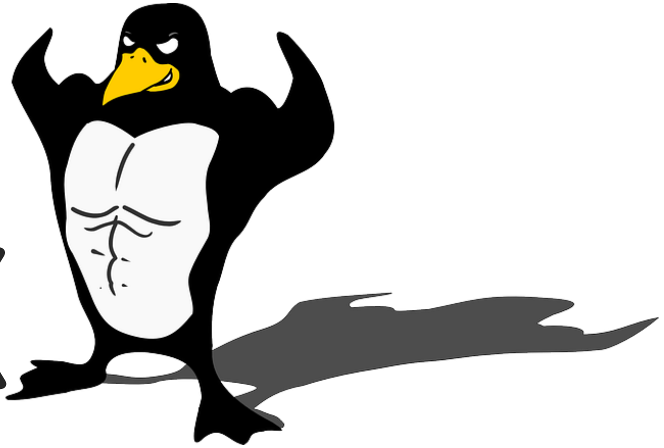
OPERATING SYSTEMS II

DPL. ING. CIPRIAN PUNGILĂ, PHD.

LINUX



An In-Depth Profile

A DEEPER LOOK INSIDE THE FAMOUS OPERATING SYSTEM

AFTER A LECTURE BY TIM WOOD

BIBLIOGRAPHY

1. A. TANNENBAUM, “MODERN OPERATING SYSTEMS”, 3RD EDITION, 2008, PRENTICE HALL
2. SILBERSCHATZ, GALVIN, AND GAGNE, "OPERATING SYSTEM CONCEPTS", 8TH EDITION, 2009, WILEY

Contents

Outline

- ❑History of the operating system
- ❑Design basics
- ❑General architecture
- ❑Process scheduling
- ❑Memory management
- ❑File systems
- ❑IPC – Interprocess communication

A **review** of the information from *Operating Systems I* and how Linux behaves as a **real** operating system

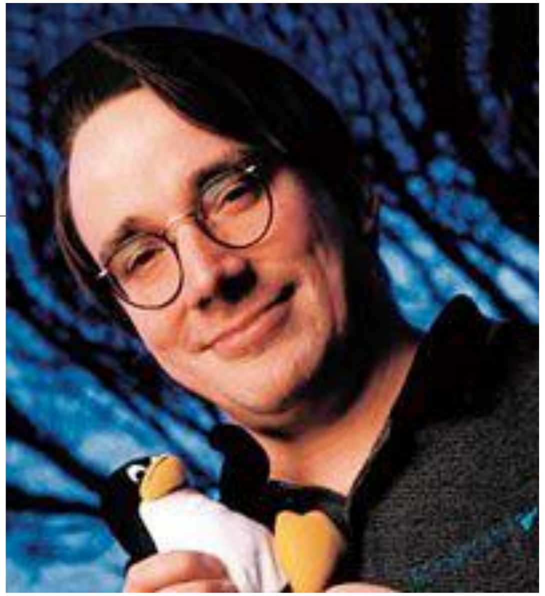
History
The Linux OS

- ❑ **Free** operating system
  - ❑ Based on UNIX standards
- ❑ **UNIX** – what is it?
  - ❑ Proprietary operating system
  - ❑ Developed in the 60s
  - ❑ Still used for mainframes
- ❑ **Linux** was first developed in 1991 by Linus Torvalds
  - ❑ Goal: providing basic UNIX functionality in a free system
- ❑ Version 0.01 (May 1991)
  - ❑ No networking
  - ❑ Ran only on 80386 compatible Intel processors and on PC hardware
  - ❑ Extremely limited device-driver support
  - ❑ Only support the Minix file system
- ❑ Version 2.6.34 (Summer 2010)
  - ❑ Most common OS for servers
  - ❑ Supports dozens of file systems
  - ❑ Runs on anything from smartphones to super computers

All contributed by the Linux community!

History

About Linus Torvalds



- Started the Linux kernel while a Masters student in Finland (1991)
- About **2% of current Linux code** was written by him
  - The rest is split between thousands of contributors!
- Message on first Linux release:
  - “PS… It is NOT portable (uses 386 task switching etc) and it probably will never support anything other than AT-harddisks, as that’s all I have :-(“

