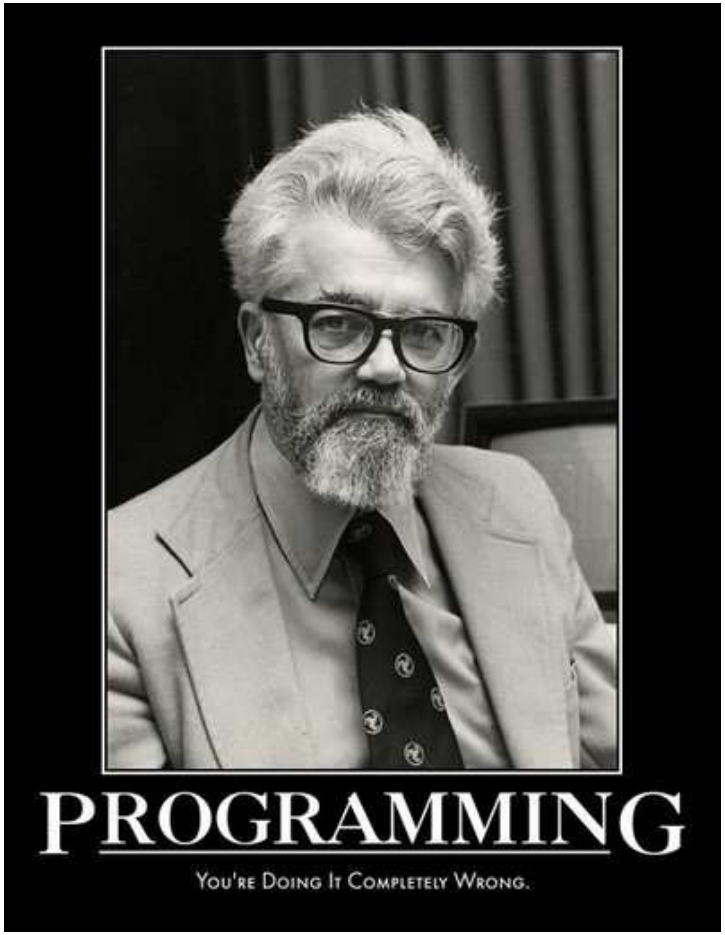
# CS631 - Advanced Programming in the UNIX Environment

Department of Computer Science
Stevens Institute of Technology
Jan Schaumann
jschauma@stevens.edu
http://www.cs.stevens.edu/~jschauma/631/

# Programming

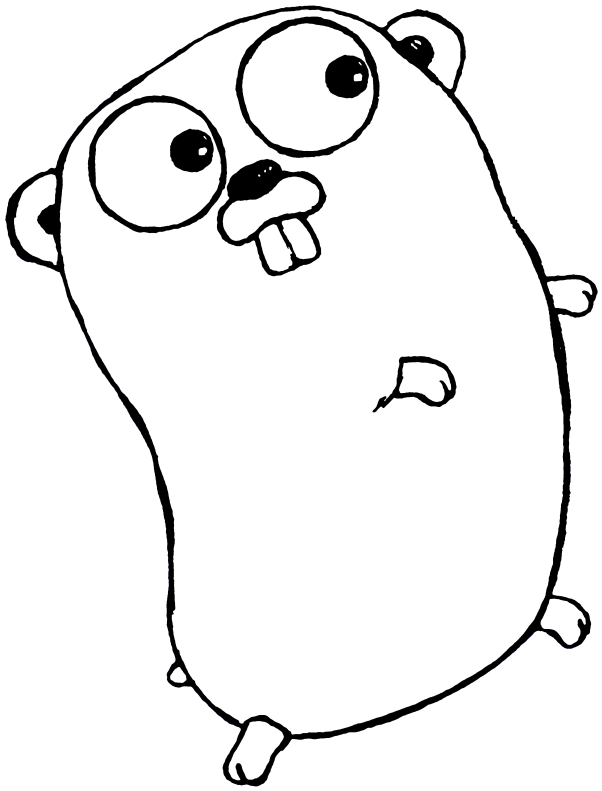



http://is.gd/Z4CLky

# Programming

http://cm.bell-labs.com/who/dmr/chist.html

# Extra Fun!



http://golang.org

# In a nutshell: the ”what”

```
$ ls /bin
[           csh         ed          ls          pwd         sleep
cat         date        expr        mkdir       rcmd        stty
chio        dd          hostname    mt          rcp         sync
chmod       df          kill        mv          rm          systrace
cp          domainname  ksh         pax         rmdir       tar
cpio        echo        ln          ps          sh          test
$
```

