The UNIX Time-Sharing System

CS380L: Emmett Witchel
thanks to Mike Dahlin

People who log in as a player of one of the games find themselves limited to the game and unable to investigate the presumably more interesting offereings of UNIX as a whole.

5.3% chess [CPU usage of chess (5th most popular program in terms of cycles executed), just behind the shell (5.8%), and in front of list directory (3.3%)].

–D. M. Ritchie and K. Thompson. "The UNIX Time-Sharing System." p. 372. *Communications of the ACM*, 17(7), July 1974, pp. 365–375.

1 Preliminaries

1.1 Outline

- Unix
  - Theme: Simplicity and elegance
  - File I/O
  - Process management
  - Lessons

2 Theme: Simplicity and elegance

- The "Third system"
  - THE: Among the first systems
    - Worked, rigid heirarchies
    - Very simple, lacked features & usability
  - Multics: The "second system effect"
    - Visionary
    - Ungainly
  - Unix
    - "Elegance, taste, craftsmanship"
    - Minimum functionality and implementation, yet...
    - Power, and...
    - The pieces fit together seamlessly
    - How many times can one say: "this is just totally obvious"
    - This paper nearly summarizes undergrad OS...if you *understand* it, you understand most of the basics of modern OS's

Feature	Unix 1973	Linux 2005	Linux 2016
Min cost system	$40,000	$100	$5 (Raspberry Pi Zero)
Applications	C compiler assembler debugger `YACC` form letter generator (?!)	C compiler assembler debugger `YACC`	C compiler assembler debugger `YACC`
Memory	144KB	2GB	32GB
Memory (min)	50KB	500KB (4MB embedded systems, e.g., ttylinux)	∼1MB for microYocto 32-bit, 1.3MB for 64-bit, IoT
Disk	1MB swap, 2.5MB, 40MB	500GB, swap on partition or files	4TB disk or 128GB SSD, swap on partition or files
File names	Up to 14 characters	Up to 255 characters	Up to 255 characters
mkdir	setuid program	user program (`mkdir` syscall)	user program (`mkdir` syscall)
File creation	`create` syscall	`O_CREAT` flag to `open` syscall	`O_CREAT` flag to `open` syscall
File block size	512B	4KB	4KB
Max file size	1MB	4TB (NTFS is 2TB)	16TB (ext4)
Static lines of code	10K	4.2M	20.6M (4.3 2015-11-01) [12M (Sloccount on 3.16.1)]
Intellectual property	AT&T propietary	Open source	Open source
Person-years	2	4,528 `http://www.dwheeler.com/-essays/-linux-kernel-cost.html`	8,000 ($1 Trillion)

3 Unix 1974 vs. Linux 2005

Look carefully students, this is what systems success beyond your wildest dreams looks like.

4 File I/O

The most important role of UNIX is to provide a file system (p. 366)

- Hierarchical name space
  - *strict* hierarchy across directories
  - QUESTION: What would get much more complex if you allow non-hierarchy?
  - Disallowing multiple links to directories →
    - Easier search
    - **Easier garbage collection** – no cycles
  - Engineering "taste" – give up a tiny bit of generality for a big savings in complexity
  - Eventually augmented with soft links, but soft links don't increment link count, so they can dangle.
  - Windows NT/2K/XP does without real soft links (shortcuts are interpreted by applications, not the OS).
- Directories are files
  - A certain elegance, but does this really help anything? What about network file systems?

  * Review: what is an inode? What is in an inode?
  * What is a directory? What is in a directory?
  * How do I find the inumber for file /foo/bar?
  * How do I find the inode for inumber 49824?
  * How do I read the third block of file /foo/bar?
  * What is the difference between a hard link and a symbolic link?

