# UNIX Time Sharing System

Original paper by Dennis M. Ritchie and Ken Thompson

Discussion led by Edward and Firn

## Paper outline

1. Introduction
2. Hardware and Software Environment
3. The File System
4. Implementation of the File System
5. Processes and Images
6. The Shell
7. Traps
8. Perspective
9. Statistics

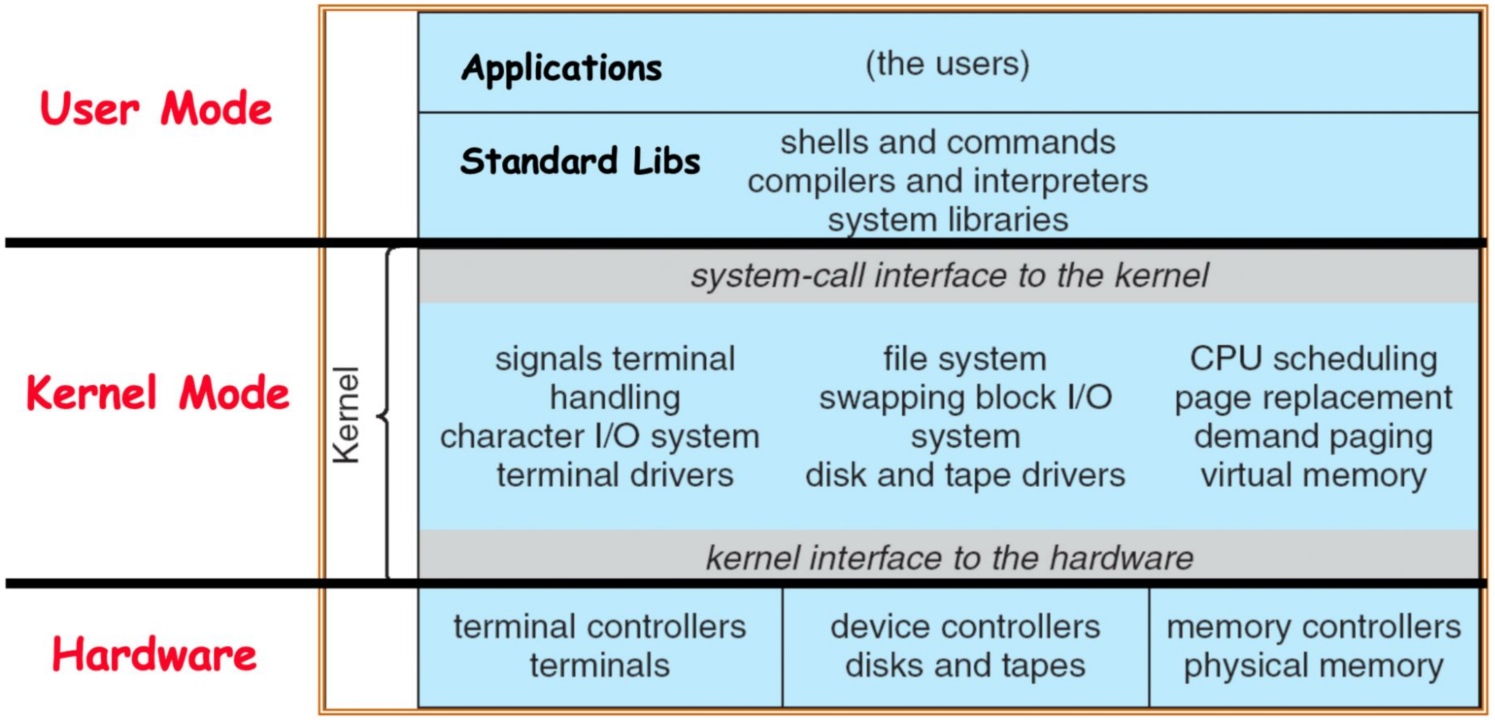
## 1. Introduction

- They’ve been iterating on Unix
- Unix is relatively cheap
- The system is self-supporting
- The system is used for both computer systems research and document preparation

## 2. Hardware and Software Environment

- Supports many devices (all now defunct)
- OS occupies a third of the core mem of their system
- Written in C
    - easy to understand, modify, and maintain

# UNIX System Structure



Credit slide from UC Berkeley EECS 262a: Advanced Topics in Computer Systems

## 3. The File System

- Ordinary files
- Directories
  - “/” in /alpha/beta
  - linking
- Special files (/dev)
- Mounting
- File access control through “protection bits”
- I/O
  - filep = open(name, flag) (flag: read/write/update)
  - location = seek(filep, base, offset)

## 4. Implementation of the File System

- I-nodes (good to be aware of what info these store!)
    - Owner
    - Protection bits
    - Address
    - Size
    - Last modification
    - # of links
    - Is this a directory?
    - Is this a special file?
    - large/small?
- Implementation is mostly straightforward
- User sees I/O as synchronous and unbuffered, but it isn’t

## 5. Processes and Images

- Image – “the current state of a pseudo computer”
- Process – “the execution of an image”
- Program layout (text, data, stack)
- Process primitives
    - processid = fork(label)
    - filep = pipe()
    - execute(file, arg1, arg2, ..., argn)
    - processid = wait()
    - exit(status)

## 6. The Shell

- Interesting metaphor
- command arg1 arg2 … argn
- No concept of a system PATH, but /bin/ is looked up
- Shell syntax, multitasking with background commands, scripting
- Implementation
- The `init` program
- Using other programs as the Shell

