Name: Nguyễn Phan Yến Ngân Student ID: 14597

**Chapter 3: Processes**

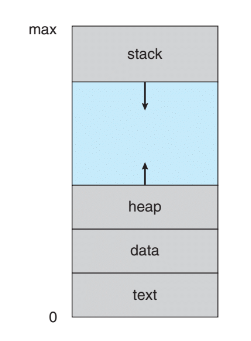
**3.1 Process Concepts**

**3.1.1 The Process**

- Process is a program in execution

- Status of current activity of a process represented by **program counter** and **contents of processor’s registers**

- Memory layout of a process:



* The memory layout divided into many sections which include:
* Text section: executable code
* Data section: global variabes
* Heap section: memory dynamically allocated during prgram run time
* Stack section: tempory data storage when invoking functions
* Sizes of text and data fixed; stack and heap can change dynamically during execution.
* Stack: Function called -> Activation record pushed onto stack

Control returned -> Activation record popped from stack

* Heap: Grow as memory dynamically allocated

Shrink when memory returned to system

* Stack and heap grow toward one another, they do not overlap
* Program is a passive entity, while process is an active entity
* Program becomes process when executable file loaded into memory
* 2 processes are 2 separate execution sequences even if associated with same program.

**3.1.2 Process State**

- New: Process created

- Running: Instructions executed

- Waiting: Wait for events occurred

- Ready: Wait to be assigned

- Terminated: Finished execution

**3.1.3 Process Control Block**

- Represent a process

- Contains pieces of information:

* Process state
* Program counter: Indicates address of next executable instruction
* CPU registers: Vary depending on computer architecture. This information must be saved when have interruption
* CPU-scheduling information: Includes a process priority
* Memory-management information: Includes such items as value of base and limit registers, etc
* Accounting information: Includes amount of CPU and real toem used, time limits, etc
* I/O status information: Includes list of I/O devices, openfiles, etc

**3.1.4 Threads**

- Process is a program performs a single thread of execution

- This allows process to perform only one task at a time

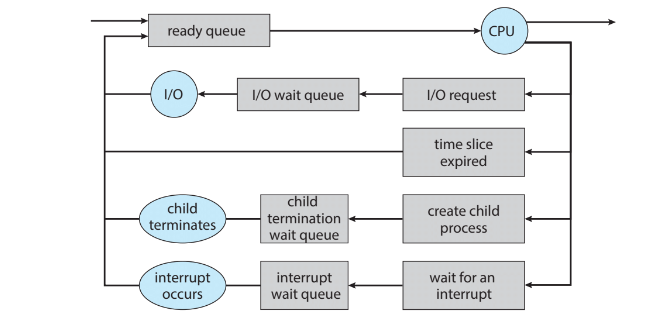
**3.2 Process Scheduling**

**3.2.1 Scheduling Queus**

- Processes are put into a ready queue when enter the system -> ready and wait to execute

- When process allocated CPU core, it executes and eventually terminates, interrupted or wait, which will be placed in a wait queue.

- A queuing diagram representing process scheduling



* Process can issue I/O request and placed in an I/O wait queue
* Process create new child process and placed in wait queue until that child terminates
* Prcoess can be removed forcibly and put back in ready queue

**3.2.2 CPU Scheduling**

- Role of CPU Scheduler: select from among processes in ready queue and allocate CPU core to one of them

- Must select a new process for CPU frequently

**3.2.3 Context Switch**

- When interruption occurs, system needs to save current context of running process to restore

- The contxt includes value of CPU registers, process state and memory management information.

- Generically, we perform a state save of current state of CPU core (whether in kernel or user mode) then a state restore to resume

- Context switch: A task requires performing state save of current process and state restore of different process to switch CPU core to another process

**3.3 Operations on Processes**

**3.3.1 Process Creation**

- A process may create several new processes -> forming a tree of processes

- When a child process created, it needs certain resources to accomplish task

- It can obtain directly from operating system or constrained to a subset of resources of parent process

- Restricting a child process to subset of parent’s resources prevents any process from overloading

- When new process created, 2 possibilities exist:

* Parent continues to execute with its children
* Parent waits until some or all children terminated

- 2 address-space possibilities:

* Child process is a duplicate of parent
* Child process has new program loaded into it

**3.3.2 Process Termination**

- A process finished final statement and ask operating system to delete it -> it terminates

- Termination can occur in other situations: a process cause termination of another process

- A parent may terminate execution of its children because:

* Child exceeded its usage of the allocated resources
* Task assigned to child no longer required
* Parent exiting -> child no longer allow to exist

**3.3.2.1 Android Process Hierarchy**

- Classify from most to least important:

