Information Technology Institute

Operating System Fundamentals

Chapter Four

PROCESSES

Table of Content

- Process Concept
- Process Scheduling
- Operations on Processes
- Threads
- Cooperating Processes
- Inter-process Communication

PROCESS CONCEPT

Process Concept

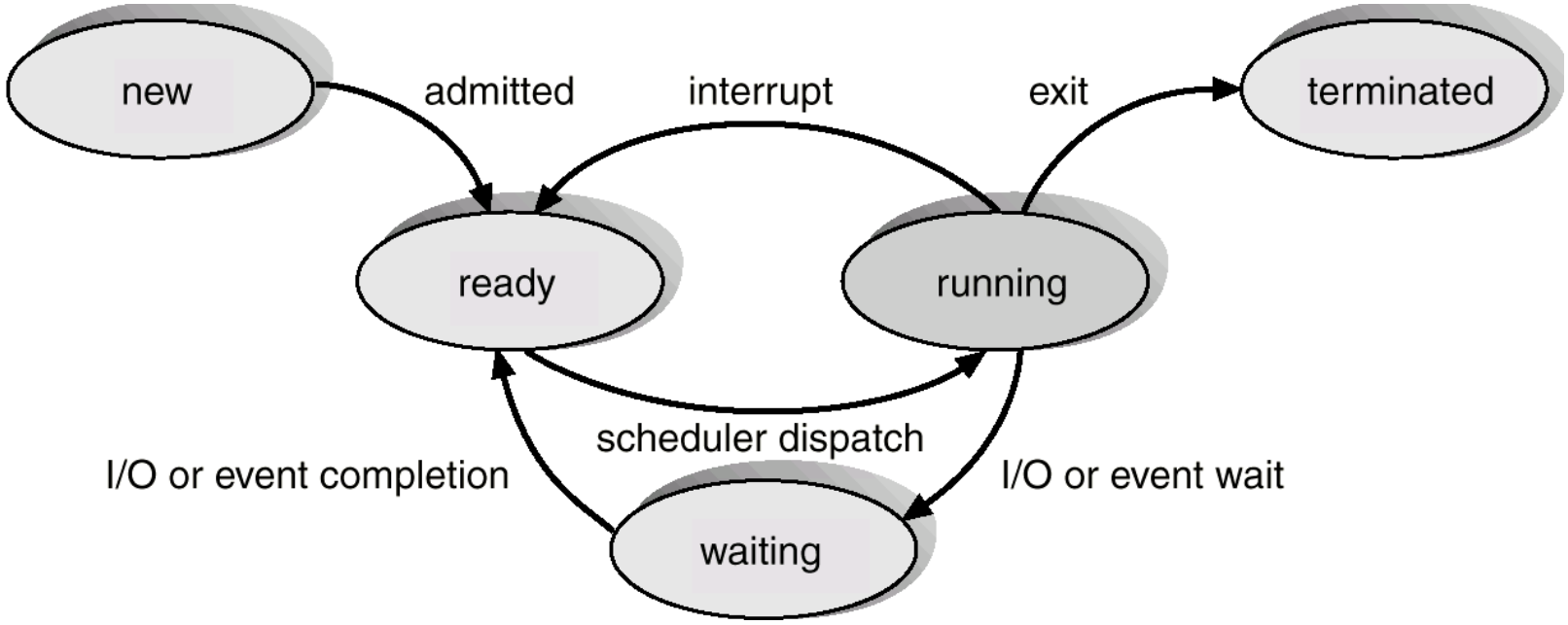
- Process – a program in execution; process execution must progress in sequential fashion.
- An operating system executes a variety of programs:
  - Batch system
  - Time-shared systems

* Textbook uses the terms job and process almost interchangeably

Process Contents

- Text section
  - Program instructions
- Program counter
  - Next instruction
- Stack
  - Local variables
  - Return addresses
  - Method parameters
- Data section
  - Global variables