# In a nutshell: the ”what”

```
$ grep "(int" /usr/include/sys/socket.h
int accept(int, struct sockaddr * __restrict, socklen_t * __restrict);
int bind(int, const struct sockaddr *, socklen_t);
int connect(int, const struct sockaddr *, socklen_t);
int getsockopt(int, int, int, void * __restrict, socklen_t * __restrict);
int listen(int, int);
ssize_t recv(int, void *, size_t, int);
ssize_t recvfrom(int, void * __restrict, size_t, int,
ssize_t recvmsg(int, struct msghdr *, int);
ssize_t send(int, const void *, size_t, int);
ssize_t sendto(int, const void *,
ssize_t sendmsg(int, const struct msghdr *, int);
int setsockopt(int, int, int, const void *, socklen_t);
int socket(int, int, int);
int socketpair(int, int, int, int *);
$
```

# In a nutshell: the ”what”

- gain an understanding of the UNIX operating systems
- gain (systems) programming experience
- understand fundamental OS concepts (with focus on UNIX family):
  - multi-user concepts
  - basic and advanced I/O
  - process relationships
  - interprocess communication
  - basic network programming using a client/server model

# In a nutshell: the ”how”

```
static char dot[] = ".", *dotav[] = { dot, NULL };
struct winsize win;
int ch, fts_options;
int kflag = 0;
const char *p;

setprogname(argv[0]);
setlocale(LC_ALL, "");

/* Terminal defaults to -Cq, non-terminal defaults to -1. */
if (isatty(STDOUT_FILENO)) {
    if (ioctl(STDOUT_FILENO, TIOCGWINSZ, &win) == 0 &&
        win.ws_col > 0)
        termwidth = win.ws_col;
    f_column = f_nonprint = 1;
} else
    f_singlecol = 1;

/* Root is -A automatically. */
if (!getuid())
    f_listdot = 1;

fts_options = FTS_PHYSICAL;
while ((ch = getopt(argc, argv, "1ABCFLRSTWabcdfghiklmnopqrstuwx")) != -1) {
    switch (ch) {
    /*
     * The -1, -C, -l, -m and -x options all override each other so
     * shell aliasing works correctly.
     */
    case '1':
        f_singlecol = 1;
```

# In a nutshell: the ”how”

```
$ $EDITOR cmd.c
$ cc -Wall -g -o cmd cmd.c
$ ./cmd
$ echo "Hooray!"
Hooray!
$
```

# In a nutshell: the ”how”

```
$ $EDITOR cmd.c
$ cc -Wall -g -o cmd cmd.c
cmd.c: In function ‘main’:
cmd.c:19: error: parse error before "return"
$
```

# In a nutshell: the ”how”

```
$ $EDITOR cmd.c
$ cc -Wall -g -o cmd cmd.c
cmd.c: In function ‘main’:
cmd.c:19: error: parse error before "return"
$ $EDITOR cmd.c
$ cc -Wall -g -o cmd cmd.c
$ ./cmd
Memory fault (core dumped)
$
```

# In a nutshell: the ”how”

```
$ $EDITOR cmd.c
$ cc -Wall -g -o cmd cmd.c
cmd.c: In function ‘main’:
cmd.c:19: error: parse error before "return"
$ $EDITOR cmd.c
$ cc -Wall -g -o cmd cmd.c
$ ./cmd
Memory fault (core dumped)
$ echo "!@#!@!!!??#@!"
!@#!@!!!??#@!
$ gdb ./cmd cmd.core
Program terminated with signal 11, Segmentation fault.
Loaded symbols for /usr/libexec/ld.elf_so
#0  0xbbbc676a in __findenv () from /usr/lib/libc.so.12
(gdb)
```

# In a nutshell

The ”why”:

- understanding how UNIX works gives you insights in other OS concepts
- system level programming experience is invaluable as it forms the basis for most other programming and even *use* of the system
- system level programming in C helps you understand general programming concepts
- most higher level programming languages (eventually) call (or implement themselves) standard C library functions

