Operating Systems:
History and Three Easy Pieces

COMP400727: Introduction to Computer Systems

Hao Li
Xi’an Jiaotong University

Today

- **History of Operating Systems**
- Three Easy Pieces

Earliest days: One batch job at a time

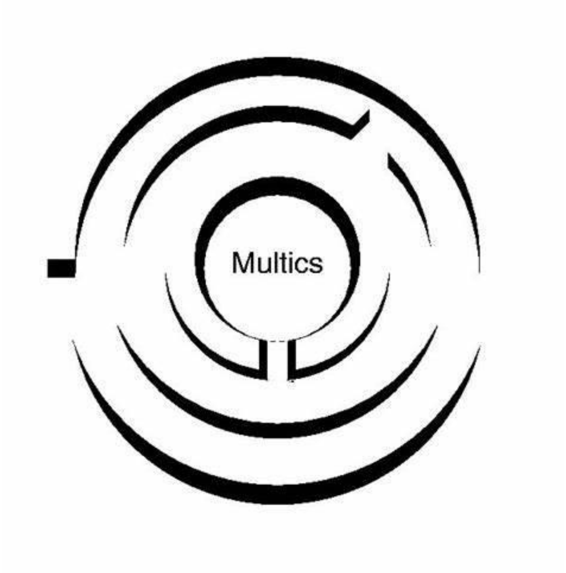
IBM 704 at Langley Research Center (NASA), 1957
https://commons.wikimedia.org/w/index.php?curid=6455009

1960s: Operating System for Multitasking

at&t

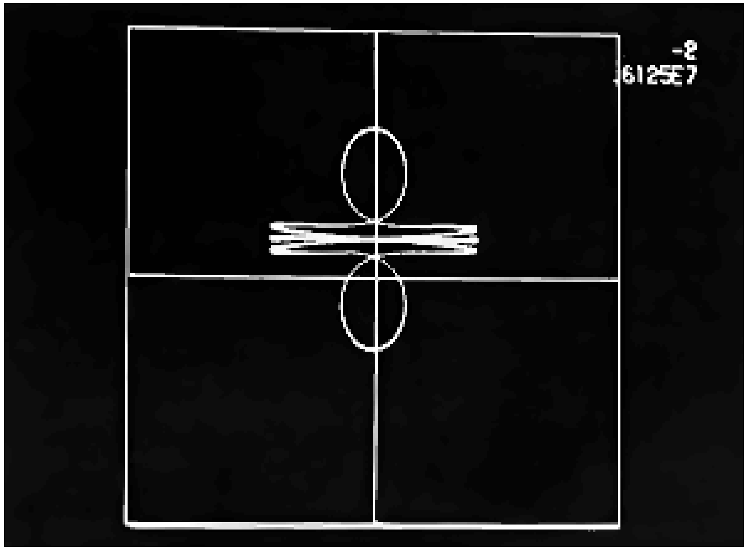
Multics

1960s: MULTICS Fails



1969: BELL Labs in AT&T

Ken Thompson
(1943-)



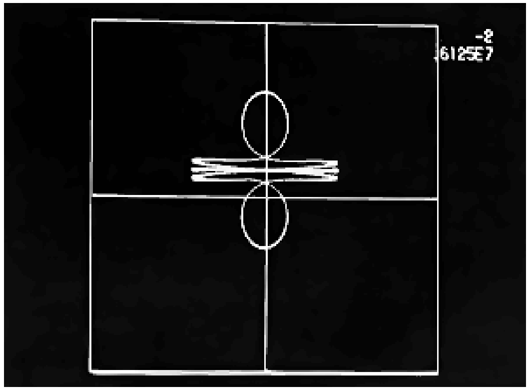
Space Travel Game

1969: BELL Labs in AT&T

Digital Equipment Corporation PDP-7

1969: How to Play Space Travel in PDP-7?

