# Unix System Programming

**Specially Design For system Programing**

**BY**
**Wasim Malik**

# Outline

- Important course information
  - Objective
  - Roadmap
  - Requirement & policy
- Brief history of GNU/Linux
- GNU/Linux Architecture
- Getting started
  - Log in, simple command, shell

# You’ve written programs…

- Lot of times, one can use existing tools to implement new functionalities

For Example,What kind of applications?

- Real world applications are complicated:
  - generate input & output, or have GUI

    Like whatsapp type apps?

  - communicate with other program (local or remote)

    ?

  - use multiple processes or threads for improved interactivities
  - Needs to be profiled/tested to improve performance

The main difference between Time-Sharing Systems and Multiprogrammed Batch Systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.

Resource: https://www.tutorialspoint.com/time-sharing-operating-system

# About this course

Time-sharing enables many people, located at various terminals, to use a particular computer system at the same time.

Processor’s time is shared among multiple users simultaneously is termed as time-sharing.

- UNIX: time-sharing operating system, consisting of
  - kernel (program that controls and allocates syst system)

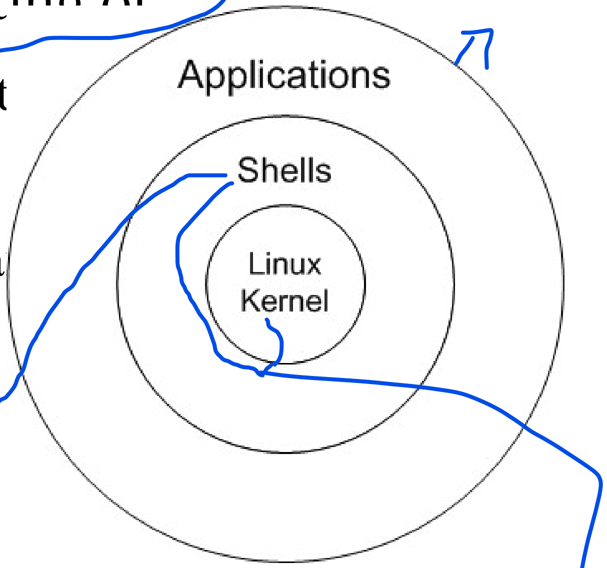
Kernal Doc: https://www.ionos.com/digitalguide/server/know-how/what-is-a-kernel/

  - Essential programs: compilers, editors, comma
- Linux is a variation of Unix

LINUX is developed from UNIX and This is open source.

  - programming environment is very similar

Operating System



shell (that is, the user interface)

How do I become a good kernel developer?
In kernel development every day a problem will arise and you have to solve it in a way that the code runs fast and you have a small amount of memory. Competitive programming is very beneficial for a kernel developer. You can use GeeksforGeeks to practice competitive programming.

Resource doc: https://www.geeksforgeeks.org/5-tips-to-make-a-career-as-a-linux-kernel-developer/#:~:text=5%20Tips%20to%20Make%20a%20Career%20as%20a,kernel%20...%205%205.%20Do%20some%20Competitive%20Programming

Kernel developers focus on interfaces, data structures, algorithms, and optimization for the core of the operating system.

You can picture the kernel as a seed or pit and the shell as the fruit that surrounds the pit.

In Linux, the graphic interface is independent from the kernel? //Confusion

When a computer powers up, the kernel is the first thing that’s loaded into the RAM. This happens in a protected area, the bootloader, so that the kernel can’t be changed or deleted.

Confusion!

Confusion!

Application programs run separately from the kernel in the operating system and merely draw on its functions. Without the kernel, communication between programs and hardware wouldn't be possible.

NOTE: Several processes can run simultaneously thanks to the multitasking kernel.
But it's generally the case that only one action can be processed by the CPU at one time
– unless you're using a multicore system. The rapid change in processes that gives the
impression of multitasking is taken care of by the scheduler.

# About this course

- Many levels of programming:

NOTE: Linux systems and Android devices use a Linux kernel. Windows uses the NT kernel, which various subsystems draw on. Apple uses the XNU kernel.

Stronger awareness about hardware and operating system

System programmers write daemons, utilities, and other tools for automating common or difficult tasks.