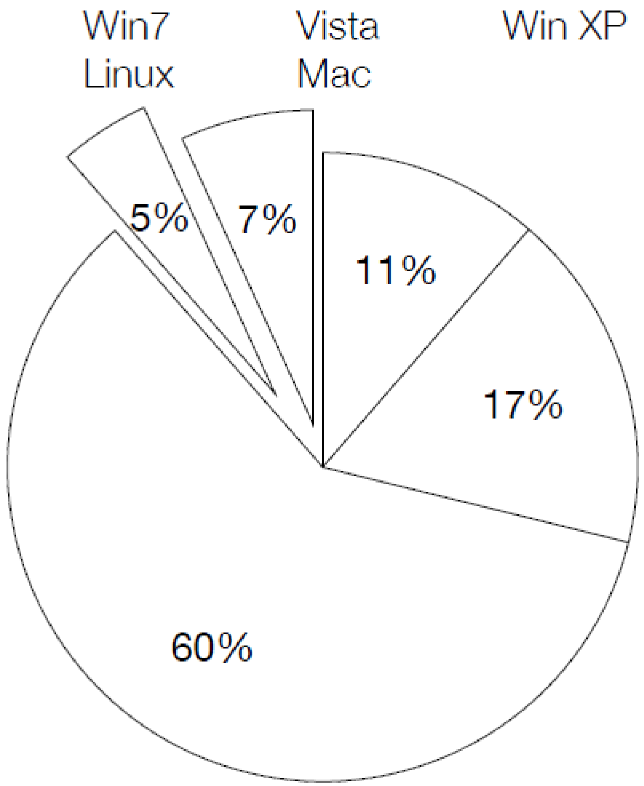
Now supports almost every hardware platform possible!

Who uses it?

Statistics for Linux (2010)