Applications



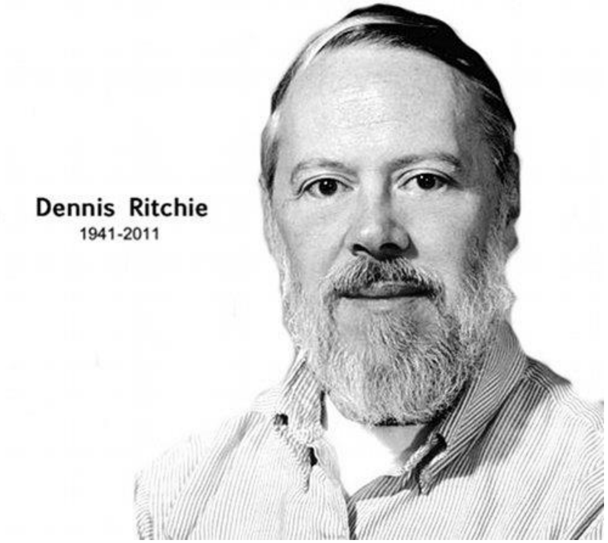
Operating System

Hardware

with Assembly

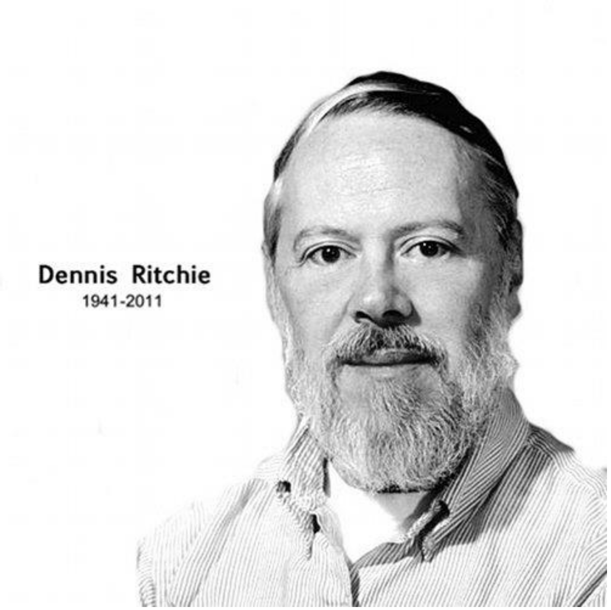
in ONE month!

1969: BELL Labs in AT&T



Dennis Ritchie
(1941-2011)

1969: BELL Labs in AT&T

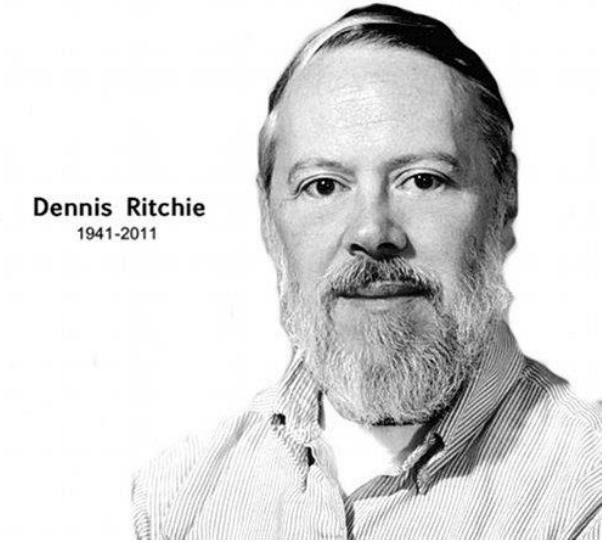


“Your OS is much worse than MULTICS”

“Call it UNICS”

1971: Re-Architect UNICS

- B Language: extended from BCPL
- A high-level programming language



Dennis Ritchie
1941-2011

- C Language: extended from B
- Decouple PL from hardware
- You are still using it!

