Department of
Computer Engineering
Faculty of Engineering
University of Sri Jayewardenepura

# Processes

## CO2206 - Operating Systems

Akarshani Amarasinghe

# Introduction

# Introduction (1)

- All modern computers often do several things at the same time.
- Eg: a Web server
  - Requests come in from all over asking for Web pages.
  - When a request comes in, the server checks to see if the page needed is in the cache.
  - If it is, it is sent back; if it is not, a disk request is started to fetch it.
  - However, from the CPU's perspective, disk requests take an eternity.
  - While waiting for a disk request to complete, many more requests may come in.
  - If there are multiple disks present, some or all of the newer ones may be fired off to other disks long before the first request is satisfied.
- Clearly, some way is needed to model and control this concurrency.
- **Processes** (and especially threads) can help here.

# Introduction (2)

- Eg: a user PC
  - When the system is booted, many processes are secretly started, often unknown to the user.
  - A process may be started up to wait for incoming emails.
  - Another process may run on behalf of the antivirus program to check periodically if any new virus definitions are available.
  - In addition, explicit user processes may be running, printing files and backing up the user's photos on a USB stick, all while the user is surfing the Web.
- All this activity has to be managed, and a multiprogramming system supporting multiple processes comes in very handy here.

# Pseudo Parallelism

- In any multiprogramming system, the CPU switches from process to process quickly, running each for tens or hundreds of milliseconds.
- While at any one instant the CPU is running only one process, in the course of 1 second it may work on several of them, giving the illusion of parallelism.
- This is called "**Pseudo Parallelism**".

# The Process Model

# What is a Process?

- All runnable software on the computer is organized into a number of **sequential processes**.
- A process is just an instance of an executing program.
  - It consists:
    - Program counter
    - Registers
    - Variables
- The CPU switches from program to program in rapid switching manner back and forth is called **multiprogramming**.

# Multiprogramming

Department of
Computer Engineering
Faculty of Engineering
University of Sri Jayewardenepura

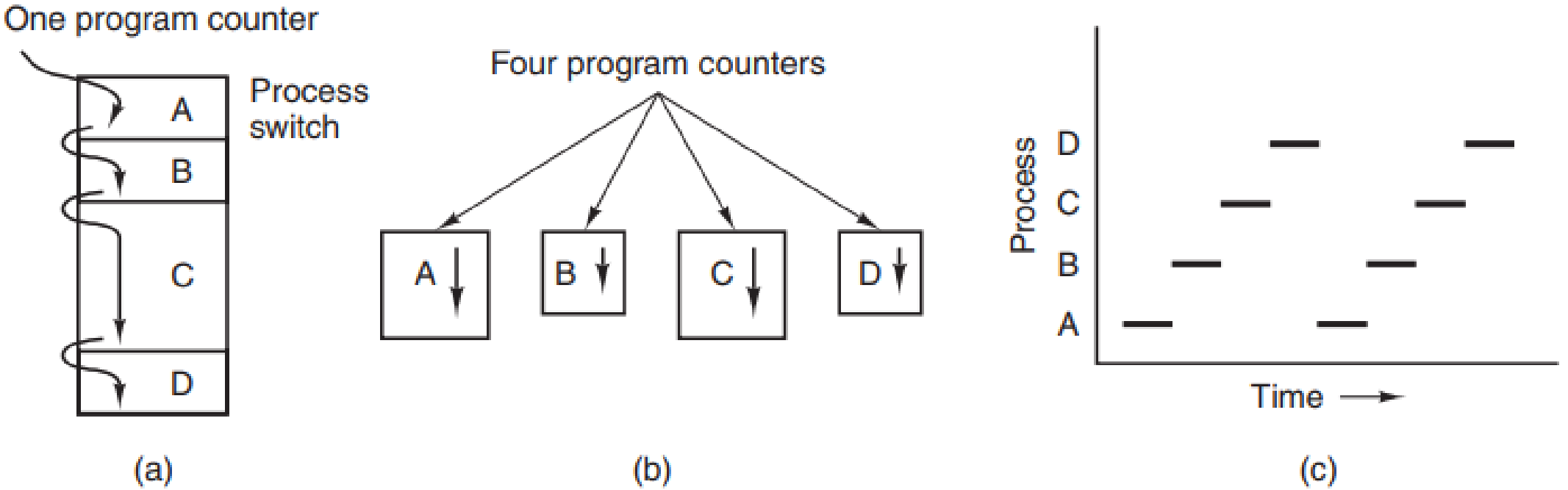



**Figure 2-1.** (a) Multiprogramming four programs. (b) Conceptual model of four independent, sequential processes. (c) Only one program is active at once.

- There is only one physical program counter, so when each process runs, its logical program counter is loaded into the real program counter.
- When it is finished (for the time being), the physical program counter is saved in the process' stored logical program counter in memory.