Low-level:
kernel programming
device driver programming

System programming

Application programming

High-level:
Web programming

Device drivers use the interfaces and data structures written by the kernel developers to implement device control and IO.

application programming aims to produce software which provides services to the user directly

An operating system is software installed on a computer's hard drive.

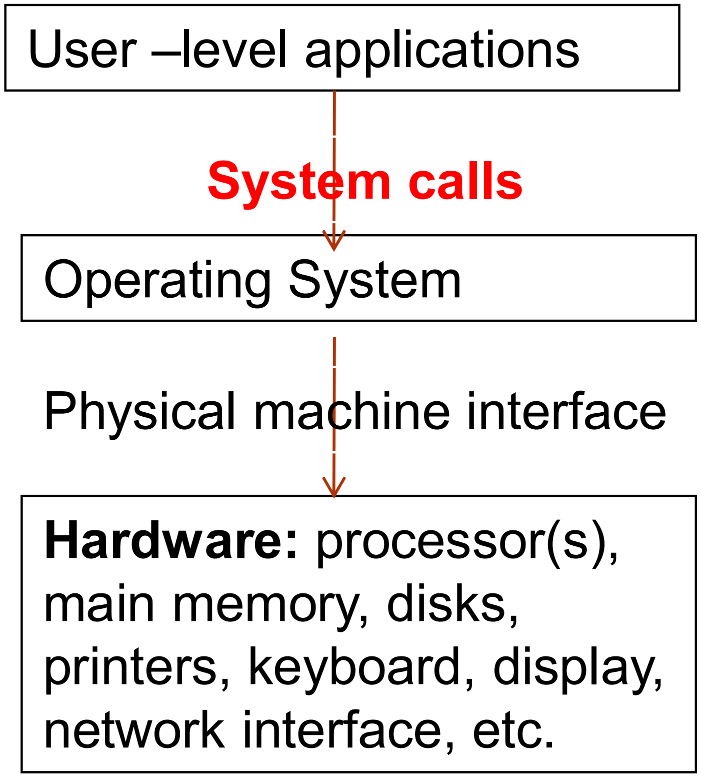
An operating system is the primary software that manages all the hardware and other software on a computer.

# Operating System, Kernel

- operating system: two different meanings
  - the entire package consisting of central software managing a computer's resources and all of accompanying standard software tools, such as command-line interpreters, graphical user interfaces, file utilities, and editors.

    اس کے ساتھ
  - central software that manages and allocates computer resources (i.e., CPU, RAM, and devices).
- kernel is often used as a synonym for second meaning

# What is Operating System?



User –level applications

**System calls**

Operating System

Physical machine interface

**Hardware:** processor(s), main memory, disks, printers, keyboard, display, network interface, etc.

- From app. programmer's point of view:
  - O.S. manages hardware resources
  - O.S. provides user programs with a simpler interface, i.e. system calls
    - cnt=read(fd, buffer,nbytes)
    - getc() etc.

System calls written in C language...

- We will encounter OS concepts, inevitably.

# Kernel Functionalities: Process scheduling

is we speak processor the CPU?

- Managing one or more central processing units (CPUs),
- Unix: a preemptive multitasking operating system
  - multiple processes (i.e., running programs) can simultaneously reside in memory and each may receive use of the CPU(s).
  - Preemptive: scheduler can preempt (or interrupt) a process, and resume its execution later => to support interactive responses
    - the processors are allowed to spend finite chunks of time (*quanta, or timeslices*) per process

preemptive multitasking operating system divides the overall operating and computing time between processes, and the switching of resources between different processes occurs through predefined criteria. Preemptive multitasking is also known as time-shared multitasking.

