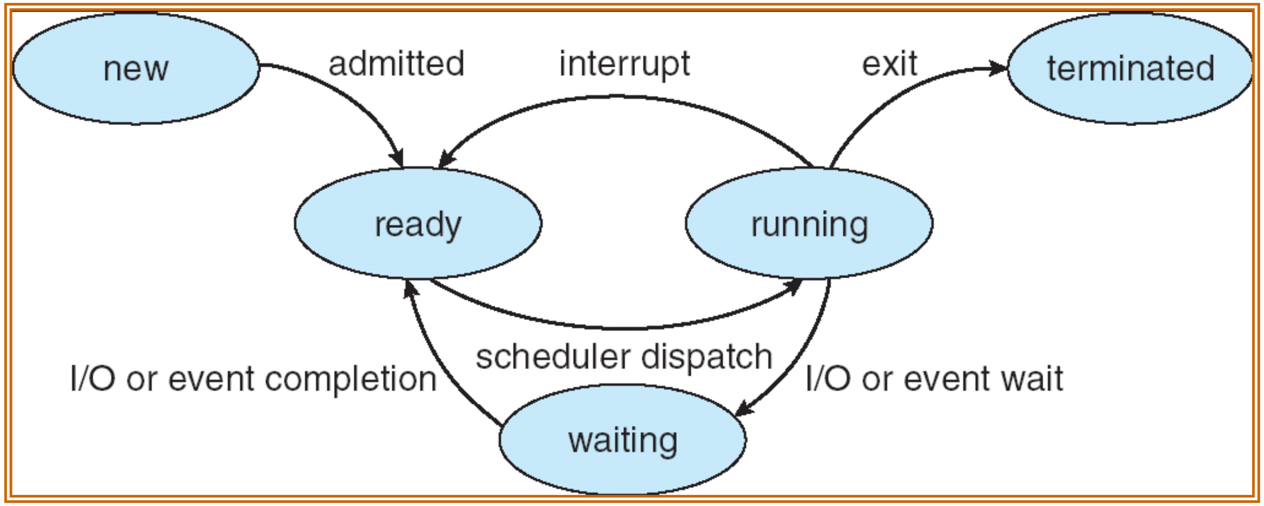
- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- Textbook uses the terms ***job*** and ***process*** almost interchangeably
- **Process** – a program in execution; process execution must progress in sequential fashion
- A process includes:
  - program counter
  - stack
  - data section
  - code
  - heap
  - allocated memory

Processes and Scheduling

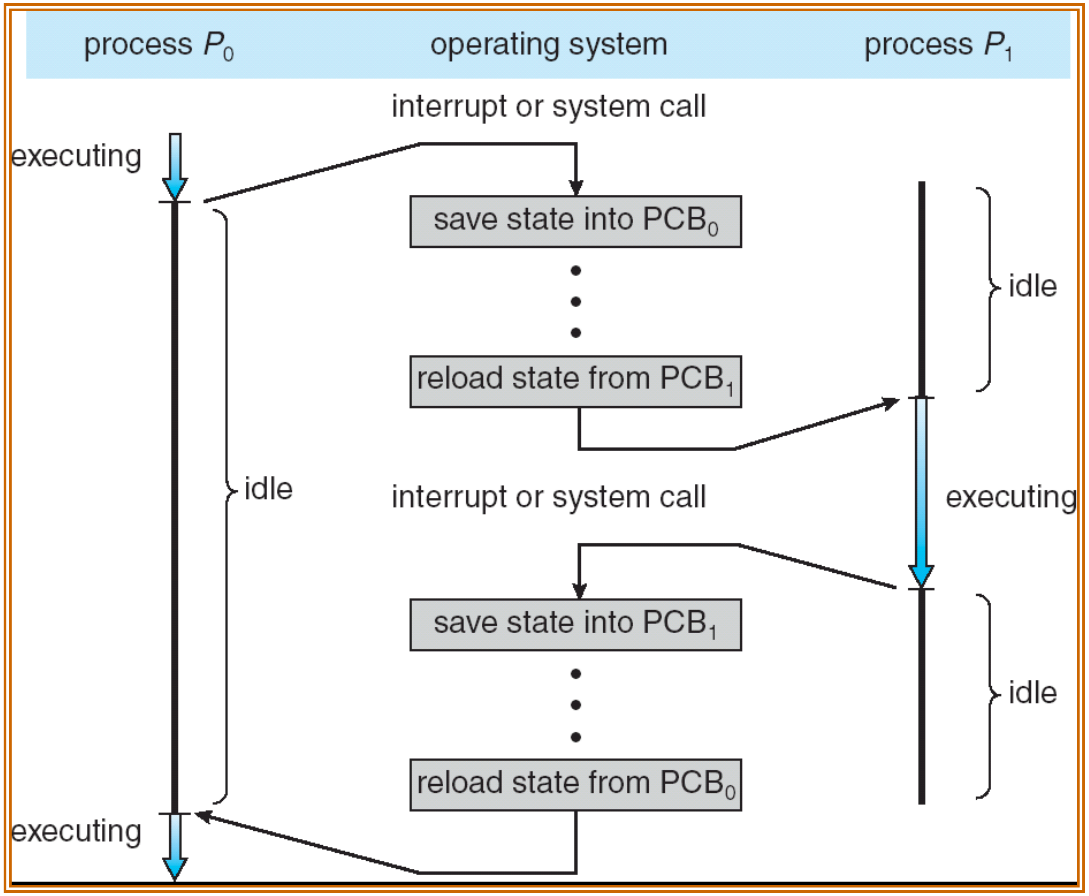
- Processes have *isolation* from each other
  - Address space
  - Security context
  - Termination protection
- Processes are scheduled separately from each other
- One process blocking or being pre-empted allows another to run
- On some systems, a process can be composed of several *threads* which share the process
- On a multiprocessor, processes can and do run simultaneously

