Chapter 3:

- Programs executed by the OS: 1- Batch system (Jobs) بشتغلوا بالترتيب
  2- Time Shared Sys (user program or tasks)
- Process: a program in execution
  - ① text section (code)
  - ② Program Counter
  - ③ Stack
  - ④ Data Section
  - ⑤ Heap
- (passive) Program becomes a process (active) when executable file loaded into memory.
- Process states: new, running, waiting, ready, terminated. (Scheduler dispatch; admitted)
- Process control Block (PCB):
  - Process State, Process number, program counter registers, memory limits, list of open files, Accounting info CPU Scheduling info, Memory management, I/O Status info
- Process Scheduler selects among available processes for next execution on CPU
- Scheduling queues: Job queue, Ready queue, Device queues.
- Short-term Scheduler (CPU Scheduler) (selects among which process should be executed next and allocates CPU)
  - 1) Sometimes the only scheduler in the system
  - 2) IS invoked frequently (milliseconds - fast)
- Long-term Scheduler (Job Scheduler) - Strives for good process mix - next page
  - 1) Selects which processes should be brought into the ready queue.
  - 2) Invoked infrequently (seconds, minutes)
  - 3) Control degree of multiprogramming

- Processes can be either:
  - 1) I/O-bound process (has short CPU burst)
  - 2) CPU-bound Process (has longer CPU burst)

- Medium-term scheduler: (used if degree of multiprogramming needs to decrease)
  - Swapping (Remove from memory → Store on disk → bring back from disk)

- foreground - controlled via user (Single usually)
  background process - in memory, running but not on the display and with limits (Multiple)

- Context switch: Save the state of the old process → load the saved state of the new process
- Context of a process represented in the PCB
- Context Switch time is overhead.
  - time dependent on hardware support.

⇒ multiple sets of registers per CPU → multiple contexts loaded at once
Process identified and managed via a process identifier (id)

- when having a parent process and a child process
  - 1) Resource sharing options: [Share all or Share subset or Share nothing]
  - 2) Execution options: [Concurrently, Sequentially]

- Address Space: → Child duplicate of parent or Child has a program loaded into it.

- In UNIX: 1) Fork() System call creates new process
Fork() returns a zero for the newly created child, and a positive value to the parent.
2) exec() system call used after fork()
⇒ replaces the process' memory space with a new program.

- exit() System call asks the OS to delete the process after executing the last statement.
  - 1) Returns status data from child to parent (via wait())
  - 2) Recources are deallocated.
- abort() System call to terminate the child via the parent.
- Cascading termination: All children, grand children, etc, are terminated.
- If no parent is waiting for the process then it is a Zombie

the parent process may wait for termination of a child process by using the wait() system call the call return status information and the pid of the terminated process.

- If parent termenated without invoking wait(), process is orphan
- Processes may be independent or cooperating process
  - Independent process cannot effect or be effected by the execution of another process
  - Cooperating process can be effect or be effected by other processes, Like sharing data.

→ For: 1) Information Sharing, 2) Computation Speedup
3) Modularity 4) Convenience

- they need inter-process communication (IPC)
  - 1) Shared Memory
  - 2) Message passing

- unbounded-buffer and bounded-buffer

→ in location (must be free) out location (cannot be free)
buffer is empty in = out
buffer is full (in + 1) % BUFFER-SIZE = out

- Shared memory: An area of memory shared among the processes that wish to communicate.
- the communication is under control of the users processes not the operating system

- Mechanism for processes to communicate and to Synchronize their actions.
- Message passing → IPC provides Send(message), receive(message))
- process communicate with each other without resorting to shared variables

Message system
- for P and Q to communicate (the message size is either fixed or variable)
  - ① Establish a communication link.
  - ② Exchange messages via send/receive.

- Physical communication link
  - Shared memory, System bus, Network

- logical Communication link — Automatic or
  - Direct or Indirect, Synchronous or Asynchronous, ~~Auto~~ explicit buffering

- Communication link properties: (Direct Communication)
  1) are established automatically
  2) A link is associated with one pair of comm. processes
  3) Between each pair there exists exactly one link.
  4) the link may be unidirectional or bi-directional. usually this

- Processes can communicate only if they share a mailbox
  - each mailbox has a unique id.

- Communication link properties: (InDirect)
  1) link established only if processes share a common mailbox.
  2) a link may be associated with many processes.
  3) each pair may share several communication links.
  4) Link may be bidirectional or unidirectional.

- Create a new mailbox(Port) → Send and receive → destroy a mailbox

direct — process who send or receive

Send(X, message) Receive(X, message) Solution for who gets the message

= mail box in indirect

Blocking send: the sender is blocked until the message is received
Blocking receive: the receiver is blocked until a message is available
non-Blocking send: the sender sends the message and continue.
non-Blocking receive: the receiver receives: A valid message or Null message

Chapter 3- Cont:

- Message passing may be Blocking or non-Blocking (considered synchronous / is considered Asynchronous)
- if both send and receive are blocking we call it rendezvous
- Implementation of queue of messages (Buffering)
  - 1) Zero capacity (Sender must wait for receiver)
  - 2) Bounded capacity (Sender must wait if link is full)
  - 3) Un-Bounded capacity (Sender never wait)
- Create a Shared memory POSIX (Sender →) shm-fd = shm-open(name, O CREATE | O RDWR, 0666);
  - Shm fd = Shm open(name, O RDONLY, 0666); → receiver
  - Set the size of that Shared memo ftruncate (shm_fd, size);
  - (Sprintf) to write to it ⇒ sprintf(shared memory, "writing to shared memo");
- System calls needed for message transfer: msg-send(), msg-receive(), msg-rpc()
  - /* memory map the shared object */ ptr: mmap(0, size, PROT, WRITE, MAP SHARED, shm fd, 0);
- Create a mailbox: port_allocate()
- If mailbox is full → 1) wait infinitely
  - 2) wait at most n milliseconds
  - 3) return immediately
  - 4) temporarily cache a message.
- Message-passing centric via advanced local procedure call (LPC)
  - 1) works between processes on the same System (only)
  - 2) uses Ports (mail box for ex) to establish and maintaine comm... Channels
  - 3) Communicate as the following.
    - the client opens a handle to the sub-sys connection port object → client sends Connection request
    - the server creates two private Communication ports, and return the handle to one of them to the client
    - the client and server uses the port handle to communicate.

- Socket is defined as an endpoint of communication.
  - Concatination of IP address and port.
- the Soket 161.25.19.8 : 1625 port on host 161.25.19.8
- all ports below 1024 are well-known used for standard services
- three types of Socket: 1) connection-oriented (TCP)
  2) Connectionless (UDP)
  3) Multicast Socket
- Remote Procedure call (RPC): abstract procedure calls processes on network systems.

Stubs - client-side proxy for the actual procedure on the server.
The client-side stub locates the server and marshalls the parameters
The Server-side stub this message, unpacks the marshalled parameters, and performs the procedure on the Server
Remote communication has more failure scenarios than local Messages can be delivered exactly once rather than at most once

- Pipes: allows two processes to communicate. Standard Producer-Consumer Style
  - ordinary pipe: 1) cannot be accessed from outside the process.
    (anonymous pipes) 2) a parent creates it to communicate with its child.
    3) are unidirectional
  - named pipe: 1) can be accessed without a parent-child relation
    2) Communication is bidirectional
    3) can be used by several pipes

No parent child relationship is necessary between the Communicating processes.

- Proceduer writes to one end (the write end of the pipe)
  Consumer reads from the other end (the read end of the pipe)