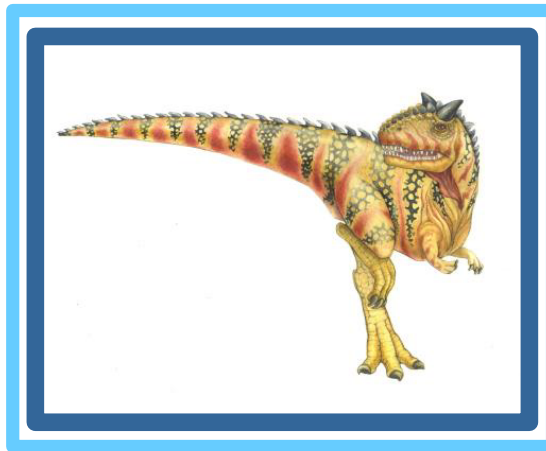# Chapter 3: Processes

Chapter 3: Processes

- Process Concept

- Process Scheduling

- Operations on Processes

- Interprocess Communication

- Communication in Client-Server Systems

Objectives

☐ To introduce the notion of a process -- a program in execution, which forms the basis of all computation

☐ To describe the various features of processes, including scheduling, creation and termination, and communication

☐ To explore interprocess communication using shared memory and message passing

☐ To describe communication in client-server systems

Process Creation

- **Parent** process create **children** processes, which, in turn create other processes, forming a **tree** of processes

- Generally, process identified and managed via a **process identifier** (**pid**)

- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources

- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

Process Creation (Cont.)

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it

- UNIX examples
  - `fork()` system call creates new process
  - `exec()` system call used after a `fork()` to replace the process' memory space with a new program
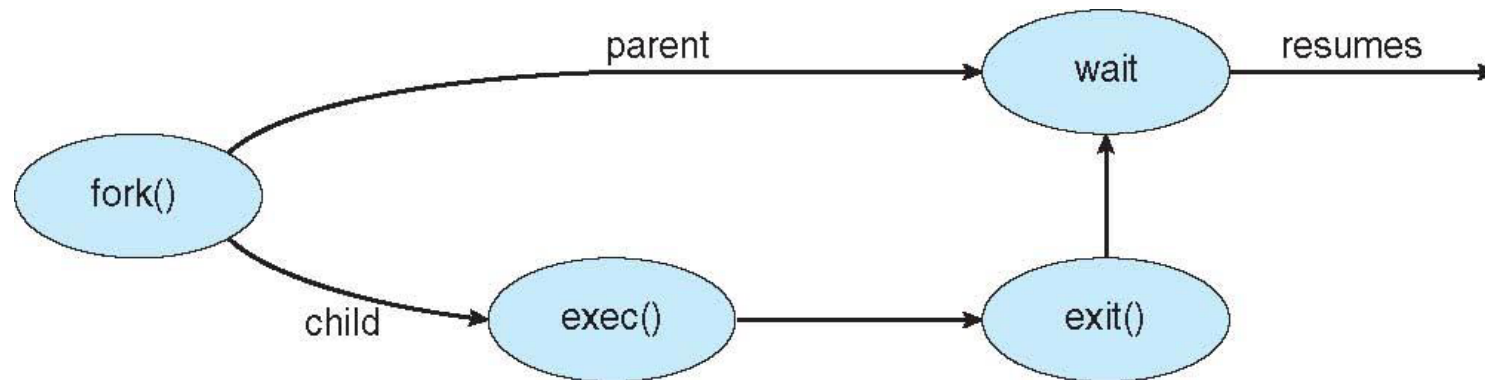
Process Termination

- Process executes 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 operating system

- Parent may terminate the execution of children processes using the `abort()` system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

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

  ```
  pid = wait(&status);
  ```

