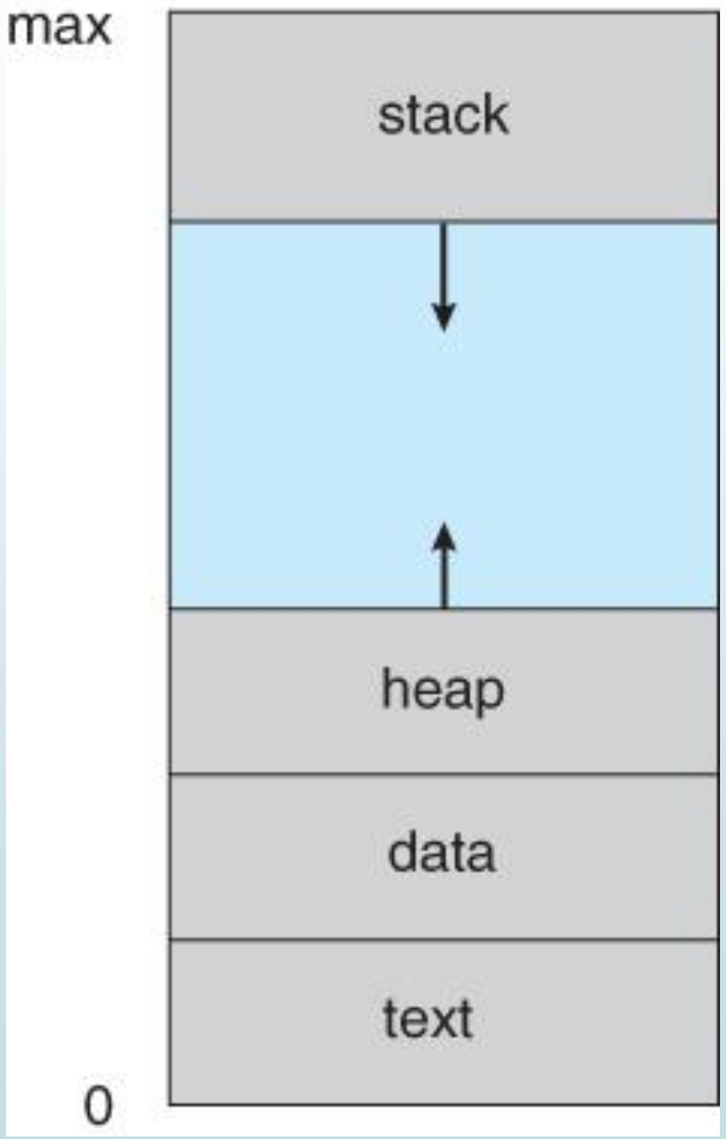
Operating System Concepts and Compiler Designs

Lecture 02

Processes Introduction

- A process is **a program during execution**.
  - Program = static file (image)
  - Process = executing program = program + execution state.
- A process is the **basic unit of execution** in an operating system
  - Each process has a number, its process identifier (pid).

Process in Memory



Process Creation

- Operating systems need some way to create processes.
- There are **four principal events** that cause processes to be created:
  - **System initialization**.
  - Execution of a process creation system **call by a running process**.
  - A **user request** to create a new process.
  - **Initiation of a batch job**.

Process Creation Cont…

- A process may create several processes using **create-process()** system call during the course of execution.
- The creating process is called as **parent process**, whereas the new processes are called as **children** of that parent process.
- in general , a **process requires resources** (CPU, Memory, Files, I/O devices) to complete its tasks.

Process Creation Cont…

- A child process may be obtain the resources **from the operating system** or it may be **from the parent process**.
- The parent process have to **partition** its resources among its children or it may be able to **share** the resources.
- Restricting a child process may help to prevent from system overloading.

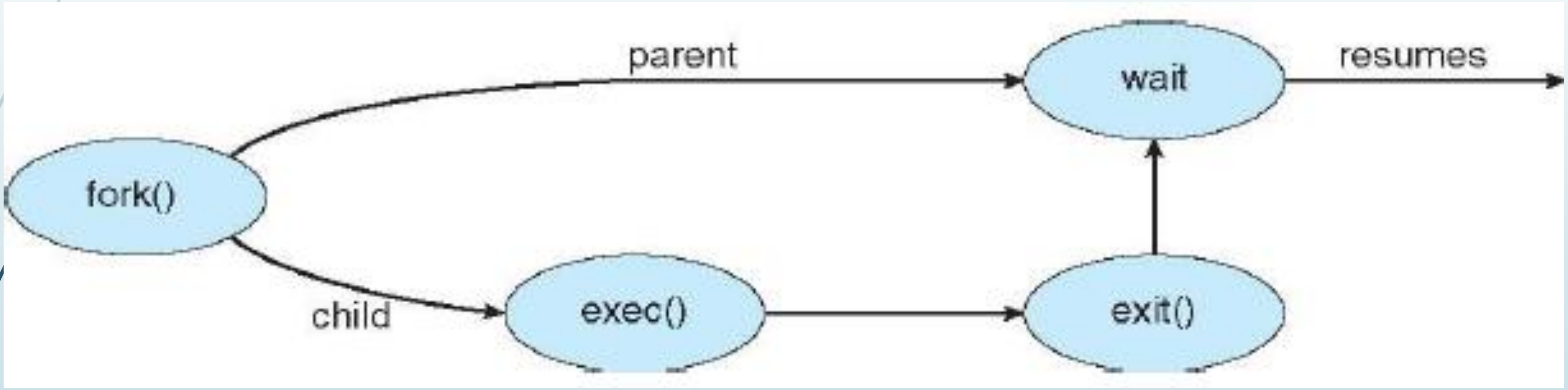
Process Creation Cont…

- When a process creates a child process, there are **two possibilities** exists to execute it,
  - The parent **continues to execute concurrently** with its children.
  - The parent **waits until** some or all of its children have **terminated**.

