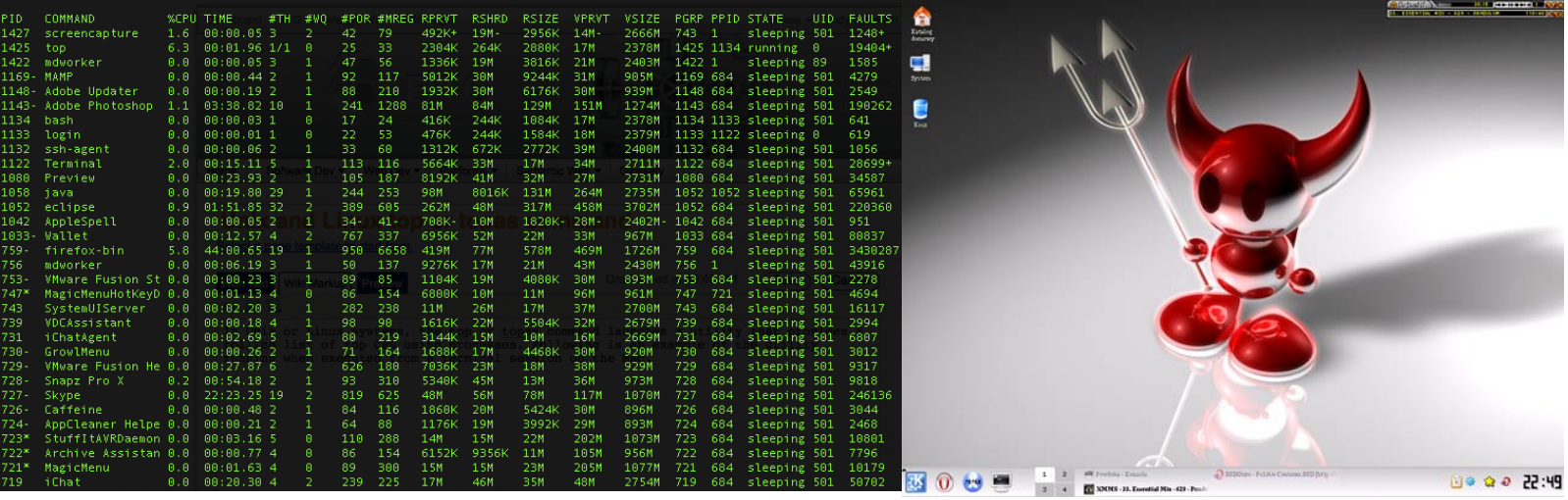


PID	COMMAND	%CPU	TIME	#TH	#WQ	#POR	#MREG	RPRVT	RSHRD	RSIZE	VPRVT	VSIZE	PGRP	PPID	STATE	UID	FAULTS
1427	screencapture	1.6	00:00.05	3	2	42	79	492K+	19M-	2956K	14M-	2666M	743	1	sleeping	501	1248+
1425	top	6.3	00:01.96	1/1	0	25	33	2304K	264K	2880K	17M	2378M	1425	1134	running	0	19404+
1422	mdworker	0.0	00:00.05	3	1	47	56	1336K	19M	3816K	21M	2403M	1422	1	sleeping	89	1585
1169-	MAMP	0.0	00:00.44	2	1	92	117	5012K	30M	9244K	31M	905M	1169	684	sleeping	501	4279
1148-	Adobe Updater	0.0	00:00.19	2	1	88	210	1932K	30M	6176K	30M	939M	1148	684	sleeping	501	2549
1143-	Adobe Photoshop	1.1	03:38.82	10	1	241	1288	81M	84M	129M	151M	1274M	1143	684	sleeping	501	190262
1134	bash	0.0	00:00.03	1	0	17	24	416K	244K	1084K	17M	2378M	1134	1133	sleeping	501	641
1133	login	0.0	00:00.01	1	0	22	53	476K	244K	1584K	18M	2379M	1133	1122	sleeping	0	619
1132	ssh-agent	0.0	00:00.06	2	1	33	60	1312K	672K	2772K	39M	2400M	1132	684	sleeping	501	1056
1122	Terminal	2.0	00:15.11	5	1	113	116	5664K	33M	17M	34M	2711M	1122	684	sleeping	501	28699+
1080	Preview	0.0	00:23.93	2	1	105	187	8192K	41M	32M	27M	2731M	1080	684	sleeping	501	34587
1058	java	0.0	00:19.80	29	1	244	253	98M	8016K	131M	264M	2735M	1052	1052	sleeping	501	65961
1052	eclipse	0.9	01:51.85	32	2	389	605	262M	48M	317M	458M	3702M	1052	684	sleeping	501	220360
1042	AppleSpell	0.0	00:00.05	2	1	34-	41-	708K-	10M	1820K-	28M-	2402M-	1042	684	sleeping	501	951
1033-	Wallet	0.0	00:12.57	4	2	767	337	6956K	52M	22M	33M	967M	1033	684	sleeping	501	80837
759-	firefox-bin	5.8	44:00.65	19	1	950	6658	419M	77M	578M	469M	1726M	759	684	sleeping	501	3430287
756	mdworker	0.0	00:06.19	3	1	50	137	9276K	17M	21M	43M	2430M	756	1	sleeping	501	43916
753-	VMware Fusion St	0.0	00:00.23	3	1	89	85	1104K	19M	4080K	30M	893M	753	684	sleeping	501	2278
747*	MagicMenuHotKeyD	0.0	00:01.13	4	0	86	154	6800K	10M	11M	96M	961M	747	721	sleeping	501	4694
743	SystemUIServer	0.0	00:02.20	3	1	282	238	11M	26M	17M	37M	2700M	743	684	sleeping	501	16117
739	VDCAssistant	0.0	00:00.18	4	1	98	90	1616K	22M	5504K	32M	2679M	739	684	sleeping	501	2994
731	iChatAgent	0.0	00:02.69	5	1	80	219	3144K	15M	10M	16M	2669M	731	684	sleeping	501	6807
730-	GrowlMenu	0.0	00:00.26	2	1	71	164	1688K	17M	4468K	30M	920M	730	684	sleeping	501	3012
729-	VMware Fusion He	0.0	00:27.87	6	2	626	180	7036K	23M	18M	30M	929M	729	684	sleeping	501	9317
728-	Snapz Pro X	0.2	00:54.18	2	1	93	310	5340K	45M	13M	36M	973M	728	684	sleeping	501	9818
727-	Skype	0.0	22:23.25	19	2	819	625	48M	56M	78M	117M	1070M	727	684	sleeping	501	246136
726-	Caffeine	0.0	00:00.48	2	1	84	116	1860K	20M	5424K	30M	896M	726	684	sleeping	501	3044
724-	AppCleaner Helpe	0.0	00:00.21	2	1	64	88	1176K	19M	3992K	29M	893M	724	684	sleeping	501	2468
723*	StuffItAVRDaemon	0.0	00:03.16	5	0	110	288	14M	15M	22M	202M	1073M	723	684	sleeping	501	10801
722*	Archive Assistan	0.0	00:00.77	4	0	86	154	6152K	9356K	11M	105M	956M	722	684	sleeping	501	7796
721*	MagicMenu	0.0	00:01.63	4	0	89	300	15M	15M	23M	205M	1077M	721	684	sleeping	501	10179
719	iChat	0.0	00:20.30	4	2	239	225	17M	46M	35M	48M	2754M	719	684	sleeping	501	5070