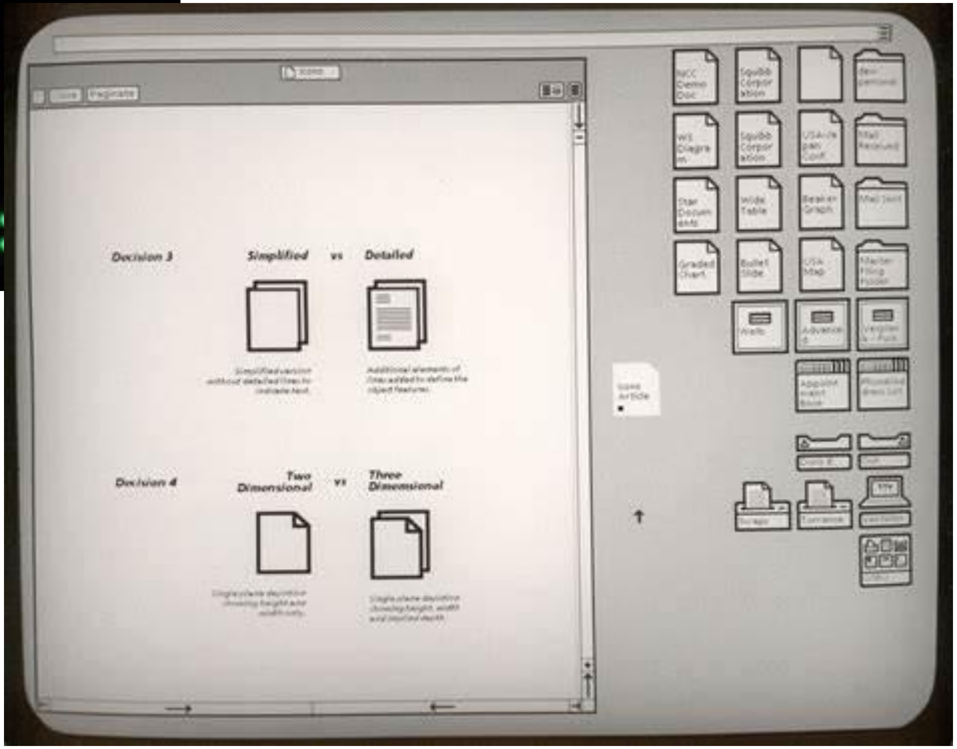
UNICS → UNIX

1973: Xerox PARC

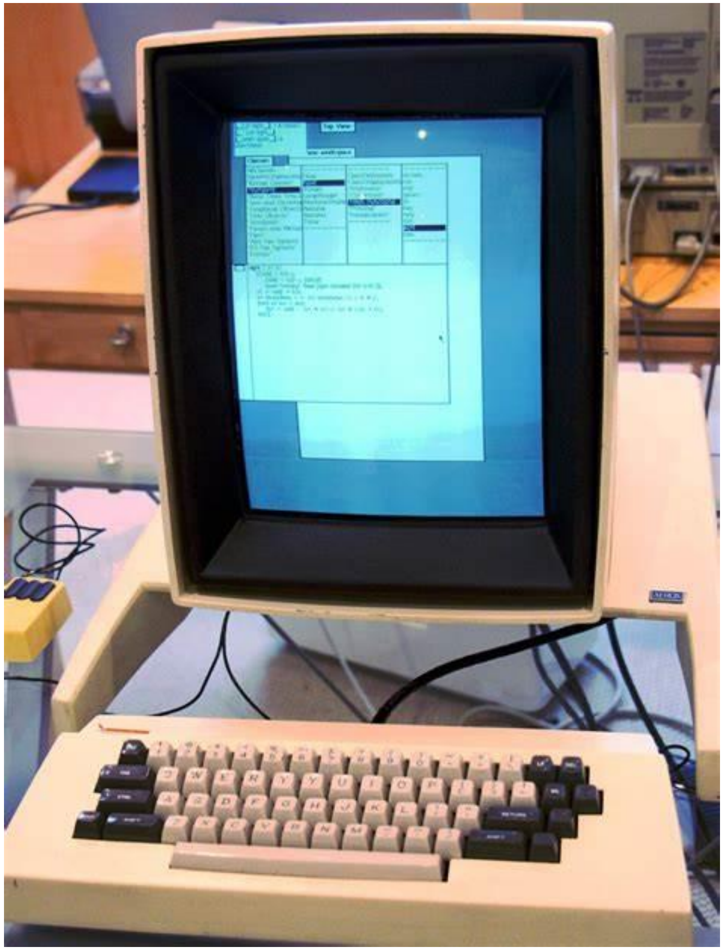
1973: Xerox PARC

```
************************************************
*                                              *
*               AT&T UNIX(TM) pc               *
*                                              *
*             Loader Version 3.51              *
*                                              *
*          Copyright (C) 1985, 1986            *
*          AT&T                                *
*          All Rights Reserved                 *
*                                              *
************************************************
Searching floppy disk...

Searching hard disk...
****************************************************
****************************************************
**********
```



1979: Xerox PARC Alto



Steve Jobs (1955-2011)

1983: Apple Lisa

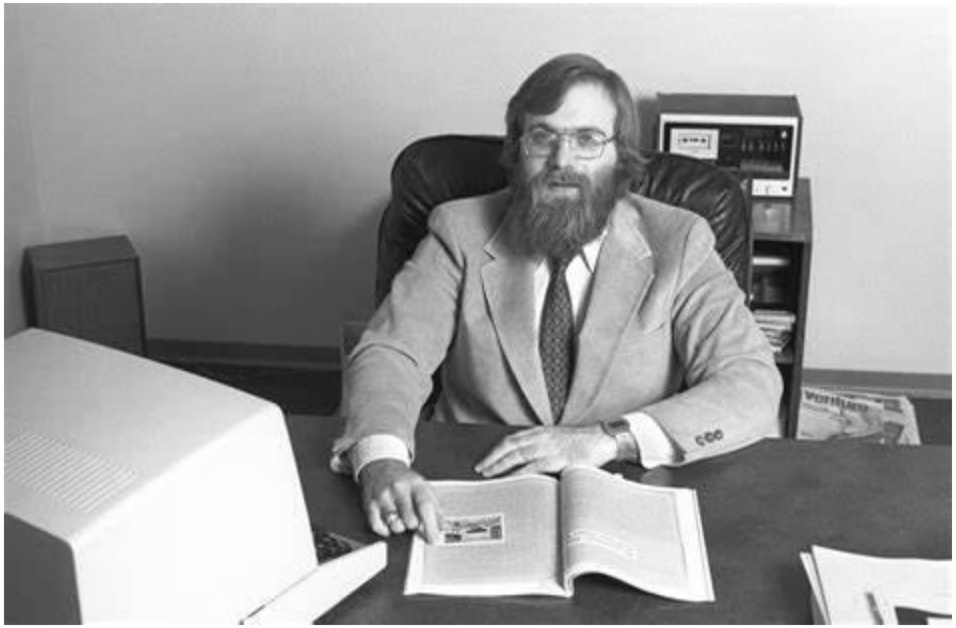
1980: A Quick and Dirty Operating System