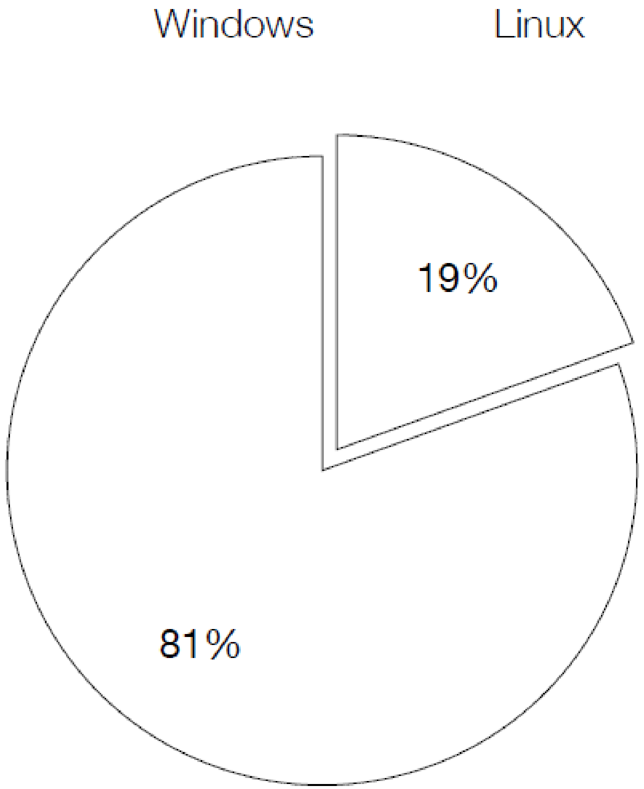
Web users

source: http://www.w3schools.com



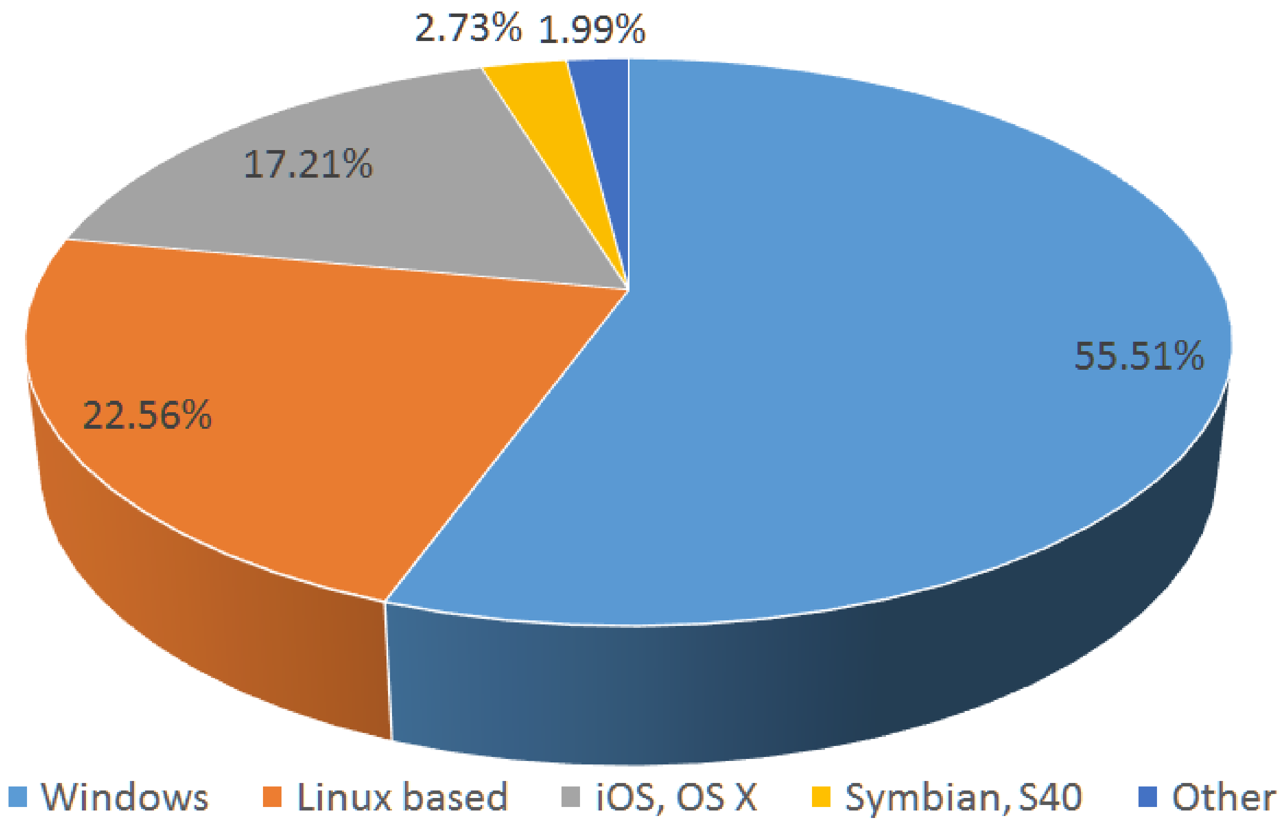
Web Servers

source: http://news.netcraft.com



Who uses it?

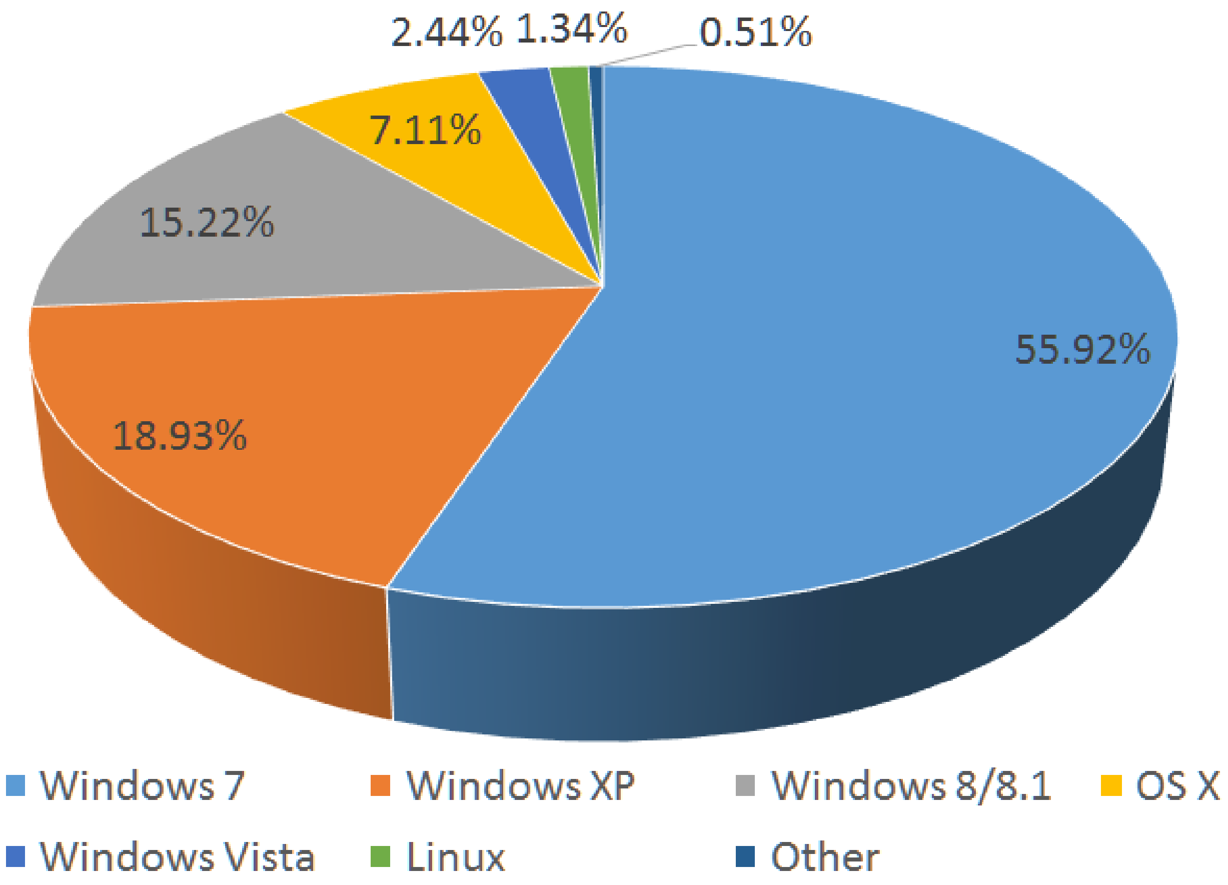
Web Users (December, 2014)



Source: Wikipedia

Who uses it?