“The Unix Philosophy”

Dr. Michael S. Brown
Associate Professor
School of Computing

NUS
National University
of Singapore

NERD!

http://www.youtube.com/watch?v=dFUlAQZB9Ng

One word summary of Unix?

Simplicity

"UNIX is very simple, it just needs a genius to understand its simplicity.“

Dennis Ritchie (Unix co-creator)

Historical Perspective

- First generation of computers (1930-1950)
  - Only a handful of computers in existance
    - E.g. Zuse Z3, Colossus, Harvard Mark 1, ENIAC
  - These computers are glorified calculators
  - Run “programs” in batches
  - Almost all are “government” sponsored machines

The Harvard Mark I

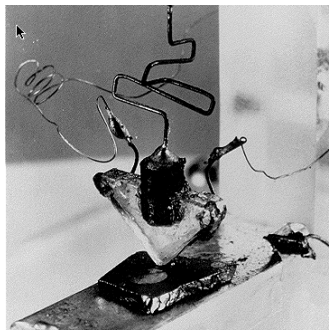
Historical Perspective

- Second gen computers (1950-1960)
  - Commercial Computers
    - IBM, Remington Rand, Burroughs, Honeywell
  - Mainly used for calculation and statistics
  - Very expensive

- But things are happening
  - Transistors replace vacuum tubes
  - Disk storage, printers being developed
  - High level languages developed

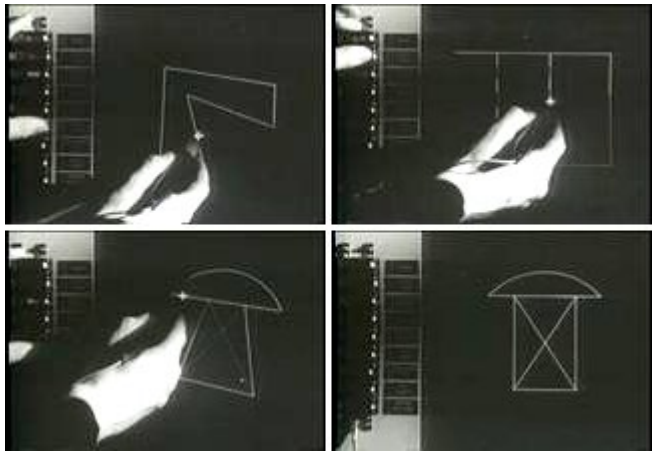
    Cobol (Common Business-Oriented Language),

    Fortran (Formula Translator)



Historical Perspective

- Third Generation (1960-1970)
  - More players enter the market
    - Bell Labs, GE, DEC, IBM, HP, Data General, Commodore
  - Uses are going beyond calculations
    - Ivan Sutherland introduces GUI, Graphics
    - Computers are used for Text Editing, type-setting



History of “Unics”

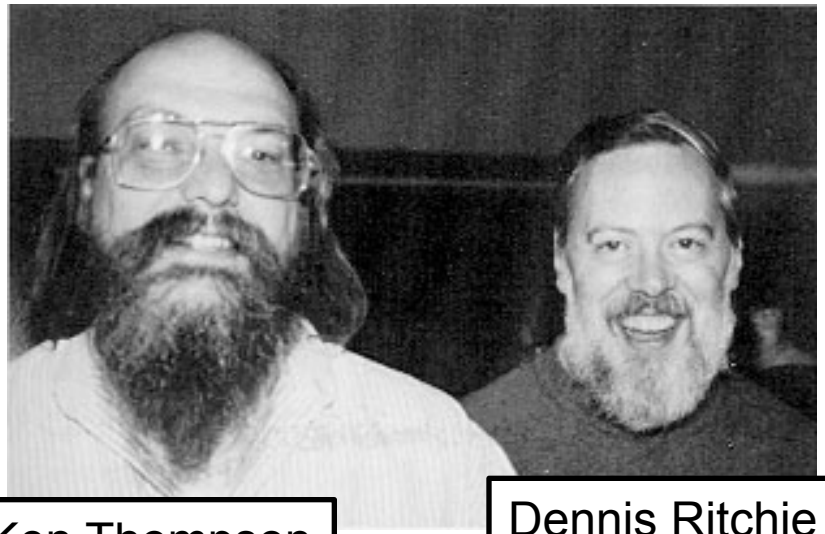
The “Fathers” of Unix

```
$ man “Ken Thompson”
 BS, MS Com-Sci: UC-Berkley
 Employers:
 Bell Labs
 Entrisphere, Inc
 Google

Notes:
1983 Turning Award
Winner

Developed UTF-8
encoding scheme.

Created the “B” programming
Langauge.
```