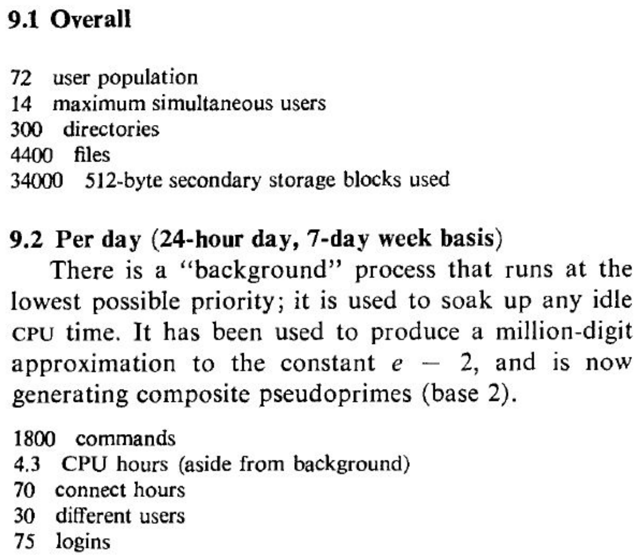
## 7. Traps

- (error handling control flow)
- Default is to dump the process image to a file named “core” in the current directory, other behaviors can be specified

## 8. Perspective

1. Designed for programmers by programmers
2. Hardware constraints led to “elegant” designs
3. The system’s designers were forced to use the system

## 9. Statistics



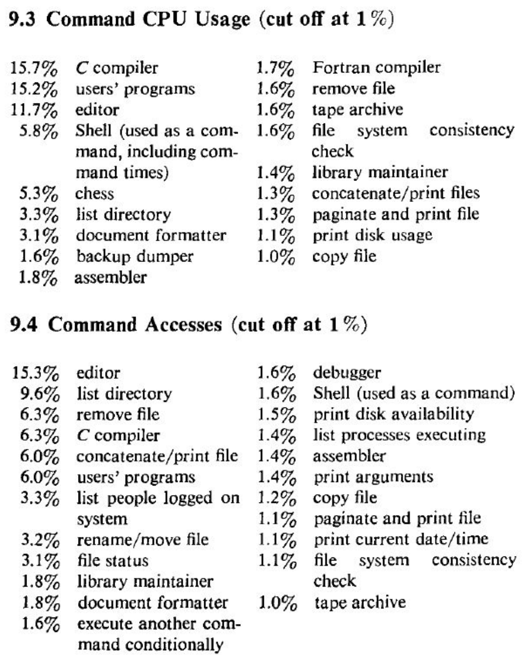
### 9.1 Overall

| | |
|---|---|
| 72 | user population |
| 14 | maximum simultaneous users |
| 300 | directories |
| 4400 | files |
| 34000 | 512-byte secondary storage blocks used |

### 9.2 Per day (24-hour day, 7-day week basis)

There is a "background" process that runs at the lowest possible priority; it is used to soak up any idle CPU time. It has been used to produce a million-digit approximation to the constant *e* − 2, and is now generating composite pseudoprimes (base 2).

| | |
|---|---|
| 1800 | commands |
| 4.3 | CPU hours (aside from background) |
| 70 | connect hours |
| 30 | different users |
| 75 | logins |

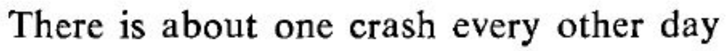


### 9.3 Command CPU Usage (cut off at 1%)

| | | | |
|---|---|---|---|
| 15.7% | *C* compiler | 1.7% | Fortran compiler |
| 15.2% | users' programs | 1.6% | remove file |
| 11.7% | editor | 1.6% | tape archive |
| 5.8% | Shell (used as a command, including command times) | 1.6% | file system consistency check |
| | | 1.4% | library maintainer |
| 5.3% | chess | 1.3% | concatenate/print files |
| 3.3% | list directory | 1.3% | paginate and print file |
| 3.1% | document formatter | 1.1% | print disk usage |
| 1.6% | backup dumper | 1.0% | copy file |
| 1.8% | assembler | | |

### 9.4 Command Accesses (cut off at 1%)

| | | | |
|---|---|---|---|
| 15.3% | editor | 1.6% | debugger |
| 9.6% | list directory | 1.6% | Shell (used as a command) |
| 6.3% | remove file | 1.5% | print disk availability |
| 6.3% | *C* compiler | 1.4% | list processes executing |
| 6.0% | concatenate/print file | 1.4% | assembler |
| 6.0% | users' programs | 1.4% | print arguments |
| 3.3% | list people logged on system | 1.2% | copy file |
| | | 1.1% | paginate and print file |
| 3.2% | rename/move file | 1.1% | print current date/time |
| 3.1% | file status | 1.1% | file system consistency check |
| 1.8% | library maintainer | | |
| 1.8% | document formatter | 1.0% | tape archive |
| 1.6% | execute another command conditionally | | |



There is about one crash every other day

# Online discussion summary

- Q1: "it's basically unchanged, but..."
- (what do each of these require from the OS? If they don’t require anything right now, how could their presence affect the design of the OS?)
    - advances in memory
    - specialized hardware like FPGAs and TPUs
    - graphical user interfaces/OSs
    - TCP/IP, networking, inter-process communication
    - journaling/fault-tolerant file systems
    - differences in I/O behavior
    - additional file access concepts (groups)
    - soft links
    - containers
    - system-like applications (browsers)

## Group Discussion (bit.ly/550-unix-discussion)

- What was the goal of this paper?
- Are you convinced that this is a good system?
    - Why or why not?
- What mistakes or flaws of the system that the authors make?
    - How would you improve the Unix system?