Classic Systems:
Unix and THE

Presented by Hakim Weatherspoon

M. Satyanarayanan

- Systems faculty at Carnegie-Mellon University
- Andrew Project
  - Distributed computing environment begun in 1983
  - IT joint venture between CMU and IBM
  - Focused on workstations: client-server

- Lead Andrew File System
- Inspired CODA and another 20 years of research

The UNIX Time-Sharing System

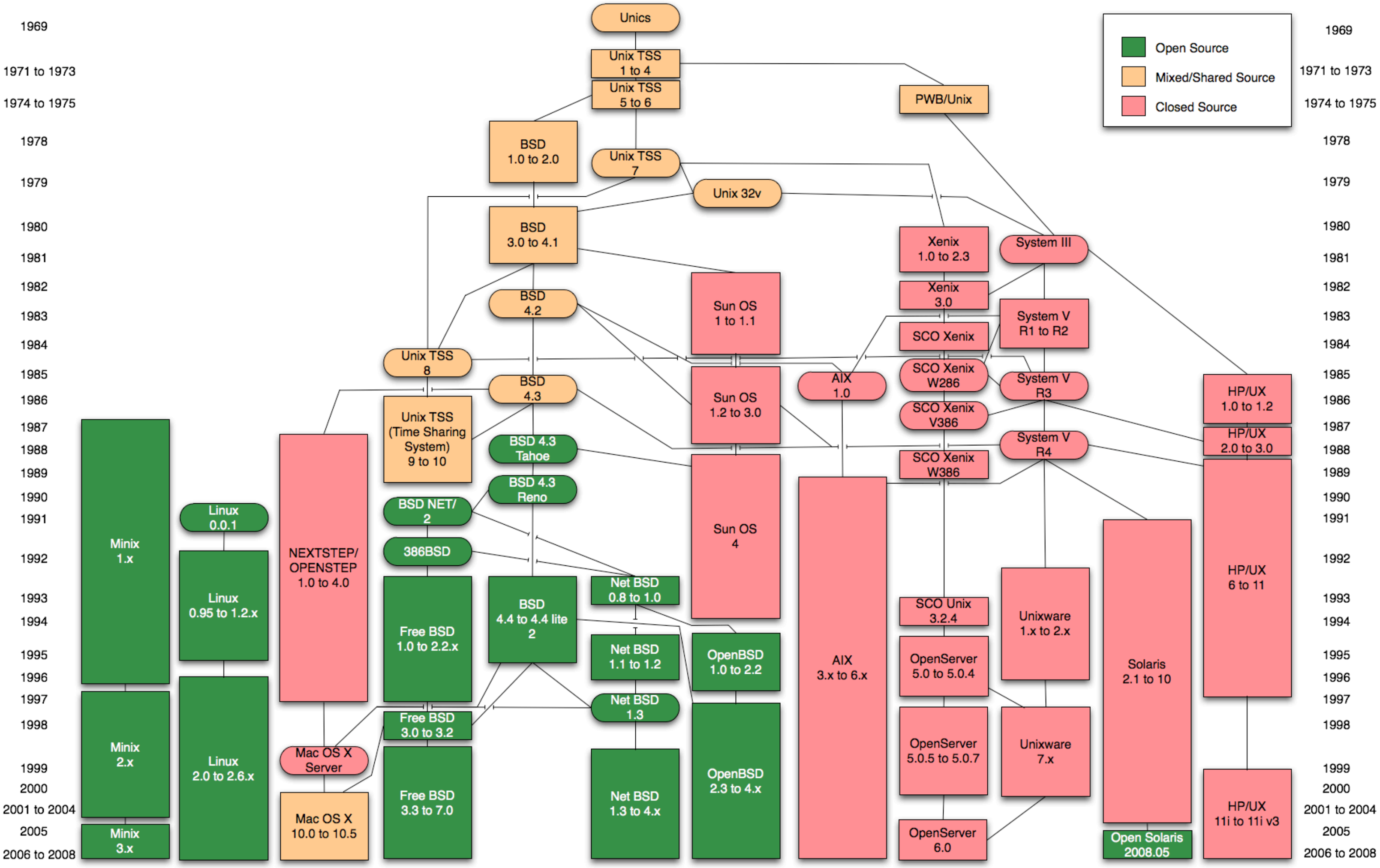
Dennis Ritchie and Ken Thompson

- Background of authors at Bell Labs
  - Both won Turing Awards in 1983

- Dennis Ritchie
  - Key developer of *The C Programming Lanuage*, Unix, and Multics
- Ken Thompson
  - Key developer of the B programming lanuage, Unix, Multics, and Plan 9
  - Also QED, ed, UTF-8