Tim Paterson
(1956-)

- **Develop an OS with 4 months**
- **For 16bit Intel 8086 CPU**
- **QDOS**
  - Quick and Dirty Operating System

1980: A Quick and Dirty Operating System

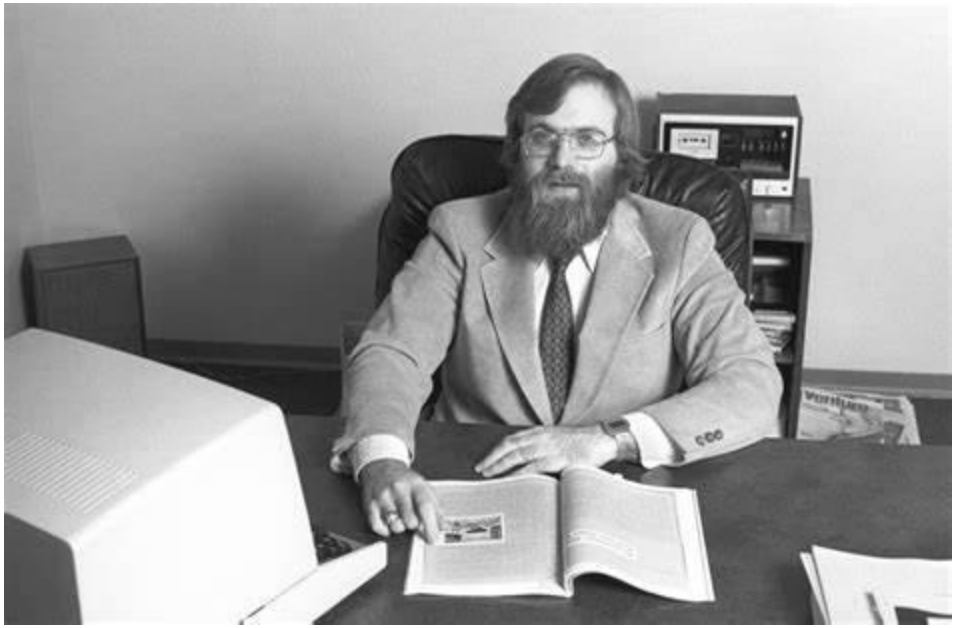
Tim Paterson
(1956-)



Paul Allen
(1953-2018)

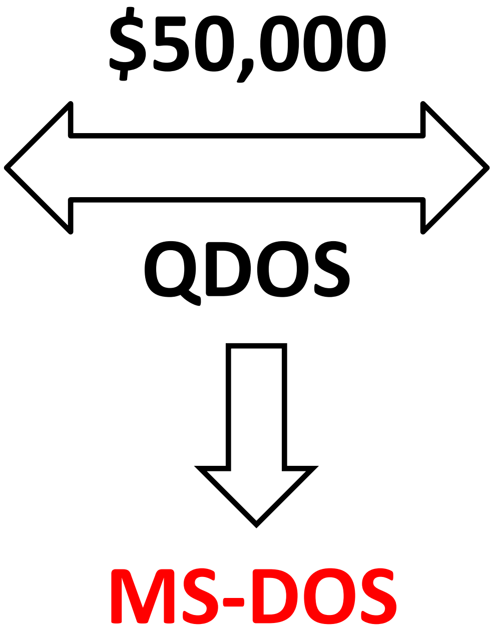
1981: Microsoft

Bill Gates
(1955-)



Paul Allen
(1953-2018)

