Faculty of Computer Science, Dalhousie University *7-Sep-2018*

CSCI 2132 — Software Development

Lecture 2: Introduction to UNIX and Unix-like Operating Systems

Location: Chemistry 125 Instructor: Vlado Keselj
Time: 12:35 – 13:25

Previous Lecture

- Course Introduction
  - logistics and administrivia
  - important dates, course description
  - evaluation scheme and criteria
  - textbooks
  - lectures, exams, assignments, lab work
- Academic integrity policy
- Culture of respect
- Main learning objectives
- Motivation: Why UNIX, Why C
- Tentative list of course topics

Part I

Unix Operating System

2 Introduction to Unix

2.1 Operating System Overview

Slide notes:

Part 1: Unix Operating System

- Reading: Unix book, Chapter 1
- In the first part we will refresh our general knowledge about operating system,
- learn more details about the Unix-style operating systems
- learn about shell as a command-line interface
- learn about the file system
- learn about utilities and some tools used in software development

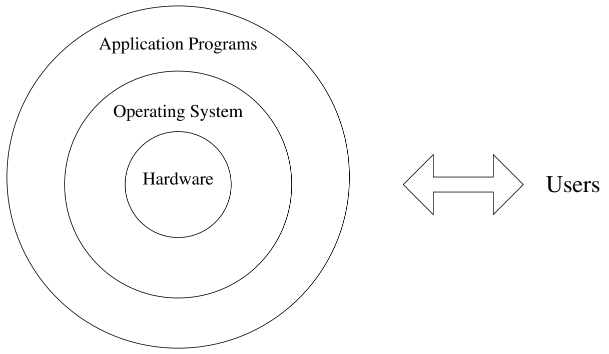
Some Functions of an Operating System

- Provides an interface between application programs and the hardware
  - E.g., reads keyboard, writes to screen, writes and reads from disks, sends data to printer, communicates with network card, . . .

  - Hides the complexity of hardware interfaces from application programs
  - Protects the hardware from user mistakes and programming errors (to prevent crashes)
- Manages the hardware resources of the computer systems
  - CPU time, disk space, memory access, ...
- Protects user's programs and data from each other (security issues)
- Supports inter-process communications and sharing
- Provides resource sharing among users and processes

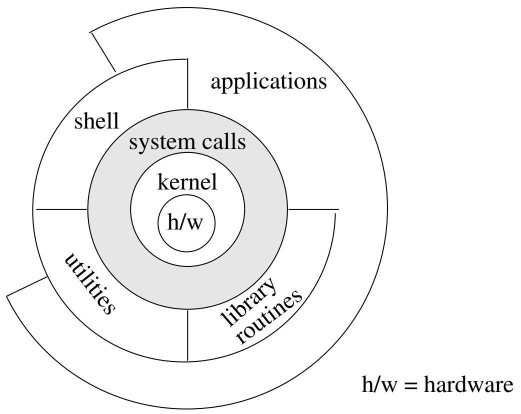
Overview of Unix-style Architecture

- The "Onion Skin Model" of Operating System



A more-detailed Onion-skin Model

Adapted from "Advanced Programming in the UNIX Environment" by W. Richard Stevens and Stephen A. Rago:



This diagram looks like the onion skin model, but with some further details specific to UNIX.

The **kernel** is the hub of the operating system. It allocates time and memory to programs, handles file storage, and deals with many other tasks critical to the functioning of a computer. It also handles communications in response to system calls. A **system call** is how a program requests a service from the kernel. A system call is usually implemented as a library function with the same or very similar name. At the machine code level, a system call usually works by preparing the appropriate parameters in the CPU registers and by making a special jump into the kernel code. This jump may be implemented as a software interrupt, and the CPU typically changes state from a "user" mode into a "protected" mode, which has more privileges. Some examples of system calls are deleting a file, or making a directory.

The **shell** is a text-based interface to the UNIX system. It is a command-line interpreter that interprets user commands and arranges for their execution. It is usually assumed that a shell is text-based, but GUI (graphical user interface) window managers act as shells in many ways, and are sometimes also called shells.