Process State



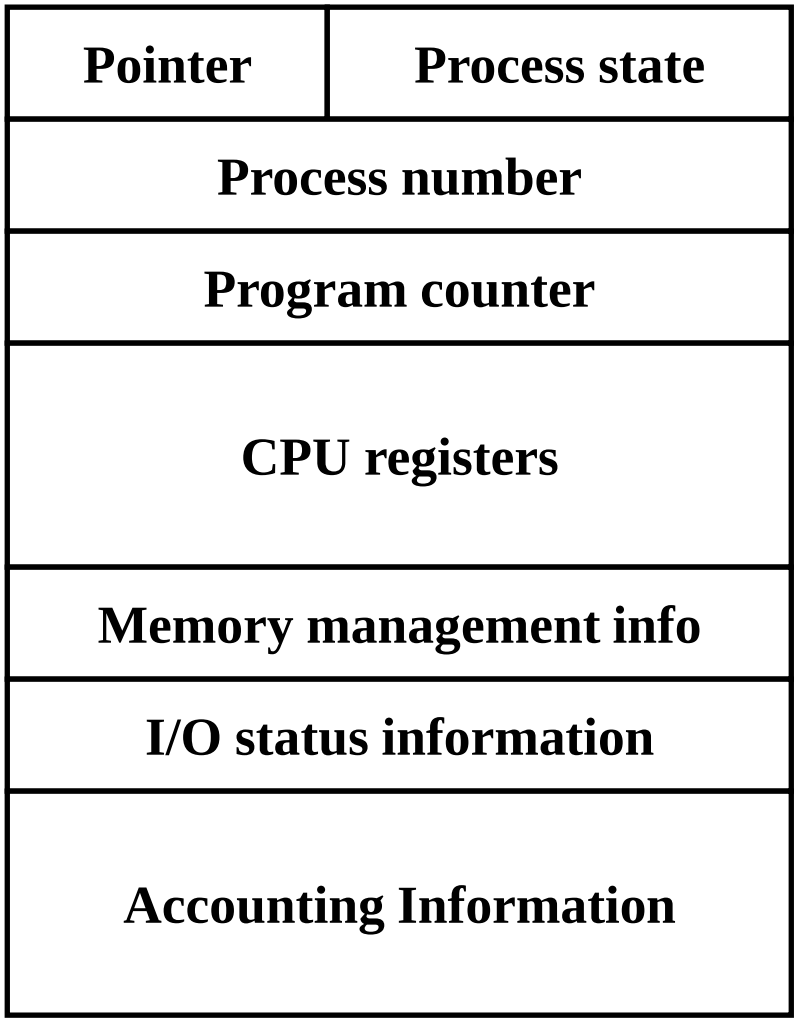
Process State Cont'd

- 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.