# About this class

Textbook:

- "Advanced Programming in the UNIX Environment", by W. Richard Stevens, Stephen A. Rago (2nd Edition)

Help:

- `http://lists.stevens.edu/cgi-bin/mailman/listinfo/cs631apue`
- `https://twitter.com/#!/cs631apue`

Grading:

- 5 homework assignments, worth 20 points each
- 1 midterm project, worth 100 points
- 1 final exam, worth 100 points
- 1 final project, worth 200 points
- no curve

# Syllabus

- 2013-08-26: Introduction, UNIX history, UNIX Programming Basics
- 2013-09-09: File I/O, File Sharing
- 2013-09-16: Files and Directories, Filesystems
- 2013-09-23: System Data Files, Time & Date, Process Environment
- 2013-09-30: Process Control, Signals
- 2013-10-07: Signals
- 2013-10-15: Interprocess Communication
- 2013-10-21: Advanced I/O: Nonblocking I/O, Polling, and Record Locking
- 2013-10-28: Daemon Processes, final project discussion
- 2013-11-04: UNIX tools: make(1), gdb(1), revision control, etc.
- 2013-11-11: Encryption
- 2013-11-18 – 2013-11-25: Code reading and discussions
- 2013-12-02: Final Exam

# linux-lab.cs.stevens.edu

http://tarantula.phy.stevens.edu/wiki/index.php/Linux_Lab

# UNIX History

# UNIX history

`http://www.unix.org/what_is_unix/history_timeline.html`

`http://www.levenez.com/unix/`

- Originally developed in 1969 at Bell Labs by Ken Thompson and Dennis Ritchie.
- 1973, Rewritten in C. This made it portable and changed the history of OS
- 1974: Thompson, Joy, Haley and students at Berkeley develop the Berkeley Software Distribution (BSD) of UNIX
- two main directions emerge: BSD and what was to become “System V”

# UNIX history

- 1984 4.2BSD released (TCP/IP)
- 1986 4.3BSD released (NFS)
- 1991 Linus Torvalds starts working on the Linux kernel
- 1993 Following the settlement of USL vs. BSDi: NetBSD, then FreeBSD are created
- 1994 Single UNIX Specification introduced
- 1995 4.4BSD-Lite Release 2 (last CSRG release); OpenBSD forked off NetBSD
- 2000 Darwin created (derived from NeXT, FreeBSD, NetBSD)
- ...

Some of todays main Unix versions:

- mostly BSD: *BSD, Linux, Mac OS X
- mostly SysV: Solaris, HP-UX, IRIX

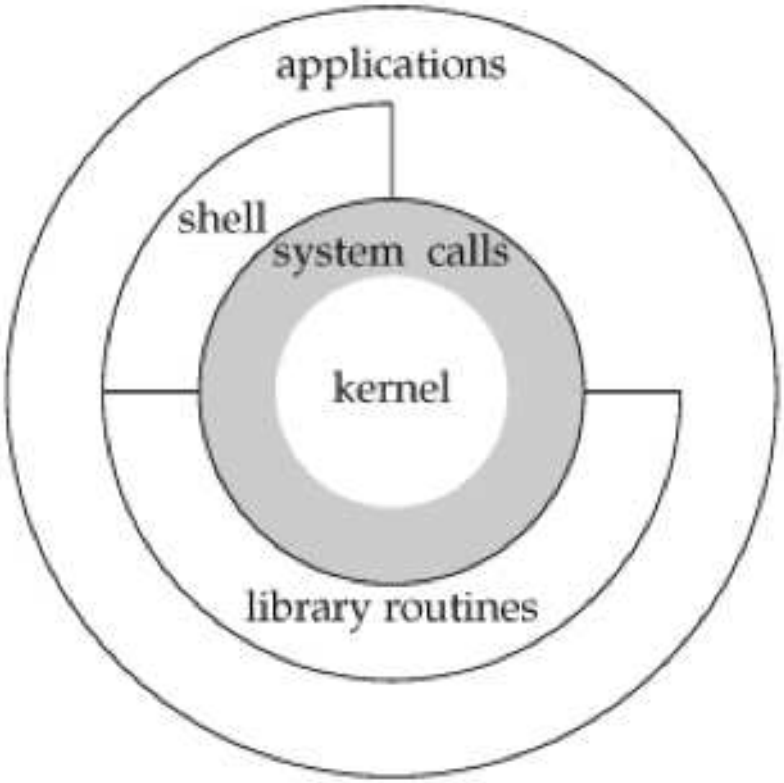
# Some UNIX versions

## More UNIX (some generic, some trademark, some just unix-like):

| | | | | |
|---|---|---|---|---|
| 1BSD | 2BSD | 3BSD | 4BSD | 4.4BSD Lite 1 |
| 4.4BSD Lite 2 | 386 BSD | A/UX | Acorn RISC iX | AIX |
| AIX PS/2 | AIX/370 | AIX/6000 | AIX/ESA | AIX/RT |
| AMiX | AOS Lite | AOS Reno | ArchBSD | ASV |
| Atari Unix | BOS | BRL Unix | BSD Net/1 | BSD Net/2 |
| BSD/386 | BSD/OS | CB Unix | Chorus | Chorus/MiX |
| Coherent | CTIX | Darwin | Debian GNU/Hurd | DEC OSF/1 ACP |
| Digital Unix | DragonFly BSD | Dynix | Dynix/ptx | ekkoBSD |
| FreeBSD | GNU | GNU-Darwin | HPBSD | HP-UX |
| HP-UX BLS | IBM AOS | IBM IX/370 | Interactive 386/ix | Interactive IS |
| IRIX | Linux | Lites | LSX | Mac OS X |
| Mac OS X Server | Mach | MERT | MicroBSD | Mini Unix |
| Minix | Minix-VMD | MIPS OS | MirBSD | Mk Linux |
| Monterey | more/BSD | mt Xinu | MVS/ESA OpenEdition | NetBSD |
| NeXTSTEP | NonStop-UX | Open Desktop | Open UNIX | OpenBSD |
| OpenServer | OPENSTEP | OS/390 OpenEdition | OS/390 Unix | OSF/1 |
| PC/IX | Plan 9 | PWB | PWB/UNIX | QNX |
| QNX RTOS | QNX/Neutrino | QUNIX | ReliantUnix | Rhapsody |
| RISC iX | RT | SCO UNIX | SCO UnixWare | SCO Xenix |
| SCO Xenix System V/386 | Security-Enhanced Linux | Sinix | Sinix ReliantUnix | Solaris |
| SPIX | SunOS | Tru64 Unix | Trusted IRIX/B | Trusted Solaris |
| Trusted Xenix | TS | UCLA Locus | UCLA Secure Unix | Ultrix |
| Ultrix 32M | Ultrix-11 | Unicos | Unicos/mk | Unicox-max |
| UNICS | UNIX 32V | UNIX Interactive | UNIX System III | UNIX System IV |
| UNIX System V | UNIX System V Release 2 | UNIX System V Release 3 | UNIX System V Release 4 | UNIX System V/286 |
| UNIX System V/386 | UNIX Time-Sharing System | UnixWare | UNSW | USG |
| Venix | Wollogong | Xenix OS | Xinu | xMach |