- If no parent waiting (did not invoke `wait()`) process is a **zombie**

- If parent terminated without invoking `wait`, process is an **orphan**
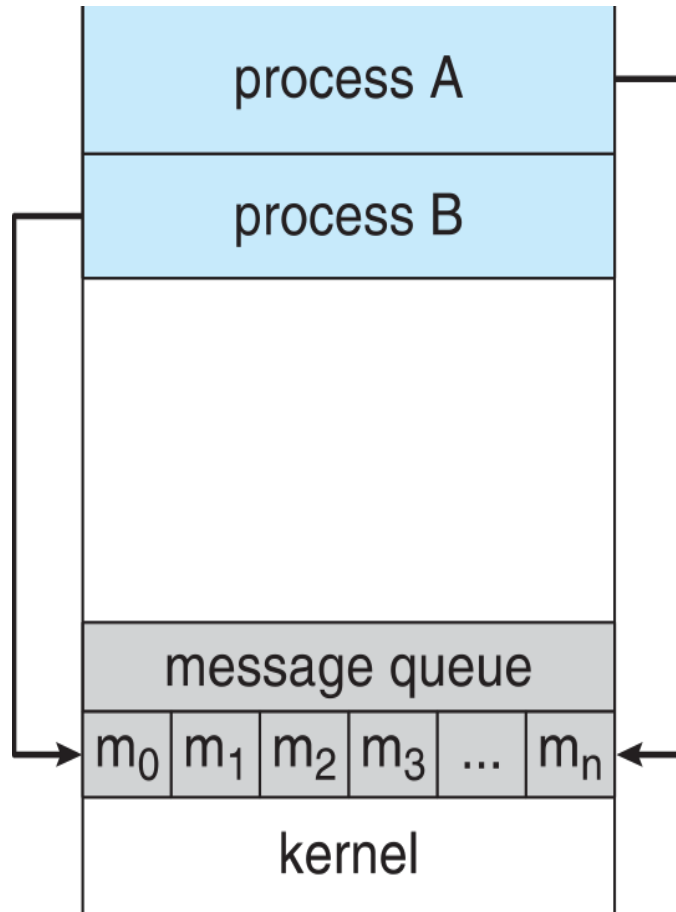
Interprocess Communication

- Processes within a system may be *independent* or *cooperating*

- *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

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

- Cooperating processes need **interprocess communication** (**IPC**)

- Two models of IPC
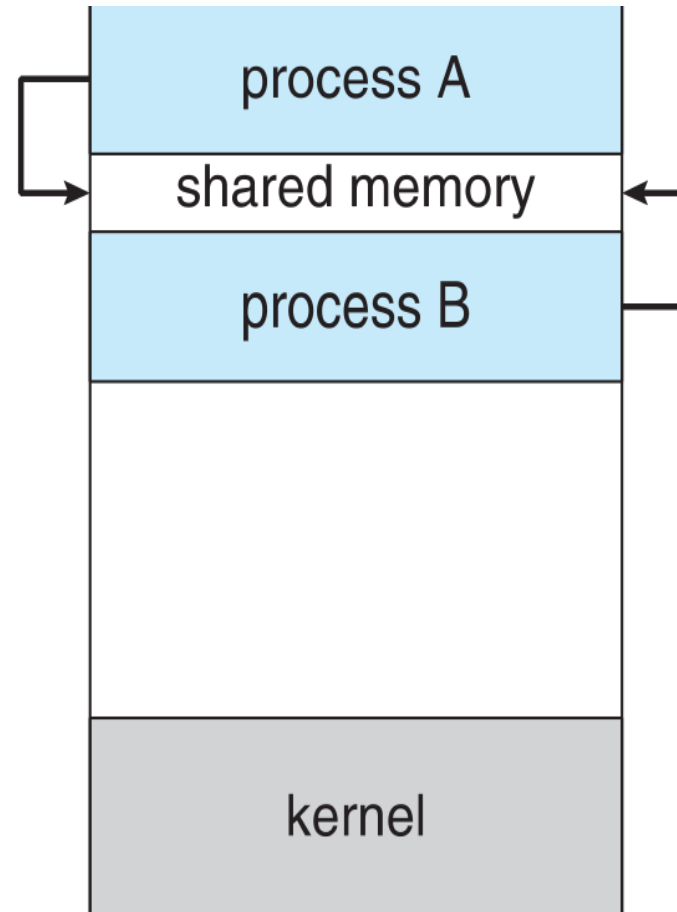   - **Shared memory**
    - **Message passing**

Communications Models

(a) Message passing. (b) shared memory.

(a)

(b)

Interprocess Communication – 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.

Interprocess Communication – 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
- The message passing can be:
  - Direct
  - Indirect
  - Synchronized

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 (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