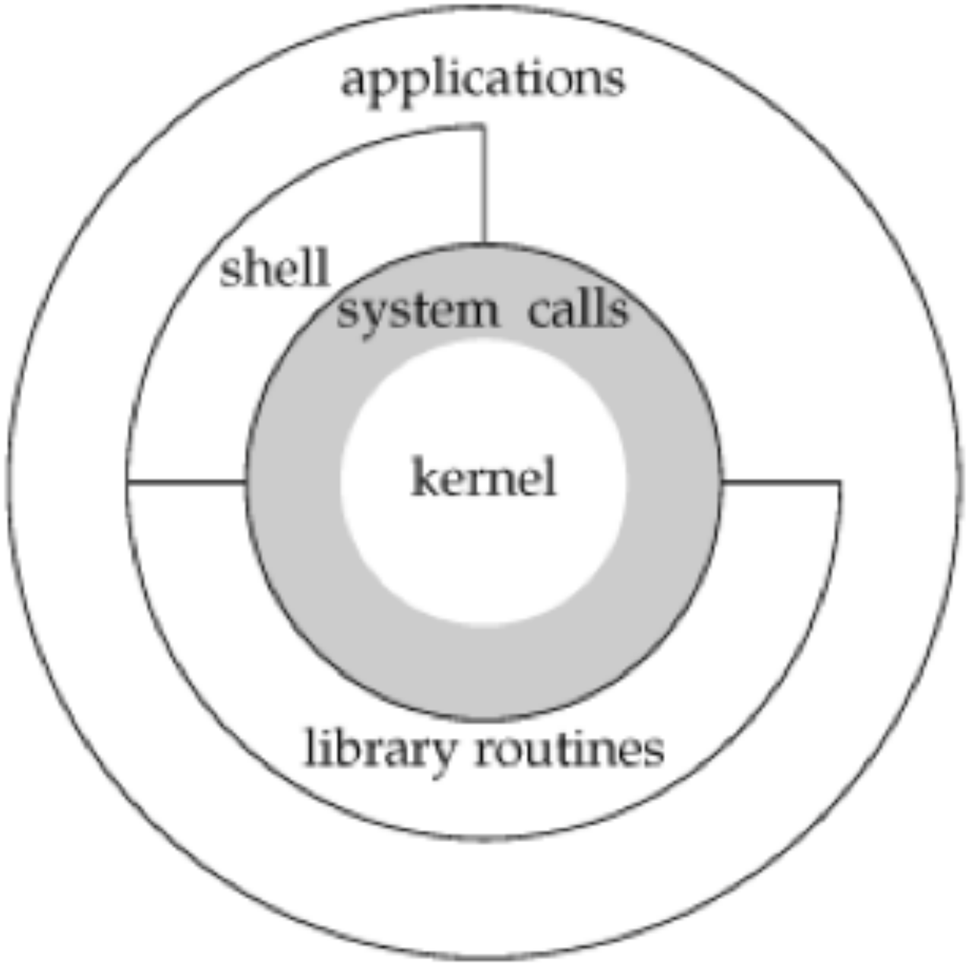
Advanced Programming in the UNIX Environment

Week 01, Segment 3: UNIX Basics

Department of Computer Science
Stevens Institute of Technology

Jan Schaumann
jschauma@stevens.edu
https://stevens.netmeister.org/631/

UNIX Basics: OS Design



System Calls and Library Functions, Standards

System Calls and Library Functions

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

Standards

- IEEE POSIX (1003.1-2008) / SUSv4

System Calls and Library Functions, Standards

```
man printf

man 3 printf

man write

man 3 write

man 2 write

man fprintf
```

System Calls and Library Functions, Standards

https://pubs.opengroup.org/onlinepubs/9799919799/

System Calls and Library Functions, Standards

The C Standard:

- ANSI C (X3.159-1989), aka C89, aka C90; C99 (ISO/IEC 9899); C11 (ISO/IEC 9899:2011); C17 (ISO/IEC 9899:2018); C23 (ISO/IEC 9899:2024), [to be continued…]

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
- lookup of constant error values via two convenient functions:

  `strerror(3)` and `perror(3)`

Let's write some code already!

```
$ ftp https://stevens.netmeister.org/631/apue-code.tar.gz
$ tar zxvf apue-code.tar.gz
$ cd apue-code/01
$ cc welcome.c
$ ./a.out
Welcome to CS631 Advanced Programming in the UNIX Environment, jschauma!
$
```

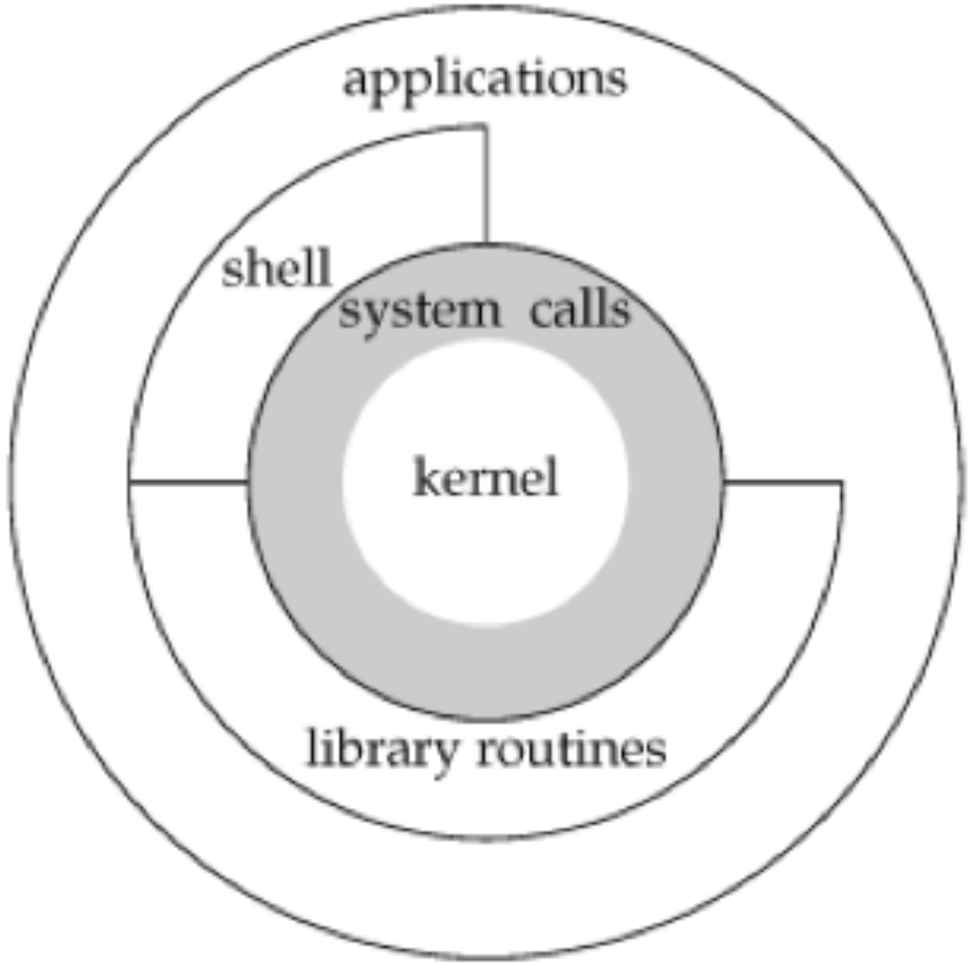
Let's write some code already!

```
$ echo "CFLAGS='-Wall -Werror -Wextra'" >> ~/.shrc
$ echo "alias cc='cc \${CFLAGS}'" >> ~/.shrc
```

See also:

https://kristerw.blogspot.com/2017/09/useful-gcc-warning-options-not-enabled.html

UNIX Basics: OS Design



What exactly is a shell?

```
$ cd 01
$ more simple-shell.c
$ cc -Wall -Werror -Wextra simple-shell.c
$ ./a.out
$$ ls
$$ ls -l
$$ exit
$$ ^D
$
```