* Foreground process: current process visible on screen, representing application user currently interacting with
* Visible process: Not directly visible on foreground but performing an acitivity foreground refering to
* Service process: Similar to background but performing an activity apparent to user
* Background process: Performing an activity not apparent to user
* Empty process: Holds no active components associated with any application

**3.4 Interprocess Communication**

- Independent process: Does not share data with other processes

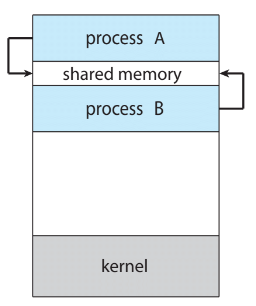
- Cooperating process: Affect or be affected by other processes

- Allow process cooperation will have many advantages: information sharing, computation speedup and modularity

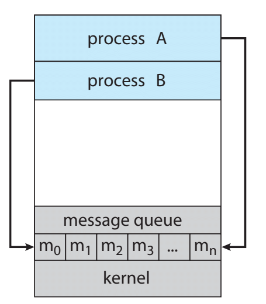
- Cooperating processes require an interprocess communication (IPC) which allow them to exchange data

- 2 fundamental models of interprocess communication: shared memory and message passing

- Shared memory: Faster. Once shared-memory established, all access treated as routine memory access, no assistance from kernel required



* Message passing:Useful for exchange smaller amounts of data



3.5 IPC in Shared-Memory Systems

- A shared-memory region resides in address space of process ceating shared-memory segment. Other wish to communicate with this segment must attach it to their address space.

- Shared-memory requires two or more processes agree to remove the restriction of no other process can access one’s memory.

=> Can exchange information by reading and writing data in shared areas

**3.6 IPC in Message-Passing Systems**

- Message passing provides a mechanism to allow processes to communicate and synchronize their actions without sharing same address space.

- Message sent by process can be either fixed or variable in size.

- If 2 processes want to communicate, must send to and receive messages from each other

=> A communication link must exist

- Several methods to implement a link: direct/indirect communication, synchronous/asynchronous communication, automatic/explicit buffering

**3.6.1 Naming**

- Direct communication: each process explicitly name recipient or sender of communication

- Indirect communication: messages sent to and received from mailboxes, or ports

**3.6.2 Synchronization**

- Message passing may be either blocking or nonblocking:

* Blocking send: Sending process is blocked until message received by receiving process or mailbox
* Nonblocking send: Sending process sends message and resumes operation
* Blocking receive: Receiver blocks until message available
* Nonblocking receive: Receiver retrieves either a valid message or null

**3.6.3 Buffering**

- Messages exchanged by communicating processes reside in temporary queue, which can be implemented in 3 ways:

* Zero capacity: Queue has maximun length of zero
* Bounded capacity: Queue has finte length n
* Unbounded capacity: Queue’s length is potentially infinite

**3.7 Examples of IPC Systems**

**3.7.1 POSIX Shared Memory**

- POSIX shared memory organized using memory-mapped files, which associate the region of shared memory with a file

**3.7.2 Mach Message Passing**

- Mach kernel supports creation and destruction of mutiple tasks.

- Most communication Mach is carried out by messages, which are sent to and received from mailboxes – called ports.

- Ports are finite in size and unidirectional

- Associated with each port is collection of port rights that identify capabilities necessary for a task to interact with port

**3.7.3 Windows**

- Message passing in Windows called the advanced local procedure call facility

- 2 types of ports: connection ports and communication ports

**3.7.4 Pipes**

- Pipe acts as a conduit allowing 2 processes to communicate

- 2 common types of pipes

**3.7.4.1 Ordinary Pipes**

- Allow 2 processes to communicate in standard producer-consumer fashion

-> Unidirectional, allow only one way communication

**3.7.4.2 Named Pipes**

- Communication can be directional, no parent-child relationship required -> Stronger than ordinary pipes

**3.8 Communication in Client-Server Systems**

**3.8.1 Sockets**

- An endpoint for communication

- A pair of processes communicating over a network employs a pair of sockets - one for each process

- Identified by an IP address concatenated with port nuber

- In general, sockets use client-server architecture

**3.8.2 Remote Procedure Calls**

- Different from IPC, messages in RPC are well structured -> no longer just packets of data

- A port in this context is simply a number included at the start of a message packet

- The semantics of RPC allows a client to invoke a procedure on a remote host as it would invoke a procedure locally.

- RPC system hides details that allow communication to take place by providing a stub on client side

**3.8.2.1 Android RPC**

- Although RPC typically associated with client-server computing in distributed system, they can also be used as a form of IPC between processes running on same system

- Android operating system has a rich set of IPC mechanisms contained in its binder framework, which includes RPC

- Android defines an application component as basic building block providing utility to an Android application -> combine multiple application components to provide functionality

- One such component is a service, has no user interface but runs in background