This diagram also shows that applications programs can use the shell, utilities, and library routines. They can also perform system calls to request service from the kernel directly.

2.2 A Brief History of UNIX

A Brief History of Unix: Multics

– Multics OS started in 1964, 5 years before Unix
– Developed by Ken Thompson, Dennis Ritchie, and others
– Collaboration of MIT, AT&T (Bell Labs), and GE for GE-645 computer
– Advances system with many features and an idea of "computing utility"
– Hardware did not keep up with sofware, so it was slow and expensive to run
– AT&T withdrew from the project and Ken Thompson started working on a new system

The work on the operating system Multics ((MULTiplexed Information and Computing Service) was started in 1964, 5 years before UNIX. It was developed by Ken Thompson, Dennis Ritchie, Doug McIlroy, and J. F. Ossanna. It was a multi-user OS developed by MIT, Bell Labs, and GE for the GE-645 system, with an idea of building a "computing utility," somewhat like the current idea of "cloud" computing.

The system had many advanced capabilities, including security mechanisms. It included the concept of dynamic linking, hot-swapping of components, hierarchical file system, and multi-ring security. It relied on some special features of the GE-645 system.

The hardware at the time was not up to the demands of th software, so it was slow and expensive to run. As an anecdotal example, Ken Thompson wrote a game Space Travel, which costed $75 dollars to play.

After AT& withdrew from the project, Ken Thompson started working on a new system.

UNIX: Created in 1969 by Ken Thompson

Ken Thompson; Dennis Ritchie and Ken Thompson (sitting) at PDP-11

UNIX: 1969– Development

– 1969: Implemented for an old PDP-7 in assembly language on a GE system
– 1970-3: PDP-11, C language, reimplemented in C, pipes, called it UNIX (Brian Kernighan)

- 1973-9: Source code available to universities, PDP-11 machines, very popular, very quickly
- 1980s: Commercialization, System V, BSD, GNU (1985)
- 1991: Linux (by, Linus Tornvalds), or GNU/Linux, new code, distributions
- Other Unix/Linux-based OS's: Chrome, Android, MacOS, etc.

UNIX was created in 1969 by Ken Thompson. Some lessons learned from MULTICS are used in the creation of UNIX, so MULTICS can be regarded as a predecessor of UNIX.

At that time, Ken Thompson, Dennis Ritchie, Rudd Canaday started talking about a system that had:

1. a developer friendly environment
2. encouraged "fellowship among users"
3. file system for multiple users
4. was efficient and simple

The first implementation of UNIX was developed for a PDP-7 computer. The system was written in an assembly language on a GE system. After being assembled, it was copied on a paper tape, carried to the PDP-7 and ran.

UNIX on PDP-11 (1970s)

- From 1970 to 73
  1. ported it to PDP-11
  2. designed and implemented the C language
  3. reimplemented it in C
  4. **pipes** were implemented in Unix
  5. called it Unix (suggested by Brian Kernighan)
- 1970s
  1. Unix source code was made available to universities
  2. Licensed to universities
  3. Universities were buying PDP-11's
  4. Unix became very popular very quickly

In the period from 1970 to 1973, UNIX was ported to a new computer, PDP-11. During this period, the C programming language was designed, and a compiler was implemented on UNIX. After this, the UNIX operating system itself was re-written in C. The system was named UNIX, at a suggestion by Brian Kernighan, and pipes were implemented in the system.

1980s Unix Wars

- Many companies (IBM, Bell, Sun, etc) developed their own versions of Unix; Xenix was available for PCs, being licensed by Microsoft from AT&T
- Ceased to be free, lots of commercialization
- Not a good time for the hobbyist (or academic)
- Different flavours/versions emerged
  - System V (AT&T) vs. BSD Unix (UC Berkeley)
- 1985 GNU begins development of a free Unix, except kernel; **G**NU's **N**ot **U**NIX
- 1991 Linus Torvalds develops the Linux kernel

Linux

- 1991: Linus Torvalds announced the project
- Open-source, UNIX-like OS kernel

– Does not share code with UNIX
– Usable in 1992
– Essentially GNU/Linux
– Various distributions available: Fedora, Ubuntu, etc.