Doc: https://www.geeksforgeeks.org/difference-between-preemptive-and-cooperative-multitasking/

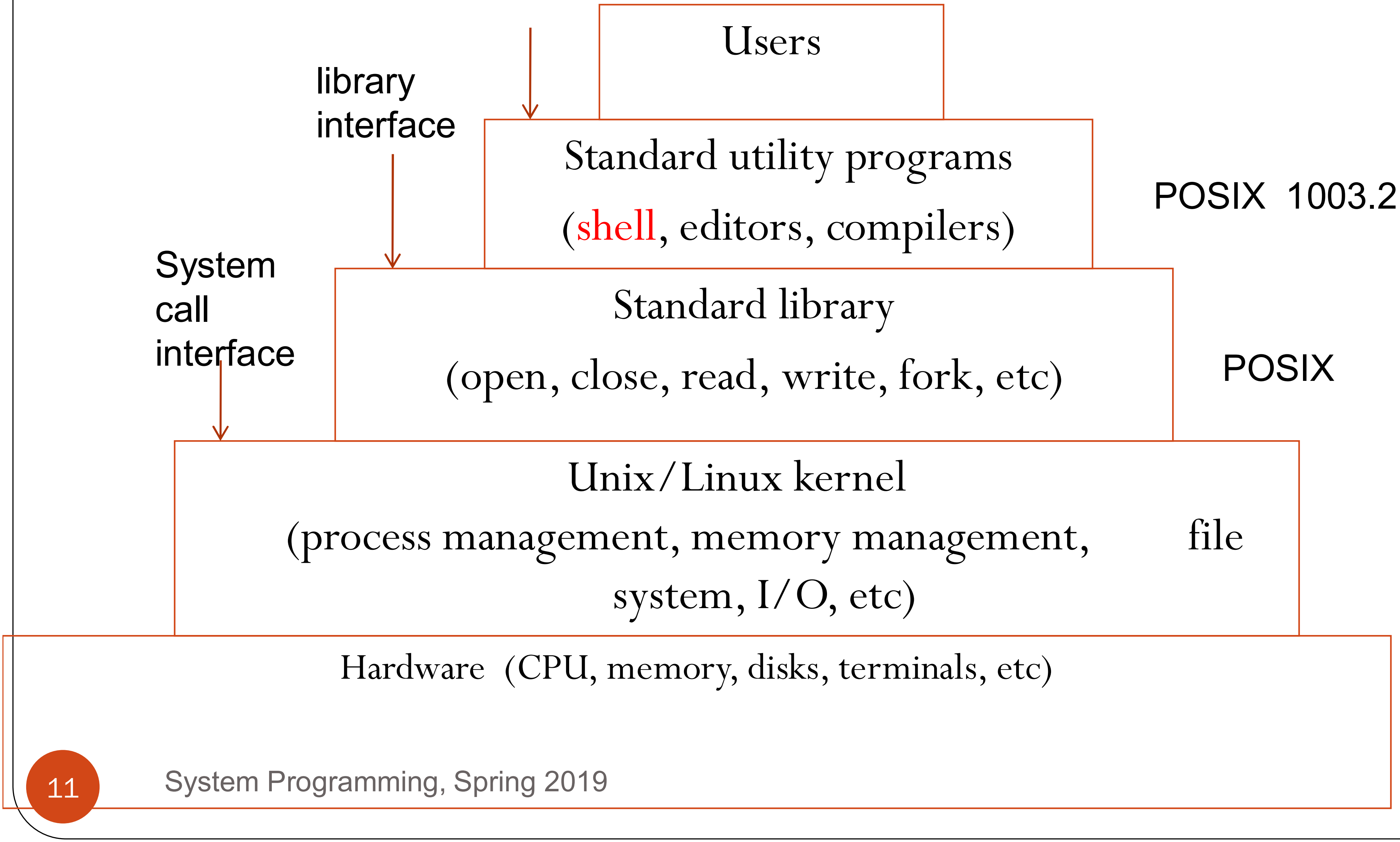
# Kernel Functionalities: Memory management

- Manage physical memory (RAM) to be shared among processes in an equitable and efficient fashion
- Virtual memory management:
  - Processes are isolated from one another and from the kernel, so that one process can't read or modify the memory of another process or the kernel. kept little portion of video in RAM that part is running!
  - Only part of a process needs to be kept in memory, thereby lowering the memory requirements of each process and allowing more processes to be held in RAM simultaneously.
  - better CPU utilization, since it increases the likelihood that, at any moment in time, there is at least one process that the CPU(s) can execute.

