Part II. Process Management

- Chapter 3 Processes
- Chapter 4 Threads
- Chapter 5 CPU Scheduling
- Chapter 6 Process Synchronization
- Chapter 7 Deadlocks

Chapter 3. Processes

- Process Concept
- Process Scheduling
- Operations on Processes
- Cooperating Processes
- Inter-Process Communication (IPC)

1. 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**
  - An instance of execution of a program that is loadable into the main memory.
  - Process execution must progress in sequential fashion.
- A process includes:
  - **Text section**
  - **CPU status** – **Program counter** and **registers**
  - **Stack** – for local variables
  - **Data section** – for global variables
  - **Heap** – dynamically allocated memory
  - See Figure 3.1.

Process State

- As a process executes, it changes ***state.***
  - **new**: The process is being created.
  - **ready**: The process is waiting to be assigned to a processor.
  - **running**: Instructions are being executed.
  - **waiting**: The process is waiting for some event to occur, e.g., i/o completion and reception of a signal.
  - **terminated**: The process has finished exec.ution.
  - See Figure 3.2.
  - ☺ ***Where is a process when it is in the ready state, or in the running state?***

Process Control Block (PCB)

- The operating system must know specific information about processes in order to manage and control them.
- **Process state information**
  - CPU registers
  - Program counter
- **Process control information**
  - CPU scheduling information
  - I/O status information
  - Memory-management information
  - Accounting information

Process Control Block - continued

- This collection of process information is kept in and accessed through the **Process control block (PCB)**, also called the task control block (TCB).
  - Process state – new, ready, …
  - Program counter
  - CPU registers
  - CPU scheduling information – priority, pointers to queues, …
  - Memory-management information – base and limit registers, addresses of …
  - Accounting information – the amount of CPU, time limits, account numbers, …
  - I/O status information – allocated i/o devices, open files, …
  - See Figure 3.3.
  - ☺ ***Where are they located?***
- ☺ ***What is a PCB?***

CPU Switch From Process to Process

- See Figure 3.4.
  - Operating system, especially process scheduler, that could be invoked from an interrupt service routine.
  - Interrupt from i/o devices or system call from any other process

2. Process Scheduling

- ☺ *Where are PCBs?*
  - Queues in main memory

- ***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

- Process migration between the various queues, by **process scheduler**
- See Figure 3.6.
- See Figure 3.7.

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
- See Figure 3.8.

Schedulers - continued

- Short-term scheduler is invoked **very frequently** (milliseconds) ⇒ (must be fast).
- Long-term scheduler is invoked very **infrequently** (seconds, minutes) ⇒ (may be slow).
- The long-term scheduler controls the *degree of multiprogramming* (the number of processes in main memory).
  - 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
  - ☺ *Which one first?*
  - ☺ *How to mix them?*

Context Switch

- Context of a process – PCB
- **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
  - E.g., several set of registers => switching of register sets => very fast

3. Operations on Processes

- Process creation
- Process termination

Process Creation

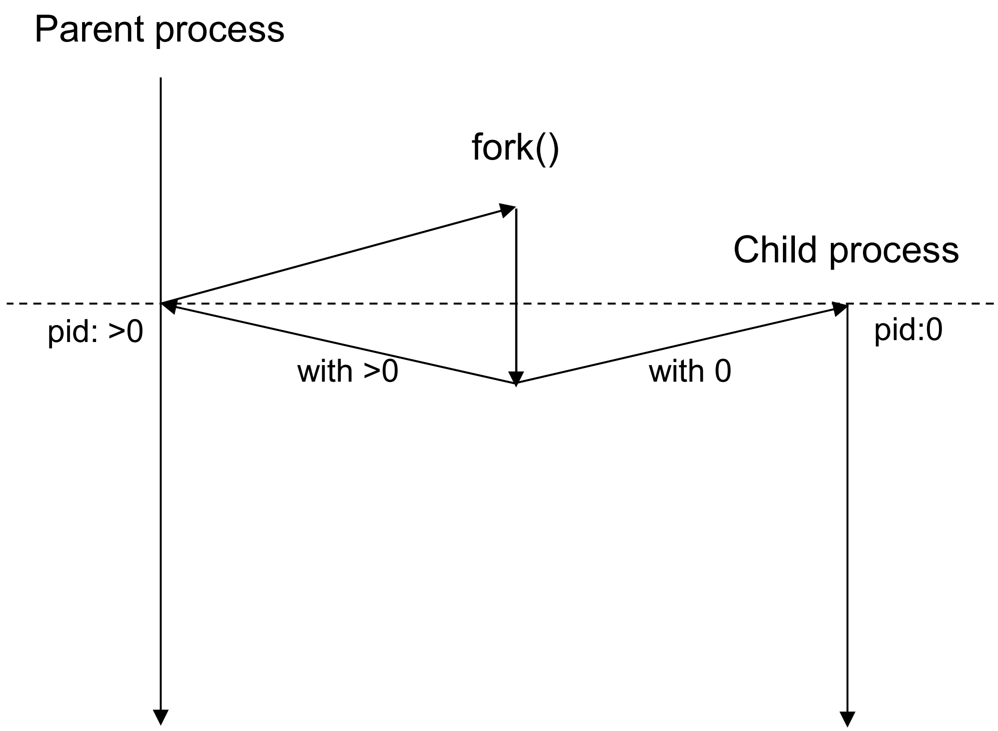
- **Parent process** creates **children processes**, which, in turn create other processes, forming a tree of processes.
- ☺ *What does creation of a process mean ???*
  - Creation of a new PCB, and put it into the job queue
- ☺ *How can a parent process create a child process???*
  - System call
- Resource sharing
  - Parent and children share all resources, or
  - Children share subset of parent's resources, or
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently, or
  - Parent waits until children terminate
- Address space
  - Child duplicate of parent, or
  - Child has a program loaded into it