More Reading about UNIX History

– You can read a bit more about UNIX history in the book by Nemeth *et al.*, *Unix and Linux System Administration Handbook* in the section 'A Brief History of System Administration'
– UNIX had many advanced features from early days, such as concurrent execution

Even though it was one of the earliest operating systems and designed in a simple and elegant way, it is interesting that Unix had many advanced features that were seen much later in other operating systems. One of these features is ability to run several processes concurrently, or in the "same time" so to speak. This feature is not easy to implement since a computer is inherently a sequential machine: A CPU executes one instructions and moves to the next one, or jumps to another location in memory.

2.3 The UNIX Philosophy and Main Features

Unix Philosophy

– Write programs that...
    – handle text streams; because that is a universal interface
    – work together; because then they can be easily combined
    – do one thing and do it well
– This allows for simple, elegant, and robust solutions
– Programs (utilities) can be combined into pipes
– A typical user is a programmer
    – can decompose problems into subproblems, used to concise syntax, understands data flow

The Context: When UNIX was created

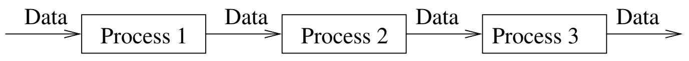
– The hardware was small and slow
    1. If you had multiple users, they could not run large programs, no sufficient memory or CPU power
    2. Programs had to be small
– The users were programmers
    1. Know how to decompose a problem into subproblems
    2. Used to short concise syntax
    3. Know how to think about data flow

UNIX Paradigm

– Many small programs for doing different things
– All programs have the same interface (text)
    1. Input is a text stream (optional)
    2. Output is a text stream
    3. Programs are also called "filters"
– To do bigger tasks "combine" small programs together
– Use pipes to combine these small programs
– Create a single "pipeline" of filters connected by pipes

The Concept of Pipeline

- Pipes are used to specify that the output of one process is to be used as the input to another process



- Example:
  ```
  who | sort
  ```
- Symbol '`|`' is called 'pipe'
- Related to every process having three default I/O channels: stdin, stdout, and stderr (standard input, standard output, standard error output)

A pipe is a mechanism in which the standard output of one process is directed into the standard input of another process. A pipeline is a sequence of several running processes, where each process's standard output is re-directed into the standard input of the next process. The first process does not have its standard input re-directed, and the last process does not have its standard output re-directed. The processes are executed concurrently.

Douglas McIlroy:

This is the Unix Philosophy:
Write programs that do one thing and do it well.
Write programs to work together.
Write programs that handle text streams, because that is a universal interface.

It can be described as a software "Toolbox philosophy":

1. A standard Unix environment has many small programs (tools)
2. Pipes are used to combine the tools
3. The tools can be combined in many different ways to solve many problems

Unix Unifying Principles: A general principle that Unix follows is that the design should be kept simple and elegant, as much as possible. As a consequence, there are several more detailed unifying principles, such as follows:

1. **Files and Processes:** Rather than having a large variety of named constructs, most things are either a process or a file.
2. **Pipes** are used to connect programs, by passing output of one program to input of another program.
3. **Process creation and communication:** Creating new processes, i.e., process spawning, and interprocess communication are well-supported, flexible and cheap.
4. **Communication between programs is simple:** The programs do not need to know about each other. As McIllroy stated, the programs are designed "to operate not specifically with each other, but with programs as yet unthought of".

Some Notable Features of UNIX

- Allows many users to access a computer system at the same time
- Shares CPU's, memory, and disk space in a fair and efficient manner among competing processes (CPU time is split in "slices", typically 1/10 seconds)
- Processes and peripherals talk to each other even on different machines
- Provides a well-defined set of system calls similar to library routines
- Very portable