Program Design

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

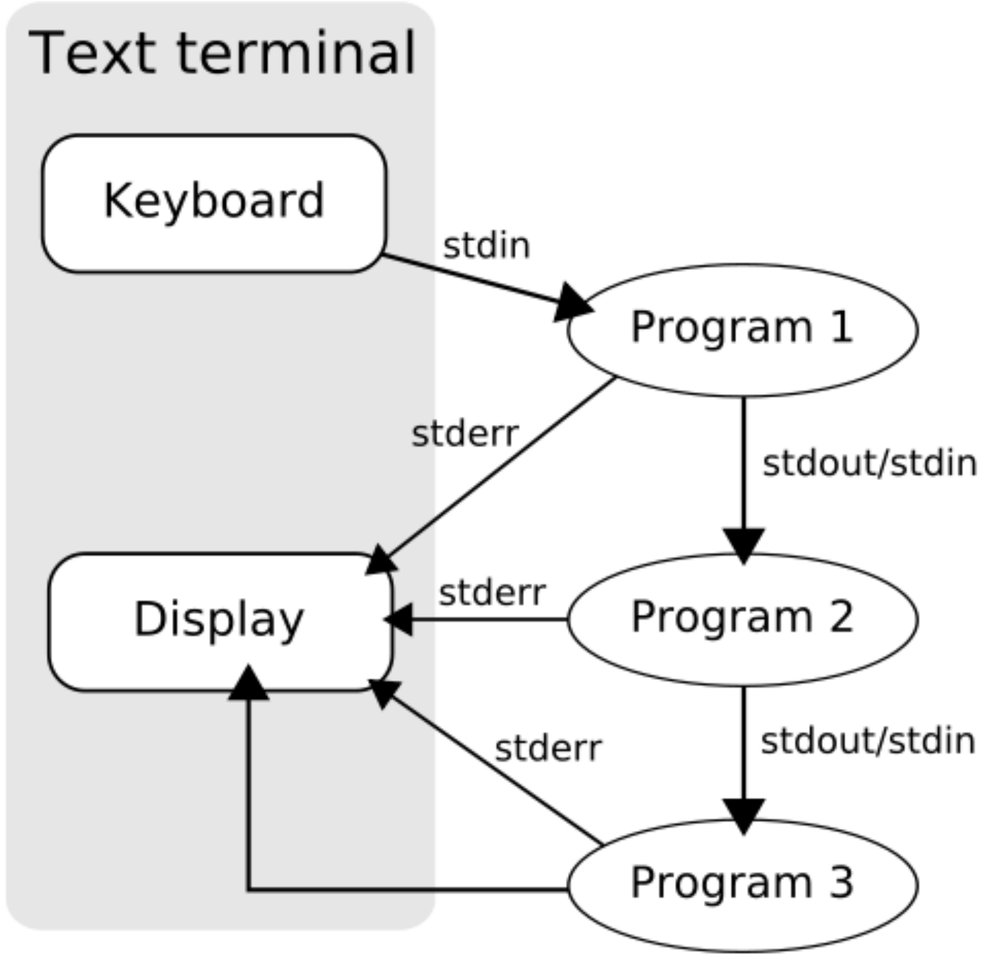
Program Design

https://en.wikipedia.org/wiki/Unix_philosophy

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

UNIX Basics: Pipes



UNIX Basics: Pipes

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

```
$ URL="https://dumps.wikimedia.org/other/pageviews/2024/2024-09/pageviews-20240901-000000.gz"
for f in $(curl -L ${URL} | zgrep -i "^en " | sort -k3 -n | tail -10 |
                    sed -e 's/en \(.*\) [0-9]* [0-9]*/\1/'); do
        links -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
```

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

```
$ cd 01
$ more simple-ls.c
$ cc -Wall -Werror -Wextra simple-ls.c
$ ./a.out .
$
```

User Identification

User IDs and group IDs are numeric values used to identify users on the system and grant permissions appropriate to them.

```
$ whoami
$ id -un
```