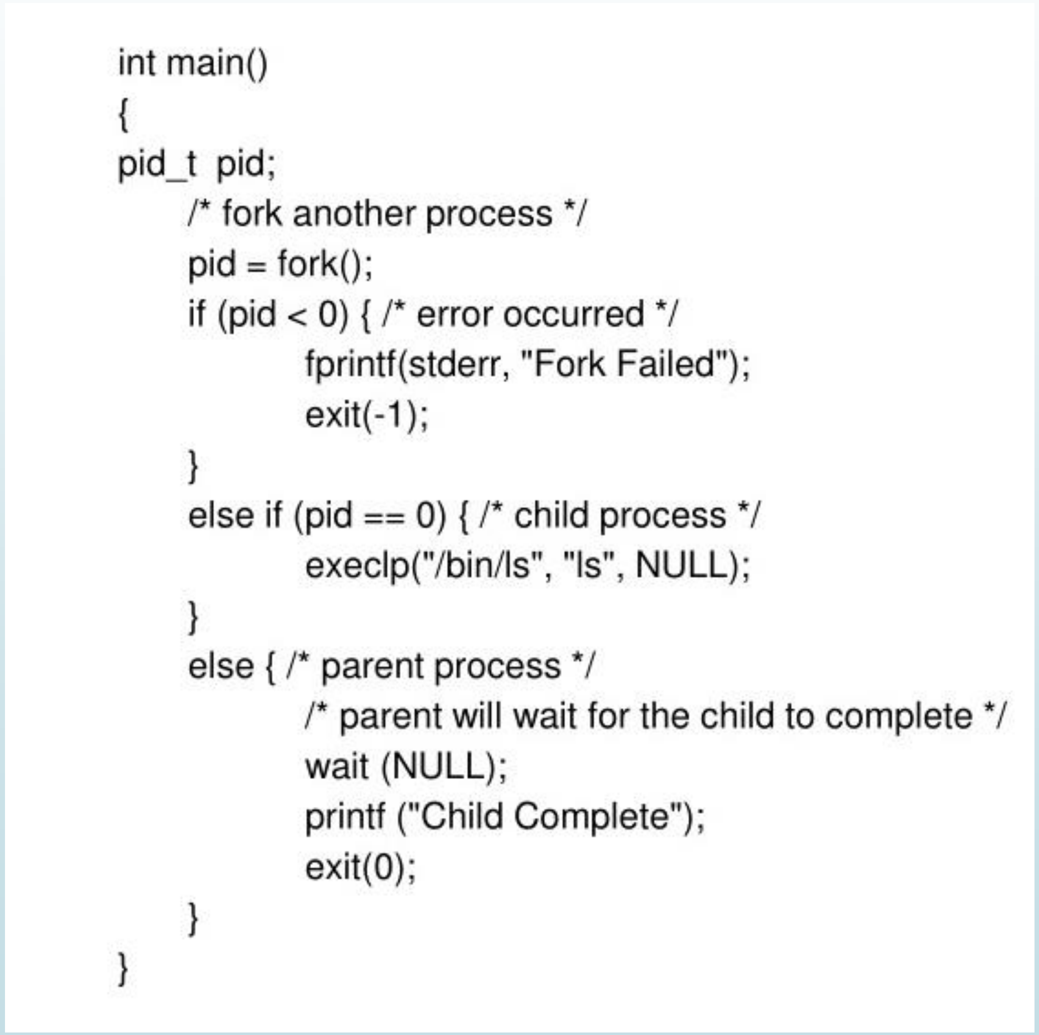
Process Creation Cont…

- Address space of a new process
  - Child process **is a duplicate** of parent
  - Child **has a program loaded into it**
- UNIX OS examples
  - Each process is identified by its **process identifier**, a unique integer.
  - fork() system call creates new process
  - execlp() system call used after a fork() to replace the process' memory space with a new program

Process Creation Cont…



Forking a Separate Process in C program



```
int main()
{
pid_t  pid;
      /* fork another process */
      pid = fork();
      if (pid < 0) { /* error occurred */
                fprintf(stderr, "Fork Failed");
                exit(-1);
      }
      else if (pid == 0) { /* child process */
                execlp("/bin/ls", "ls", NULL);
      }
      else { /* parent process */
                /* parent will wait for the child to complete */
                wait (NULL);
                printf ("Child Complete");
                exit(0);
      }
}
```

Process Termination

- Nothing lasts forever!
- The process will terminate, usually due to one of the following conditions
  - **Normal exit** (voluntary)
  - **Error exit** (voluntary)
  - **Fatal error** (involuntary)
  - **Killed by another process** (involuntary)