- How important are hard links? They save space, but they confuse users. They frustrate space accounting :).

```
danko:/usr/lib/i386-linux-gnu/dri> ls -il *.so
20897983 -rw-r--r-- 5 root root 5477116 May 15 04:43 i915_dri.so
20897983 -rw-r--r-- 5 root root 5477116 May 15 04:43 i965_dri.so
20885369 -rw-r--r-- 1 root root 4262808 May 15 04:42 nouveau_dri.so
20897983 -rw-r--r-- 5 root root 5477116 May 15 04:43 nouveau_vieux_dri.so
20897983 -rw-r--r-- 5 root root 5477116 May 15 04:43 r200_dri.so
20897984 -rw-r--r-- 1 root root 3488564 May 15 04:42 r300_dri.so
20897982 -rw-r--r-- 1 root root 3996492 May 15 04:42 r600_dri.so
20897983 -rw-r--r-- 5 root root 5477116 May 15 04:43 radeon_dri.so
20885367 -rw-r--r-- 1 root root 3356880 May 15 04:42 radeonsi_dri.so

danko:/usr/lib/git-core> ls -il git* | head -n 45
20891152 -rwxr-xr-x 1 root root 1577256 Mar 19 09:43 git*
20891810 lrwxrwxrwx 1 root root       3 Mar 19 09:43 git-add -> git*
20890454 -rwxr-xr-x 1 root root   36841 Mar 19 09:43 git-add--interactive*
20890466 -rwxr-xr-x 1 root root   23060 Mar 19 09:43 git-am*
20891782 lrwxrwxrwx 1 root root       3 Mar 19 09:43 git-annotate -> git*
20896021 lrwxrwxrwx 1 root root       3 Mar 19 09:43 git-apply -> git*
20891824 lrwxrwxrwx 1 root root       3 Mar 19 09:43 git-archive -> git*
20890468 -rwxr-xr-x 1 root root   11997 Mar 19 09:43 git-bisect*
20891792 lrwxrwxrwx 1 root root       3 Mar 19 09:43 git-bisect--helper -> git*
20891788 lrwxrwxrwx 1 root root       3 Mar 19 09:43 git-blame -> git*

danko:~> mkdir tmp
danko:~> touch tmp/foo
danko:~> ls -l tmp
total 0
-rw-rw-r-- 1 witchel osa 0 Sep  4  2014 foo
danko:~> ls -ld foo
drwxrwxr-x 3 witchel osa 4096 May  5 23:53 foo/
danko:~> chmod 0 tmp
danko:~> cat tmp/foo
cat: tmp/foo: Permission denied
[status 1]
danko:~> chmod +x tmp
danko:~> ls -ld tmp
d--x--x--x 2 witchel osa 4096 Sep  4 01:20 tmp/
danko:~> cat tmp/foo
danko:~> rm tmp/foo
rm: cannot remove 'tmp/foo': Permission denied
[status 1]
danko:~> chmod +w tmp
danko:~> rm tmp/foo
```

```
danko:~> touch quux
danko:~> ls -l quux
-rw-rw-r-- 1 witchel osa 0 Sep  4  2014 quux
danko:~> ln quux tmp/quux
danko:~> chmod 0 tmp
danko:~> chmod 777 quux
danko:~> chmod 777 tmp
danko:~> ls -l tmp
total 0
-rwxrwxrwx 2 witchel osa 0 Sep  4 11:22 quux*
```

- In order to be able to list, read or write a file, you need execute permission on the directories leading to that file (e.g., on a, b, and c for /a/b/c/execute_me.py).
- File owner can always chmod, does not need write or execute permission in enclosing directory.
- Return value of read/write. Short reads, short writes, EWOULDBLOCK.
- Writing blocks of data far more efficient than individual bytes.
- What are sparse files? Why are they needed?
- `lockf`, `flock`, `fcntl`, why is Unix file locking such a mess? (The contradictions of fine-grained sharing)
- (slides for editor update, `rename`)
- Untyped data (byte oriented)
  - *The structure of files is controlled by the programs which use them, not by the system.* (p. 366)
  - What are the advantages of the "resource fork" ala Macintosh?
  - Where to store access control lists for a user-level implementation of a new file system?
  - Memory also "untyped": *Another important aspect of programming convenience is that there are not "control blocks" with a complicated structure partially maintained by and depended on by the file system or other system calls. Generally speaking, the contents of a program's address space are the property of the program, and we have tried to avoid placing restrictions on the data structures within that address space.* (p. 374)
  - IRIX had the PRDA (process data area). The OS puts useful data values on this page so user programs can read them without system calls. Generally considered a wart, but an understandable one.

