AI224: Operating Systems — Lab 1

Dr. Karim Lounis

Jan - Jun 2025

Unix ... GNU/Linux

MULTICS

Everything started in the ‘60s when punched cards were still used. A cooperative project, called MULTICS (Multiplexed Information and Computing Service), started in 1964, led by MIT, General Electric, and AT&T Bell Labs, aimed at the development of a time-sharing operating system, Multics, to operate on GE-645 mainframes. The project failed in 1969.



Some employees working on punched-cards computers (on the left), a GE-645 mainframe (in the center), and Denis Ritchie and Ken Thomson (from AT&A Bell Labs) working on a PDP-11/20 (on the right).

Unix

- Ken Thomson, who worked on the project decided to continue working on the project to, at least, create something out of the project aches.
- He formed a team in **1970** and built Unics, which later became Unix.

  The first version of Unics (Unix v1.0) was fully written in assembly and was run on DEC PDP-11/20 minicomputers.
- In 1972, Denis Retchie, a programmer from the team, developed the C programming language.

  In the meantime, the team was progressing and upgrading Unix to v4.0.
- They decided to rewrite Unix in C, creating Unix v5.0 in **1972**.
- At that time, AT&T was forbidden from entering the computer market after some legal issues.

Unix Live Free or Die

- AT&T decided to license the source code of Unix to third commercial and academic parties.

  E.g., UC Berkeley, Microsoft, IBM, DEC, HP and Sun Microsystems.
- In **1977**, the University of California Berkeley was licensed the source code and started developing the system further creating **BSD**.
- In **1980**, Microsoft created Xenix based on Unix v7.0.
- In **1984**, AT&T and Bell Labs got separated and AT&T started selling a commercial Unix version called **System V**.
- From System V and BSD, many versions of Unix were created.

  E.g., HP-UX (1984) and IBM AIX (1986) were both based on System V. Whereas, the ULTRIX (1984) from DEC (for PDP-11 and VAX) and Sun Microsystems Solaris (1981) were based on BSD.

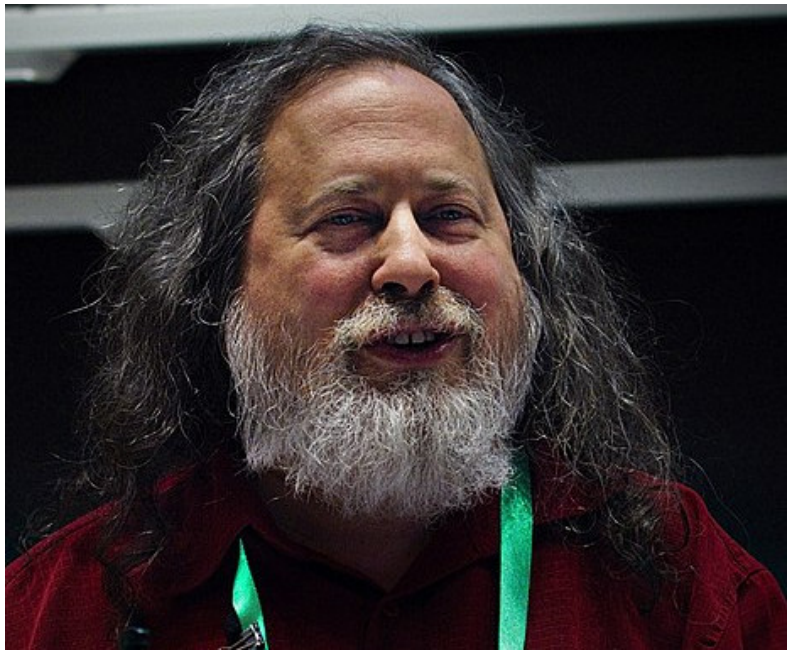
Unix Live Free or Die

- The UC Berkeley started replacing AT&T files with their own files to become separate from AT&T and System V. It managed to publish its source code and create FreeBSD in **1983**.

  E.g., Of course, there was a lawsuit with AT&T and got dropped.

GNU Project

- In **1983**, Richard Stallman founded the GNU foundation (which stands for GNU is Not Unix) with the aim of developing an operating system as a free software replacement for Unix (1984).