Desktop Users (December, 2014)



Source: Wikipedia

Design Basics
Principles

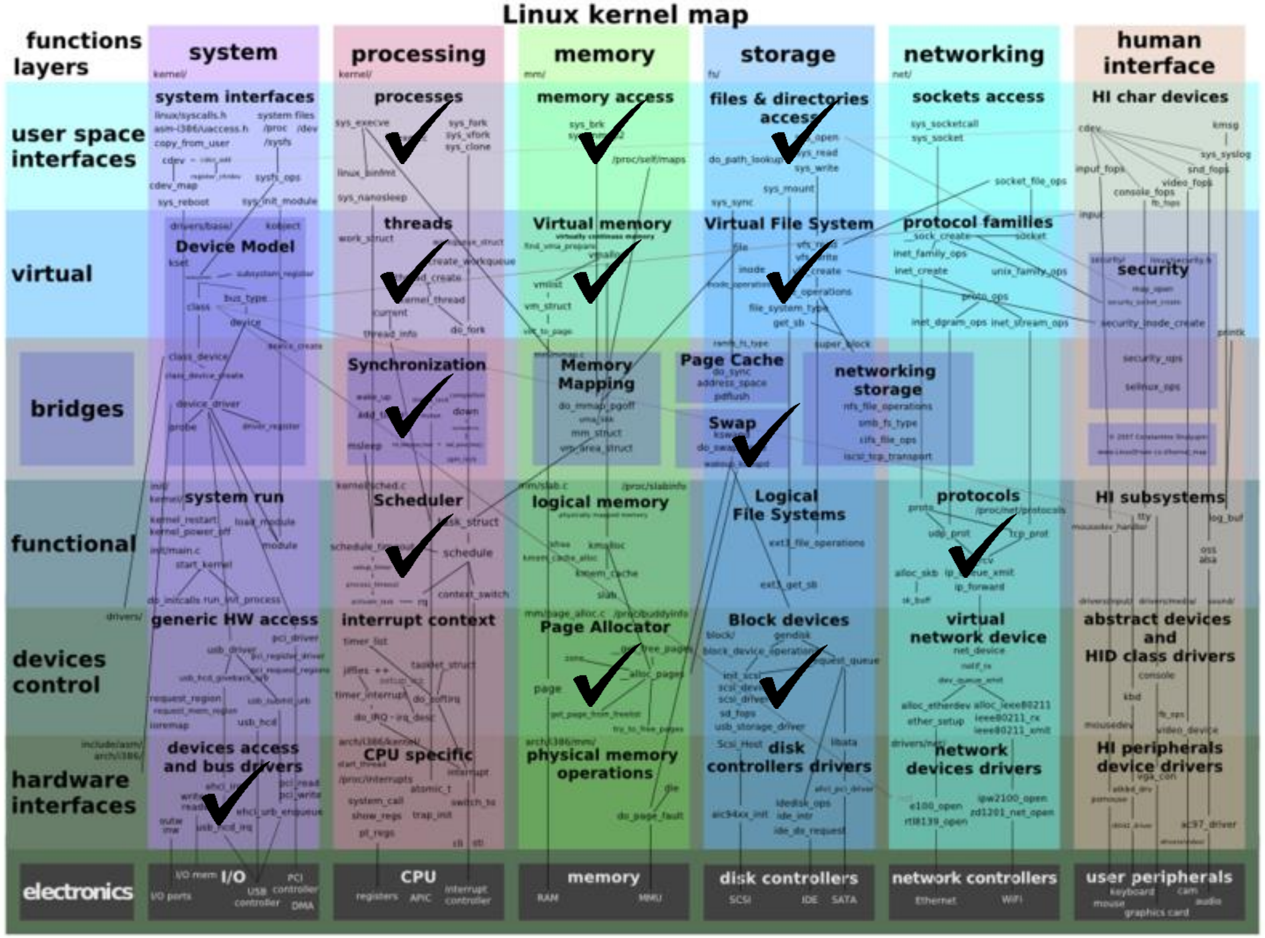
- Linux is a multiuser, multitasking operating system with a full set of **UNIX-compatible** tools
- Its file system adheres to traditional UNIX semantics, and it fully implements the standard UNIX networking model
- Main design goals are **speed, efficiency**, and **standardization**
- The Linux kernel is distributed under the GNU General Public License (GPL), as defined by the **Free Software Foundation**
  - “Anyone using Linux, or creating their own derivative of Linux, may not make the derived product proprietary; software released under the GPL must include its source code”