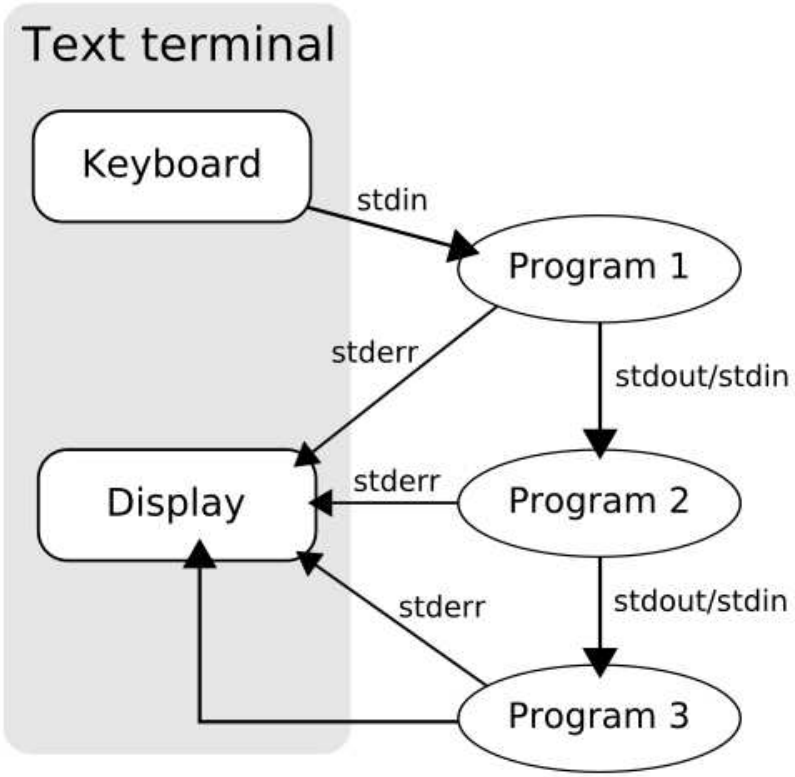
# UNIX Basics

# UNIX Basics: Architecture



# UNIX Basics: Pipelines

Say ”Thank you, Douglas McIlroy!”




http://is.gd/vGH09J

# Program Design

”Consistency underlies all principles of quality.”
Frederick P. Brooks, Jr

# Program Design

`https://secure.wikimedia.org/wikipedia/en/wiki/Unix_philosophy`

UNIX programs...

- ...are simple
- ...have a manual page
- ...follow the element of least surprise
- ...accept input from `stdin`
- ...generate output to `stdout`
- ...generate meaningful error messages to `stderr`
- ...have meaningful exit codes

# Boot/Login process

```
[...]
total memory = 768 MB
avail memory = 732 MB
timecounter: Timecounters tick every 10.000 msec
mainbus0 (root)
[...]
boot device: xbd3
root on xbd3a dumps on xbd3b
mountroot: trying lfs...
mountroot: trying ffs...
root file system type: ffs
init: copying out path ‘/sbin/init’ 11
[...]
Starting local daemons:.
Starting sendmail.
Starting sshd.
Starting snmpd.
Starting cron.

NetBSD/amd64 (panix.netmeister.org) (console)

login:
```

# Boot/Login process

```
[...]
total memory = 768 MB
avail memory = 732 MB
timecounter: Timecounters tick every 10.000 msec
mainbus0 (root)
[...]
boot device: xbd3
root on xbd3a dumps on xbd3b
mountroot: trying lfs...
mountroot: trying ffs...
root file system type: ffs
init: copying out path ‘/sbin/init’ 11
[...]
Starting local daemons:.
Starting sendmail.
Starting sshd.
Starting snmpd.
Starting cron.

NetBSD/amd64 (panix.netmeister.org) (console)

login: jschauma
Password:
```

# Boot/Login process

```
[...]
total memory = 768 MB
avail memory = 732 MB
timecounter: Timecounters tick every 10.000 msec
mainbus0 (root)
[...]
boot device: xbd3
root on xbd3a dumps on xbd3b
mountroot: trying lfs...
mountroot: trying ffs...
root file system type: ffs
init: copying out path '/sbin/init' 11
[...]
Starting local daemons:.
Starting sendmail.
Starting sshd.
Starting snmpd.
Starting cron.

NetBSD/amd64 (panix.netmeister.org) (console)

login: jschauma
Password:
Last login: Sat Sep 10 14:27:56 2011 on console
Copyright (c) 1982, 1986, 1989, 1991, 1993
    The Regents of the University of California.  All rights reserved.

NetBSD 5.0.2 (PANIX-VC) #2: Tue Oct 19 16:30:57 EDT 2010

Welcome to NetBSD!

$
```

# Soooo... what exactly is a ”shell”?

```
$ wget http://www.cs.stevens.edu/~jschauma/631/simple-shell.c
$ more simple-shell.c
$ cc -Wall -o mysh simple-shell.c
$ ./mysh
$$ /bin/ls
[...]
$$ ^D
$
```

# Files and Directories

- The UNIX **filesystem** is a tree structure, with all partitions mounted under the root (/). File names may consist of any character except / and NUL as pathnames are a sequence of zero or more filenames separated by /'s.

# Files and Directories

- The UNIX **filesystem** is a tree structure, with all partitions mounted under the root (/). File names may consist of any character except / and NUL as pathnames are a sequence of zero or more filenames separated by /'s.
- Directories are special "files" that contain mappings between *inodes* and *filenames*, called directory entries.