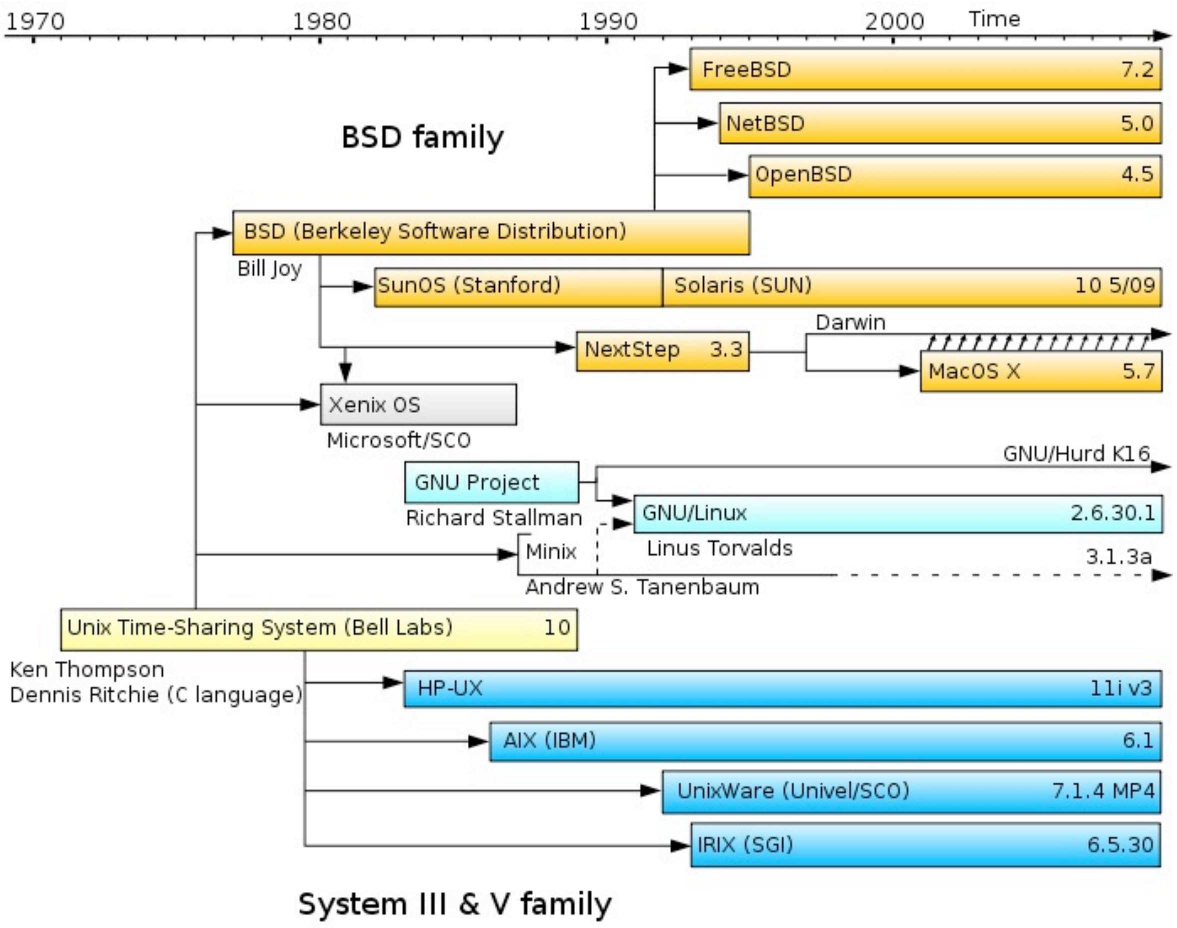
The UNIX Time-Sharing System

Dennis Ritchie and Ken Thompson



The UNIX Time-Sharing System

Dennis Ritchie and Ken Thompson



The UNIX Time-Sharing System

Dennis Ritchie and Ken Thompson

- Classic system and paper
  - described almost entirely in 10 pages

- Key idea
  - elegant combination of a few concepts that fit together well

System features

- Time-sharing system
- Hierarchical file system
- Device-independent I/O
- Shell-based, tty user interface
- Filter-based, record-less processing paradigm