Process Termination Cont…

- Process executes last statement and then asks the operating system to delete it using the **exit()** system call.
  - Returns status data from child to parent (via **wait()**)
  - Process' resources are de-allocated by operating system

Process Termination Cont…

- Parent may terminate the execution of children processes using the **abort()** system call. Some reasons for doing so:
  - Child has **exceeded allocated resources**
  - Task assigned to child is **no longer required**
  - The **parent is exiting** and the operating systems does not allow a child to continue if its parent terminates

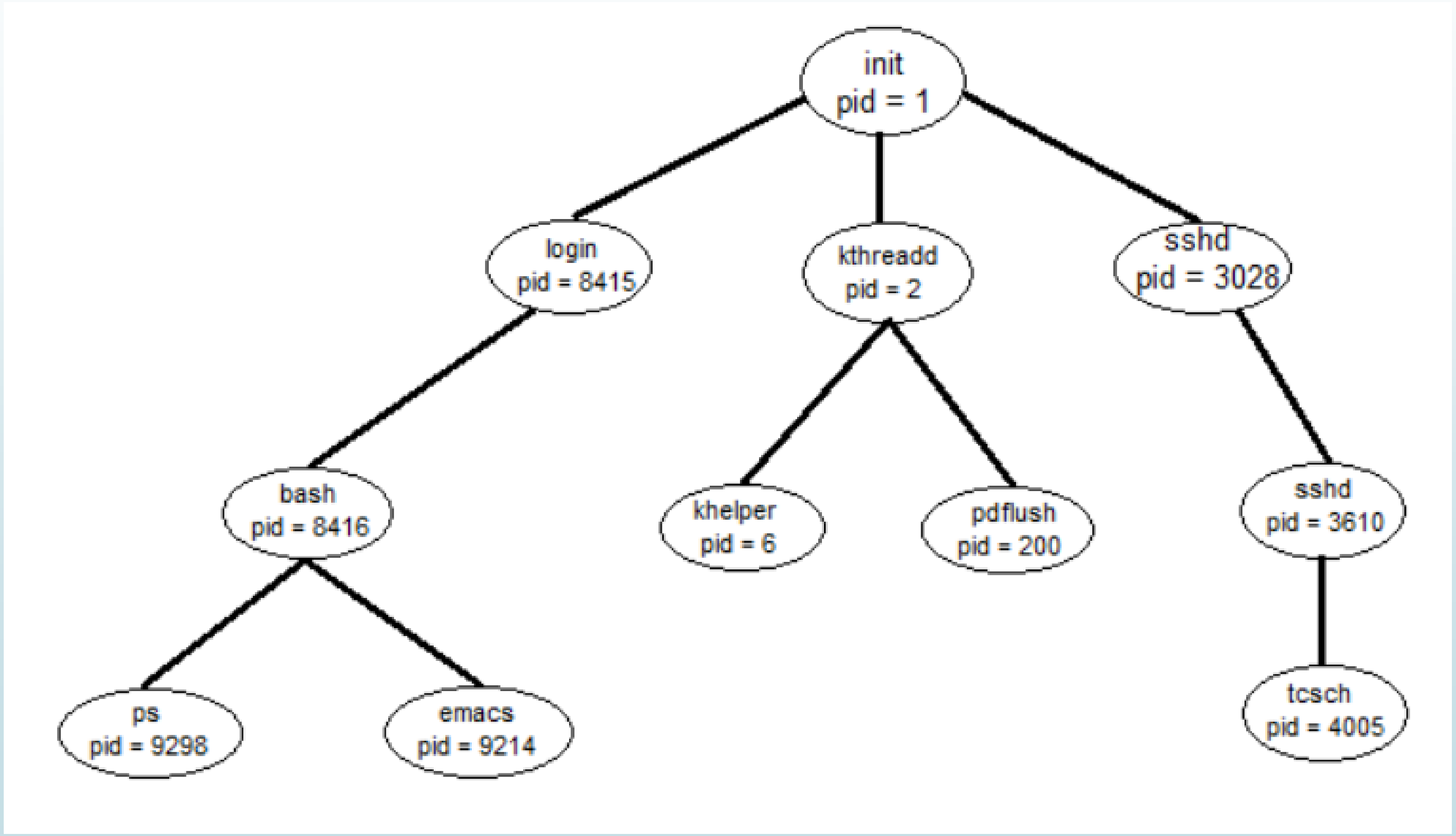
Process Hierarchies

- Association in certain ways – Parent and Children
- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- **Resource sharing options**
  - Parent and children **share all resources**
  - Children **share subset of parent's resources**
  - Parent and child **share no resources**