Process Creation - continued

- ☺ *How to create?*
- UNIX examples
  - `fork()` system call creates a new child process
    - Same text section
    - Different data section, stack, heap
    - Different CPU status
  - `exec()` system call used after a **fork** to replace the process' memory space with a new program
    - Now, different text section
    - Different data section, stack, heap
    - Different CPU status
  - `wait()` system call used to move itself off the ready queue until the termination of the child
  - See Figure 3.10 – 3.11.

C Program Forking Separate Process

- `pid = fork();`



Process Termination

- ☺ *How to terminate?*
- Process executes last statement and asks the operating system to decide it (**`exit`**).
  - Output data from child to parent (via **`wait`**)
  - Process' resources including PCB 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***

4. Cooperating Processes

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

In the game, Tetris,
One process for dropping blocks
Another process for turning blocks
☺ *How can they operate?*

Producer-Consumer Problem

- Paradigm for cooperating processes
  - *Producer* process produces information that is consumed by a *consumer* process.
  - Need a buffer that can be filled by the producer and emptied by the consumer.
    - *Unbounded-buffer* places no practical limit on the size of the buffer.
    - *Bounded-buffer* assumes that there is a fixed buffer size.
    - What kind of data type would be proper?
  - The producer and consumer must be synchronized.

5. Interprocess Communication (IPC)

Interprocess Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions

- Message-passing 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()
  - Implementation of communication link
    - physical (e.g., shared memory, hardware bus)
    - logical (e.g., logical properties)

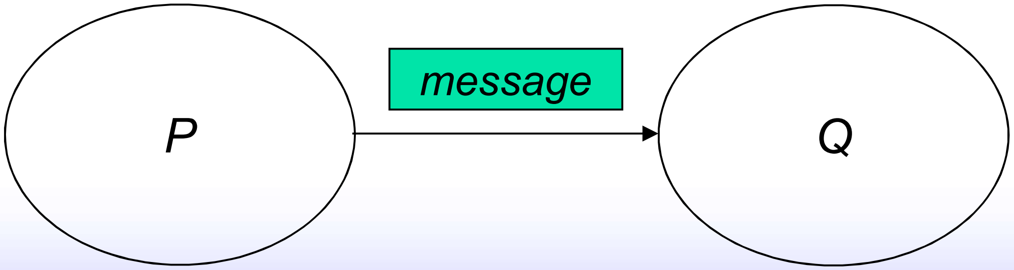
Implementation Questions

- 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?

- Two communication models
  - See Figure 3.14.

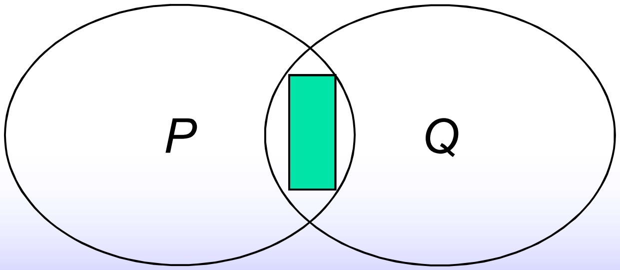
Direct Communication

- Concept of **message passing**
- 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

- Concept of **shared memory**
- 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 - continued

- 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

Indirect Communication - continued

- Mailbox sharing:
  - P_1, P_2, and P_3 share mailbox A.
  - P_1, sends; P_2 and P_3 receive.
  - Who gets the message?
- Solutions
  - Allow a link to be associated with at most two processes.
  - Allow only one process at a time to execute a receive operation.
  - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

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.
  - Also called *Rendezvous*
- **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.

In the game, Tetris,
One process for dropping blocks
Another process for turning blocks
☺ *Synchronous or asynchronous?*

Buffering

- Queue of messages attached to the link; implemented in one of three ways:
  1. Zero capacity – 0 messages
     Sender must wait for receiver (rendezvous).
  2. Bounded capacity – finite length of *n* messages
     Sender must wait if link full.
  3. Unbounded capacity – infinite length
     Sender never waits.

Other IPC Mechanisms

- UNIX
  - Pipes – FIFO
    - `$ cat /etc/passwd | grep mlee`
  - Signals
    - To notify a process of an event
    - Asynchronous
    - By one process to another, including itself; by the kernel to a process
    - `kill(int pid, int sig)`
    - `raise(int sig)`
    - `signal(int sig, SIGARG func)`
      - Can ignore or catch the signal, depending on `func`