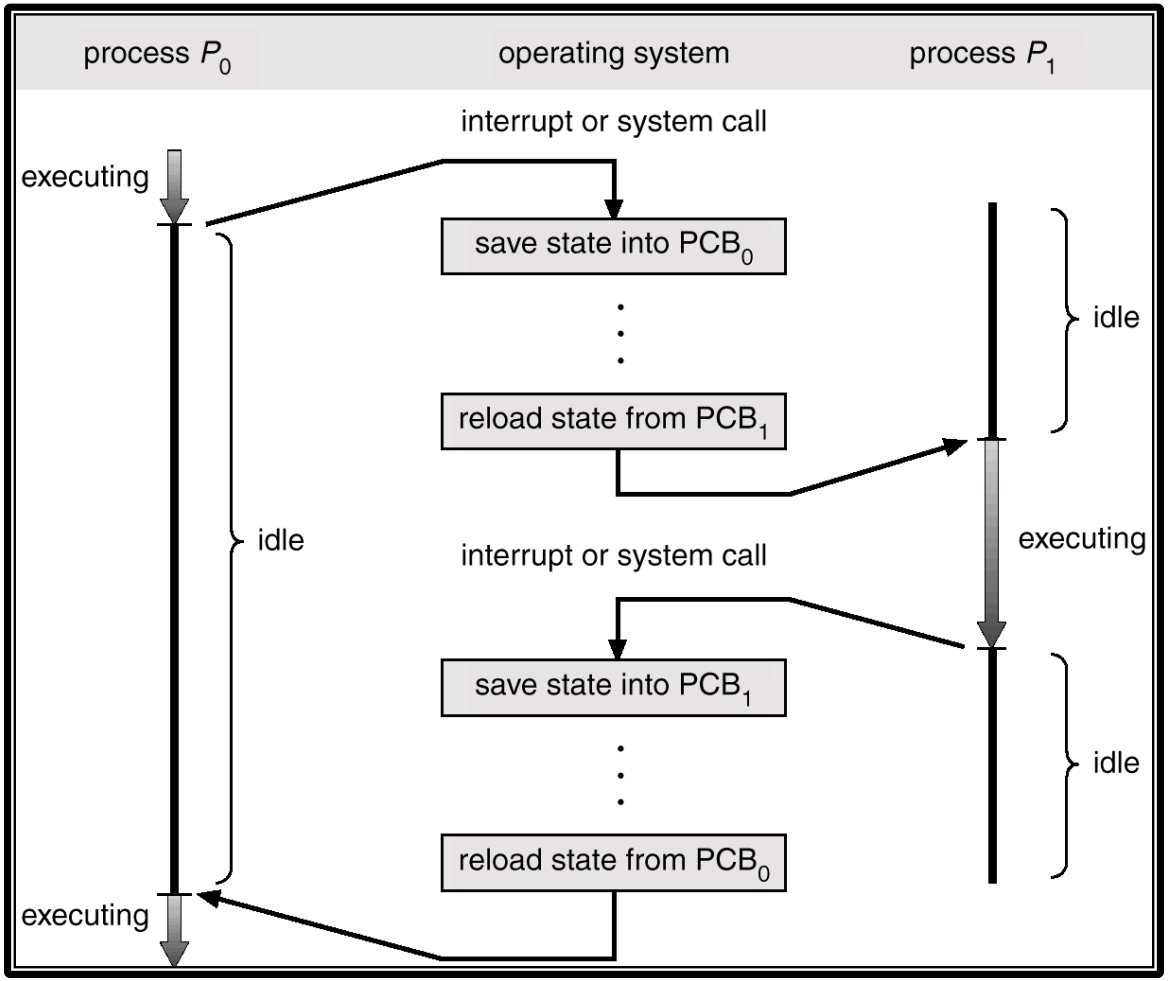
Process Control Block (PCB)

- Information associated with each process.
  - Process state
  - Program counter
  - CPU registers
  - CPU scheduling information
  - Memory-management information
  - Accounting information
  - I/O status information