Kernel vs distributions
Variants

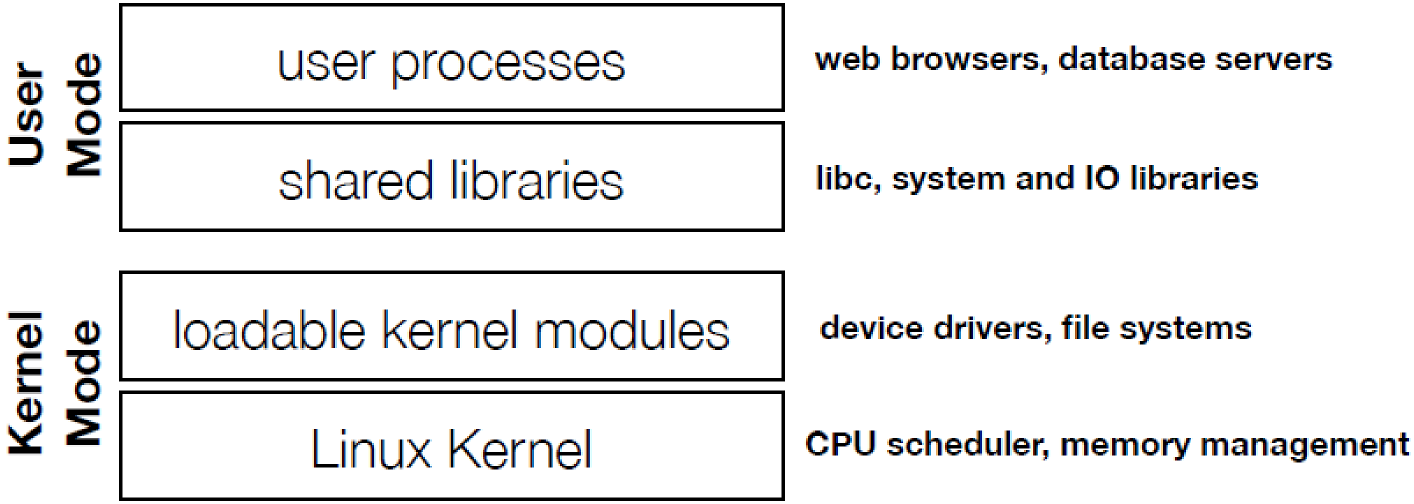
- The **Linux kernel** is the core part of the operating system
  - scheduler, drivers, memory managers, etc.
- A **Linux distribution** is the kernel plus the software needed to make the system actually usable
  - user interface, libraries, all user level programs, etc.

The Linux Operating System

Kernel Map



The Linux Operating System
Structure



User Mode

user processes — web browsers, database servers

shared libraries — libc, system and IO libraries

Kernel Mode

loadable kernel modules — device drivers, file systems

Linux Kernel — CPU scheduler, memory management

Linux
Structure

- ❑Linux separates **user** and **kernel mode** to provide protection and abstraction
  - ❑The OS functionality is split between the main **Linux Kernel** and optional **kernel modules**
- ❑**Linux Kernel** - all code that is needed as soon as the system begins: CPU scheduler, memory managers, system call / interrupt support, etc
  - ❑A *monolithic kernel* - benefits?
- ❑**Kernel modules** - extensions to the kernel that can be dynamically loaded or unloaded as needed: device drivers, file systems, network protocol, etc
  - ❑Provides some *modularity* - benefits?
- ❑Can specify whether each OS component is compiled into the kernel or built as a module, if you build your own version of Linux from source code

Linux

Kernel Modules

- ❑Pieces of functionality that can be **loaded and unloaded** into the OS
  - ❑Does not impact the rest of the system
  - ❑OS can provide protection between modules
  - ❑Allows for minimal core kernel, with main functionality provided in modules
- ❑Very handy for **development and testing**
  - ❑Do not need to reboot and reload the full OS after each change
- ❑Also, a way around Linux's **licensing restrictions**: kernel modules do not need to be released under the GPL
  - ❑Would require you to release all source code

Linux

Kernel Modules

- Kernel maintains tables for modules such as:
  - Device drivers
  - File systems
  - Network protocols
  - Binary formats
- When a module is loaded, add it to the table so it can **advertise its functionality**
- Applications may interact with kernel modules through system calls
- Kernel must **resolve conflicts** if two modules try to access the same device, or a user program requests functionality from a module that is not loaded
- Not all functionality can be implemented as modules – *examples?*