# Files and Directories

- The UNIX **filesystem** is a tree structure, with all partitions mounted under the root (/). File names may consist of any character except / and NUL as pathnames are a sequence of zero or more filenames separated by /'s.
- Directories are special "files" that contain mappings between *inodes* and *filenames*, called directory entries.
- All processes have a current working directory from which all relative paths are specified. (Absolute paths begin with a slash, relative paths do not.)

# Listing files in a directory

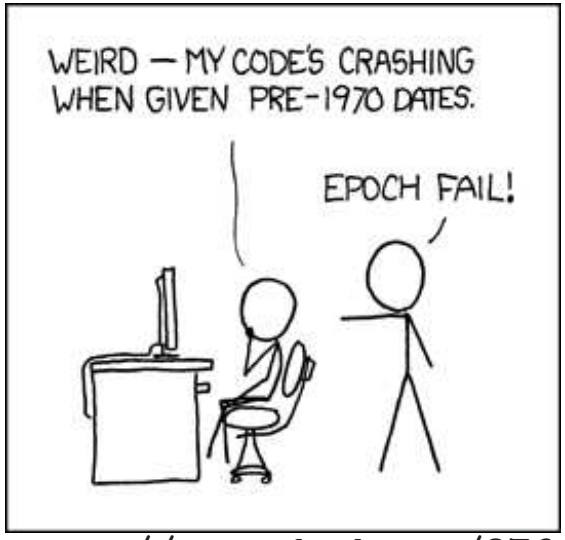
```
$ wget http://www.cs.stevens.edu/~jschauma/631/simple-ls.c
$ more simple-ls.c
$ cc -Wall -o myls simple-ls.c
$ ./myls .
[...]
$
```

# User Identification

- *User ID*s and *group ID*s are numeric values used to identify users on the system and grant permissions appropriate to them.
- *Group ID*s come in two types; *primary* and *secondary*.

# Unix Time Values

*Calendar time*: measured in seconds since the UNIX epoch (Jan 1, 00:00:00, 1970, GMT). Stored in a variable of type `time_t`.




`https://www.xkcd.com/376/`

`https://secure.wikimedia.org/wikipedia/en/wiki/Year_2038_problem`

# Unix Time Values

*Process time*: central processor resources used by a process. Measured in *clock ticks* (`clock_t`). Three values:

- clock time
- user CPU time
- system CPU time

```
$ time grep -r _POSIX_SOURCE /usr/include >/dev/null
```

# System Calls and Library Functions, Standards

## System Calls and Library Functions

- *System calls* are entry points into kernel code where their functions are implemented. Documented in section 2 of the manual (e.g. `write(2)`).
- *Library calls* are transfers to user code which performs the desired functions. Documented in section 3 of the manual (e.g. `printf(3)`).

## Standards

- ANSI C (X3.159-1989) C89, C9X/C99 (ISO/IEC 9899), C11 (ISO/IEC 9899:2011)
- IEEE POSIX (1003.1-2008) / SUSv4

# Standard I/O

- Standard I/O:
  - **file descriptors**: Small, non-negative integers which identify a file to the kernel. The shell can redirect any file descriptor.
  - kernel provides **unbuffered** I/O through e.g. `open read write lseek close`
  - kernel provides **buffered** I/O through e.g. `getc putc fopen fread fwrite`

# Standard I/O: cat(1)

```
$ for f in cat cat2; do
    wget http://www.cs.stevens.edu/~jschauma/631/simple-${f}.c
    cc -Wall -o my${f} simple-${f}.c
    ./my${f} <simple-${f}.c
done
[...]
$
```