Process Control Block (PCB) Cont'd



CPU Switch From Process to Process



PROCESS SCHEDULING

Process Scheduling

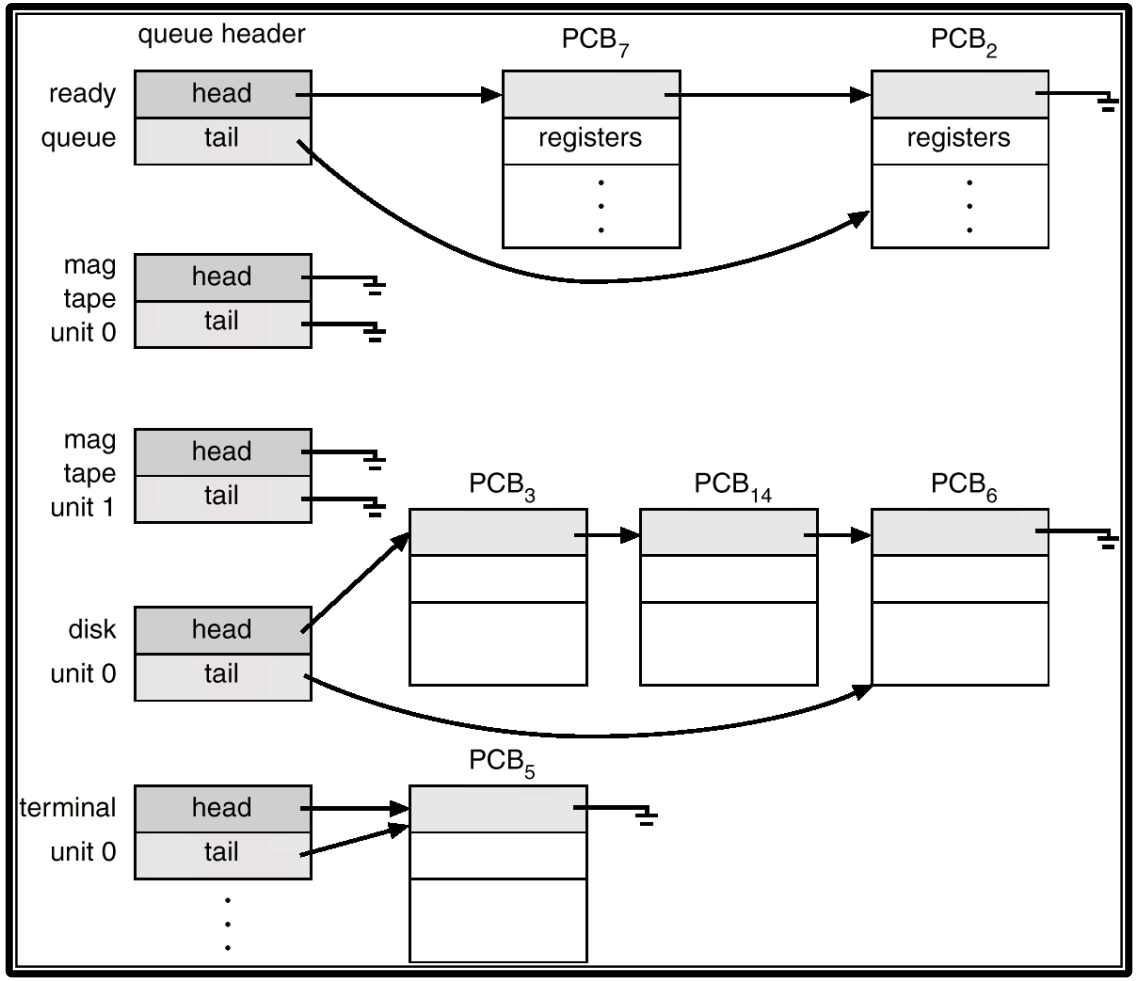
- Multi-Programming systems
  - Some processes executing at all times
  - Maximize CPU utilization
- Time-Sharing systems
  - Switch the CPU among processes frequently
  - Users can interact with a program while it’s executing
  - Virtually run processes at the same time