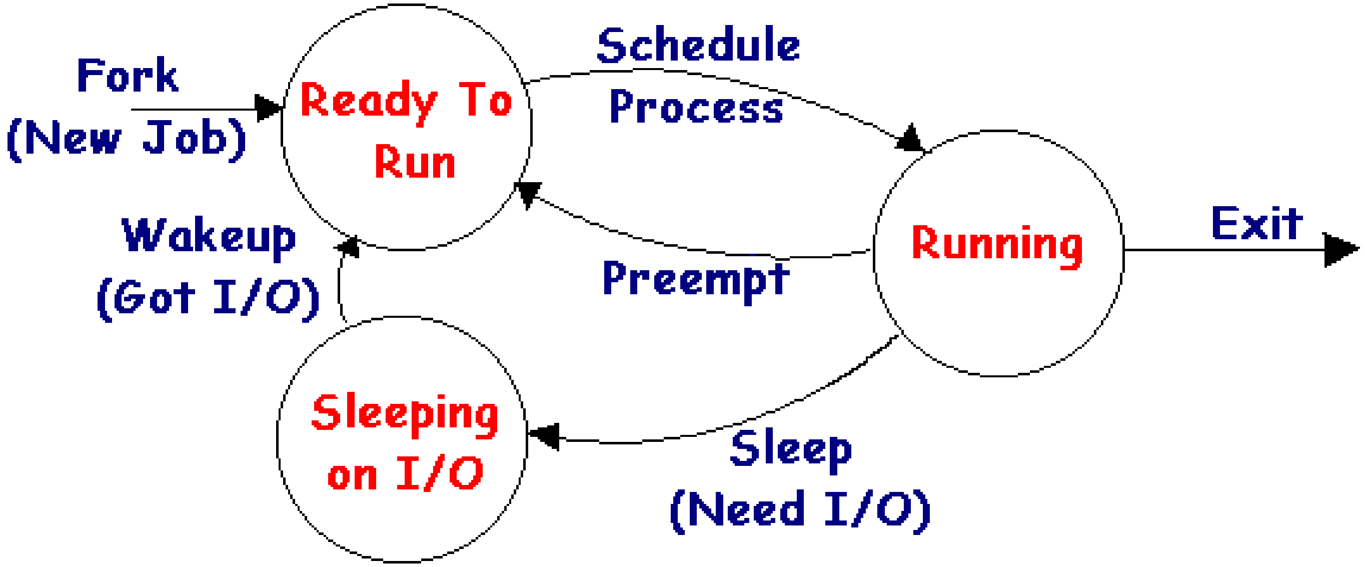
Linux

Process Management

- Processes are created using the fork/clone and execve functions
  - **fork** - system call to create a new **process**
  - **clone** - system call to create a new **thread**
    - Actually just a process that shares the address space of its parent
  - **execve** - run a new program within the context created by fork/clone
  - Often programmers will use a library such as *Pthreads* to simplify API
- Linux maintains information about each process:
  - Process Identity
  - Process Environment
  - Process Context