# Other OS functionalities …

- The kernel provides a file system on disk, allowing files to be created, retrieved, updated, deleted, and so on.
- Creation and termination of processes
- Peripheral device: standardizes and simplifies access to devices, arbitrates access by multiple processes to each device
- Networking: transmits and receives network packets on behalf of user processes.
- Support system call interfaces: processes can request the kernel to perform various tasks using kernel entry points known as system calls.
  - Second part of this course: Unix system call API

# Layers in UNIX/Linux System



Users

Standard utility programs

(shell, editors, compilers)

library interface

POSIX 1003.2

Standard library

(open, close, read, write, fork, etc)

System call interface

POSIX

Unix/Linux kernel

(process management, memory management, file system, I/O, etc)

Hardware (CPU, memory, disks, terminals, etc)

# Goal of this course

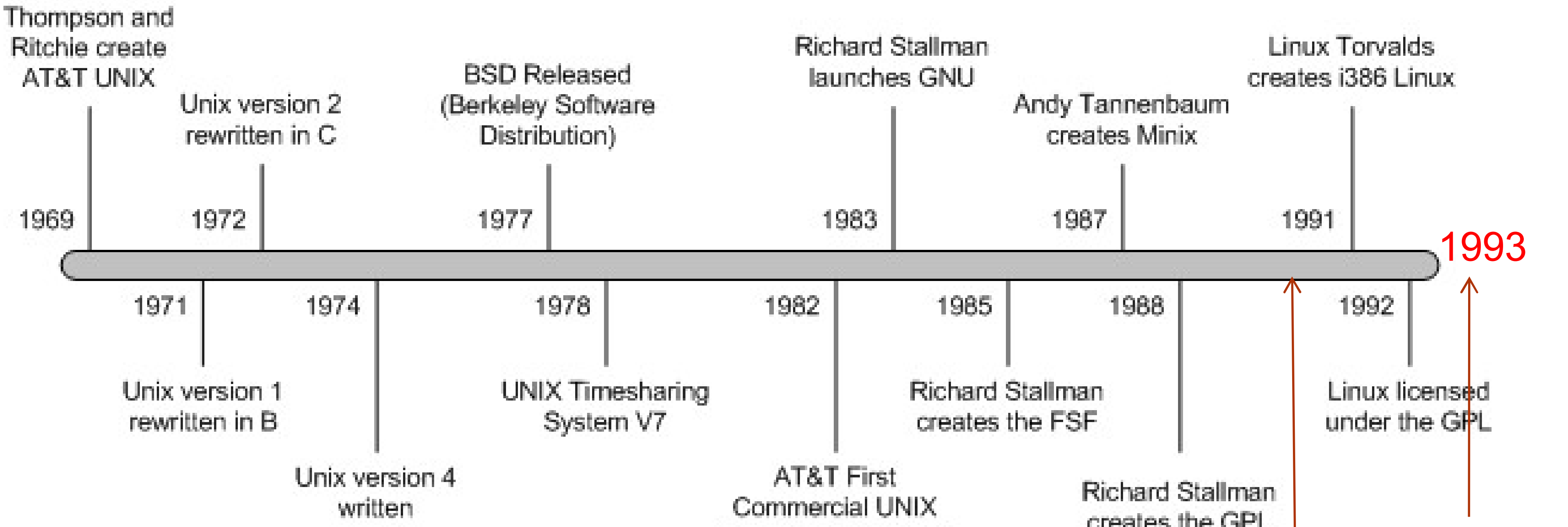
- Learn tools needed for develop application in GNU/Linux
  - A working understanding about UNIX
  - Basic commands, shell scripting
  - GNU tools for app. development
    - compiler, debugger, make, version control, library, testing/profiling tools
  - System calls provided in Unix:
    - to request services from operating system

# Roadmap: a top-down approach

- Get started topics
  - Basic concepts & useful commands
- vi,emacs, sed, awk
- Bash programming
- Basic GNU Tools
  - Compiler chain, make, debugger (gdb)
- Unix system calls
- Advanced GNU Tools
  - Library, gcov, gprof, version control tools