1981: MS-DOS

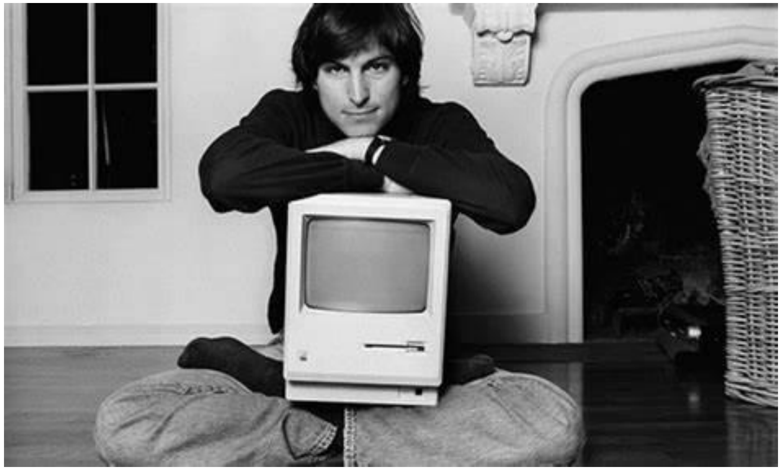


$50,000

QDOS

MS-DOS

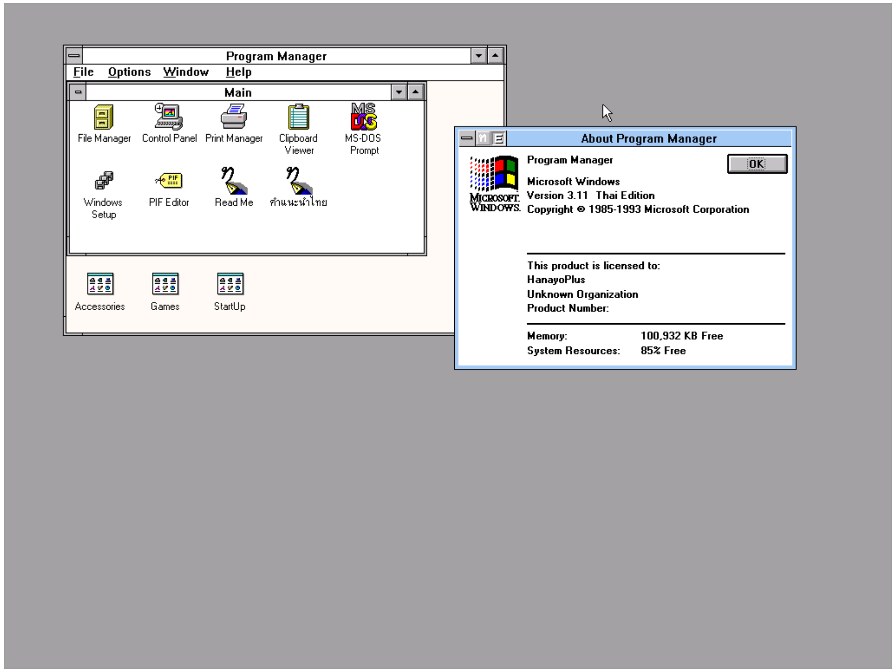
1990: Should OS and Hardware Decouple?



MS-DOS:
IBM, Intel, Apple, …

Apple:
One OS for One Machine

1990: Windows



Program Manager

Microsoft Windows
Version 3.11 Thai Edition
Copyright © 1985-1993 Microsoft Corporation

This product is licensed to:
HanayoPlus
Unknown Organization
Product Number:

Memory: 100,932 KB Free
System Resources: 85% Free

Windows 3.0

1990: Two Thieves

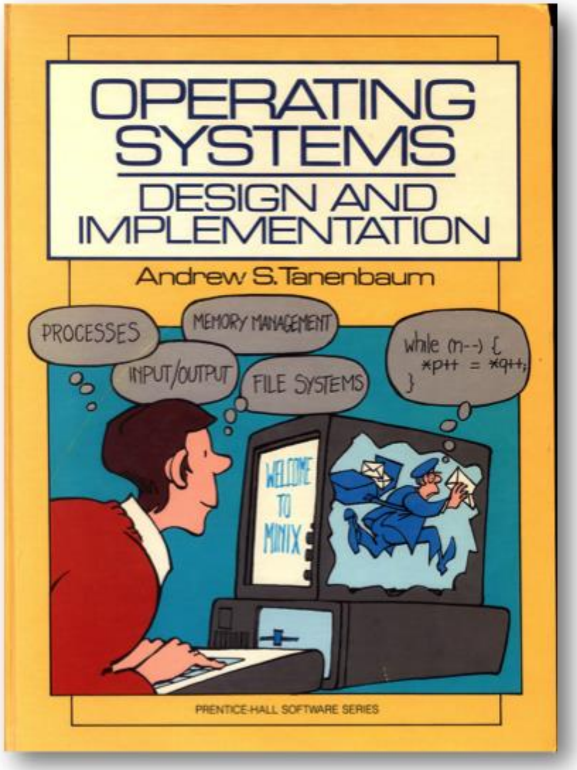
You Are Stealing
Our Operating System!

Well, Steve, I think there's more than one way of looking at it. I think it's more like we both had this rich neighbor named Xerox and I broke into his house to steal the TV set and found out that you had already stolen it.

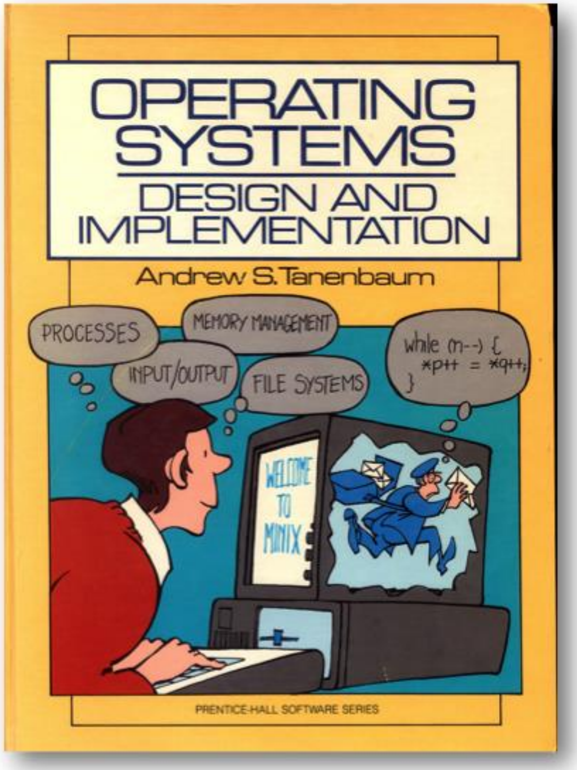
1991: A New and Open OS

Andrew Tanenbaum
(1944-)