Ken Thompson

Dennis Ritchie

```
$ man “Dennis Ritchie”
 BS, Physics/Math: Harvard
 Employers:
 Bell Labs

Notes:
1983 Turning Award
Winner

Created the “C” programming
Language.
```

Engineers at Bell Labs, 1960s

Bell Laboratories

Multics

- **Mult**iplexed **I**nformation and **C**omputing **S**ervice
- A time-sharing OS started in 1964
- Partners: MIT, GE, Bell Labs
- Multics
  - Was growing too complicated and convoluted
  - Bell exited the project in 1969
  - Ken and Dennis worked on this project
  - Ken and Dennis wanted to make a simpler version
    - While working on Multics, they had already started, working on a "small" PDP-7 and experimenting with notions of processes, command-line interpreters, hierarchical file system, and utility programs

PDP-7

Unix (Unics)

- **Un**iplexed **I**nformation and **C**omputing **S**ervice (Unics)
  - Some people have purported the Unix was originally a play on Multics, i.e “Unics”
  - Dennis Ritchie states this is a myth, the term was Unix from the beginning . . however, you do wonder how they go to Unix?
- Driving idea of Unix
  - Portable
    - OS up to this point were wedded to the computer used
    - They wrote Unix in C, with minimal assembly code
  - Mutli-tasking, multi-user
  - Simple and light-weight OS (Kernel)
  - Use of “flat” plain text-files
  - Hierarchical file system
  - Large number of “software tools”

From the men themselves

http://www.youtube.com/watch?v=7FjX7r5icV8

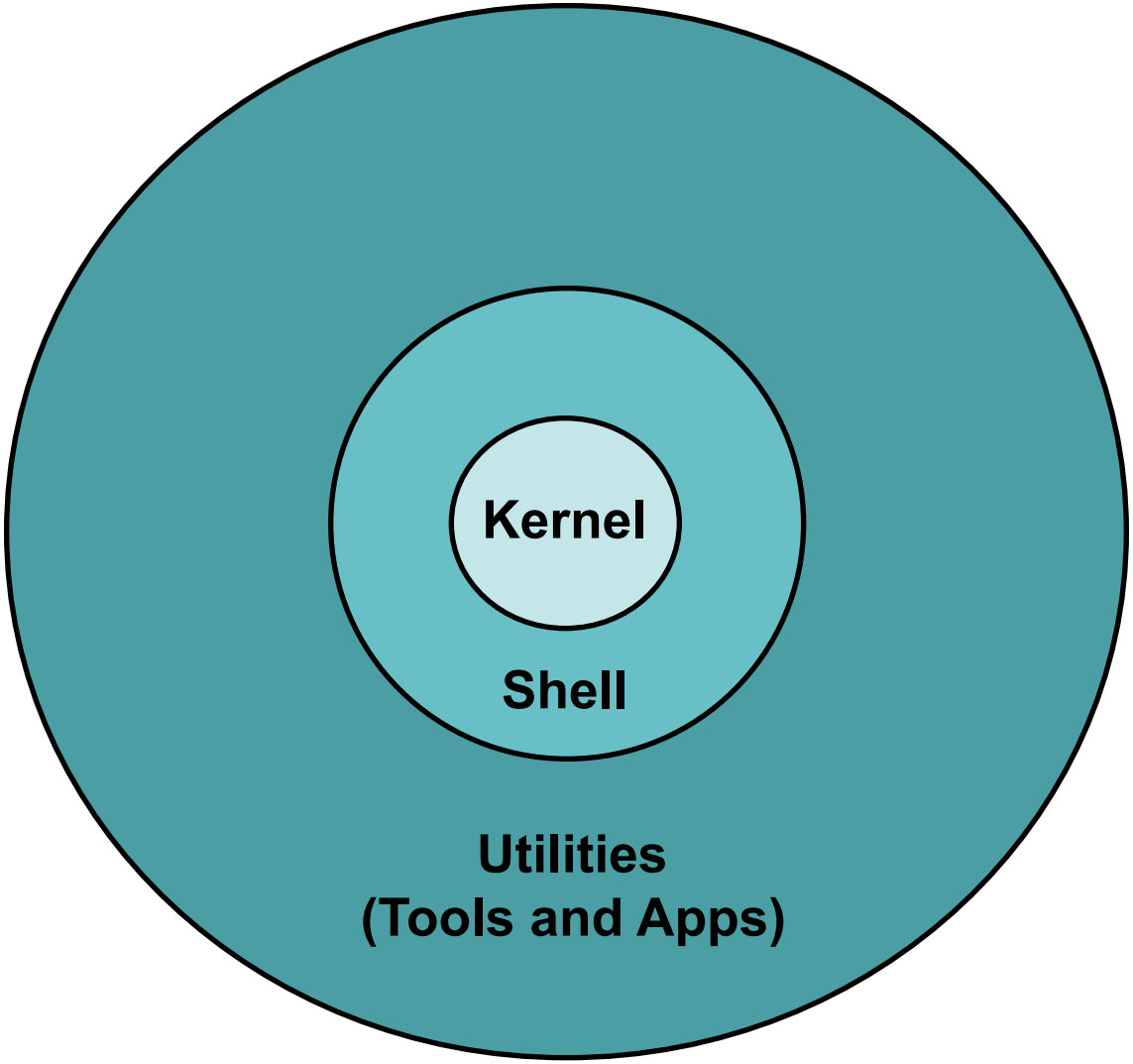
“Modern” OS features proposed in Unix

(1) Processes
(2) Device/Files Unification
(3) Shell (and pipes)
(4) File system

Multics

- Many of the ideas in Unix, were inspired by Multics
  - Processes
  - Hierarchal file system
  - “Virtual Memory”
- The problem with Multics?
  - The system was too complicated!
    - Tried to do everything, to be too versatile, too flexible, and failed
  - One of the driving philosophies in Unix was to “keep is simple”