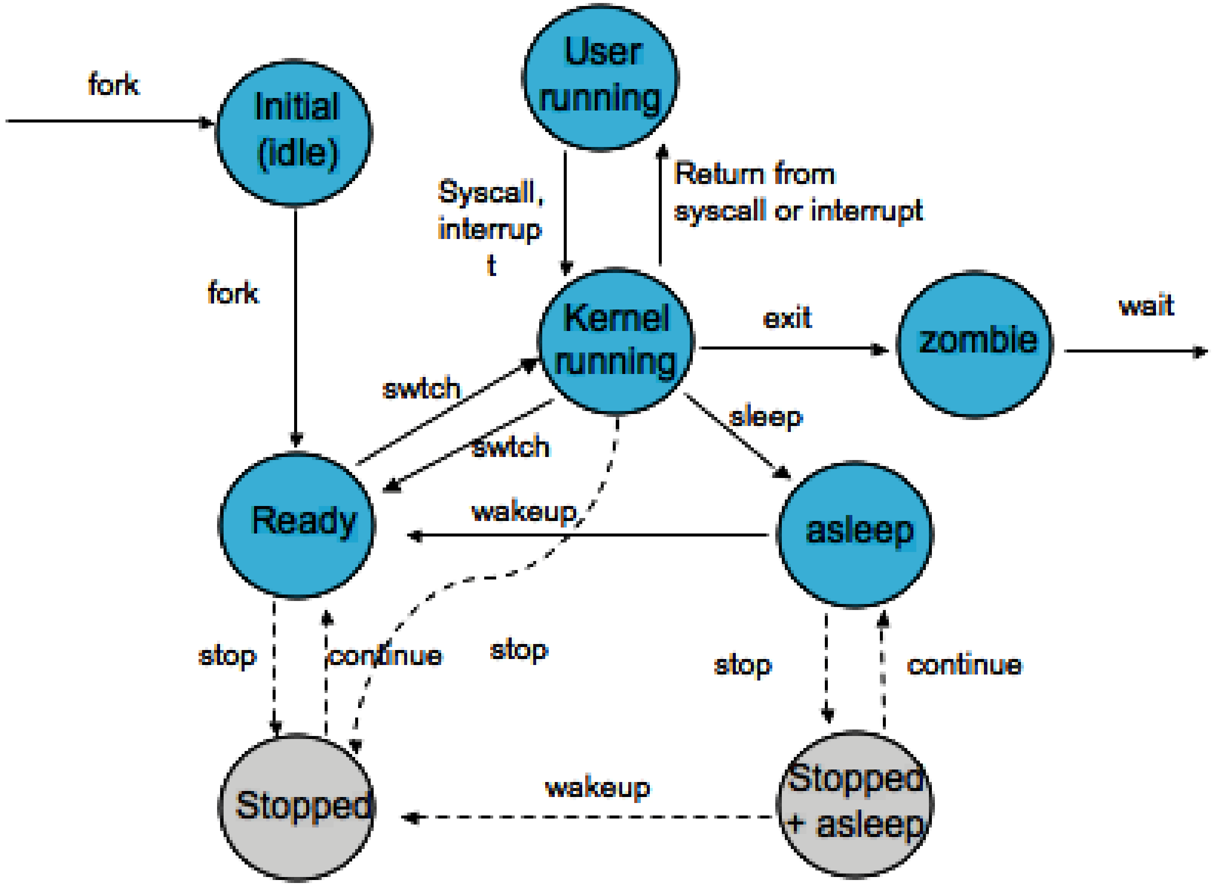
Linux

Process States



Linux

Process States



Linux
Process Identity

- **General information** about the process and its owner
- **Process ID (PID)** - a unique identifier, used so processes can precisely refer to one another
  - `ps` -- prints information about running processes
  - `kill PID` -- tells the OS to terminate a specific process
- **Credentials** - information such as the user who started the process, their group, access rights, and other permissions info

Linux

Process Environment

- Stores static data that can be customized for each process
- Argument Vector - list of parameters passed to the program when it was run
  - `head -n 20 file.txt` -- start the “head” program with 3 arguments
- **Environment Vector** - a set of parameters inherited from the parent process with additional configuration data
  - the current directory, the user’s path settings, terminal display parameters
- These provide a simple and flexible way to pass data to processes
  - Allows settings to configured per-process rather than on a system oruser-wide level

Linux
Process Context

- ❑The **dynamically changing state** of the process
- ❑**Scheduling context** - all of the data that must be saved and restored when a process is suspended or brought into the running state
- ❑**Accounting information** - records of the amount of resources being used by a process
- ❑**File table** - list of all files currently opened by the process
- ❑**Signal-handler table** - lists how the process should respond to signals
- ❑**Virtual memory context** - describes the layout of the process’s address space

Linux

Process Scheduling

- The Linux scheduler must allocate CPU time to both user processes and kernel tasks (e.g. device driver requests)
- **Primary goals**: **fairness** between processes and an emphasis on good performance for **interactive (I/O bound) tasks**
- Uses a **preemptive scheduler**
  - What happens if one part of the kernel tries to preempt another?
    - Prior to Linux 2.4, all kernel code was non-preemptable
    - Newer kernels use locks and interrupt disabling to define critical sections

Linux

Process Scheduling

- ❑Scheduler implementation has changed several times over the years
- ❑Kernel 2.6.8: **O(1) scheduler**
  - ❑Used **multi-level feedback queue** style scheduler
  - ❑Lower priority queues for processes that use up full time quantum
  - ❑All scheduling operations are O(1), constant time, to limit scheduling overhead even on systems with huge numbers of tasks
- ❑Kernel 2.6.23: **Completely Fair Scheduler**
  - ❑Uses red-black trees instead of run queues (not O(1))
  - ❑Tracks processes at nano-second granularity -> more accurate fairness

Linux

Memory Management

- User processes are granted memory using **segmented demand paging**
  - Virtual memory system tracks the address space both as a set of regions (segments) and as a list of pages
- Pages can **be swapped out** to disk when there is memory pressure
  - Uses a modified version of the Clock algorithm to write the **least frequently** used pages out to disk
- Kernel memory is either paged or statically allocated
  - Drivers reserve contiguous **memory** regions
  - The **slab allocator** tracks chunks of memory that can be re-used for kernel data structures

Linux

Caches

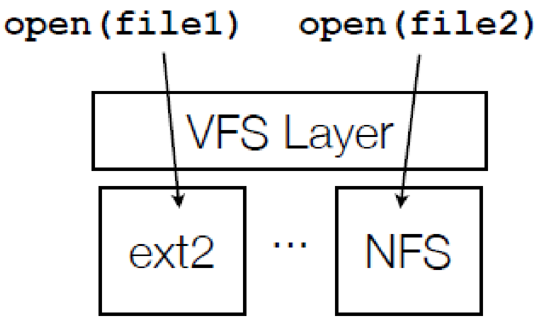
- Linux maintains caches to improve I/O performance
- **Buffer Cache** - stores data from **block devices** such as disk drives
  - All pages brought from disk are temporarily stored in buffer cache in case they are accessed again
- **Page Cache** - caches entire pages of I/O data
  - Can store data from both **disks and network** I/O packets
- Caches can significantly improve the speed of I/O at the expense of RAM
  - Linux automatically resizes the buffer and page caches based on how much memory is free in the system