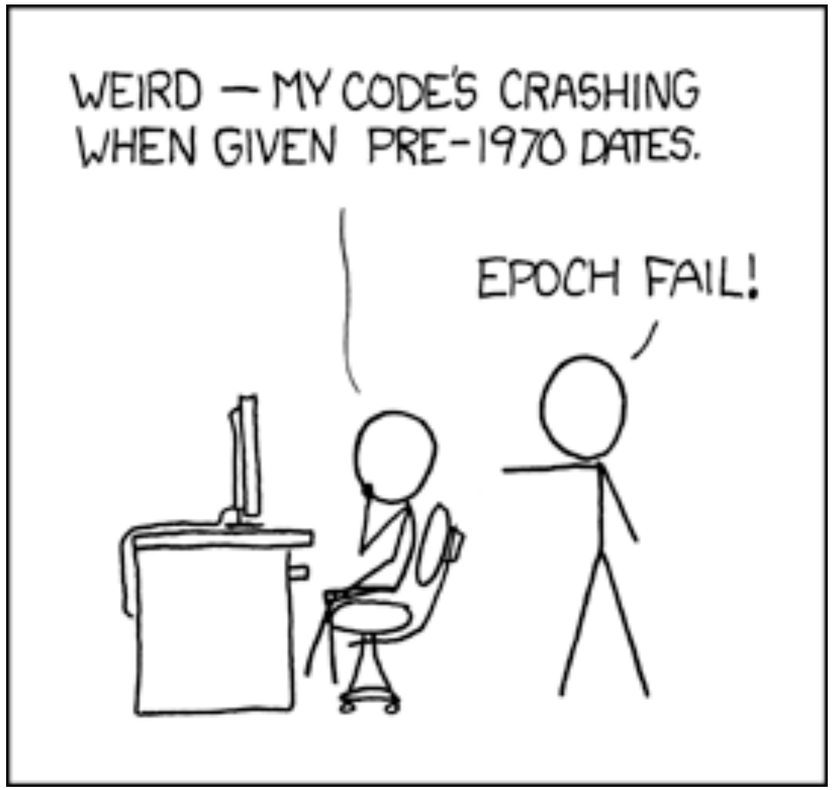
Group IDs come in two types; primary and secondary.

```
$ groups
$ id [-Gn]
```

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
1598535631
```



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

Unix Time Values: Year 2038 Problem

Binary : 01111111 11111111 11111111 11110000

Decimal : 2147483632

Date : 2038-01-19 03:13:52 (UTC)

Date : 2038-01-19 03:13:52 (UTC)

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 find /usr/src/usr.bin -name '*.[ch]' -exec cat {} \; | wc -l
$ time find /usr/src/usr.bin -name '*.[ch]' -print | xargs cat | wc -l
```

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(2), read(2), write(2), lseek(2), close(2)
- kernel provides **buffered** I/O through *e.g.* fopen(3), fread(3), fwrite(3), getc(3), putc(3)

```
$ cc simple-cat.c
$ a.out
$ a.out simple-cat.c
$ a.out <simple-cat.c >/tmp/copy
$ diff -bu simple-cat.c simple-cat2.c
```

Processes

- Programs executing in memory are called processes.
- Programs are brought into memory via one of the `exec(3)` / `execve(2)` 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 via the `fork(2)`, `exec(3)`, and `waitpid(2)` functions.

```
$ proctree
$ echo $$
$ cc -Wall -Werror -Wextra pid.c
$ ./a.out
```

Signals

Signals notify a process that a condition has occurred. Signals may be:

- allowed to cause the default action
- intentionally and explicitly ignored
- caught and control transferred to a user-defined function

```
$ cc -Wall -Werror -Wextra  simple-shell.c
$ ./a.out
^C
$ cc -Wall -Werror -Wextra simple-shell2.c
```

Homework

- make sure your NetBSD VM is set up
- read `intro(2)` and `intro(7)`
- read chapters 1 & 2 in Stevens as well as the linked chapter on UNIX history and basics as you review Week 1
- review and repeat all the code exercises from this lecture segment
- read chapter 3 in Stevens to prepare for Week 2
- join the class Slack and say hi

Next week: File I/O and File Sharing