    ```
    /* magicpage.c -- check for a mapping at 0x200000, IRIX's PRDA */
    #include <stdio.h>

    char *prda = (char *)0x200000;

    int main(int argc, char **argv) {
        char c;
        printf("trying %p\n", prda);
    ```

```
        c = *prda;
        printf("ok\n");
        return 0;
    }
```

– Linux has `vdso` (and for backward compatibility `vsyscall`), a page mapped into each user address space that has code to implement fast system calls (e.g., gettimeofday, time and getcpu). The format of the page is...an ELF dynamic shared object(!)

- File creation (UNIX vs. IBM system 360)
  - **UNIX**
    echo > /tmp/foo
    redirection rocks
  - **IBM system 360 job control language**

```
//PDSCRTJ1 JOB SIMOTIME,ACCOUNT,CLASS=1,MSGCLASS=0,NOTIFY=CSIP1,
//             COND=(0,LT)
//* *******************************************************************
//*                   This program is provided by:                    *
//*                    SimoTime Enterprises, LLC                      *
//*           (C) Copyright 1987-2005 All Rights Reserved             *
//*             Web Site URL:   http://www.simotime.com               *
//*                   e-mail:   helpdesk@simotime.com                 *
//* *******************************************************************
//*
//* Subject: Define a PDS using the IEFBR14 with a DD Statement
//* Author:  SimoTime Enterprises
//* Date:    January 1,1998
//*
//* Technically speaking, IEFBR14 is not a utility program because it
//* does nothing. The name is derived from the fact that it contains
//* two assembler language instruction. The first instruction clears
//* register 15 (which sets the return code to zero) and the second
//* instruction is a BR 14 which performs an immediate return to the
//* operating system.
//*
//* IEFBR14's only purpose is to help meet the requirements that a
//* job must have at least one EXEC statement. The real purpose is to
//* allow the disposition of the DD statement to occur.
//*
//* For example, the following DISP=(NEW,CATLG) will cause the
//* specified DSN (i.e. PDS) to be allocated.
//* Note: a PDS may also be referred to as a library.
//*********************************************************************
//*
//IEFBR14  EXEC PGM=IEFBR14
//TEMPLIB1 DD  DISP=(NEW,CATLG),DSN=SIMOTIME.DEMO.TEMPLIB1,
//             STORCLAS=MFI,
//             SPACE=(TRK,(45,15,50)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=800,DSORG=PO)
//*
```

  - How about a key to some of the terms
    * Top line is job metadata, the scheduling class, and how error messages are to be reported to the user.
    * EXEC line specifies program. IEFBR14 is a label to refer to the job.
    * TEMPLIB1 is a new dataset to be created as a side effect of running the job.

    - NEW,CATLG means this is a new dataset that should persist after the job is finished.
    - DSN is dataset name. OS360 has a 3 level "heirarchy."
    - STORCLAS specifies the symbolic name of the unit and volume of the disk pack which stores the data and some file metadata defaults (like RECFM, LRECL). For SMS (storage management system).
    - SPACE specifies the size of the dataset, here in tracks (blocks, and cylinders are also available. In this case allocate 45 tracks initially, increment by 15 when the dataset needs to grow (though a dataset can only be grown 15 times), and use up to 50 tracks if this is a partitioned data set (directory). Rule of thumb, one directory block for every 6 entries.
      Disks in IBM land have 512 byte blocks, a device-dependent number of sectors per track (17, 35, 75), a device dependent number of tracks (up to 1024), a device dependent number of heads, and a cylinder which contains as many tracks as there are heads.
    - DCB is the data control block
    - RECFM specifies the record format. FB is fixed block records (variable length records, undefined length records and others also available).
    - LRECL is the logical record length, here 80 characters
    - BLKSIZE is the size of the data control block itself (i.e., "inode").
    - DSORG specifies the data set organization. PO is partitioned organization (the default).