- 1987: Andy Tanenbaum
  - Includes source code for Minix (“toy” Unix)

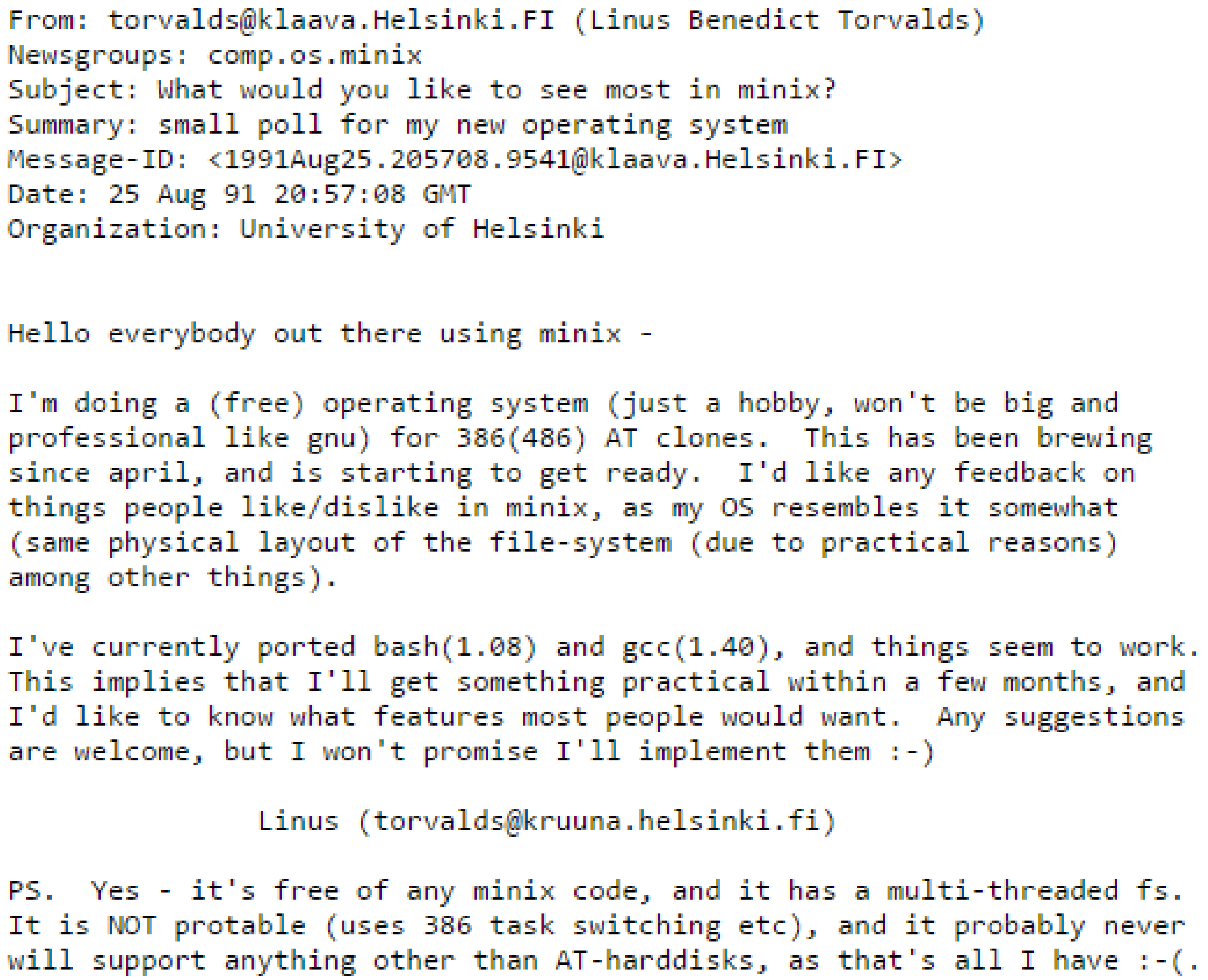


1991: A New and Open OS



Linus Torvalds
(1969-)