# Process vs. Program (1)

- Eg: baking a cake scenario (1)
  - A person is baking a birthday cake for his young daughter.
  - He has a birthday cake recipe and a kitchen well stocked with all the input: flour, eggs, sugar, extract of vanilla, and so on.
  - The recipe is the program (an algorithm), the person is the processor (CPU), and the cake ingredients are the input data.
  - The process is the activity consisting of our baker reading the recipe, fetching the ingredients, and baking the cake.

# Process vs. Program (2)

- Eg: baking a cake scenario (2)
  - Now imagine that his son comes running in screaming his head off, saying that he has been stung by a bee.
  - The person records where he was in the recipe (the state of the current process is saved), gets out a first aid book, and begins following the directions in it.
  - The processor being switched from one process (baking) to a higher-priority process (administering medical care), each having a different program (recipe versus first aid book).
  - After caring his son, the person goes back to his cake, continuing at the point where he left off.

# A Process is an Activity

- A process is an activity of some kind.
- It has a program, input, output, and a state.
- A single processor may be shared among several processes, with some scheduling algorithms being accustomed to determine when to stop work on one process and service a different one.

# Process Creation

# Four Principle Events

- Four principal events cause processes to be created:
  1. System initialization.
  2. Execution of a process-creation system call by a running process.
  3. A user request to create a new process.
  4. Initiation of a batch job.
- Processes that stay in the background to handle some activity such as email, Web pages, news, printing, and so on are called **daemons**.
- In all the above cases, a process executes a system call to create a new process.

# "fork"

- UNIX has one system call to create a new process: **fork**.
- Creates an exact clone of the calling process.
- After a fork, the parent and child have the same memory image, same environmental strings, same open files, etc.
- After a process is created, the parent and the child have their own distinct address spaces.
- If either process changes a word in its address space, the change is not visible to the other processes.
- In UNIX, the child's initial address space is a copy of the parent's but in Windows, these spaces are different from the start.

# Process Termination

# Conditions for a Process Termination

1. Normal exit (voluntary)
   - Most processes terminate because they have completed their work.
2. Error exit (voluntary)
   - No such file exists errors, pop-up error messages.
3. Fatal error (involuntary)
   - Due to a program bug.
   - Sometimes errors may be caught and handled.
4. Killed by another process (involuntary)
   - Execution of a system call like kill.

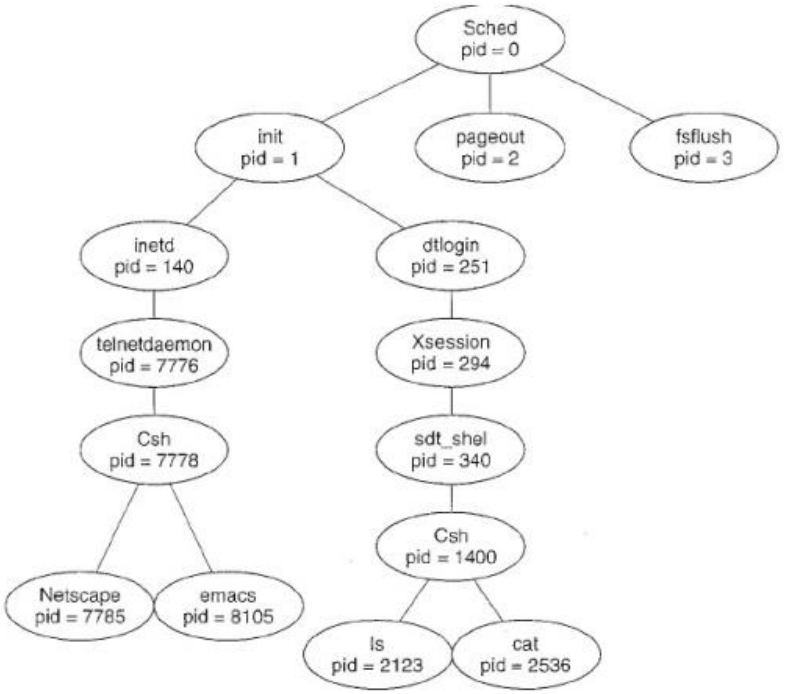
# Process Hierarchies

# Importance of a Hierarchy (1)

- A child process can itself create more processes, forming a process hierarchy.
  - A process has only one parent.
- In UNIX, a process and all its children and further descendants form a process group.
  - A signal may be delivered to all members processes of such group.
  - Individuals can catch a signal, ignore the signal, or take the default action.

# Importance of a Hierarchy (2)

- At the start-up, processes work as a hierarchy/tree to correctly initialize the system such as a chain execution by ***init*** in UNIX.
- Windows has no process hierarchy, instead a special token is held by the parent which can control the child.



