CS631 - Advanced Programming in the UNIX Environment

Department of Computer Science
Stevens Institute of Technology
Jan Schaumann
jschauma@stevens.edu
https://stevens.netmeister.org/631/
New Rules

Close your laptops!
New Rules

Close your laptops!

Open your eyes!

(Mind, too.)
So far, so good...

What questions do you have?
About this class

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

It is *not* called:

- “An Introduction to Unix”
- “An Introduction to Programming”
- “An introduction to C”
In a nutshell: the "what"

$ ls /bin
[ csh ed ls pwd sleep
cat date expr mkdir rcmd stty
cpio domainname ksh pax rmdir test
cpio doma
chcat date expr mkdir rcmd stty
chnmod df kill mv rm systrace
chcp domainname ksh pax rmdir test
c
cp
cp
$
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
static char dot[] = ".", *dotav[] = { dot, NULL };  
struct winsize win;  
int ch, fts_options;  
int kflag = 0;  
const char *p;  

setprogsuffix(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, "1ABCFLRSTabcdfghiklmnopqrstuwx")) != -1) {  
    switch (ch) {  
        /*  
        * The -1, -C, -1, -m and -x options all override each other so  
        * shell aliasing works correctly.  
        */  
        case '1':  
            f_singlecol = 1;  
    }}
Writing code

We will write a fair bit of code in this class.

Writing code is *communication*.

Make sure your code is

- clearly structured
- well-formatted
- uses a consistent coding style (indentation, placement of braces, etc.)
- variables, functions etc. are sensibly named
- comments are used only when necessary, explaining the *why*, not the *what*

See [https://stevens.netmeister.org/631/style](https://stevens.netmeister.org/631/style).
**In a nutshell: the "how"**

We will write a fair bit of code in this class.

Our reference platform will be NetBSD 8.1:

- install VirtualBox ([https://www.virtualbox.org](https://www.virtualbox.org))
- download NetBSD 8.1 ([https://www.netbsd.org](https://www.netbsd.org))
- set up a new VM
- configure the VM to let you SSH in
- do all your coding in this VM

```
$ ssh apue
apue$ $EDITOR cmd.c
```
In a nutshell: the "how"

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

Now compile and run this program:

```bash
$ ftp https://stevens.netmeister.org/631/welcome.c
$ cc welcome.c
$ ./a.out
$ echo "Hooray!"
Hooray!
$ 
```
In a nutshell: the "how"

```bash
$ $EDITOR welcome.c
$ cc welcome.c
welcome.c: In function 'main':
welcome.c:5:81: warning: implicit declaration of function 'getlogin' [-Wimplicit-function-declaration]
    printf("Welcome to CS631 Advanced Programming in the UNIX Environment, %s!\n", getlogin());

welcome.c:6:1: error: expected ';' before '{'} token
  }
  ^

$`

```
In a nutshell: the "how"

$ $EDITOR welcome.c
$ cc welcome.c
welcome.c: In function 'main':
welcome.c:5:81: warning: implicit declaration of function 'getlogin' [-Wimplicit-function-declaration]
    printf("Welcome to CS631 Advanced Programming in the UNIX Environment, %s!\n", getlogin());
$ ./a.out
Memory fault (core dumped)
$
Programming

https://i.imgur.com/WnpMFjX.jpg

https://i.imgur.com/ZyeC0.jpg
In a nutshell: the "how"


```bash
$ export CFLAGS="-ansi -g -Wall -Werror -Wextra -Wformat=2 -Wjump-misses-init 
    -Wlogical-op -Wpedantic -Wshadow"

$ cc $CFLAGS welcome.c
welcome.c: In function 'main':
  printf("Welcome to CS631 Advanced Programming in the UNIX Environment, %s!\n", getlogin());

welcome.c:5:81: error: implicit declaration of function 'getlogin' [-Werror=implicit-function-declaration]
  printf("Welcome to CS631 Advanced Programming in the UNIX Environment, %s!\n", getlogin());

welcome.c:5:9: error: format '%s' expects argument of type 'char *', but argument 2 has type 'int' [-Werror=format=]
  printf("Welcome to CS631 Advanced Programming in the UNIX Environment, %s!\n", getlogin());

welcome.c:5:9: error: format '%s' expects argument of type 'char *', but argument 2 has type 'int' [-Werror=format=]

welcome.c:4:10: error: unused parameter 'argc' [-Werror=unused-parameter]
  main(int argc, char **argv) {

welcome.c:4:23: error: unused parameter 'argv' [-Werror=unused-parameter]
  main(int argc, char **argv) {

welcome.c:6:1: error: control reaches end of non-void function [-Werror=return-type]
```

cc1: all warnings being treated as errors

$ $EDITOR welcome.c
$ cc $CFLAGS welcome.c
$ ./a.out
Welcome to CS631 Advanced Programming in the UNIX Environment, jschauma!