Linux

File Systems

- **Virtual File System** layer provides a standard interface for file systems
  - Supports **file, inode**, and **file-system** objects
  - Lets the OS treat all files identically, even if they may be on different devices or file systems

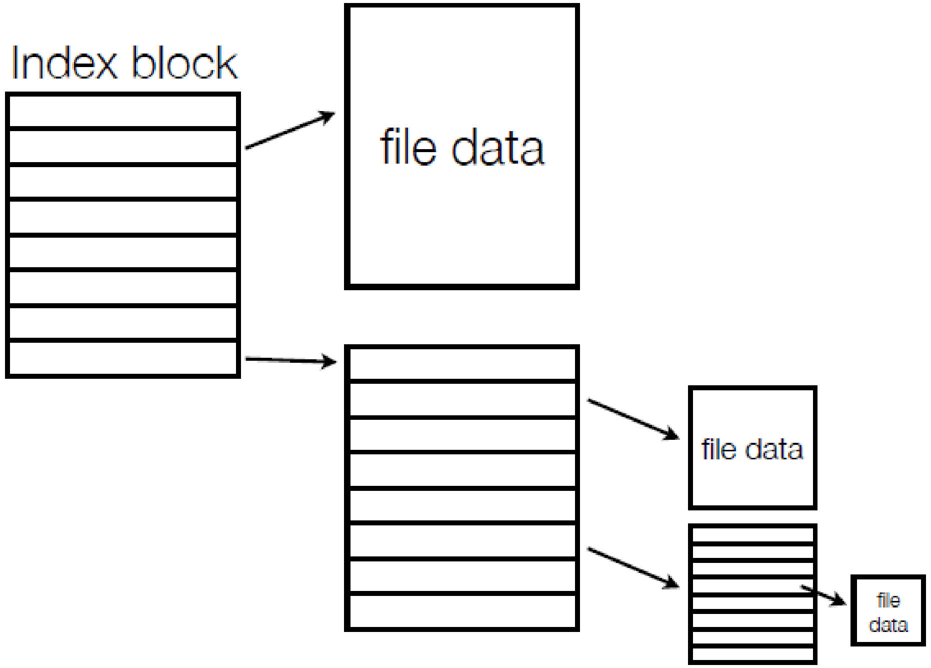
User view	VFS mapping
/bin/	--> normal ext2 file system
/boot/	--> read only disk partition
/home/	--> network mounted file system



- Each file system implements its own functionality for how to use these objects

Linux
File Systems

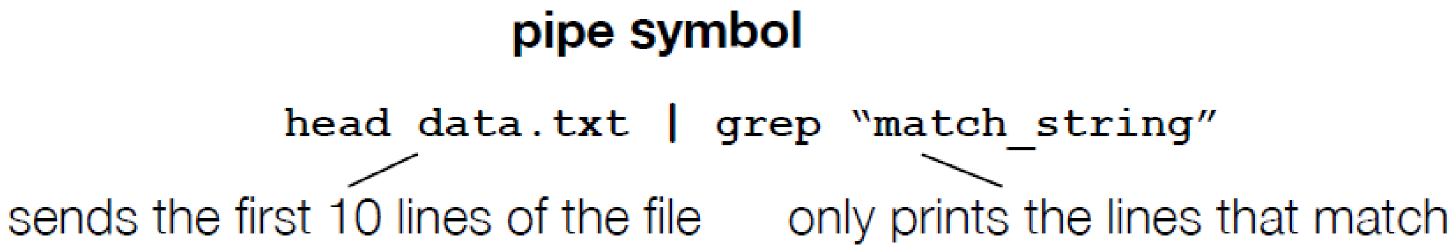
- **ext2fs** and **ext3fs** are the most common Linux file systems
  - But it supports dozens more
- Uses **multi-level indexes** to store and locate file data
  - Up to 3 levels of indirection
  - Allows for very large files
  - Still has good performance for small files
- Uses (small) 1KB blocks on disk
  - Allocator places blocks **near each other** to maximize read performance



Linux

IPC – Interprocess Communication

- **Simplest way to send a stream of data from one process to another?**
- **Pipes** - simple communication channel between a pair of processes
  - First process can send messages to second process
  - Linux sets up the pipe and manages the communication between the processes



pipe symbol

```
head data.txt | grep "match_string"
```

sends the first 10 lines of the file

only prints the lines that match

Linux
IPC – Interprocess Communication

- ❑**Signals** - used to alert a process of an event
  - ❑just raises a flag, carries no extra information
  - ❑Each process has a signal table which tells how it responds to signals
  - ❑**Ctrl+C** = send cancel/kill signal to a process (usually)
    - ❑ process can register its own functions to call when a signal is received
- ❑**Shared Memory** - very fast data sharing between processes
  - ❑Process can map a region from another's address space
  - ❑Requires additional mechanisms such as **locks** to be used safely

Linux

An In-Depth Profile

- ❑Questions? ☺