Process Scheduling Queues

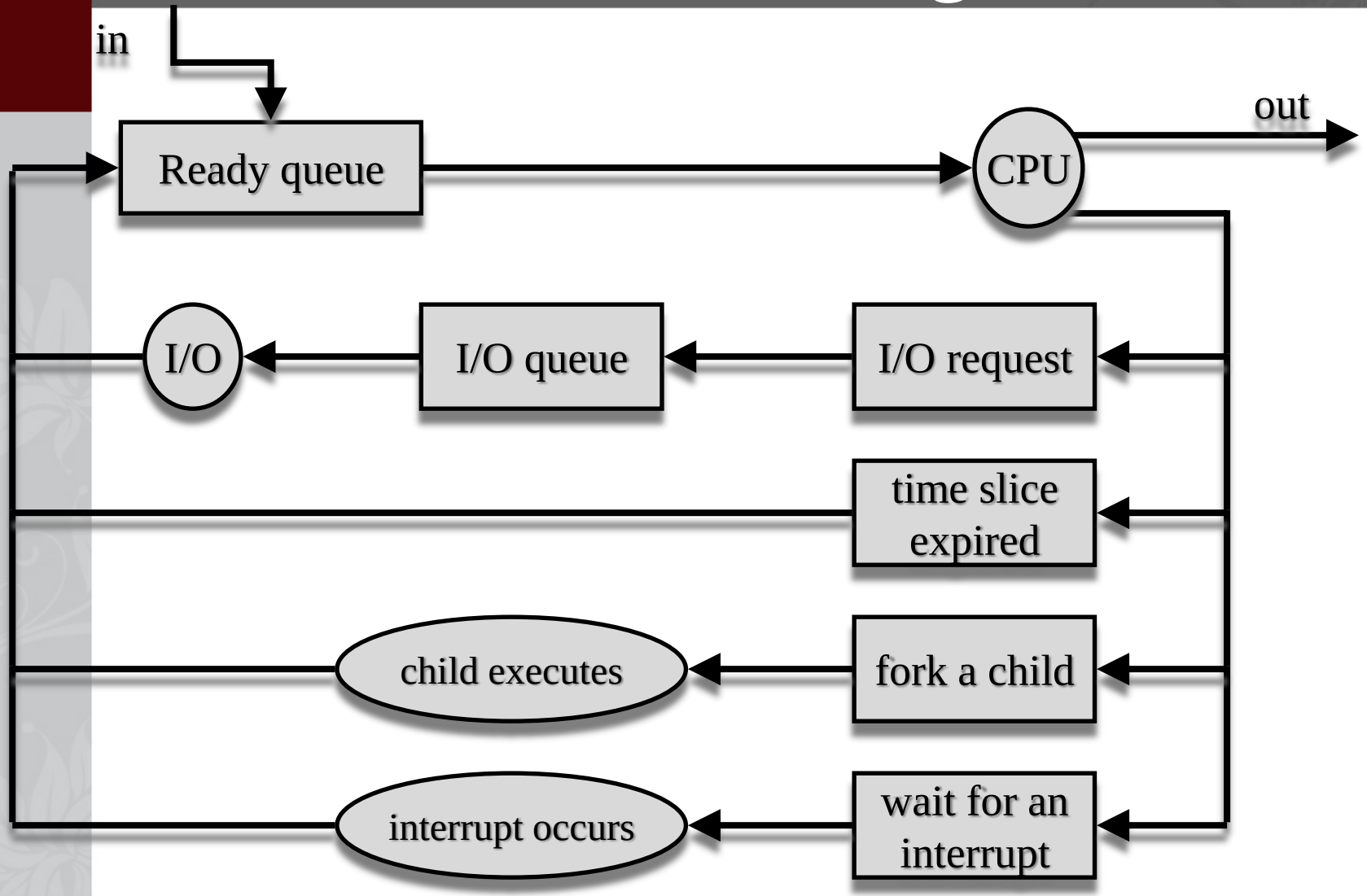
- Job queue
  - set of all processes in the system.
- Ready queue
  - set of all processes residing in main memory which is ready and waiting to execute.
- Device queues
  - set of processes waiting for an I/O device.