Process Hierarchies Cont…

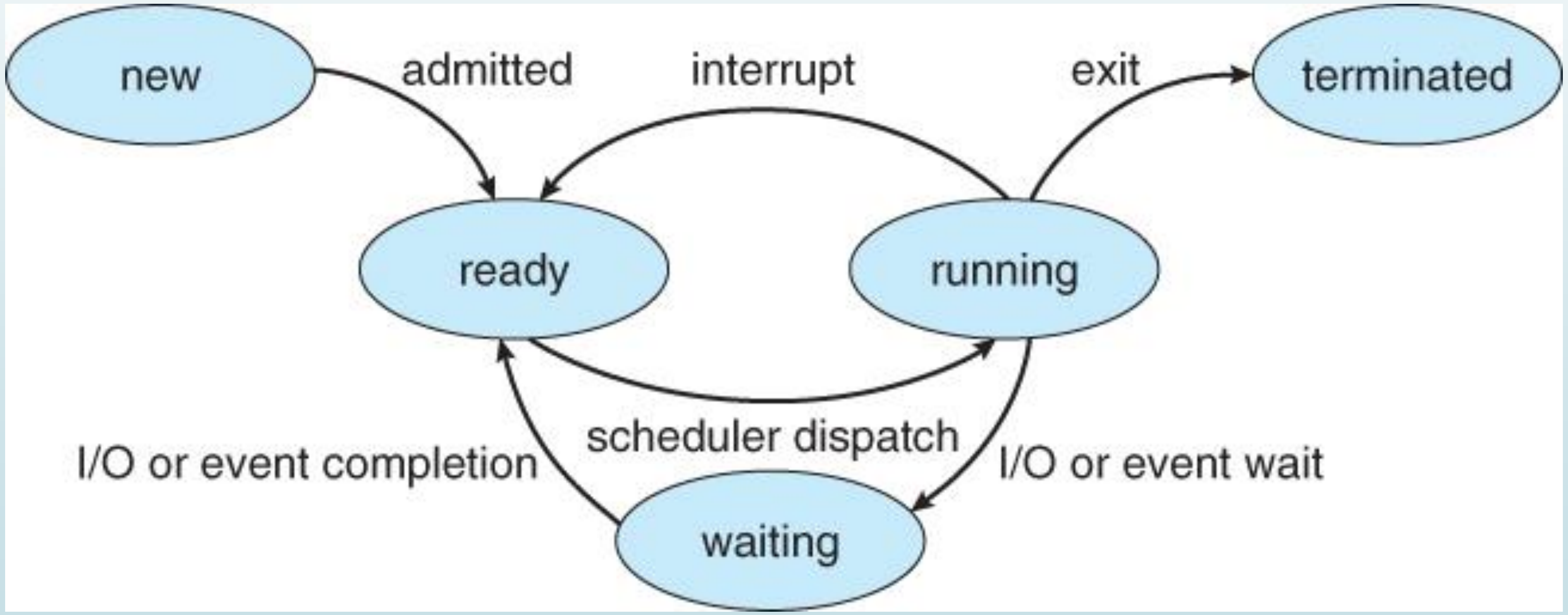
- Execution options
  - Parent and children **execute concurrently**
  - Parent **waits until children terminate**

Process Tree in LINUX



Process States

- When process executes, it changes its state



Process States Cont…

- **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**

Inter-Process Communication

- Processes within a system may be **independent or cooperating**
- **Cooperating process can affect or be affected by other processes**, including sharing data.
- Reasons for cooperating processes:
  - **Information sharing**
  - **Computation speedup**
  - **Convenience**

Inter-Process Communication Cont…

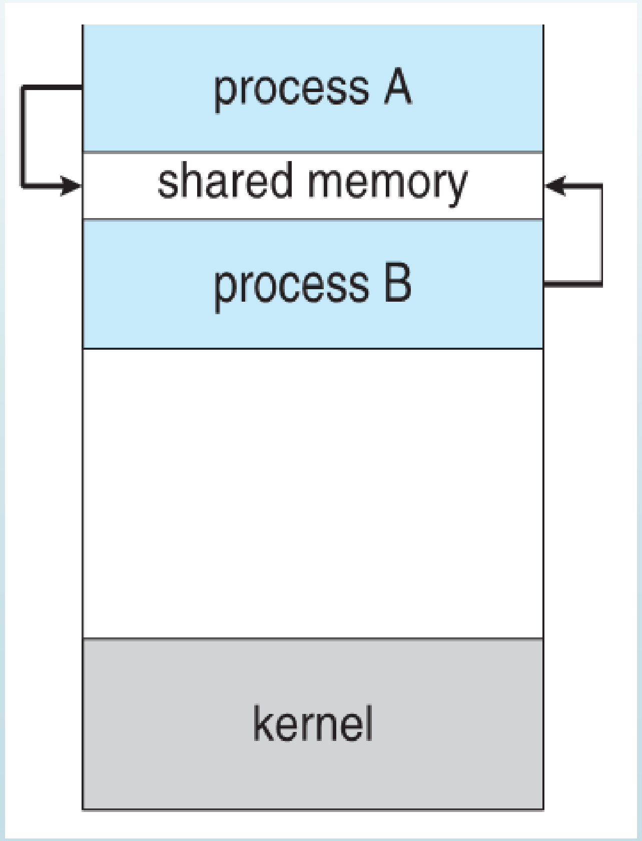
- Cooperating processes need **inter-process communication (IPC)**
- Two models of IPC
  - **Shared memory**
  - **Message passing**