Unix Design




```
$ man “kernel”

Controls the systems.
Main tasks:
Process management
Device communication
File system
```

```
$ man “shell”

User interface to
communicate with the system.
Includes a set of basic
shell-utilities.  Facilitates
scripting.
```

```
$ man “utilities”

The largest set of Unix.
Tools developed to assist
the user in their tasks,
e.g. text editor, plotting,
etc. .
```

(1) Process Management

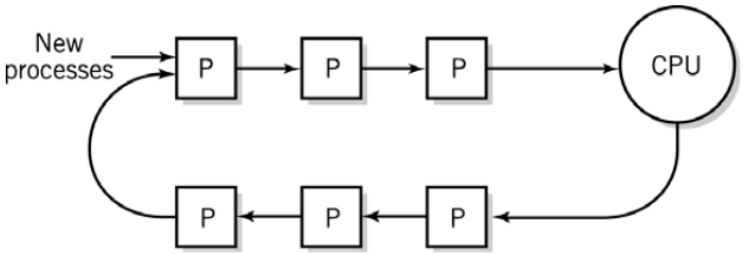
- Process management
- Process is an executing program
  - Its code+data
- Each process has a unique ID (PID)
- Multiple users can run multiple processes at once
  - Time-sharing, multi-user system
- Processes have a hierarchy (parent-child)

Unix Processes

- **Process components**

Unique ID (PID)
Parents ID (PPID)
Code (instructions)
Machine registers
Global data
Run-time data (stack)
Open files (file descriptors)
Environment variables
Credentials for security

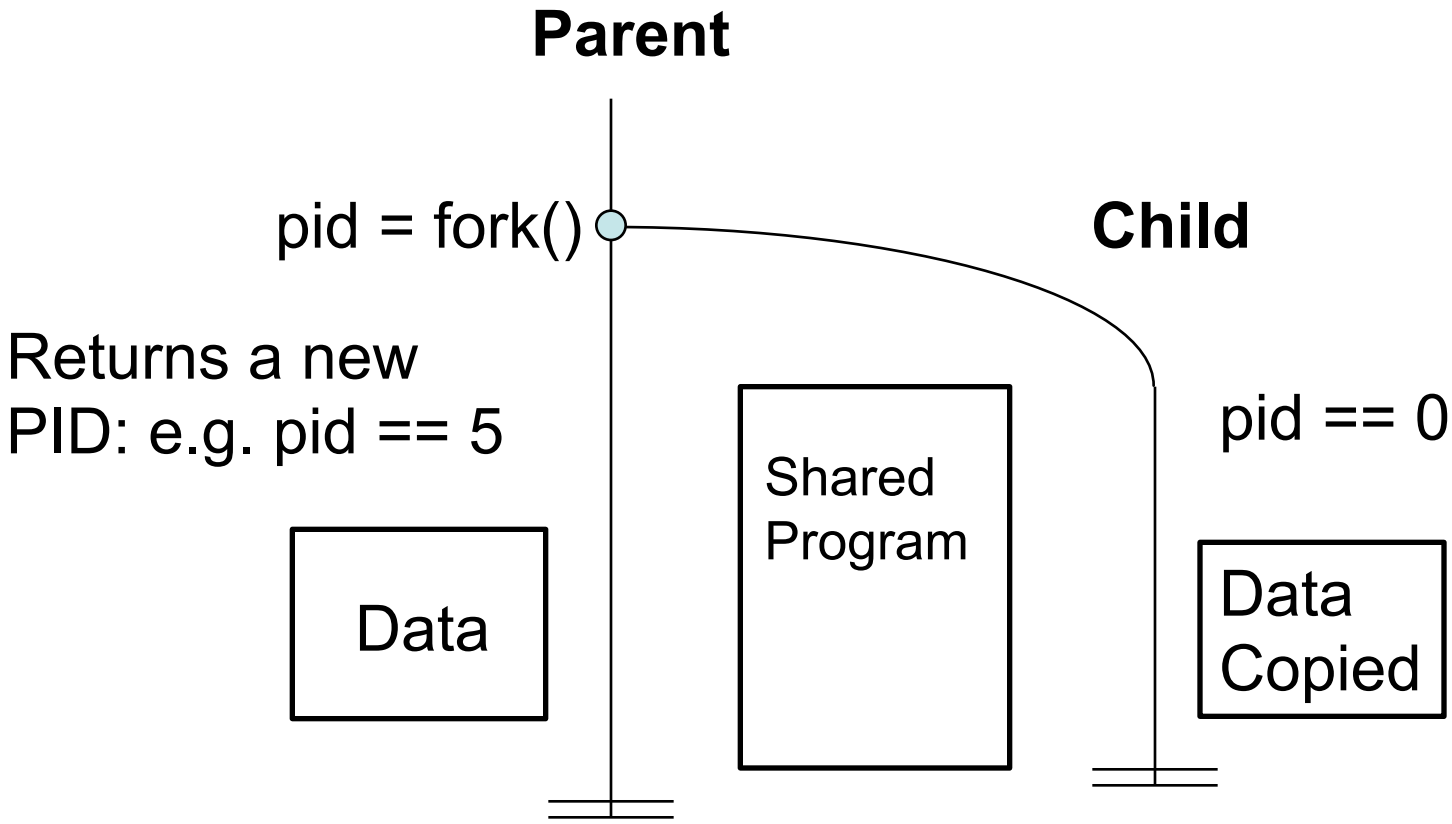
- Kernel manages this information
- Each process gets a small time on the CPU, in a round robin fashion (time-sharing)