* Process migration between the various queues

Ready Queue And Various I/O Device Queues



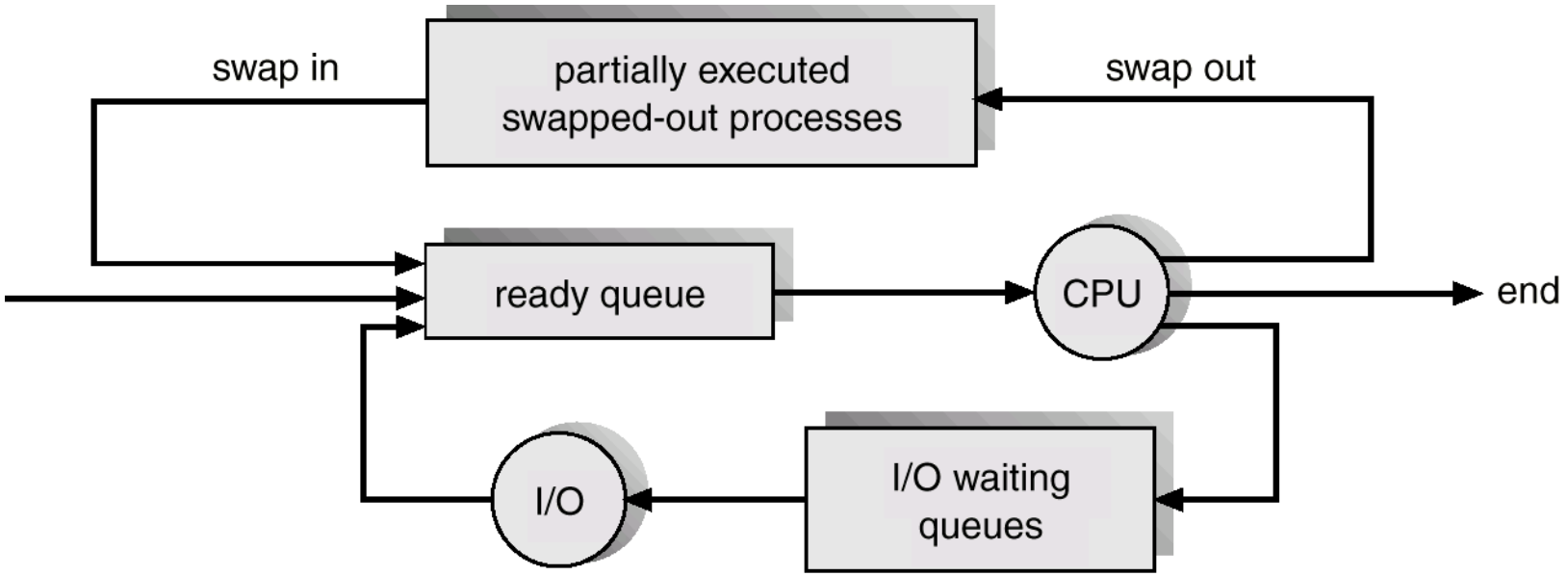
Representation of Process Scheduling



Schedulers

- Long-term scheduler (or job scheduler)
  - Selects which processes should be brought into the ready queue.
  - Executes infrequently
  - May be absent in some O/S
- Short-term scheduler (or CPU scheduler)
  - Selects which process should be executed next and allocates CPU.
  - Executes frequently

Addition of Medium Term Scheduling



Schedulers Cont'd

- 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.
- Proper System performance
  - Mix of CPU & I/O bound processes
- Improper system performance
  - All processes are I/O bound
  - All processes are CPU bound

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.