- Device-independent I/O
  - *There is a threefold advantage in treating I/O devices this way: file and device I/O are as similar as possible; file and device names have the same syntax and meaning, so that a programming expecting a file name as a parameter can be passed a device name; finally, special files are subject to the same protection mechanism as regular files.* (p. 367)
    I'd say (3) is reasonably compelling, (1) is mildly useful, though high-performance implementations tend to treat e.g., network I/O differently from disk. (2) seems useless to dangerous. The "ioctl" interface for device-specific functionality is terrible.
  - Simple owner/group/other permissions remarkably flexible and useful.
  - "Pipes are not a completely general mechnism since the pipe must be set up by a common ancestor of the process." Now, named pipes in the file system. Though sockets are more general than pipes.
  - Chillaxing with Thompson and Ritchie, "A program which is used rarely or which does no great volume of I/O may quite reasonably read and write in units as small as it wishes."
  - Problems with /dev and their solution with udev (http://lwn.net/Articles/65197/) (note, udev is now integrated with systemd)
    - A static /dev is unwieldy and big. It would be nice to only show the /dev entries for the devices we actually have running in the system.
    - We are (well, were) running out of major and minor numbers for devices.
    - The database of major/minor numbers in the kernel must match the database in `/dev` or the user can not access device functionality (e.g., kernel built with CONFIG_TUN and "mknod /dev/net/tun c 10 200")

    - Users want a way to name devices in a persistent fashion (i.e. "This disk here, must _always_ be called "boot_disk" no matter where in the scsi tree I put it", or "This USB camera must always be called "camera" no matter if I have other USB scsi devices plugged in or not.") With /dev, the first USB printer is lp0, and the second is lp1 (same major number, incremented minor number). But add a USB hub, and the device names could be swapped.
    - Userspace programs want to know when devices are created or removed, and what /dev entry is associated with them.
  - udev has replaced devfs, which replaced /dev. So it is a "deep" change that took a while to settle down.
    - using udev, the /dev tree only is populated for the devices that are currently present in the system.
    - udev emits D-BUS messages so that any other userspace program (like HAL) can listen to see what devices are created or removed. It also allows userspace programs to query it's database to see what devices are present and what they are currently named as (providing a pointer into the sysfs tree for that specific device node.)
  - Network cards are not in `/dev`

- File descriptors are a little piece of brilliance.
  - Filter programs do not know the name of input or output files.
  - "Handle with access rights" – that is a capability, which is an abstraction that makes protected sharing easier.
  - How many file descriptors can you open?
  - File descriptors are just integers, why can't a user program forge one?

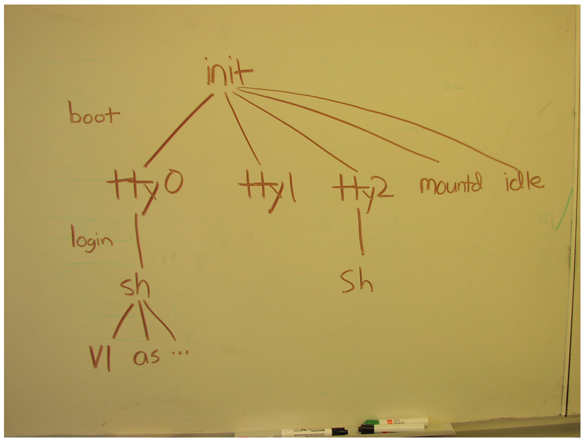
- Set-User-ID (rights amplification)
  - Coarse-grained sharing – "execute a program as someone else"
  - v. Multics rings – fine grained sharing – "execute a procedure as someone else"
  - Minimalist: Need to have process == principal anyhow
  - Just add setuid to that basic mechanism rather than invent an orthogonal authentication model
  - But, given "psychological acceptability" constraint, are there limits to how fine-grained we can (correctly/conveniently) divide programs?
  - Also "Make common case fast. Make uncommon case correct." Common case is – procedure call to same code base. How much extra mechanism do you want (complexity, cost, speed penalty in common case) for uncommon case?
  - Compare power of approach
    - **Question**: suppose you have a "protected subsystem" S in multics that has data D_S that only it can access and to procedures S_{P1} that can be accessed by A and B and S_{P2} that can only be accessed by A
      - How would you arrange this in Multics?
      - Can you arrange this in Unix? How?

    * Long list of buggy kernel modules can crash Linux—the chickens of simplicity have come home to roost.
    * The single root user is a big Achilles heel in the Linux/DAC permissions system. Compromise of any process running as root means compromise of entire OS even though the program probably only needed one feature of root (ambient authority).
    * Andriod operating system installs each application under a unique user id. Why?