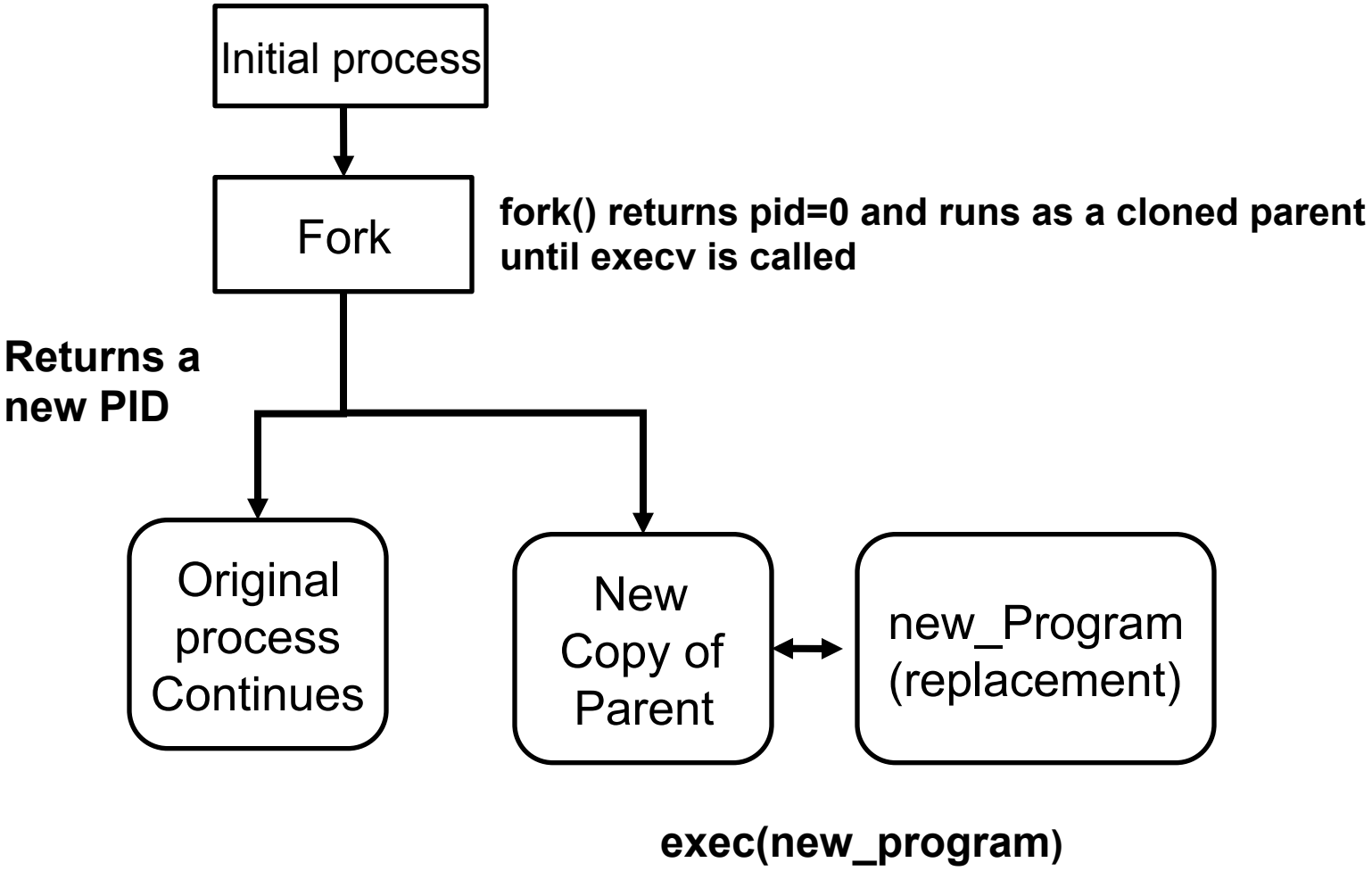
Process Creation

- A process can create a new process
- Creation strategy?
  - `fork()`
    - makes an exact copy of the process that calls fork()
  - `exec*()`
    - loads a program image to reply current process code

Fork()

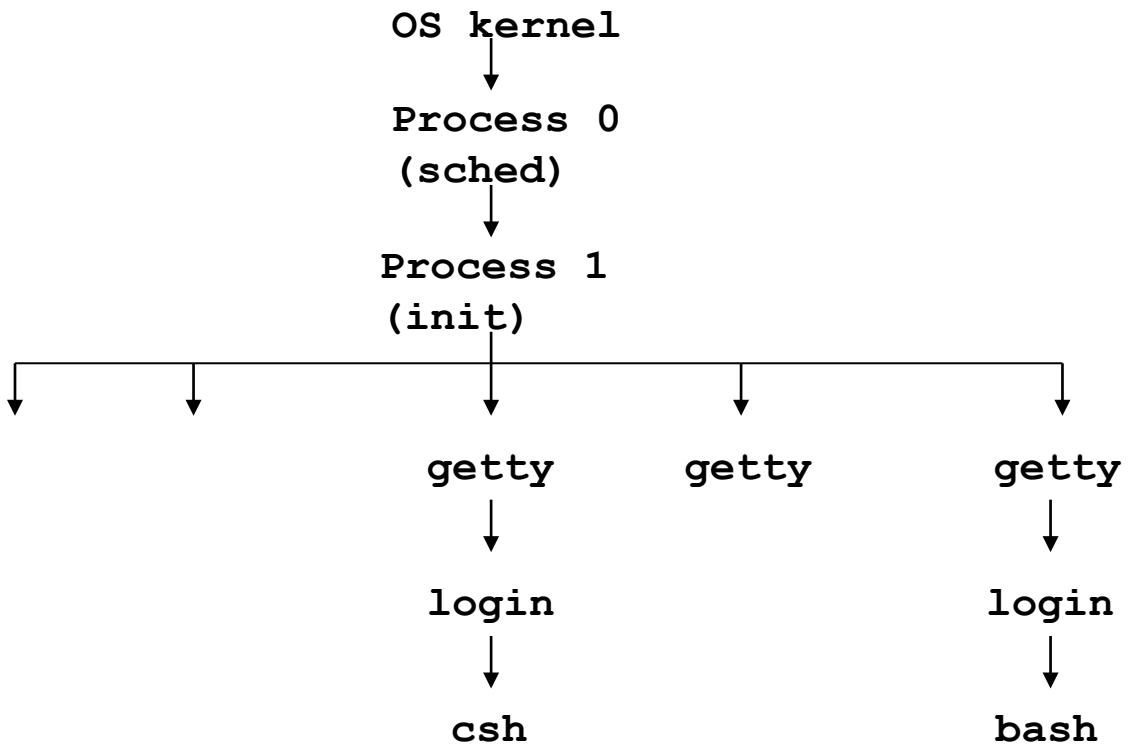

Parent
pid = fork()
Child
Returns a new
PID: e.g. pid == 5
pid == 0
Shared
Program
Data
Data
Copied