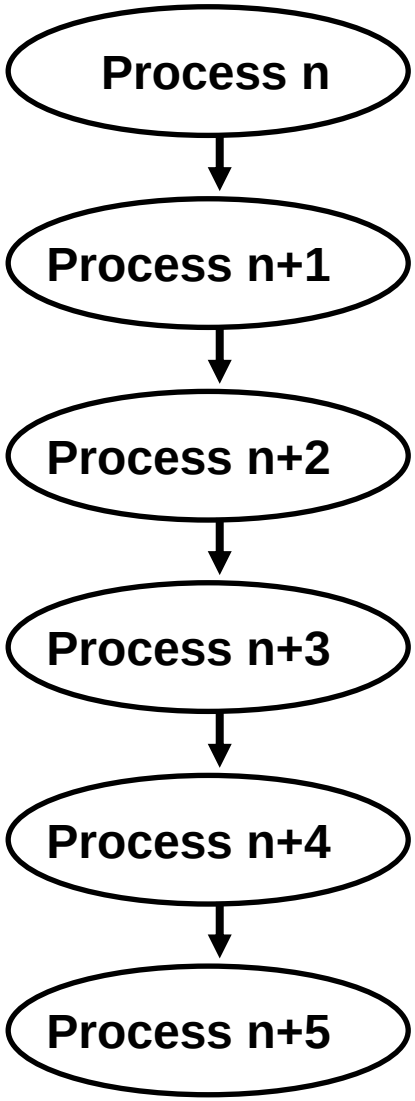
OPERATIONS ON PROCESSES

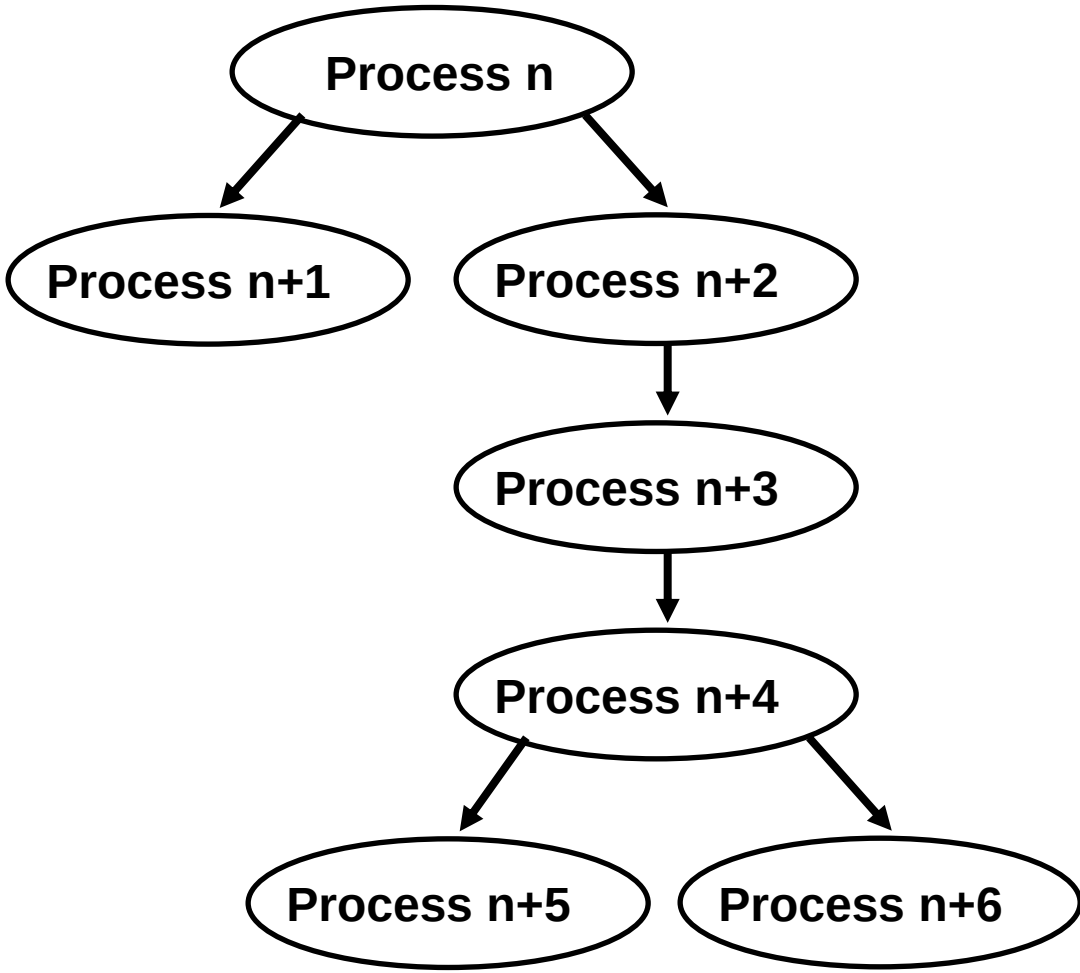
Operations on Processes

- Process Creation
- Process Termination

* Process resources
  - CPU time
  - Memory
  - Files
  - I/O devices

Process Creation





Process Creation Cont'd

- Parent process create 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.
- Execution
  - Parent and children execute concurrently.
  - Parent waits until children terminate.

Process Creation Cont'd

- Address space
  - Child duplicate of parent.
  - Child has a program loaded into it.
- UNIX examples
  - **fork** system call creates new process
  - **exec** system call used after a **fork** to replace the process' memory space with a new program.