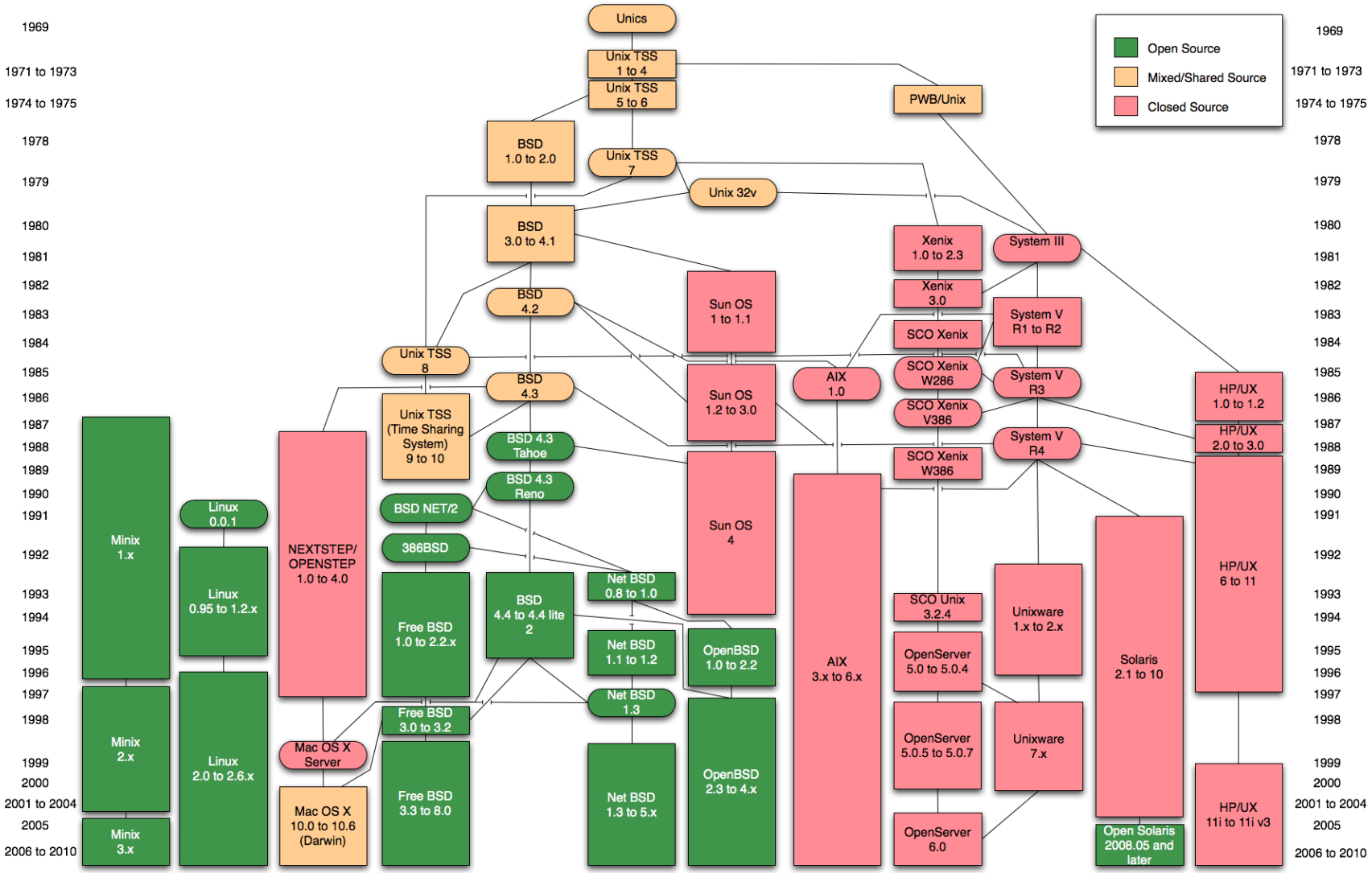
Richard Stallman (Left) and the GNU project logo (right).

GNU Project

- In the early '90s, the GNU system was almost complete but missing an important part, which was the kernel (GNU Hurd was legging behind).
- Few year earlier (1987), somewhere in The Netherlands, Andrew Tanenbaum wrote MINIX, a Unix-like operating system, particularly developed as an educational tool to teach operating systems and get rid of AT&T licence issues.
- At that time, in Finland, Linus Torvald, a student, who used MINIX and knew about Unix wanted to overcome the deficiencies of MINIX and wrote his own operating system (with a kernel called **Linux**) which he wanted to make public.