Fork() + Exec()


Initial process
Fork
fork() returns pid=0 and runs as a cloned parent until execv is called
Returns a new PID
Original process Continues
New Copy of Parent
new_Program (replacement)
exec(new_program)

Process Hierarchy

- `init` – the "mother" process
- `getty` – login process



(2) File/Device Unification

- Another Unix idea was the use of a single (file) interface for both files, devices, and even process communication
- Files
  - Can read and write characters or “blocks” of data
- Devices
  - Character devices (read characters, not “seekable”)
    - /dev/pts10
  - Block devices (read blocks of data, “seekable”)
- Pseudo Device
  - /dev/zero (produces zeros)
  - /dev/null (ignores all input, has no output)
  - /dev/random (produces a stream of random output)
- In Unix, files, devices all are interfaced the same
  - This makes it very convenient for writing code

(3) Unix Shell

- Command Line interpreter
  - **Is just another program**
  - BUT, it can be used to tie many programs together
    - Thanks to process creation and hierarchy
      - Parent of all processes in a shell are the shell!
  - Also, allows “shell scripting”
    - Sequential execution of shell commands
    - Evolved into full programming language

It is surprisingly easy to write your own basic shell using fork+exec. (Even pipes isn’t too hard)

(3) Unix Shell

- Common Shell Commands

sort Sorts lines in ascending, descending and unique order
grep Searches for regular expressions in strings or files
basename Strips the path from a path string to leave just the filename
dirname Removes the file from a path string to leave just the pathname
cut Chops up a text string by characters or fields
wc Count the characters, words, or lines
[(*test*)] Predicate or conditional processor
tr 'a' 'b' Transform characters
expr Simple arithmetic processor
bc Basic Calculator
eval Evaluate variables
echo Output strings
date Create date strings
nawk Manipulate text strings
head | tail Access lines in files

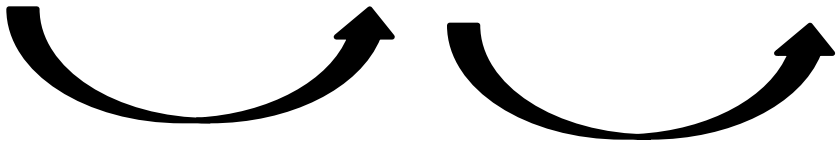
- Flow control

if, for, while, case, basic math

Pipes

- Unix also introduced the notion of “pipes”

```
cat file.txt | wc -l > output.txt
```



Output of process ‘cat’

Because the input of process ‘wc’

Output of process wc is input to file output.txt

Pipes was a powerful way to link the *stdout* of a program to the *stdin* of a other.

This is the cornerstone of Unix getting processes to “work together”.

(4) Unix File System

- Directory Structure
  - Tree-based structure
  - Allows “soft” and “hard links”
  - Some Unix systems allow graph structure (i.e. cycles, this can be dangerous)
- Directory Structure is well known
  - Unix systems are standard
  - Files kept in standard directories

  /usr/bin /sys/bin etc . . .

Unix File System

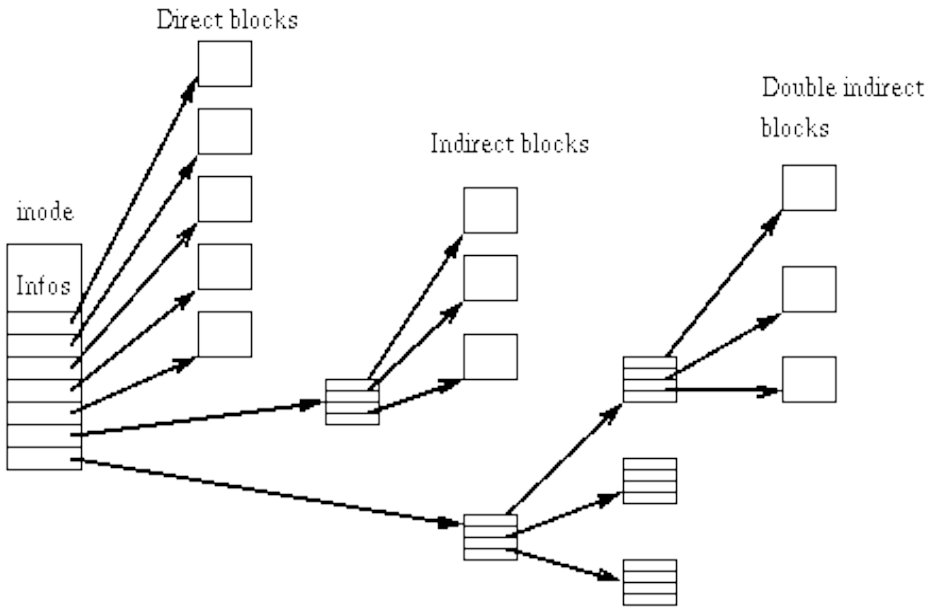
Boot Block	Super Block	I-nodes	Data Blocks

```
$ man "boot block"
Boot strap code to get the
system booted.
```

```
$ man "super block"
Meta-data about the
file system, hard drive,
tuning parameters.
```

```
$ man "inodes"
Meta-data (permissions),
and link to data blocks.
This is the "file"
```

```
$ man "data"
The actual data!
```