Version 3 Unix

- 1969: Version 1 ran PDP-7
- 1971: Version 3 Ran on PDP-11's
  - Costing as little as $40k!
- < 50 KB
- 2 man-years to write
- Written in C

File System

- Ordinary files (uninterpreted)
- Directories (protected ordinary files)
- Special files (I/O)

Directories

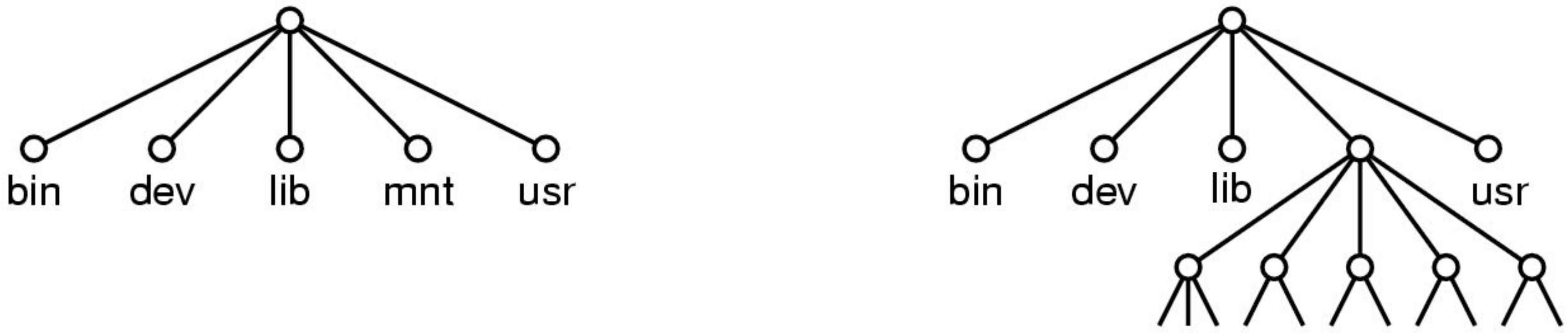
- root directory
- path names
- rooted tree
- current working directory
- back link to parent
- multiple links to ordinary files

Special Files

- Uniform I/O model
  - Each device associated with at least one file
  - But read or write of file results in activation of device
- Advantage: Uniform naming and protection model
  - File and device I/O are as similar as possible
  - File and device names have the same syntax and meaning, can pass as arguments to programs
  - Same protection mechanism as regular files

Removable File System

- Tree-structured
- *Mount*'ed on an ordinary file
  - – Mount replaces a leaf of the hierarchy tree (the ordinary file) by a whole new subtree (the hierarchy stored on the removable volume)
  - – After mount, virtually no distinction between files on permanent media or removable media



(a)

(b)

Protection

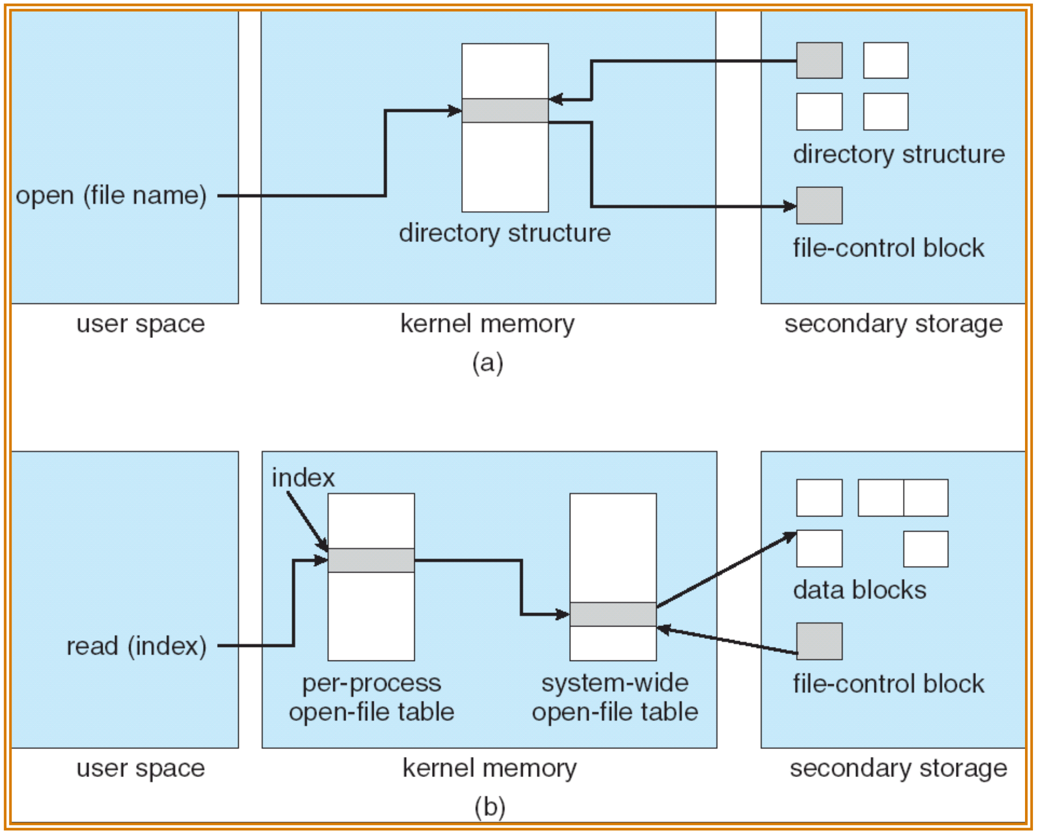
- User-world, RWX bits
- set-user-id bit
- super user is just special user id