Inter-Process Communication Cont…

- **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
  - Convenience

Inter-Process Communication : Shared Memory

- Shared memory

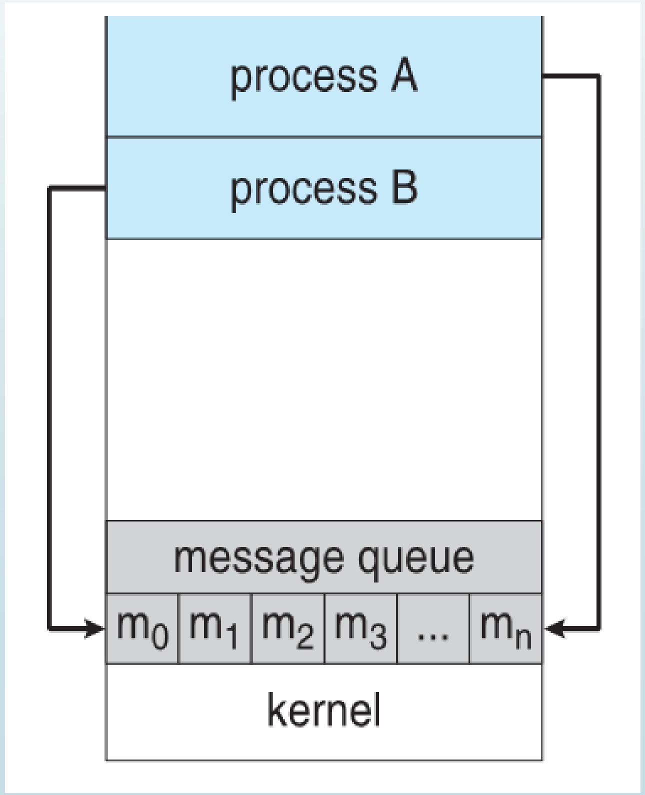


Inter-Process Communication : Shared Memory

- An **area of memory shared among the processes** that wish to communicate
- The **communication is under the control of the users processes not the operating system**.
- Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory

Inter-Process Communication : Message Passing

- **Message passing**



Inter-Process Communication : Message Passing

- Mechanism for processes **to communicate and to synchronize their actions**
- Message system – processes communicate with each other without resorting to shared variables
- IPC facility provides two operations:
  - **send(message)**
  - **receive(message**)
- The message size is either fixed or variable

Inter-Process Communication : Message Passing

- If processes P and Q wish to communicate, they need to:
  - **Establish a communication link** between them
  - **Exchange messages** via send/receive

Inter-Process Communication : Message Passing

- Implementation **issues**:
  - **How are links established**?
  - Can a link be **associated with more than two processes**?
  - **How many links can there** be between every pair of communicating processes?
  - What is the **capacity of a link**?
  - Is the **size of a message** that the link can accommodate fixed or variable?
  - Is a link **unidirectional or bi-directional**?

Inter-Process Communication : Message Passing

Implementation of communication link

- **Physical**:
  - Shared memory
  - Hardware bus
  - Network
- **Logical**:
  - Direct or indirect
  - Synchronous or asynchronous
  - Automatic or explicit buffering

Inter-Process Communication : Message Passing - 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
  - Between each pair there exists **exactly one link**
  - The link **may be unidirectional, but is usually bi-directional**

Inter-Process Communication : Message Passing - Indirect Communication

- 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**

Inter-Process Communication : Message Passing - Indirect Communication

- **Mailbox sharing**
  - P1, P2, and P3 share mailbox A
  - P1, sends; P2 and P3 receive
  - Who gets the message?
- **Solutions**
  - Allow a link to be associated with **at most two processes**
  - Allow **only one process at a time to execute a receive operation**
  - Allow the system to select arbitrarily the receiver. **Sender is notified who the receiver was.**

Inter-Process Communication : Message Passing - Synchronization

- Message passing may be either **blocking or non-blocking**
- Blocking is considered synchronous
  - **Blocking send** -- the sender is blocked until the message is received
  - **Blocking receive** -- the receiver is blocked until a message is available
- Non-blocking is considered asynchronous
  - **Non-blocking send** -- the sender sends the message and continue
  - **Non-blocking receive** -- the receiver receives: A valid message, or Null message

Inter-Process Communication : Message Passing – Buffering

- **Queue of messages attached to the link.**
- implemented in one of three ways
  - **Zero capacity** – no messages are queued on a link. 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

Communications in Client-Server Systems

- Sockets
- Remote Procedure Calls
- Pipes
- Remote Method Invocation (Java)