Diagram of 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 a process
  - **terminated**: The process has finished execution



CPU Switch From Process to Process



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

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

Bounded-Buffer – Shared-Memory Solution

- Shared data

```
int *in = (int *) data++;
int *out = (int *) data++;
#define BUFFER_SIZE 10
typedef struct {
 . . .
} item;
item *buffer[BUFFER_SIZE];
buffer = (item *) data;
*in = *out = 0;
```

- Solution is correct, but can only use BUFFER_SIZE - 1 elements

Bounded-Buffer – Producer Process

```
item nextProduced;

while (1) {
    while (((*in + 1) % BUFFER_SIZE) == *out)
          ; /* busy wait, do nothing */
    buffer[*in] = nextProduced;
    *in = (*in + 1) % BUFFER_SIZE;
}
```

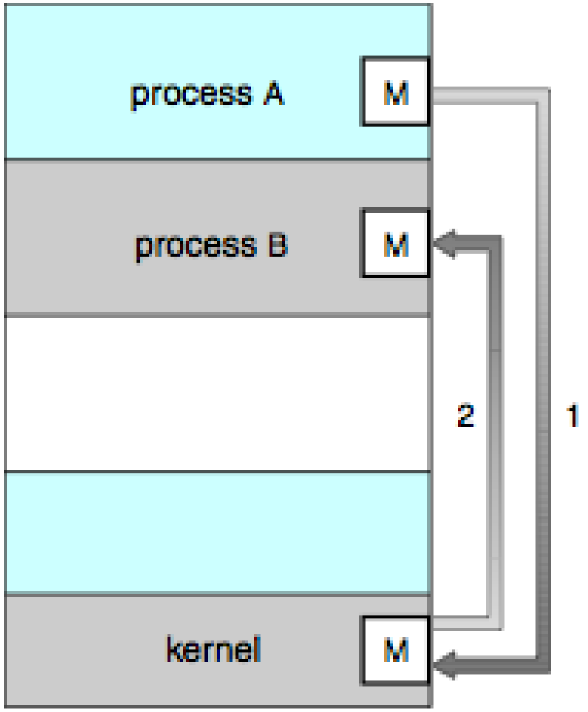
Bounded-Buffer – Consumer Process

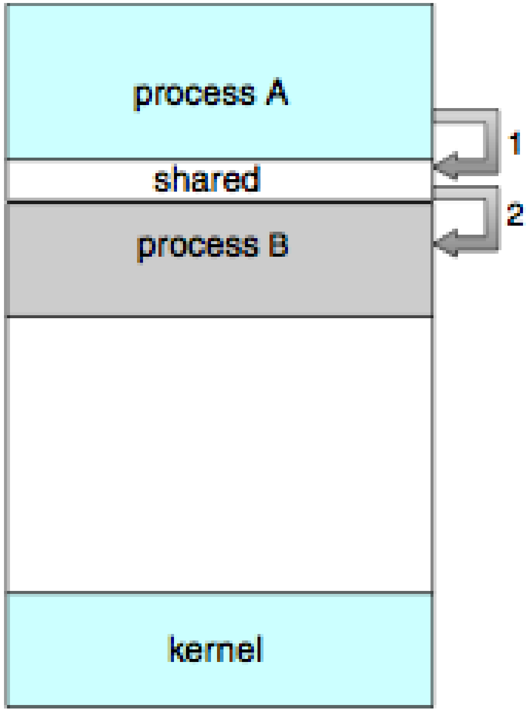
```
item nextConsumed;

while (1) {
    while (*in == *out)
          ; /* busy wait, do nothing */
    nextConsumed = buffer[*out];
    *out = (*out + 1) % BUFFER_SIZE;
}
```

Interprocess Communication

Message Passing

Shared Memory





Message Passing

- Message system – processes communicate with each other without resorting to shared variables
- Message passing 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
- Implementation of communication link
  - physical (e.g., shared memory, hardware bus)
  - logical (e.g., logical properties)

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

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

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