- Mount
  - Removable storage; expand storage;
  - Engineering simplification: No cross-volume links allowed

5 Process management

- Theme
  - Primitives, not "solutions"
  - "Happily, all of this mechanism meshes very nicely..."
- A process is an executing program (or image). Code, heap and stack.
- Building blocks
  - Fork, exec, wait
  - File I/O structure
    * Fork'd child shares parent's open files
    * $\rightarrow$ pipe, std I/O, redirection, filters
    * "Pipes are not a completely general mechnism since the pipe must be set up by a common ancestor of the process." Now, named pipes. Though sockets are more general than pipes.
    * Coarse grained sharing of programs: cat foo | grep "bar" | sort | tail -10
  - Shell, bg execution (being able to suspect a program is just brilliant (though comes later) and absent from Windows).
  - Running a program in the background is great. Why do we need `nohup`?
  - Standard in and standard out. What a great idea, enables redirection and pipelines.
  - Shell turned out to be a pretty bad programming language though.
    * Elegant process structure enables communication
    * Signals as another form of inter-process communication.



∗ Shell pseudo-code

```
shell(){
  while(got = read(STDIN, buffer, ...)){
    command, args, redirection, bg = parse(buffer);
    if(pid = fork()){
      /* I am the child */
      if(redirection){
        close stdin and/or stdout and open specified files
      }
      exec(command, args);
      /* Only reached if error on exec */
      exit(-1);
    }
    /* I am the parent */
    if(!bg && donePid != pid){
      donePid = wait();
    }
}
```

∗ HW Question: How to add pipes?

6 Fight the man, with UNIX!

Our goals throughout the effort, when articualted at all, have always concerned themselves with building a comfortable relationship with the machine and with exploring ideas an inventions in operating systems. We have not been faced with the need to satisfy someone else's requirements, and for this freedom we are grateful. (p. 374)

- Dewey: “Rock ain't about doing things perfect. Who can tell me what it's really about? Frankie?”
  Frankie: “Uh? Scoring chicks?”
  Dewey: “No. See? No. Eleni?”
  Eleni: “Getting wasted?”
  Dewey: “No. Come on. No.”
  Leonard: “Sticking it to The Man?”

Dewey: "Yes!"
–School of Rock (2003)

- *The current version of UNIX avoids the issue by not charging any fees at all* (p. 369)
- Computer scientists like to build scalable, fault-tolerant, best-effort systems, all of which conflict with central authority. Technical successes like Internet routing, the web, UNIX-style accounting and NFS eschew the central authority embraced by failures like AFS, and IBM system 360 accounting. Of course, sometimes central authority is a good thing, like the domain name service or ethernet card numbering.