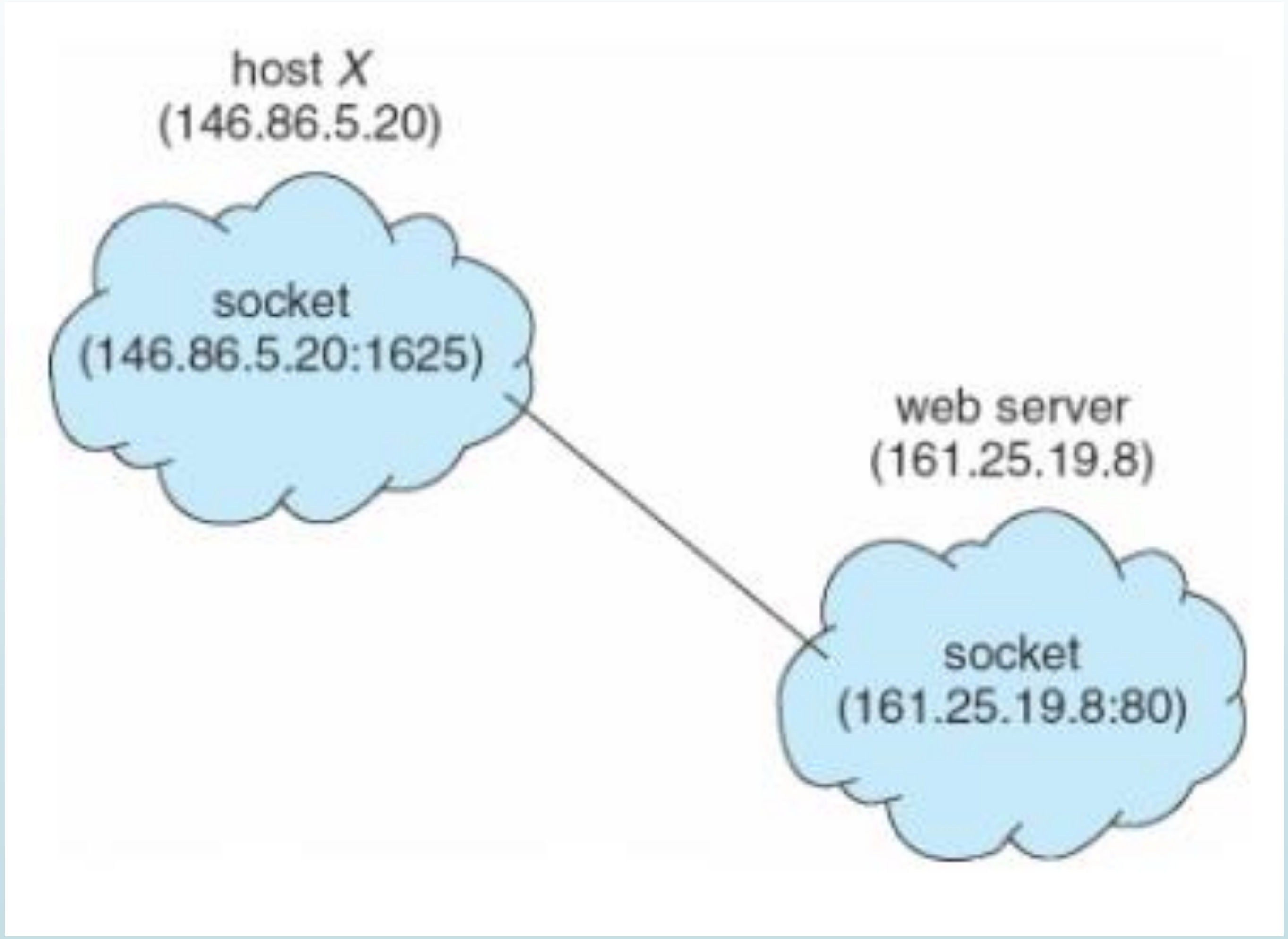
Sockets

- A socket is defined as **an endpoint for communication**
- Concatenation of IP address and port – **a number included at end of message packet** to differentiate network services on a host
- The socket 161.25.19.8:1625 refers to **port 1625 on host 161.25.19.8**

Sockets Cont…

- **Communication consists between a pair of sockets**
- All ports below 1024 are well known, used for standard services
- Special IP address **127.0.0.1** (loopback) used to refer to system on which process is running

Socket Communication



Pipes

- **Acts as a conduit allowing two processes to communicate**
- Issues:
  - Is communication **unidirectional or bidirectional**?
  - In the case of two-way communication, is it **half or full-duplex**?
  - **Must there exist a relationship** (i.e., parent-child) between the communicating processes?
  - **Can the pipes be used over a network**?

Pipes Cont…

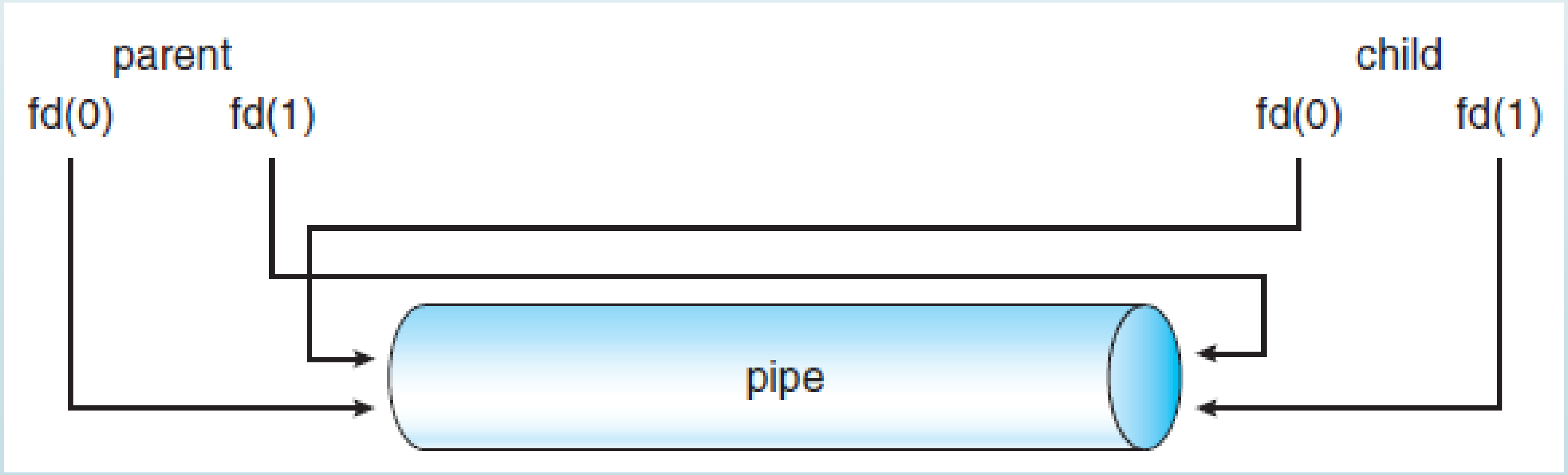
- **Ordinary pipes**
  - **Cannot be accessed from outside the process** that created it. Typically, a parent process creates a pipe and uses it to communicate with a child process that it created.
- **Named pipes**
  - **Can be accessed without a parent-child relationship.**