Uniform I/O Model

- open, close, read, write, seek
  - Uniform calls eliminates differences between devices
- other system calls
  - close, status, chmod, mkdir, ln
- bytes, no records

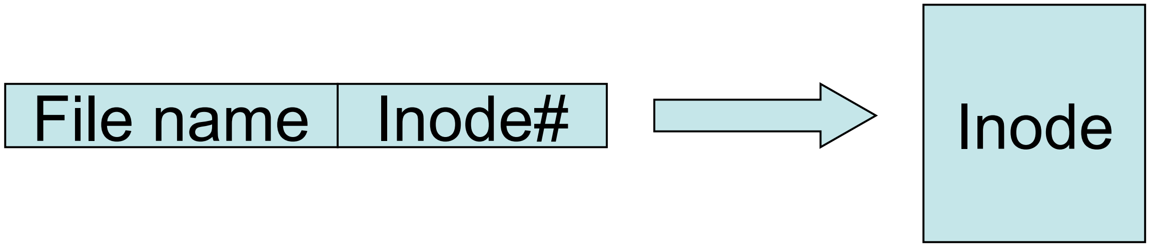
File System Implementation

- table of i-nodes
- path name scanning
- mount table
- buffered data
- write-behind



I-node Table

- short, unique name that points at file info.
- allows simple & efficient fsck
- cannot handle accounting issues



Processes and images

- text, data & stack segments
- process swapping
- pid = fork()
- pipes
- exec(file, arg1, ..., argn)
- pid = wait()
- exit(status)

The Shell

- cmd arg1 ... argn
- stdio & I/O redirection
- filters & pipes
- multi-tasking from a single shell
- shell is just a program

- Trivial to implement in shell
  - Redirection, background processes, cmd files, etc

Traps

- Hardware interrupts
- Software signals
- Trap to system routine

Perspective

- Not designed to meet predefined objective
- Goal: create a comfortable environment to explore machine and operating system
- Other goals
  - Programmer convenience
  - Elegance of design
  - Self-maintaining

“THE”-Multiprogramming System
Edsger W. Dijkstra

- Received Turing Award in 1972
- Contributions
  - Shortest Path Algorithm, Reverse Polish Notation, Bankers algorithm, semaphore’s, self-stabilization
- Known for disliking ‘goto’ statements and using computers!

“THE”-Multiprogramming System
Edsger W. Dijkstra

- Never named “THE” system; instead, abbreviation for "Technische Hogeschool Eindhoven”

- Batch system (no human intervention) that supported multitasking (processes share CPU)
  - THE was *not* multiuser

- Introduced
  - software-based memory segmentation
  - Cooperating sequential processes
  - semaphores

Design

- Layered structure
  - Later Multics has layered structure, ring segmentation
- Layer 0 – the scheduler
  - Allocated CPU to processes, accounted for blocked proc’s
- Layer 1 – the pager
- Layer 2 – communication between OS and console
- Layer 3 – managed I/O
- Layer 4 – user programs
- Layer 5 – the user

Perspective

- Layered approach
  - Design small, well defined layers
  - Higher layers dependent on lower ones
    - Helps prove correctness
    - Helps with debugging

- Sequential process and Semaphores

Next Time

- Read and write review:
- Do Lab 1 due yesterday
- Project Proposal due this Thursday
  - Email and talk to me before Thursday
- Check website for updated schedule