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/
New Rules

Close your laptops!
New Rules

Close your laptops!

Open your eyes!

(Mind, too.)
About this class

The class is called “Advanced Programming in the UNIX Environment”.

It is *not*:

- an introduction to Unix
- an introduction to programming
- an introduction to C
What is it?

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

See also:

http://www.cs.stevens.edu/~jschauma/631/#source-code
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 "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
In a nutshell: the "how"

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

setprogmname(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"

Open your laptops!
Ignore Facebook, Twitter, Email, Tinder, ...

Now compile and run this program:

http://www.cs.stevens.edu/~jschauma/631/welcome.c
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"

```bash
$ $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)
If programming languages were weapons

C is an M1 Garand standard issue rifle, old but reliable.

https://i.imgur.com/WnpMFjX.jpg
https://i.imgur.com/ZyeCO.jpg
About this class

Textbook:

Help:
- http://lists.stevens.edu/cgi-bin/mailman/listinfo/cs631apue
- IRC: #cs631apue on Freenode
- https://twitter.com/#!/cs631apue

Grading:
- 4 homework assignments, worth 25 points each
- 1 midterm project, worth 100 points
- 1 final project (group work), worth 200 points
- 1 final programming assignment (individual), worth 100 points
- no curve
Syllabus

- Introduction, UNIX history, UNIX Programming Basics
- File I/O, File Sharing
- Files and Directories
- Filesystems, System Data Files, Time & Date
- UNIX tools: make(1), gdb(1), revision control, etc.
- Process Environment, Process Control
- Process Groups, Sessions, Signals
- Interprocess Communication
- Daemon Processes, shared libraries
- Advanced I/O: Nonblocking I/O, Polling, and Record Locking
- Encryption
- Code reading, coding style, best practices
- Review
UNIX History
UNIX history

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

- 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”
Notable dates in UNIX history

- 1984 4.2BSD released (TCP/IP)
- 1986 4.3BSD released (NFS)
- 1991 Linus Torvalds starts working on the Linux kernel
- 1993 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)
- 2003 Xen; SELinux
- 2005 Hadoop; DTrace; ZFS; Solaris Containers
- 2006 AWS ("Cloud Computing" comes full circle)
- 2007 iOS; KVM appears in Linux
- 2008 Android; Solaris open sourced as OpenSolaris
Some UNIX versions

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

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

unix.png : http://www.levenez.com/unix/
linux.png : http://futurist.se/glbt/
UNIX Everywhere

Today, your desktop, server, cloud, TV, phone, watch, stereo, car navigation system, thermostat, door lock, etc. all run a Unix-like OS...
UNIX Everywhere

Today, your desktop, server, cloud, TV, phone, watch, stereo, car navigation system, thermostat, door lock, etc. all run a Unix-like OS...

...with all the risks that entails.
UNIX Basics
UNIX Basics: Architecture
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


- IEEE POSIX (1003.1-2008) / SUSv4
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:

```c
#include <string.h>
char *strerror(int errnum) Returns: pointer to message string
```

```c
#include <stdio.h>
void perror(const char *msg)
```
UNIX Basics: Pipelines

What is the longest word found on the ten most frequently retrieved English Wikipedia pages?

```
for f in $(curl -L http://is.gd/c6F2fs | zgrep -i '^en ' | sort -k3 -n | tail -10 | sed -e 's/en \([.\-]\) [0-9]* [0-9]*/\1/'); do
  dolinks -dump http://en.wikipedia.org/wiki/${f}
done |
tr '[:punct:]' ' ' |
tr '[:space:]' '\n' |
tr '[:upper:]' '[:lower:]' |
egrep '^[a-z]+$' |
awk '{ print length() " " $0; }' |
sort |
uniq |
sort -n |
tail -1
```
UNIX Basics: Pipelines

Say "Thank you, Douglas McIlroy!"

http://is.gd/vGH09J
"Consistency underlies all principles of quality."
Frederick P. Brooks, Jr
Program Design


UNIX programs...

- ...are simple
- ...follow the element of least surprise
- ...accept input from stdin
- ...generate output to stdout
- ...generate meaningful error messages to stderr
- ...have meaningful exit codes
- ...have a manual page
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
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 IDs** and *group IDs* are numeric values used to identify users on the system and grant permissions appropriate to them.

- *Group IDs* come in two types; *primary* and *secondary*.

```
$ id
```
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`.

```
$ date +%s
```

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

https://en.wikipedia.org/wiki/Year_2038_problem
Unix Time Values

*Process time*: central processor resources used by a process. Measured in *clock ticks* (\texttt{clock\_t}). Three values:

- clock time
- user CPU time
- system CPU time

```
$ time grep -r _POSIX\_SOURCE /usr/include >/dev/null
```
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.** `fopen fread fwrite getc putc`

$ wget http://www.cs.stevens.edu/~jschauma/631/simple-cat.c
$ wget http://www.cs.stevens.edu/~jschauma/631/simple-cat2.c
$ diff -bu simple-cat*.c
[...]
$
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
$ ./mypad

[...]
$ echo $$
[...]
Processes

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

[...]
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
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!

$
Signals

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

```bash
$ 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!
```
Homework

- read intro(2), Stevens 1 & 2
- follow, test and understand all examples from this lecture
- ask questions on the course mailing list and IRC channel
- bookmark these websites:
  - https://www.cs.stevens.edu/~jschauma/631/
  - http://pubs.opengroup.org/onlinepubs/9699919799/
- ensure you have access to a NetBSD system