Ordinary Pipes

- Ordinary Pipes **allow communication in standard style**
- **Writes to one end** (the write-end of the pipe)
- **Reads from the other end** (the read-end of the pipe)
- Ordinary pipes are therefore unidirectional
- Require parent-child relationship between communicating processes

Ordinary Pipes Cont…

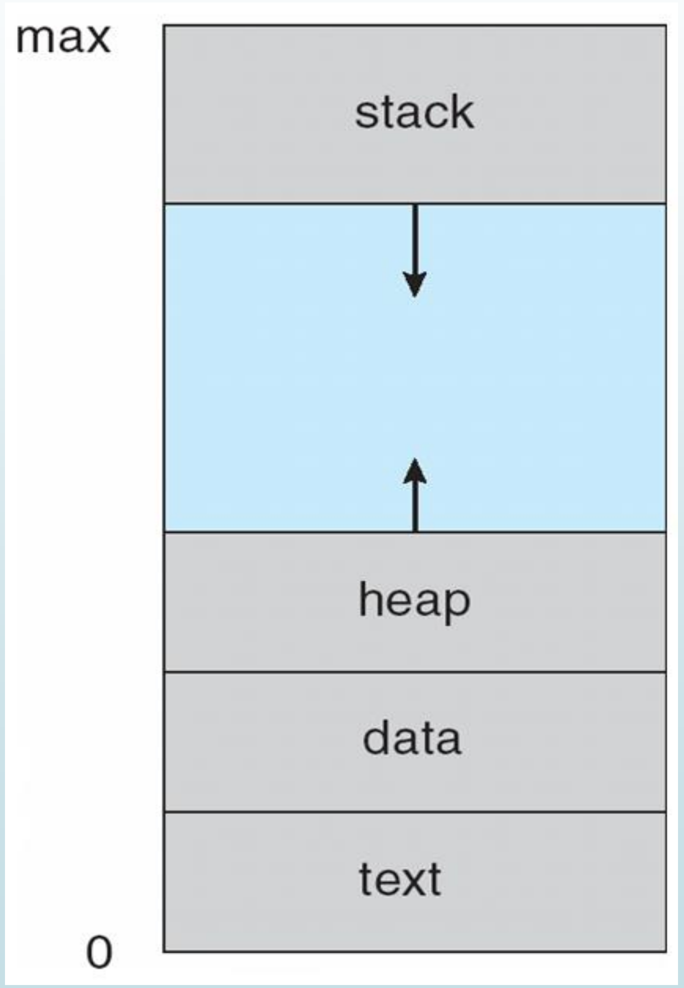
- fd : File discriptors
- fd(0) : Read end
- fd(1) : write end



Named Pipes

- Named Pipes are more powerful than ordinary pipes
- **Communication is bidirectional**
- No parent-child relationship is necessary between the communicating processes
- Several processes can use the named pipe for communication
- **Provided on both UNIX and Windows systems**

Process in Memory



Components in process

- **Text region**: **Store the converted executable code** that the processor executes.
- **Data region**: **contains static and global variables** which are going to be remains until the life time of the process
- **Stack**: **Contains the temporary data** such as function parameters, return addresses and local variables.
- **Heap**: **a memory that is dynamically allocated during the process time**.

Process's Suspend State

- Each Process to be executed must be loaded into the memory.
- When all of the **processes in main memory are in the waiting/ blocked states, the Operating system can suspend one process** by putting it in the suspend state and transferring it to secondary memory.
- The space that is freed in the main memory can then be used to bring in another process

Characteristics of Suspended Process

- The **process can not executed immediately**
- The process **may or may not be waiting** for an event.
- The process was placed into Identifying characteristics and classifying the sapphire using microscopy images and machine-learning techniques a suspend state by **parent process, or itself or operating system**
- The process may not be removed from the suspended state until its turn comes.

Process Control Block (PCB)

- **Operating System keeps an internal data structure to describe and identify each process** it manages.
- When OS creates process it creates this **process descriptor called as PCB** also known as task control block.
- **Contains many pieces of information** associated with a specific process.