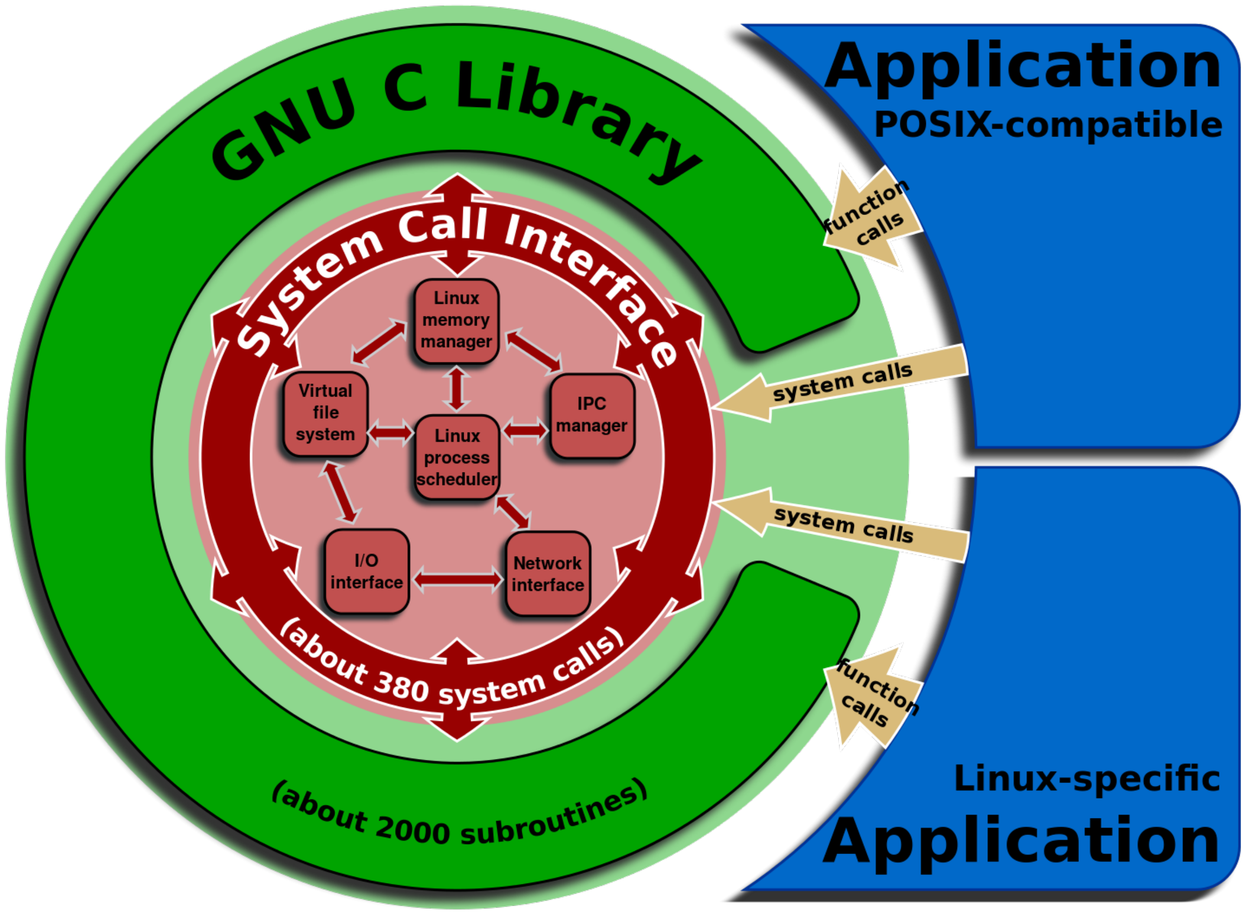


Figure 1: Applications, libraries and OSes

7 Lessons

How do you teach/learn "elegance"? I don't know. Study case studies, try to learn lessons.

When you build a system, ask yourself "What would Richie and Thompson say about my design?"

Can it be learned at all? Dijkstra attributes much of elegance to "fear", Richie and Thompson to having small machines.

7.1 Unix v. Multics v. THE

Note: Multics design and THE design pre-date Unix. Unix borrows liberally.

"The success of UNIX lies not so much in new inventions but rather in the full exploitation of a carefully selected set of fertile ideas, and especially in showing that they can be keys to the implementation of a small yet powerful operating system. (p. 374)

- Unix v. Multics
  - Implementation effort: 2 work-years v. ? work-years
  - Implementation latency: 2 years v. inf
  - Key ideas

Feature	Multix	Unix
Key abstraction	Unify file = memory	Unify I/O = file
Protection	rings (jump to memory)	suid (execute file)
Sharing	N segments; arbitrary sharing	3 segments (text, heap, stack) text is shared (RO); Communication between domains via files, pipes

  - Coarse grained (file exec) v. fine grained sharing (procedure call)
    - Elegant structure
      - unified file I/O, interprocess I/O (pipes), device I/O
      - cooperating process structure: fork, exec, shared files, shell, redirection, filters
      - setuid
    - Lampson: coarse grained sharing (pipes) is one of great success in CS; fine-grained sharing an "abject failure" (Dahlin's paraphrase of NSF workshop, SOSP keynote talks).
    - Lessons?
      - Simple primitives v. general solutions?
      - "Psychological acceptability" limits us to coarse-grained 2-entity sharing most of the time anyhow?
      - Trade performance and features for simplicity?
  - Approach
    - Multics – "we need fine grained sharing; design the 'right' mechanism for it"
    - Unix – "any system we build must have (1) process, (2) ability to read and write files from a process, (3) a user ID associated with a process for access control, (4) be able to read/write tty (and other devices); given those requirements, can we design the basic abstractions in a way that supports sufficient sharing? (Yes. In fact, now that we look at it this way, file IO and device IO are the same thing....)"
- Unix v. THE
  - Similar scale (2-3 work-years)
  - Dates of completion: THE (1967?), Unix (v1 1969, v2 1971)