Process Termination

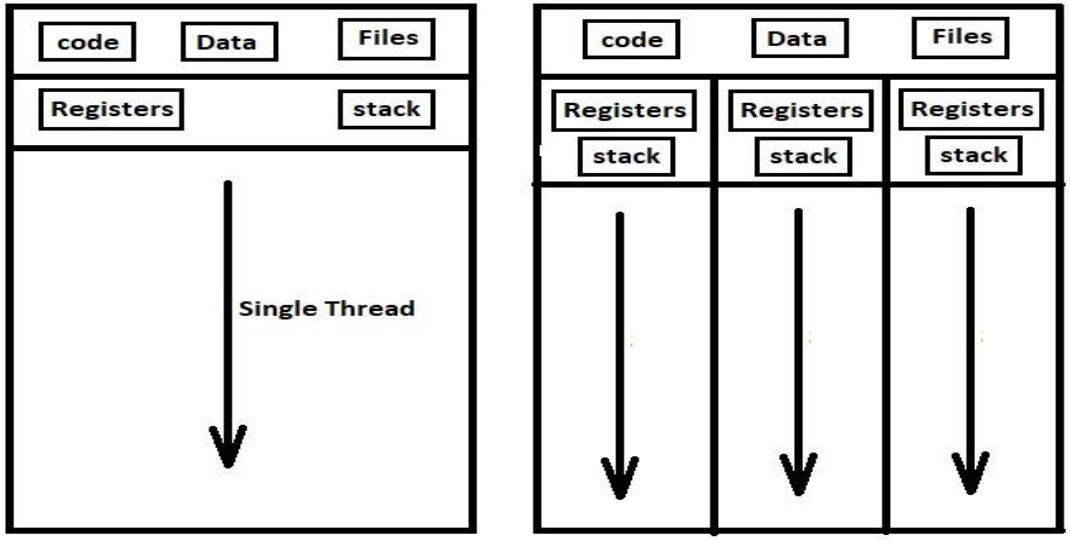
- Process executes last statement and asks the operating system to decide it (**exit**).
  - Output data from child to parent.
  - Process' resources are de-allocated by operating system.
- Parent may terminate execution of children processes (**abort**).
  - Child has exceeded allocated resources.
  - Task assigned to child is no longer required.
  - Parent is exiting.
    - Operating system does not allow child to continue if its parent terminates.
    - Cascading termination.

Threads

- Many modern operating systems now provide features for a process to contain multiple threads of control instead of a single Thread
- A thread, sometimes called a lightweight process (LWP)

Threads

- It shares with other threads belonging to the same process its **code section**, **data section**, and other operating-system resources, such as **open files**



Threads

- Benefits:

1. Responsiveness

2. Resource sharing

3. Economy

4. Utilization of multiprocessor architectures

Multi Threading Models

- Many-to-One Model:
  - Maps many user-level threads to one kernel thread. Thread
  - Management is done in user space, so it is efficient
  - The entire process will block if a thread makes a blocking system call
  - Multiple threads are unable to run in parallel on multiprocessors

Multi Threading Models

- One-to-one Model:
  - Maps each user thread to a kernel thread.
  - It provides more concurrency than the many-to-one model.
  -
  - Another thread can run when a thread makes a blocking system call.
  -
  - It also allows multiple threads to run in parallel on multiprocessors.
  - The only drawback is the overhead of creating kernel threads can burden the performance of an application.

Multi Threading Models

- Many-to-Many Model:
  - Multiplexes many user-level threads to a smaller or equal number of kernel threads.
  - Developers can create as many user threads as necessary, and the corresponding kernel threads can run in parallel on a multiprocessor.
  - Also, when a thread performs a blocking system call, the kernel can schedule another thread for execution.

Threading Issues

- The fork and exec System Calls
  - Does the new process duplicate all threads or is the new process single-threaded?
- Cancellation
  - Allocating resources to a cancelled thread or cancel a thread while updating data that is shared by another thread
- Signal Handling
  - Handling signals in single-threaded programs is straightforward; signals are always delivered to a process. However, delivering signals is more complicated in multithreaded programs, as a process may have several threads. Where then should a signal be delivered?
- Thread Pools
  - Unlimited threads could exhaust system resources, such as CPU time or memory. One solution to this issue is to use thread pools
- Thread-Specific Data
  - Each thread might need its own copy of certain data in some circumstances.

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

INTER-PROCESS COMMUNICATION

Inter-process Communication

- Message Passing
- Shared Memory

* Synchronization

Message Passing

- Messages
  - Fixed size
  - Variable size
- Communication
  - Direct:
    - send(P2, message), receive(P1, message)
  - Indirect:
    - send(ID, message), receive(ID, messages)

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.