ch2: Process

- Process: a program in execution
- Thread: a thread is the unit of execution within a process
- A process includes **program counter**, **stack**, **data section**

Process State

- **new**: being created
- **ready**: waiting to be assigned to CPU
- **running**: being executed
- **waiting**: waiting for some event to occur (e.g. I/O operations)
- **terminated**: finished execution

Process Scheduling

- **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 I/O
- **Scheduler**
    - Long term scheduler (job scheduler) : select which processes should be brought into the ready queue
        - invoked infrequently
    - Short-term scheduler (CPU scheduler) : select which process should be executed next and allocates CPU.
        - invoked frequently
- **CPU-bound** vs **I/O-bound**
    - CPU-bound spends more time doing computations
        - **Few**, **long** CPU bursts
    - i/o-bound spends more time doing I/O rather than computations
        - **Many**, **short** CPU bursts

Context Switch

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

Process Creation

- Child process may duplicate parent's memory space or load new memory contents (load a new program)
    - **fork** system calls create new processes:
        - In parent process: **fork()** returns **PID of the created child process**
        - In child process: **fork()** returns **0**
    - **exec** system calls is used after fork to replace the process' memory space with a new program and execute
- Steps:
    - **Load** a program code into the memory
    - the program's run-time **stack** is allocated
    - the program's **heap** is created
    - OS does some other initialization tasks
    - **start** the program running at main()

Process Termination

- Process finishes and asks the OS to delete it (**exit**)
    - Output data from child to parent (**wait**)
    - Process's resources are **de-allocated** by OS
- Parent may terminate child process (**abort**)
    - child has exceeded allocated resources
    - task assigned to child is no longer required
    - parent is exiting (cascading termination)

Process Cooperation

- **Independent** process cannot affect each other while **cooperating** can.
- **Advantages**:
    - Information sharing
    - Computation speed-up
    - Modularity (easy to maintain)

Inter-process Communication (IPC)

- Two operations: **send** and **receive**
- **Communication Models**
    - **Message Passing** & **Shared Memory**

Message Passing	Shared Memory
used for process communication	used for process communication
could be used in a distributed environment, like remote machines or network	for processes on a single processor or they are on the same machine
OS provides this mechanism and synchronization	requires the programmer to write programs to explicitly coordinate the access of data
time consuming	time efficient

Direct Communication

- Must name each other explicitly
    - send (p, message)
    - receive (q, message)
    - Properties:
        - Links are established automatically
        - A link is associated with one pair of communicating process
        - Between each pair there exists exactly one link
        - The link may be undirectional, but usually bi-directional

Indirect Communication

- Messages are directed and received from mailboxes (ports)
    - Each mailbox has a unique id
    - processes can communicate only if they share a mailbox
- Properties:
    - Link establishes only if processes share a common mailbox
    - A link may be assocaited with many processes
    - Each pair of processes may share several communication links
- Mailbox operations: **create**, **send and receive**, **destroy**
- Primitives:
    - send(A, message): send a message to mailbox A
    - receive(A, message): receive a message from mailbox A
- Hazards:
    - p1, p2, p3 share the same mailbox, when p1 sends a message, who gets it from the mailbox?
        - Allow a link to be established with at most two processes
        - Allow only one process at a time to execute a receive operation

        - Allow the system to select the receiver

Synchronization

- Message passing may be either **blocking** or **non-blocking**
    - **blocking** is considered **synchronous**
    - **non-blocking** is considered **asynchronous**

Client-Server Communication

- Internet socket: An endpoint for communication
    - The socket **161.25.19.8:1625** refers to port **1625** on host **161.25.19.8**
- Unix sockets:
    - Usually the same function calls as Internet sockets
    - all communication occurs entirely within OS kernel, instead of using network protocol
    - two-way FIFO queues

Remote Procedure Calls (RPC)

- RPC abstracts procedure calls between processes on networked systems (allow processes on different machines to communicate)
- The client-side locates the server and ***marshalls*** the parameters
- The server-side receives this message, **unpacks** the marshalled parameters, and performs the procedures on the server