In **1991**, he released Linux under GPL (GNU General Public Licence). The combination of Linux kernel and the incomplete GNU system made a completely free operating system, called GNU/Linux operating system.

Unix Consequences



GNU GPL

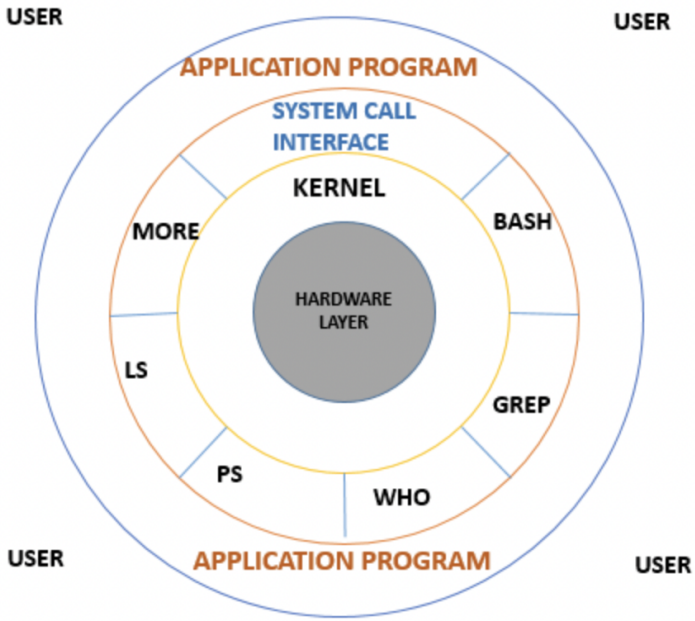
The GNU General Public License (GNU GPL) is a series of widely used free software licenses that guarantee end users the four freedoms to run, study, share, and modify the software. These GPL series are all copyleft licenses, which means that any derivative work must be distributed under the same or equivalent license terms.

Linux Kernel

- The Linux kernel is not the Unix kernel. It does not use Unix source code but uses Unix philosophy and architecture.
- Many operating systems nowadays use the Linux kernel. E.g., the Android operating system uses the Linux kernel. Also, 60% of servers outside use Linux. It also runs on embedded systems, including routers, smart home devices, video game consoles, televisions (e.g., Samsung and LG Smart TVs), automobiles (e.g., Tesla, Audi, Mercedes-Benz, Hyundai, and Toyota), and spacecraft (e.g., Falcon 9 rocket).
- Three are thousands of GNU/Linux distributions: Red Hat, Fedora, Mandriva, Arch, Ubuntu (spyware), Debian, Backtrack, Kali, Parrot, Linux Mint, Yellow dog, and Kubuntu, these are just distributions (different desktops, . . .) but the same Linux Kernel.

Unix-like Systems

- These are a family of operating systems that follow and adopt the Unix philosophy as well as the Unix architecture.
- **Unix Architecture.** A Unix system consists of a kernel (includes system calls), a shell (commands interpreter), internal commands (e.g., who, ls, ps, grep, and more), and application programs (e.g., skype, Mozilla Firefox, and more).



Unix-like Systems

- There are different variants of Unix shells: Thomson shell (1971), csh (C shell by University of California — Berkeley) and tcsh used in BSD, Bourne shell (used in Unix v7, a.k.a, sh by ATT Bell Labs), ksh (Korn shell by ATT Bell Labs), the GNU Bourne-again shell (bash), ... Zshell, etc.
- **Unix Philosophy.** Over time, the leading developers of Unix (and programs that ran on it) established a set of cultural norms for developing software; these norms became as important and influential as the technology of Unix itself, and have been termed the "Unix philosophy".
- The Unix philosophy emphasizes building simple, compact, clear, modular, and extensible code that can be easily maintained and repurposed by developers other than its creators. The Unix philosophy favors composability as opposed to monolithic design.

Unix Philosophy

The Unix philosophy is documented by Doug McIlroy in the Bell System Technical Journal from 1978:

1. Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features".
2. Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid stringently columnar or binary input formats. Don't insist on interactive input.
3. Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them.
4. Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them.