Chapter 3 summary

- An operating system executes a variety of programs:
  - Batch system – **jobs**
  - Time-shared systems – **user programs** or **tasks**
- Program is **passive** entity stored on disk (*executable file*); process is **active.**
  - Program becomes process when executable file loaded into memory.
- One program can be **several** processes.
- Process includes multiple parts:
  - The program code, also called **text section**.
  - Current activity involving **program counter**, processor registers.
  - **Stack** containing temporary data.
  - **Data section** containing global variables.
  - **Heap** containing memory dynamically allocated during run time.
- 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 processor.
  - **terminated**: The process has finished execution.
- Process Control Block (PCB):
  - **Process state** – running, waiting, etc.
  - **Program counter** – location of instruction to next execute.
  - **CPU registers** – contents of all process-centric registers.
  - **CPU scheduling information** – priorities, scheduling queue pointers.
  - **Memory-management information** – memory allocated to the process.
  - **Accounting information** – CPU used, clock time elapsed since start, time limits.
  - **I/O status information** – I/O devices allocated to process, list of open files.
- Process Scheduling:
  - Maximize CPU use, quickly switch processes onto CPU for time sharing.
  - **Process scheduler** selects among available processes for next execution on CPU.
  - Maintains **scheduling queues** of processes:
    - **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.
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the <u>ready queue</u>. (may be slow)

- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU. (must be fast)
- The long-term scheduler controls the **degree of multiprogramming**.
- Processes can be described as either:
    - **I/O-bound process** – spends more time doing <u>I/O</u>, <u>many</u> <u>short</u> CPU bursts.
    - **CPU-bound process** – spends more time doing <u>computations</u>; <u>few</u> <u>very</u> <u>long</u> CPU bursts.
- Long-term scheduler strives for good **process mix**.
- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease.
    - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**.
- <u>Context switch:</u>
    - 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 via a **context switch**. (PCB)
    - Context-switch time is **overhead**; the system does no useful work while switching.
    - The more complex the OS and the PCB, longer the **context switch**.
    - Time dependent on **hardware support**.
- **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)**.
- <u>Resource sharing options:</u>
    - Parent and children share all resources.
    - Children share subset of parent's resources.
    - Parent and child share no resources.
- <u>Execution options:</u>
    - Parent and children execute concurrently.
    - Parent waits until all or some children terminate.
- <u>Address space:</u>
    - Child duplicate of parent program.
    - Child has a program loaded into it.
- <u>UNIX examples:</u>
    - **fork()** system call creates new process.
    - **exec()** system call used after a **fork()** to replace the process' memory space with a new program.

- Process executes last statement and asks the operating system to delete it **exit()**.
- Parent may terminate execution of its children processes **abort()** for a variety of reasons:
    - Child has exceeded allocated resources.
    - Task assigned to child is no longer required.
    - If parent is exiting:
        - Some operating systems do not allow child to continue if its parent terminates.
        - All children must also terminated - **cascading termination**.
- Processes within a system may be **independent** or **cooperating**.
- Independent process <u>cannot</u> affect or be affected by other processes.
- Cooperating process <u>can</u> affect or be affected by other processes, including **sharing data**. Also, require an **interprocess communication (IPC)** mechanism.
    - Two models of IPC:
        - Message passing.
        - Shared memory.
- **Producer-Consumer Problem**: producer process produces information that is consumed by a consumer process.
- One solution to allow them to run concurrently, uses a buffer of items (**shared memory**) that can be filled by producer and emptied by consumer:
    - **Unbounded-buffer** places no practical limit on the size of the buffer.
    - **Bounded-buffer** assumes that there is a fixed buffer size.
- Message passing – processes communicate with each other without resorting to shared variables.
    - IPC facility provides at least 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.
- <u>Direct communication:</u>
    - 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 communications:
    - 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**.
- 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.
    - If both send and receive are blocking, we have a **rendezvous**.
- Buffering:
    - Queue of messages attached to the link; implemented in one of three ways:
        - **Zero capacity – 0 messages**
            - Sender must wait for receiver (rendezvous).
        - **Bounded capacity – finite length of n messages**
            - Sender must wait if link full.
        - **Unbounded capacity – infinite length**
            - Sender never waits.
- Shared memory and message passing strategies can be used for communication in client–server systems as well.
- A socket is an endpoint for communication. A socket is identified by an **IP address** concatenated with a **port number**.
- The server waits for incoming client requests by listening to a specified port
    - All ports below 1024 are **well known**, used for standard services
    - To allow a client and server on the same host to communicate, a special IP address 127.0.0.1 (**loopback**) is used to refer to itself.
- Three different types of sockets in java:
    - **Connection-oriented (TCP)**
    - **Connectionless (UDP)**
    - **MulticastSocket class** – data can be sent to multiple recipients.

- Remote procedure call (RPC) abstracts procedure calls between processes on networked 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 receives this message, unpacks the **marshalled** parameters, and performs the procedure on the server.
- **Pipes**: acts as a channel allowing two processes to communicate.
- **Ordinary Pipes** allow communication in standard producer-consumer style.
    - Allow only **unidirectional** communication.
- Require **parent-child** relationship between communicating processes.
- UNIX treats it as a special type of file and can be accessed using **read()** and **write()** system calls.
- Named pipes are more powerful tool than ordinary pipes:
    - Communication can be bidirectional.
    - No parent–child relationship is required.
    - Several processes can use it for communication.
- Both UNIX and Windows systems support it, although implementation details differ greatly.