1991: The First Email



```
From: torvalds@klaava.Helsinki.FI (Linus Benedict Torvalds)
Newsgroups: comp.os.minix
Subject: What would you like to see most in minix?
Summary: small poll for my new operating system
Message-ID: <1991Aug25.205708.9541@klaava.Helsinki.FI>
Date: 25 Aug 91 20:57:08 GMT
Organization: University of Helsinki


Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and
professional like gnu) for 386(486) AT clones.  This has been brewing
since april, and is starting to get ready.  I'd like any feedback on
things people like/dislike in minix, as my OS resembles it somewhat
(same physical layout of the file-system (due to practical reasons)
among other things).

I've currently ported bash(1.08) and gcc(1.40), and things seem to work.
This implies that I'll get something practical within a few months, and
I'd like to know what features most people would want.  Any suggestions
are welcome, but I won't promise I'll implement them :-)

               Linus (torvalds@kruuna.helsinki.fi)

PS.  Yes - it's free of any minix code, and it has a multi-threaded fs.
It is NOT protable (uses 386 task switching etc), and it probably never
will support anything other than AT-harddisks, as that's all I have :-(.
```

1991: Free Software

Richard Stallman
(1953-)

- GPL
  - Copy-Left
- GNU
  - GNU is Not Unix
  - Emacs, gcc, gdb, …

1991: Linux and Open Source Project

- Applications

- Operating Systems

Nowadays


Windows 1
1985
Windows 3.1
1992
Windows 95
1995
Windows XP
2001
Windows Vista
2006
Windows 7
2009
Windows 8
2012
Windows 10
2015


iOS
iPadOS
AR
macOS
watchOS
A
文
tvOS


archlinux
Fuduntu
Lubuntu
Mandriva
openSUSE
xubuntu
archbang
CentOS
kubuntu
core
ubuntu
ZORIN
KNOPPIX
mageia
Linux Mint
PC-BSD
sabayon
Scientific
debian
fedora
PCLinuxOS
Puppy
UE
Ultimate
Edition
Vector
bodhi
CHAKRA
FreeBSD
gentoo linux
PinguyOS
slackware

Today

- History of Operating Systems
- Three Easy Pieces

What a happens when a program runs?

- **A running program executes instructions.**
  1. The processor **fetches** an instruction from memory.
  2. **Decode**: Figure out which instruction this is
  3. **Execute**: i.e., add two numbers, access memory, check a condition, jump to function, and so forth.
  4. The processor moves on to the **next instruction** and so on.

Operating System (OS)

- **Responsible for**
  - Making it easy to **run** programs
  - Allowing programs to **share** memory
  - Enabling programs to **interact** with devices

OS is in charge of making sure the system operates correctly and efficiently.

1st Piece: Virtualization

- **The OS takes a physical resource and transforms it into a virtual form of itself.**
    - **Physical resource**: Processor, Memory, Disk …
  - The virtual form is more general, powerful and easy-to-use.
  - Sometimes, we refer to the OS as a **virtual machine**.

System call

- **System call allows user to tell the OS what to do.**
  - The OS provides some interface (APIs, standard library).
  - A typical OS exports a few hundred system calls.
    - Run programs
    - Access memory
    - Access devices

The OS is a resource manager.

- **The OS manage resources such as *CPU*, *memory* and *disk*.**
- **The OS allows**
  - Many programs to run → Sharing the CPU
  - Many programs to *concurrently* access their own instructions and data → Sharing memory
  - Many programs to access devices → Sharing disks

Virtualizing the CPU

- **The system has a very large number of virtual CPUs.**
  - Turning a single CPU into a seemingly infinite number of CPUs.
  - Allowing many programs to seemingly run at once
    → **Virtualizing the CPU**

Virtualizing the CPU (Cont.)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <assert.h>
#include "common.h"

int
main(int argc, char *argv[])
{
        if (argc != 2) {
                fprintf(stderr, "usage: cpu <string>\n");
                exit(1);
        }
        char *str = argv[1];
        while (1) {
                Spin(1); // Repeatedly checks the time and
                         returns once it has run for a second
                printf("%s\n", str);
        }
        return 0;
}
```

Simple Example(`cpu.c`): Code That Loops and Prints

Virtualizing the CPU (Cont.)

- **Execution result 1.**

```
prompt> gcc -o cpu cpu.c -Wall
prompt> ./cpu "A"
A
A
A
ˆC
prompt>
```

Run forever; Only by pressing “Control-c” can we halt the program

Virtualizing the CPU (Cont.)

- **Execution result 2.**

```
prompt> ./cpu A &  ./cpu B &  ./cpu C &  ./cpu D &
[1] 7353
[2] 7354
[3] 7355
[4] 7356
A
B
D
C
A
B
D
C
A
C
B
D
...
```

Even though we have only one processor, all four of programs seem to be running at the same time!

Virtualizing Memory

- **The physical memory is *an array of bytes*.**
- **A program keeps all of its data structures in memory.**
  - **Read memory** (load):
    - Specify an address to be able to access the data
  - **Write memory** (store):
    - Specify the data to be written to the given address

Virtualizing Memory (Cont.)

- A program that Accesses Memory (`mem.c`)

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int
main(int argc, char *argv[])
{
        int *p = malloc(sizeof(int));  // a1: allocate some
                                          memory
        assert(p != NULL);
        printf("(%d) address of p: %08x\n",
                getpid(), (unsigned) p);  // a2: print out the
                                          address of the memmory
        *p = 0;  // a3: put zero into the first slot of the memory
        while (1) {
                Spin(1);
                *p = *p + 1;
                printf("(%d) p: %d\n", getpid(), *p); // a4
        }
        return 0;
}
```

Virtualizing Memory (Cont.)

