Process Concept

- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – also called user programs or tasks
- Job – A program in execution. A job can contain some processes.
- Process – execution must progress in sequential fashion.
- A process includes:
  - Text/Code section.
  - Data section.
  - Stack section.

Process Creation

- Parent process creates children processes, which, in turn create other processes, forming a tree of processes.
- Resource sharing
  - Parent and children share all resources.
  - Children share subset of parent’s resources.
  - Parent and child share no resources.
- Execution
  - Parent and children execute concurrently.
  - Parent waits until children terminate.
- Address space
  - Child duplicate of parent.
  - Child has a program loaded into it.

A Tree of Processes On A Typical UNIX System

Process Termination

- Process executes last statement or asks the operating system to decide it (exit).
  - Output data from child to parent (via exit & wait).
  - Process’ resources are deallocated by the operating system.
- Any process may terminate execution of another processes of the same user. (kill).
- When Parent is exiting.
  - UNIX declares on its child as orphan and “init” adopts this child.
  - There are some operating systems which do not allow child to continue if its parent terminates. They make cascading termination.
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 process.
- terminated: The process has finished execution.

Process State in Linux
- R running or ready.
- S sleeping (=waiting).
- T Stopped.
- Z a defunct ("zombie") process.
- D uninterruptible sleep (IO).

Process Control Block (PCB)
Information associated with each process.
- Process state
- Process number
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O information (open files)
**CPU Switch From Process to Process**

- **Saving a Process:**
  - Process $P_0$ is executing.
  - Operating system receives an interrupt or system call.
  - State of $P_0$ is saved into its PCB.

- **Reloading a Process:**
  - Process $P_1$ is in an idle state.
  - Operating system receives an interrupt or system call.
  - State of $P_1$ is loaded from its PCB.

**Process Scheduling Queues**

- **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.

- **Processes Migrate:**
  - Processes migrate between the various queues.

**Ready Queue And Various I/O Device Queues**

- **Ready Queue:**
  - Contains processes that are ready to run.

- **I/O Device Queues:**
  - Include queues for mag tape unit 0, mag tape unit 1, disk unit 0, and terminal unit 0.

**Representation of Process Scheduling**

- **Process States:**
  - Ready queue
  - I/O queue

- **Events:**
  - I/O request
  - Time slice expired
  - Child executes
  - Fork a child
  - Interrupt occurs
  - Wait for an interrupt

- **Process Execution Flow:**
  - Process moves from ready queue to I/O queue, then to CPU when it gets I/O request.
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.
- In UNIX the Long-term scheduler does not exist.
- 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.
- 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.

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.

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

Producer-Consumer - Shared-Memory Solution

- Producer process produces information that is consumed by a Consumer process. How to implement?
- Shared data
  \[
  \begin{align*}
  &\text{var } n; \\
  &\text{type } \text{item} = \ldots; \\
  &\text{var } \text{buffer}}. \text{array}[0..n-1] \text{ of } \text{item}; \\
  &\text{in, out}: 0..n-1;
  \end{align*}
  \]
- Producer process
  ```
  \begin{align*}
  &\text{repeat} \\
  &\quad\ldots \\
  &\quad\text{produce an item in nextp} \\
  &\quad\ldots \\
  &\quad\text{while } in+1 \mod n = out \text{ do } \text{no-op} \\
  &\quad\text{buffer}[in] :=\text{nextp} \\
  &\quad\text{in} :=\text{in+1 \mod n} \\
  &\text{until false;}
  \end{align*}
  ```
Bounded-Buffer (Cont.)

- Consumer process

  ```
  repeat
      while in = out do no-op;
      nextc := buffer [out];
      out := out+1 mod n;
      ... consume the item in nextc
      ... until false;
  ```

  - Solution is correct, but can only fill up n–1 buffer.

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.

Indirect Communication

- Messages received from the resource itself.
  - Each resource has a unique id.
  - Processes can communicate only if they share a resource.

- Properties of communication link
  - Link established only if processes share a common resource.
  - A link may be associated with many processes.

- Operations
  - create a new resource.
  - send and receive messages through resource.
  - destroy a resource.

Synchronization

- Message passing may be either blocking or non-blocking.

- **Blocking** is considered *synchronous*

- **Non-blocking** is considered *asynchronous*

- `send` and `receive` primitives may be either blocking or non-blocking.
Buffering

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

- In UNIX there are:
  - Pipes – Bounded capacity
  - Messages – Unbounded capacity