- Similar abstractions

THE	Unix
Level 0: sequential execution	process
Level 1: Virtual memory	virtual memory
Level 2: Message demux	File I/O + special files + process
Level 3: Buffer I/O	Buffered I/O
Level 4: User-level process	User-level process
Level 5: Operator	Operator

- UNIX bloat, OS bloat
  - Much of multics has been put back in
  - QUESTION: Many have given up on OS security and are resorting to virtual machines for process isolation...

7.2 Quotable UNIX

- *Given the partially antagonistic desires for reasonable efficiency and expressive power, the size constraint has encouraged not only economy but a certain elegance of design.* (p. 374)
  - Both THE and Unix gain success/elegance by ruthless simplification
  - THE motiation "fear"
  - Unix motivation – hardware constraints
  - Today's motivation? (discipline?)
- *Nearly from the start, the system was able to, and did, maintain itself. This fact is more important than it might seem. If designers of a system are forced to use that system, they quickly become aware of its functional and superficial deficiencies and are strongly motivated to crrect them.* (p. 374)
  - "Eat your own dogfood"
  - "worked once" system v. "real system"

8 Torvalds

- Linux is portable in part because it was not built to be portable. Sometimes it is better to be deep than wide. Good design principles and a good development model. (I've heard it said that most tech blockbusters combine a technical innovation with a business innovation, e.g., Netflix, Google (advertising), subsidized smart phones.
- He doesn't like microkernels. Or academic research. I agree with many of his arguments (more for the former).
- "You can present a better architecture to the OS than is really available on the actual hardware platform" also true of virtualization.
- Avoid interfaces.
- Linux relies on gcc. Choose your allies/dependences carefully.
- I like his attitude toward open source/GPL. People can really get fanatical about it.

9 Singularity

Let's revisit OSes from a modern persepctive.

- First, the pervasive use of safe programming languages eliminates many preventable defects, such as buffer overruns.
- Second, the use of sound program verification tools further guarantees that entire classes of programmer errors are removed from the system early in the development cycle.
- Third, an improved system architecture stops the propagation of runtime errors at well-defined boundaries, making it easier to achieve robust and correct system behavior
- Software-isolated processes
  - Closed object space. No shared writable memory, no dynamic code loading.
  - Exchange heap is a nightmare.
  - Isolation provided by software, not hardware. Does this matter?
  - Low cost makes it practical to use SIPs as a fine-grain isolation and extension mechanism to replace the conventional mechanisms of hardware protected processes and unprotected dynamic code loading.
  - As a consequence, Singularity needs only one error recovery model, one communication mechanism, one security architecture, and one programming model, rather than the layers of partially redundant mechanisms and policies found in current systems.
- Contract-based channels
  - A channel provides a lossless, in-order message queue.
  - A channel endpoint belongs to exactly one thread at a time. Only the endpoint's owning thread can dequeue messages from its receive queue or send messages to its peer.
  - A contract consists of message declarations and a set of named protocol states.
  - Channel contracts provide a clean separation of concerns between interacting components and help in understanding the system architecture at a high level. Static checking helps programmers avoid runtime "message not-understood errors."
- Manifest-based programs
  - To start execution, a user invokes a manifest, not an executable file
  - A manifest describes an manifest-based program's code resources, its required system resources, its desired capabilities, and its dependencies on other programs.
  - The primary purpose of the manifest is to allow static and dynamic verification of properties of the MBP. For example, the manifest of a device driver provides sufficient information to allow installtime verification that the driver will not access hardware used by a previously installed device driver.
  - Additional MBP properties which are verified by Singularity include type and memory safety, absence of privileged-mode instructions, conformance to channel contracts, usage of only declared channel contracts, and correctlyversioned ABI usage.
  - Singularity MBPs is delivered to the system as compiled Microsoft Intermediate Language (MSIL) binaries. Compiled at install time.