Modern systems allow file sizes up to 8 Zettabytes

kilobyte (kB)	10^3	2^{10}
megabyte (MB)	10^6	2^{20}
gigabyte (GB)	10^9	2^{30}
terabyte (TB)	10^{12}	2^{40}
petabyte (PB)	10^{15}	2^{50}
exabyte (EB)	10^{18}	2^{60}
zettabyte (ZB)	10^{21}	2^{70}
yottabyte (YB)	10^{24}	2^{80}

Unix Spreads Quickly

- 1972, Unix ported to C language
  - Now, entire OS is almost all in C, only small amount in assembly needs to be modified for a new platform
- Bell (a telephone company) was not allowed to enter the computing market
  - Ken Thompson started sending copies to anyone how asked, full distribution 10MB
- Bell became AT&T, licensed Unix to universities, commercial firms
- AT&T releases System 7+ Unix commercially, 1979+
- Berkeley Software Distribution (BSD) starts its own Unix, 1977

Linus Torvalds

- 1991, Linux is developed for PCs

Finnish student working on “Minix” (a educational only mini-unix by Andrew Tanenbaum), he decided to write his own Unix.

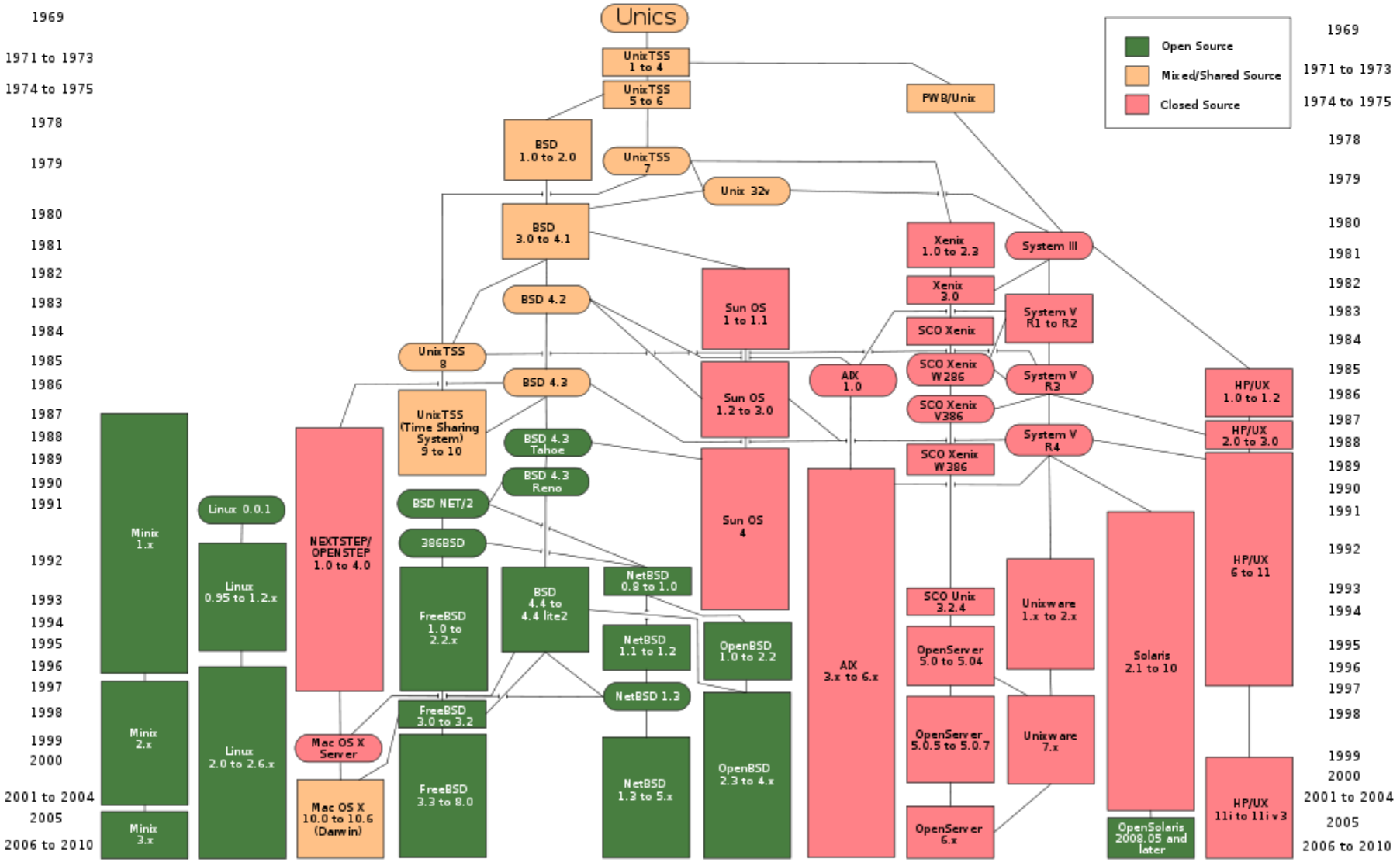
Linux, released under the GNU license.

The rest is History. . .

```
$ man “Linus”

Masters in CS, U. Helsinki
(Master thesis? LINUX)
Now, only around 2% of existing Linux kernels are from his original code.
His net worth is $20milion, due to stock options from Red Hat, etc. .
Now a US citizen.
```

The many flavors of Unix



The Unix Culture

- Unix created a culture
- People wanted a better, more efficient, computing system
- These people were logical and pragmatic in their thought
- They laid the foundation for:

“The Unix Philosophy”

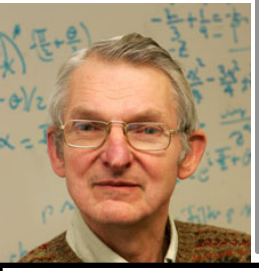
Driving Philosophy

According to Douglas McIlory, part of Unix development team at Bell Labs:

(1) Write programs that do one thing and do it well.

(2) Write programs to work together.

(3) Write programs to handle text streams, because that is a universal interface.