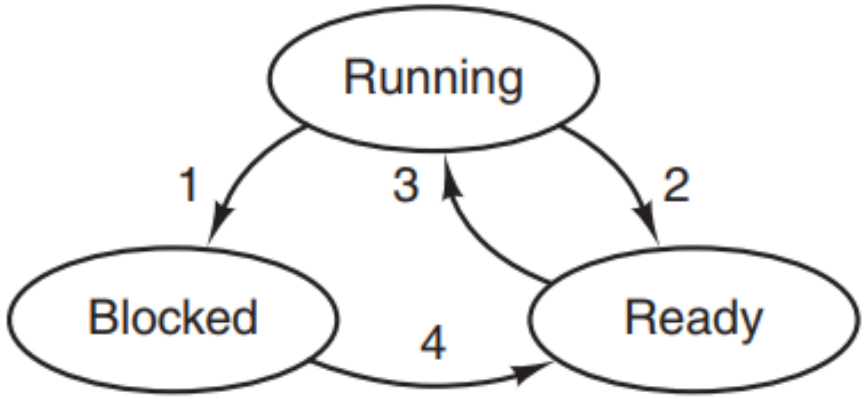
# Process States

# What is a State?

- Processes need to interact with other processes.
  - *cat* chapter1 chapter2 | *grep* tree
  - The first process runs ***cat*** and concatenates the two files; “chapter1” and “chapter2”.
  - The second process runs ***grep*** and selects all lines containing the word “tree” in the concatenated output.
- When a process blocks:
  - Logically: It cannot continue
  - Typically: It is waiting for the input that is not yet available
- Sometimes a process is conceptually ready but OS has decided to allocate CPU to another process.

# Three States

1. Running (actually using the CPU at that instant).
2. Ready (runnable; temporarily stopped to let another process run).
3. Blocked (unable to run until some external event happens).




1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available

**Figure 2-2.** A process can be in running, blocked, or ready state. Transitions between these states are as shown.

# Process State Transitions

- Many algorithms have been devised to try to balance the competing demands of efficiency for the system as a whole and fairness to individual processes.
- User processes can be blocked and unblocked from time to time.

# Implementation of Processes

# Process Table (1)



- To implement the process model, the OS maintains a table (an array of structures) called the **process table (process control blocks)** with one entry per process.
- Process table simply serves as the repository for information that may vary from process to process.
- Each entry contains important information about process's state including:
  - Program counter
  - Stack pointer
  - Memory allocation
  - State of open files
  - Accounting and scheduling information

# Process Table (2)

| Process management | Memory management | File management |
|---|---|---|
| Registers | Pointer to text segment info | Root directory |
| Program counter | Pointer to data segment info | Working directory |
| Program status word | Pointer to stack segment info | File descriptors |
| Stack pointer | | User ID |
| Process state | | Group ID |
| Priority | | |
| Scheduling parameters | | |
| Process ID | | |
| Parent process | | |
| Process group | | |
| Signals | | |
| Time when process started | | |
| CPU time used | | |
| Children's CPU time | | |
| Time of next alarm | | |

**Figure 2-4.** Some of the fields of a typical process-table entry.

# Operations on an Interrupt

- Associated with each I/O class is a location (at a fixed location near the bottom of memory) called the **interrupt vector**.
  - It contains the address of the interrupt service procedure.

1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector.
3. Assembly-language procedure saves registers.
4. Assembly-language procedure sets up new stack.
5. C interrupt service runs (typically reads and buffers input).
6. Scheduler decides which process is to run next.
7. C procedure returns to the assembly code.
8. Assembly-language procedure starts up new current process.

**Figure 2-5.** Skeleton of what the lowest level of the operating system does when an interrupt occurs.

# Process Scheduling

- **The objective of time sharing** is to switch CPU among processes so frequently that users can interact with each program it is running.
- Sometimes all the interrupt handling and details of actually starting and stopping processes are hidden away in the scheduler.

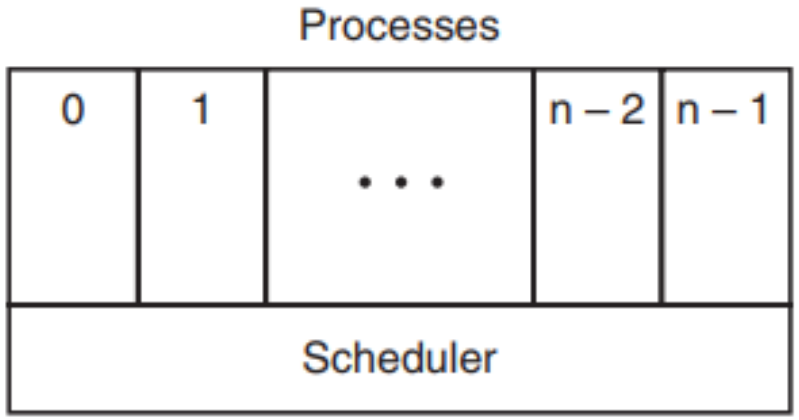



**Figure 2-3.** The lowest layer of a process-structured operating system handles interrupts and scheduling. Above that layer are sequential processes.

# Context Switch

- When an interrupt occurs, the system needs to save the current context of the process running on the CPU.
- If the above is performed, the CPU can restore the context of the process once the latter resumes.
- Context is represented in the process table of the process.
- This state saves and restores of a process is known as a **context switch**.
- At the context switch time, the system does not do any useful work.