- **The output of the program `mem.c`**

```
prompt> ./mem
(2134) memory address of p: 00200000
(2134) p: 1
(2134) p: 2
(2134) p: 3
(2134) p: 4
(2134) p: 5
^C
```

  - The newly allocated memory is at address `00200000`.
  - It updates the value and prints out the result.

Virtualizing Memory (Cont.)

- **Running `mem.c` multiple times**

```
prompt> ./mem &; ./mem &
[1] 24113
[2] 24114
(24113) memory address of p: 00200000
(24114) memory address of p: 00200000
(24113) p: 1
(24114) p: 1
(24114) p: 2
(24113) p: 2
(24113) p: 3
(24114) p: 3
...
```

  - It is as if each running program has its **own private memory**.
    - Each running program has allocated memory at the same address.
    - Each seems to be updating the value at `00200000` independently.

Virtualizing Memory (Cont.)

- **Each process accesses its own private virtual address space.**
  - The OS maps address space onto the physical memory.
  - A memory reference within one running program does not affect the address space of other processes.
  - Physical memory is a shared resource, managed by the OS.

2nd Piece: Concurrency

- The OS is juggling many things at once, first running one process, then another, and so forth.

- Modern multi-threaded programs also exhibit the concurrency problem.

Concurrency Example

- **A Multi-threaded Program (`thread.c`)**

```c
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

volatile int counter = 0;
int loops;

void *worker(void *arg) {
        int i;
        for (i = 0; i < loops; i++) {
                counter++;
        }
        return NULL;
}

int
main(int argc, char *argv[])
{
        if (argc != 2) {
                fprintf(stderr, "usage: threads <value>\n");
                exit(1);
        }
```

Concurrency Example (Cont.)

```
        loops = atoi(argv[1]);
        pthread_t p1, p2;
        printf("Initial value : %d\n", counter);

        Pthread_create(&p1, NULL, worker, NULL);
        Pthread_create(&p2, NULL, worker, NULL);
        Pthread_join(p1, NULL);
        Pthread_join(p2, NULL);
        printf("Final value : %d\n", counter);
        return 0;
}
```

- The main program creates **two threads**.
    - Thread: a function running within the same memory space. Each thread start running in a routine called `worker()`.
    - `worker()`: increments a counter

Concurrency Example (Cont.)

- `loops` determines how many times each of the two workers will increment the shared counter in a loop.
  - `loops`: 1000.

```
prompt> gcc -o thread thread.c -Wall -pthread
prompt> ./thread 1000
Initial value : 0
Final value : 2000
```

  - `loops`: 100000.

```
prompt> ./thread 100000
Initial value : 0
Final value : 143012 // huh??
prompt> ./thread 100000
Initial value : 0
Final value : 137298 // what the??
```

Why is this happening?

- **Increment a shared counter → take three instructions.**
  1. Load the value of the counter from memory into register.
  2. Increment it
  3. Store it back into memory

- **These three instructions do not execute atomically. → Problem of concurrency happen.**

3rd Piece: Persistence

- **Devices such as DRAM store values in a volatile.**
- ***Hardware* and *software* are needed to store data persistently.**
  - **Hardware**: I/O device such as a hard drive, solid-state drives(SSDs)
  - **Software**:
    - File system manages the disk.
    - File system is responsible for storing any files the user creates.

Persistence (Cont.)

- Create a file (`/tmp/file`) that contains the string “hello world”

```
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <fcntl.h>
#include <sys/types.h>

int
main(int argc, char *argv[])
{
        int fd = open("/tmp/file", O_WRONLY | O_CREAT
                      | O_TRUNC, S_IRWXU);
        assert(fd > -1);
        int rc = write(fd, "hello world\n", 13);
        assert(rc == 13);
        close(fd);
        return 0;
}
```

`open()`, `write()`, and `close()` system calls are routed to the part of OS called the file system, which handles the requests

Persistence (Cont.)

- **What OS does in order to write to disk?**
  - Figure out **where** on disk this new data will reside
  - **Issue I/O** requests to the underlying storage device

- **File system handles system crashes during write.**
  - **Journaling** or **copy-on-write**
  - Carefully <u>ordering</u> writes to disk

Design Goals

- **Build up abstraction**
  - Make the system convenient and easy to use.
- **Provide high performance**
  - Minimize the overhead of the OS.
  - OS must strive to provide virtualization <u>without excessive overhead</u>.
- **Protection between applications**
  - <u>Isolation</u>: Bad behavior of one does not harm other and the OS itself.

Design Goals (Cont.)

- **High degree of reliability**
    - The OS must also run non-stop.

- **Other issues**
    - Energy-efficiency
    - Security
    - Mobility