Welcome to CS631 Advanced Programming in the UNIX Environment, jschauma!
About this class

Textbook:


Grading:

- course participation, course notes: 50 points
- 2 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
- no late submissions
- no extra credit
- no make-up assignments
Course Notes

- create a git repository with a single text file for each lecture
- before each lecture, note:
  - what you read
  - what questions you have
- after each lecture:
  - answers you’ve found, or especially interesting new things you learned
  - what questions remain
  - what new questions arose
  - what additional reading might be relevant
- follow up on unanswered questions in class or on the mailing list
- at the end of the semester, submit all your notes

https://stevens.netmeister.org/631/course-notes.html
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
Hooray!

5 Minute Break
UNIX History
What is it?

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 IRIX/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>UNICOS</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>Venix</td>
<td>Wollongong</td>
<td>Xenix OS</td>
<td>XINU</td>
<td>xMach</td>
</tr>
</tbody>
</table>
UNIX Timeline

unix.png : https://www.levenez.com/unix/
linux.png : https://is.gd/G2zx8f
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:

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

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

Returns: pointer to message string
UNIX Basics: Pipelines

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

for f in $(curl -L https://is.gd/c6F2fs | zgrep -i "^en " | sort -k3 -n | tail -10 | sed -e 's/en \((.*\) [0-9]* [0-9]*/\1/'); do
dolinks -dump https://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!"

https://is.gd/vGH09J
Program Design

"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
Soooo... what exactly is a "shell"?

```bash
$ ftp https://stevens.netmeister.org/631/simple-shell.c
$ more simple-shell.c
$ cc $CFLAGS -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 file system 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

$ ftp https://stevens.netmeister.org/631/simple-ls.c
$ more simple-ls.c
$ cc $CFLAGS -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`.

```bash
$ 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* (*clock_t*). Three values:

- clock time
- user CPU time
- system CPU time

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

$ ftp https://stevens.netmeister.org/631/simple-cat.c
$ ftp https://stevens.netmeister.org/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.

```bash
$ pstree 2>/dev/null || proctree
$ ftp https://stevens.netmeister.org/631/pid.c
$ more pid.c
$ cc $CFLAGS -o mypid pid.c
$ ./mypid .
[...]
$ echo $$
[...]
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

$ ftp https://stevens.netmeister.org/631/simple-shell2.c
$ more simple-shell2.c
$ cc $CFLAGS -o mysh simple-shell2.c
$ ./mysh
$$ /bin/ls
[...]
$$ ^C
Caught SIGINT!
Code requirements

For all code assignments, create a plain ASCII file named checklist. In it, answer the following questions:

- Did I write all the code myself?
- Does my code follow the style guide?
- Does my code compile without warnings or errors on a NetBSD 8.x system?
- Did I provide a Makefile and a README to explain any problems or issues I encountered?
- Did I check the return value of all function calls?
- Did I send error messages to stderr?
- Did I use only meaningful and necessary comments?
- Does my program return zero on success, non-zero otherwise?
- Did I make sure that my .h files only include function forward declarations, macros, etc.?

https://stevens.netmeister.org/631/checklist
Homework

Before every lecture:
- re-read the previous week’s slides and notes
- follow up with questions on the course mailing list or in class
- prepare for class by reading the assigned chapters and slides
- come to class prepared with questions
- update your class notes

After every lecture:
- run all examples from the lecture
- update your class notes
Homework

This week:

- set up your class notes
- read intro(2), Stevens 1 & 2
- bookmark these websites:
  - https://stevens.netmeister.org/631/
  - http://pubs.opengroup.org/onlinepubs/9699919799/
- ensure you are subscribed to the class mailing list:
  - https://lists.stevens.edu/cgi-bin/mailman/listinfo/cs631apue
- ensure you have access to a NetBSD system
  - https://stevens.netmeister.org/631/netbsd.html