Douglas McIlroy

```
$ man "McIlory"
BS Cornell
PhD MIT
Bell Labs Career
Now retired, adjunct faculty at Dartmouth
```

Principals and Practices

- Rule 1. Bottlenecks occur in surprising places, so don't try to second guess and put in a speed hack until you've proven that's where the bottleneck is.
- Rule 2. Measure. Don't tune for speed until you've measured.
- Rule 3. Fancy algorithms are slow when *n* is small, and *n* is usually small. Fancy algorithms have big constants. Until you know that *n* is frequently going to be big, don't get fancy. (Even if *n* does get big, use Rule 2 first.)
- Rule 4. Fancy algorithms are buggier than simple ones, and they're much harder to implement. Use simple algorithms as well as simple data structures.
- Rule 5. If you've chosen the right data structures and organized things well, the algorithms will almost always be self-evident. **Data structures, not algorithms, are central to programming.**

Douglas McIlroy

```
$ man "Pike"
Education cannot be determined
Bell Labs, Developed early GUI
terminal for Unix in 1982 (not X)
Now at Google
(Did not receive the 1980 Olympic
Medal for Archery)
```

Some selective rules

(http://www.faqs.org/docs/artu/ch01s06.html)

Rule of Modularity

Write simple parts connected by clean interfaces.

Software can be very complicated, especially for large systems.
Its even hard for the programming to cope with the complexity.

Take a modular approach and build many small, but simple components.

These components work together through simple and clean interfaces.

**Object oriented programming uses this concept too.

Rule of Clarity

Clarity is better than cleverness.

Write code for humans to read, first!

```
void primes(int m, int t, int c) {
  int i,j;
  i = t / m;
  j = t % m;
  (i <= 1) ? primes(m,t+1,c) : (!j) ? primes(m,t+1,j) : (j == i && !c) ?
  (printf("%d\t",i), primes(m,t+1,c)) : (j > 1 && j < i) ?
  primes(m,t+1,c + !(i % j)) : (t < m * m) ? primes(m,t+1,c) : 0;
}

int main(void) {
  primes(100,0,0);
}
```

Rule of Composition

Design programs to be connected with other programs

THINK PIPE!

Stick with text streams for input and output.

Avoid the use of user input (for example, prompt for “yes”), use command line flags instead.

To make programs composable, make them independent. A program on one end of a text stream should care as little as possible about the program on the other end.

Rule of Simplicity

Design for simplicity; add complexity only where you must.

The notion of “intricate and beautiful complexities” is almost an oxymoron. Complexity leads to bugs.

Keep programs as simple as possible. Sometimes less is more.

“Worse” is “better”.

If other processing is required, consider writing another programming and exploiting composibility . . i.e. piping.

Rule of Transparency

Design for visibility to make inspection and debugging easier.

Debugging dominates development. Unix is made to have programs work together. Make it easy to understand how your program works (i.e. make it transparent).

For a program to demonstrate its own correctness, it needs to be using input and output formats sufficiently simple so that the proper relationship between valid input and correct output is easy to check.

A good ‘manual page’ with examples also helps!

Rule of Robustness

Robustness is the child of transparency and simplicity.

Program should work well. Minimize when it fails, and when it does fail, fail gracefully.

Consider situations, just as accepting empty lists/strings/etc., even in places where a human would seldom or never supply an empty string, avoids having to special-case such situations when generating the input mechanically.

Rule of Lease Surprise

In interface design, always do the least surprising thing.

The easiest programs to use are those that demand the least new learning from the user.

Stick with convention (+ means add).

Consider your audience.

Program used by sys administrators may have different common practices than a program used by programmers.

Avoid excessive features, novelty.

Rule of Silence

When a program has nothing surprising to say, it should say nothing.

I think that the terseness of Unix programs is a central feature of the style. When your program's output becomes another's input, it should be easy to pick out the needed bits. And for people it is a human-factors necessity — important information should not be mixed in with verbosity about internal program behavior. If all displayed information is important, important information is easy to find.

-- Ken Arnold

Ken Arnold

```
$ man “McIlory”
BS Berkeley
Contributor to the
original BSD Unix
distribution.

Also wrote the game “rouge”.
```

Rule of Repair

> Repair what you can — but when you must fail, fail noisily and as soon as possible.

Basically, if it is something minor, fix it (or consider handling it), but if you "fail" then make sure you cause a detectable failure to the other programs.

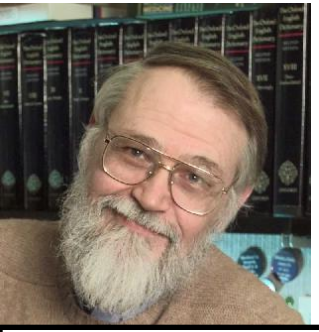
Also, if possible, make it clear why it failed.

Rule of Optimization

Prototype before polishing. Get it working before you optimize it.

"90% of the functionality delivered now is better than 100% of it delivered never!"
- Kernighan & Plauger
(Kernighan is co-author of the C Programming Book)

"Premature optimization is the root of all evil"
– Donald Knuth (Computing "Grandfather")



Brian Kernighan

Don Knuth

Unix Influence

- Apple OS
  - Built on BSD
- Chrome/OS
  - Google has hired many of the Unix people
- Unix inspirations
  - Xinu (**X**inu **I**s **N**ot **U**nix)
    Extremely light-weight OS for embedded systems

Unix Usage Today

- Hollywood (Lucas Film, ILM)
  - Processing ray-tracing renderings
  - Scripting
- Bank Industry
  - Database manipulation
  - Report creation
  - Scripting
- Super computers/clouds
  - Light-weight OS, better utilities resources than Windows

Still need windows?

- Try cygwin

  Pseudo-Unix for Windows

  Provides a great deal of functionality of Unix, on your Windows machine

Summary

- When Unix first came out, it represented a huge leap in OS design
- Established many of the modern ideas used today
- Windows has slowly copied these ideas
- There is always room for improvement
  - Don't become complacent
  - Cloud computing, very light-weight systems
- Maybe Google . . . .