Information stored in PCB

- Process Identification Number
- Priority number
- Program Counter
- Memory allocation
- I/O status information
- List of open files
- Process State
- Protection

Major Operations in Process

- **Creation**
- Scheduling
- Executing
- **Termination**

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

Process Scheduling Cont…

- Processes migrate among the various queues
- Types of schedulers
  - Long-Term Scheduler
  - Short-Term Scheduler
  - Medium-Term Scheduler

Long-term scheduler

- Long-term scheduler (or **job scheduler**) **selects which processes should be brought into the ready queue**
- Long-term scheduler is invoked infrequently (seconds, minutes) ➔ (may be slow)
- The long-term scheduler controls the degree of multiprogramming
- Long-term scheduler strives for good **process mix**

Short-term scheduler

- Short-term scheduler (or **CPU scheduler**) – **selects which process should be executed next and allocates CPU**
  - **Sometimes the only scheduler** in a system
  - Short-term scheduler is invoked frequently (milliseconds) ➔ (must be fast)
- 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

Medium-term scheduler

- 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**

Degree of Multiprogramming

- The degree of multiprogramming **describes the maximum number of processes that a single-processor system can accommodate efficiently**
- The primary factor affecting the degree of multiprogramming is **the amount of memory available to be allocated** to executing processes.

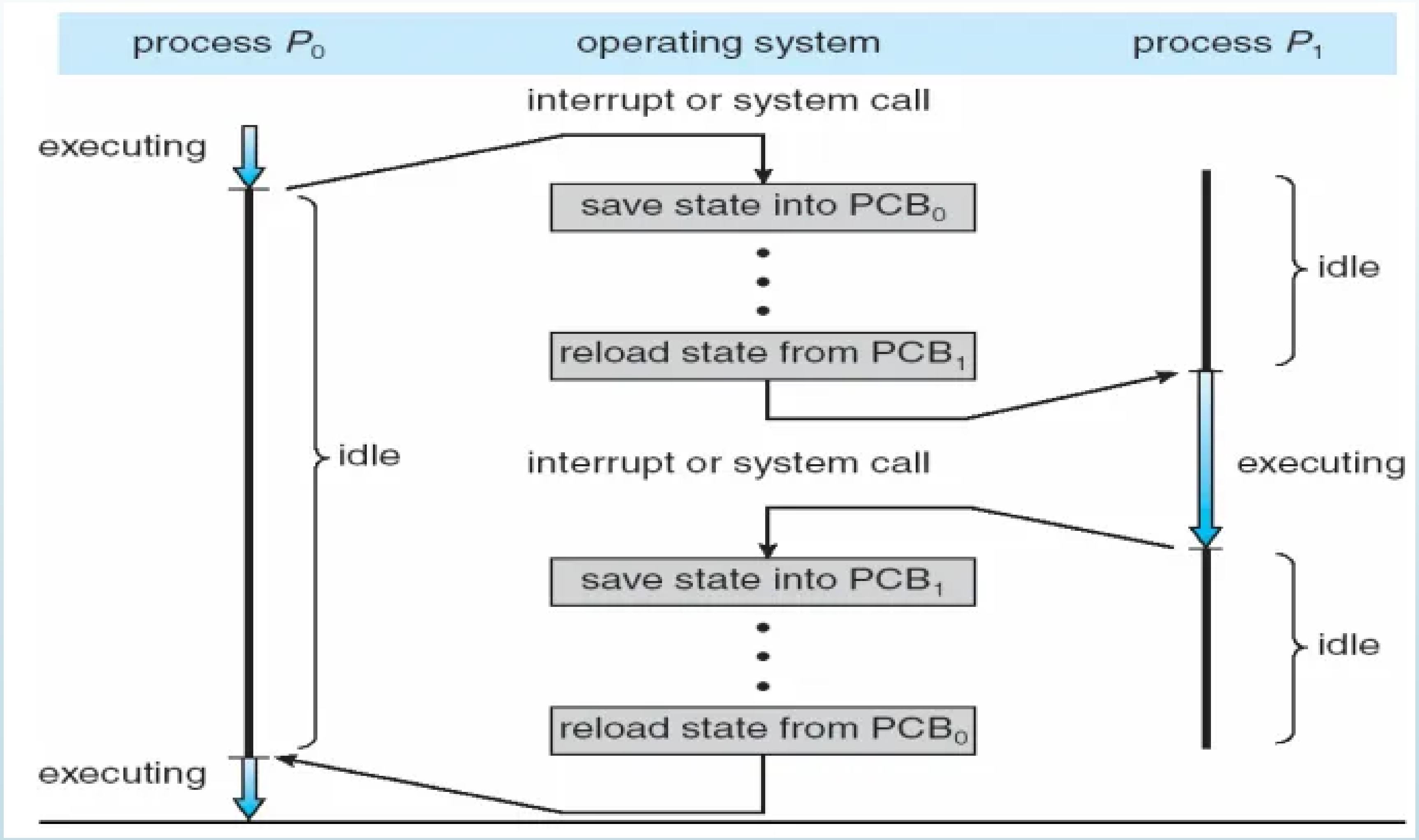
Context Switch

- **Interrupt cause the operating system to change a CPU from its current executing task and to run a kernel routine**
- Such operating happen frequently on general purpose system.
- When an interrupt occurs, the **system needs to save the current context of the process currently running on the CPU so that it can restore the context when its process is done, essential suspending the process and then resuming it**.

Context Switch Cont…

- The **context is representing in the PCB** of the process.
- Switching the CPU to another process requires performing a state save of the current process and a restore of a different process.

Context Switch Cont…



Context Switch Cont…

- **Context switching time is pure overhead**, because the system does no useful work while switching.
- The **speed of switching varies from machine to machine**, depending on the memory , number of registers that must be copied and the existence of the special instructions.
- Typically few **milliseconds**.

Thank You!

M. Janotheepan