# Now let’s get started with some background information.

# Timeline of Unix/Linux, GNU



1969 Thompson and Ritchie create AT&T UNIX

1971 Unix version 1 rewritten in B

1972 Unix version 2 rewritten in C

1974 Unix version 4 written

1977 BSD Released (Berkeley Software Distribution)

1978 UNIX Timesharing System V7

1982 AT&T First Commercial UNIX System (System III)

1983 Richard Stallman launches GNU

1985 Richard Stallman creates the FSF

1987 Andy Tannenbaum creates Minix

1988 Richard Stallman creates the GPL

1991 Linux Torvalds creates i386 Linux

1992 Linux licensed under the GPL

1993

Windows NT

Windows 3X

Dennis Ritchie

Ken Thompson

PDP-11

# GNU history

- GNU: GNU is Not Unix
- **Richard Stallman** (author of Emacs, and many other utilities, ls, cat, …, on linux)
  - 1983: development of a free UNIX-like operating system
  - Free Software Foundation (100s of Programmers)
- Free software:
  - freedom to run the program, for any purpose.
  - freedom to study how the program works and adapt it to your needs.
  - freedom to redistribute copies so you can help others.
  - freedom to improve the program and release your improvements to the public, so that everyone benefits.

# GPL License

- GNU General Public License is a free, copyleft license for software and other kinds of works…
  - “The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users.”
- Manual pages for commands include copyright info:

COPYRIGHT

Copyright © 2011 Free Software Foundation, Inc. License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.

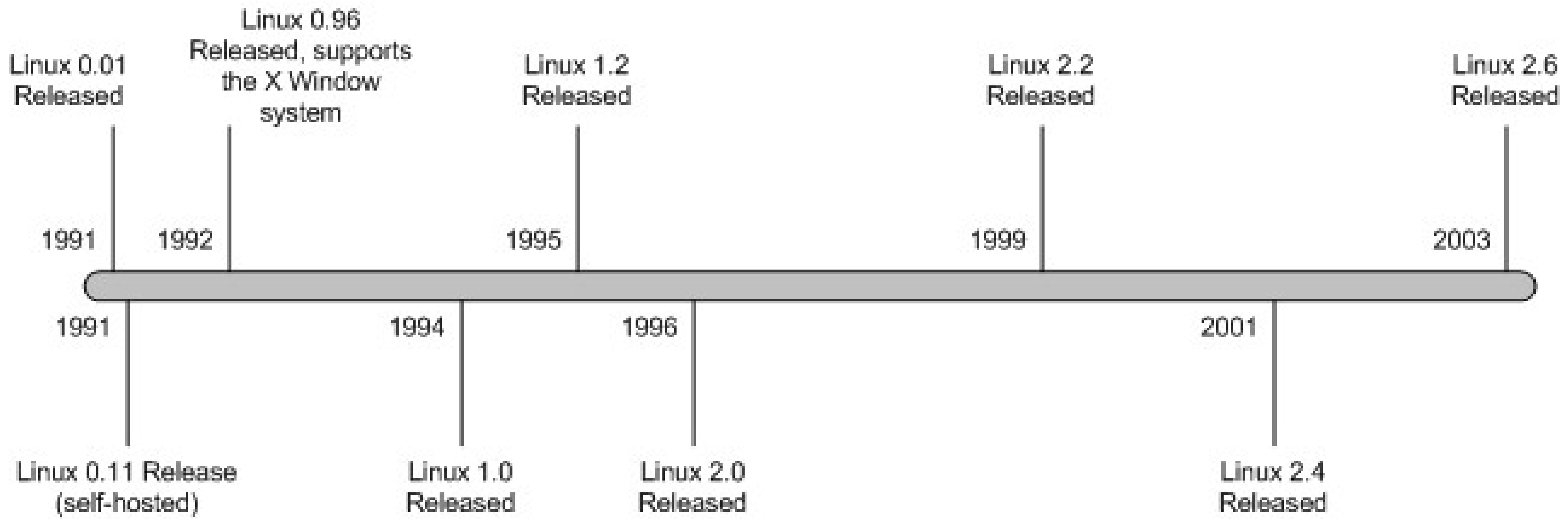
This is free software: you are free to change and redistribute it. There is NO WARRANTY, to the extent permitted by law.

