CSE 550: Systems for All

Lecture 2: The UNIX Time Sharing System

Unix Paper

Classic system and paper

All key components described almost entirely in 10 pages

- File system, processes, forking, shell, etc.

Won a Turing award for this (1983)

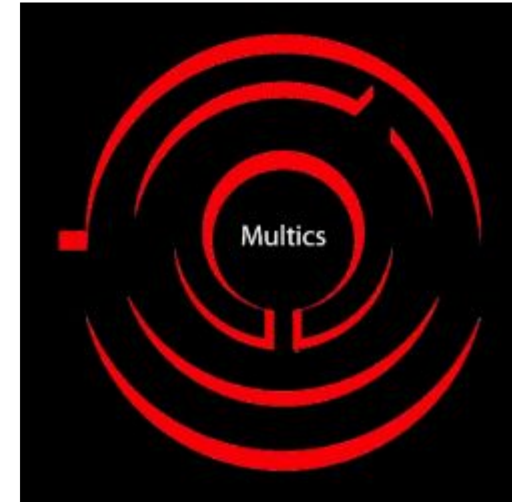
Precursor: Multics

Designed at MIT in 1964 (w/ Bell Labs and Honeywell)

- Followed Compatible Time-Sharing System also at MIT

Advanced lots of things:

- Virtual memory (implemented as three levels)
  - OS keeps relevant info on the chip
- Multi-user
- Dynamic linking
- Security
- Lots and lots and lots of other things



"The **second-system effect** (or **syndrome**) is the tendency of small, elegant, and successful systems to be succeeded by over-engineered, bloated systems, due to inflated expectations and overconfidence."

Unix

Name is pun on Multics

Platform: PDP-11 computer; operational in 1971

   Written in C instead of assembly; 33% overhead but *portable*

2 man-years to write

Defined an ecosystem — the Unix tools

Developers used/built the system for their own work (“dogfood”)

Unix Features

Time-sharing system – contrast with *batch* operating systems
Hierarchical file system
Device-independent I/O

- Blocking and non-blocking calls

Shell-based, tty user interface – original goal was text processing
Filter-based, record-less processing paradigm
Portable (written in C)

Lots of features but overall goal of ***simplicity***

Unix Philosophy (Doug McIlroy)

Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features".
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.
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.
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.

Unix Philosophy rev2 (Peter H. Salus)

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

Unix Philosophy Example 1: Shell

- Shell is just a program!
  - Can change shells if needed.
- Invoke programs: “cmd arg1 ... argn”
  - Only lightweight processing of args to program
- Tools for chaining programs:
  - Stdio and I/O redirection
  - Filters & pipes
- Multi-tasking from a single shell (& symbol)

Unix Philosophy Example 2: File System

“Important job of Unix is to provide a file system”

Everything is a file:

- Ordinary files: sequence of bytes (unstructured)
- Directories (protected ordinary files)
- Special files (I/O, system operations

Uniform I/O, naming, and protection model

- RWX for everything

Over to Edward and Firn

Next topic: Concurrency

Unix decided to not worry when it comes to files but not always possible

```
balance = getAccountBalance(number=12345)
if (balance > 100):
  withdraw(amount=100, number=12345)
```

What can go wrong here?

Dealing with concurrency

No pre-emption

Transactions (e.g., hardware-support)

Shared memory

- Semaphores – global counters
- Monitors – collocate data and code

Message passing

Mesa

Not a super-easy read

- Brush up on basics before

Example: https://teaching.csse.uwa.edu.au/units/CITS2230/handouts/Lecture09/lecture9.pdf