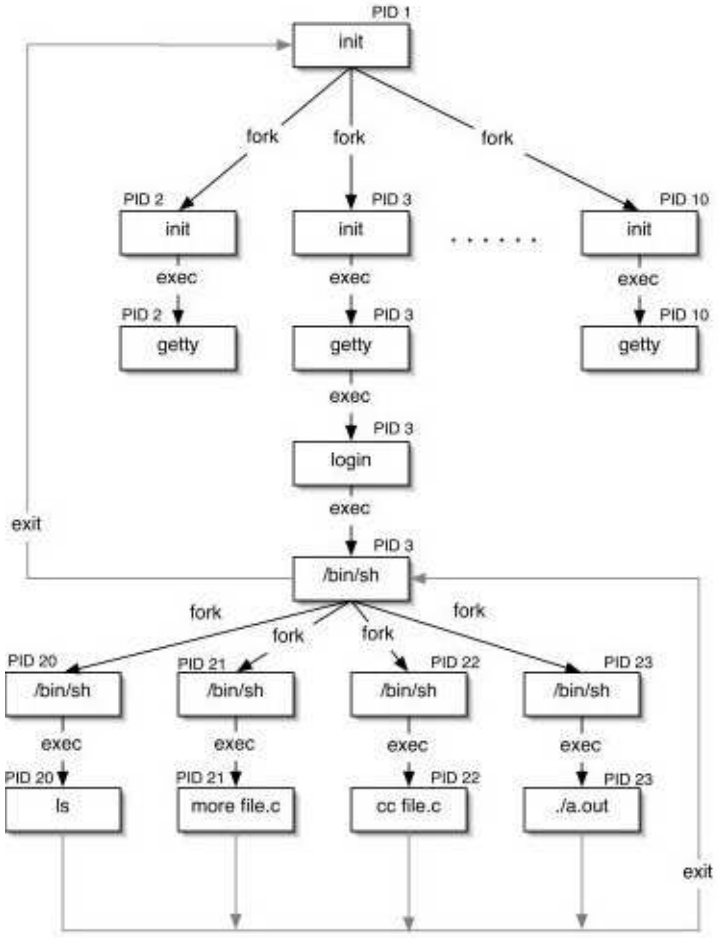
# Processes

Programs executing in memory are called *processes*.

- Programs are brought into memory via one of the six `exec(3)` functions. Each process is identified by a guaranteed unique non-negative integer called the *processes ID*. New processes can **only** be created via the `fork(2)` system call.
- **process control** is performed mainly by the `fork(2)`, `exec(3)` and `waitpid(2)` functions.

```
$ wget http://www.cs.stevens.edu/~jschauma/631/pid.c
$ more pid.c
$ cc -Wall -o mypid pid.c
$ ./mypid .
[...]
$ echo $$
[...]
```

# Processes


PID 1
init
fork
fork
fork
PID 2
init
PID 3
init
PID 10
init
exec
exec
exec
PID 2
getty
PID 3
getty
PID 10
getty
exec
PID 3
login
exec
exit
PID 3
/bin/sh
fork
fork
fork
fork
PID 20
/bin/sh
PID 21
/bin/sh
PID 22
/bin/sh
PID 23
/bin/sh
exec
exec
exec
exec
PID 20
ls
PID 21
more file.c
PID 22
cc file.c
PID 23
./a.out
exit

# Processes

```
$ pstree -hapun | more
```

# Signals

- **Signals** notify a *process* that a condition has occured. Signals may be
  - ignored
  - allowed to cause the default action
  - caught and control transferred to a user defined function

```
$ wget http://www.cs.stevens.edu/~jschauma/631/simple-shell2.c
$ more simple-shell2.c
$ cc -Wall -o mysh simple-shell2.c
$ ./mysh
$$ /bin/ls
[...]
$$ ^C
Caught SIGINT!
```

# Important ANSI C Features, Error Handling

- Important ANSI C Features:
  - function prototypes
  - generic pointers (`void *`)
  - abstract data types (e.g. `pid_t`, `size_t`)
- Error Handling:
  - meaningful return values
  - `errno` variable
  - look up constant error values via two functions:

```
#include <string.h>
char *strerror(int errnum)
```

Returns: pointer to message string

```
#include <stdio.h>
void perror(const char *msg)
```

# Homework

- read `intro(2)`, Stevens 1 & 2
- follow, test and understand all examples from this lecture
- sign up on the course mailing list (using a Stevens email address)
- bookmark these websites:
  - `http://www.cs.stevens.edu/~jschauma/631/`
  - `http://pubs.opengroup.org/onlinepubs/9699919799/`
- ensure you have an account on `linux-lab.cs.stevens.edu`; see `http://tarantula.phy.stevens.edu/wiki/index.php/Linux_Lab`

- `http://www.cs.stevens.edu/~jschauma/631/f13-intro.html`
- `http://www.cs.stevens.edu/~jschauma/631/f13-hw1.html`