# Linux history

- Linus Torvalds
  - 1991: "hobby" operating system for

  i386-based computer, while study in Univ. of Helsinki
- 1996: Linux becomes a GNU software component
- GNU/Linux: A fairer name than Linux?
  - "Most operating system distributions based on Linux as kernel are basically modified versions of GNU operating system. We began developing GNU in 1984, years before Linus Torvalds started to write his kernel. Our goal was to develop a complete free operating system. Of course, we did not develop all the parts ourselves—but we led the way. We developed most of the central components, forming the largest single contribution to the whole system. The basic vision was ours too." --- RMS

# Linux kernel versions



Use "uname –a" to check system information (including kernel version).

# Linux version history

- 1.0: only supported single-processor i386-based computer
- 1.2 support for computers using processors based on the Alpha, SPARC, and MIPS architectures.
- 2.0:  SMP support (symmetric multiple processors) and support for more types of processors
- 2.2: removed global spinlock, improved SMP support, support m68k and PowerPC architectures, new file systems (including read-only support for Microsoft's NTFS)
- 2.4.0: support for ISA Plug and Play, USB, and PC Cards, PA-RISC processor,  Bluetooth, Logical Volume Manager (LVM) version 1,  RAID support, InterMezzo and ext3 file systems.

# Linux version history

- 2.6.0 : integration of μClinux , PAE support, support for several new lines of CPUs, integration of ALSA , support for up to $2^{32}$ users , up to $2^{29}$ process IDs, increased the number of device types and the number of devices of each type, improved 64-bit support, support for file systems of up to 16 terabytes, in-kernel preemption, support for the Native POSIX Thread Library (NPTL), User-mode Linux integration into the mainline kernel sources, SELinux integration into the mainline kernel sources, …
- 3.0 : 21 July 2011. the big change was, "NOTHING. Absolutely nothing." "…let's make sure we really make the next release not just an all new shiny number, but a good kernel too." , released near the 20th anniversary of Linux

# Understanding uname

```
$ uname -a
Linux storm.cis.fordham.edu 3.6.11-1.fc16.x86_64 #1 SMP Mon Dec 17
    21:29:15 UTC 2012 x86_64 x86_64 x86_64 GNU/Linux
```

- Kernel name: Linux:
- Hostname
- Kernel release: 3.6.11-1.fc16.x86_64
- Kernel version: #1 SMP Mon Dec 17 21:29:15 UTC 2012
- Machine hardware name:  x86_64 (AMD64 instruction set)
- Processor:x86_64
- Operating system: GNU/Linux

# Unix Standardization

- Different implementations of Unix diverged:
  - Different meaning for command options
  - System calls syntax and sementatics
- POSIX, "**P**ortable **O**perating **S**ystem **I**nterface", is a family of standards specified by IEEE for maintaining compatibility between Unix systems.
  - C library level, shell language, system utilities and options, thread library
  - currently IEEE Std. 1003.1-2004
- **POSIX for Windows:**
  - Cygwin provides a largely POSIX-compliant development and run-time environment for Microsoft Windows.

# Single Unix Specification

- **1990:** X/Open launches XPG3 Brand. OSF/1 debuts.
- **1993:** Novell transfers rights to "UNIX" trademark and Single UNIX Specification to X/Open.
- **1994:** X/Open introduces **Single UNIX Specification**, separating the UNIX trademark from any actual code stream
- **1995:**X/Open introduces UNIX 95 branding program for implementations of Single UNIX Specification.
- **1996** : **Open Group** forms as a merger of OSF and X/Open.

# X/Open => Open Group

- **1997:** Open Group introduces Version 2 of Single UNIX Specification, including support for realtime, threads and 64-bit and larger processors.
- **1998:** Open Group introduces UNIX 98 family of brands, including Base, Workstation and Server. First UNIX 98 registered products shipped by Sun, IBM and NCR.
- **1999:** Open Group and IEEE commence joint development of a revision to POSIX and the Single UNIX Specification.
- **2001: Version 3 of the Single UNIX Specification unites IEEE POSIX, The Open Group and the industry efforts.**

# Today’s Unix Systems

- To be an officially Unix system, need to go through certification based on the Single Unix Specification
- Registered Unix systems: AIX, HP/UX, OS X, Reliant Unix, ….
- Linux and FreeBSD do not typically certify their distributions, as the cost of certification and the rapidly changing nature of such distributions make the process too expensive to sustain.