

CS810D - Advanced Programming in the UNIX Environment

Department of Computer Science
Stevens Institute of Technology
Jan Schaumann

`jschauma@cs.stevens.edu`

`http://www.cs.stevens.edu/~jschauma/810-APUE/`

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, "1ABCFLRSTWabcdghiklmnopqrstuvwxyz")) != -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"
$ $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 needed in almost all IT jobs
- knowing UNIX intimately is needed in almost all IT jobs
- programming in C helps you understand general programming concepts

About this class

Textbook:

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

Grading:

- 2 homework assignments, worth 20 points each
- 4 Quizzes, 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

- 2008-09-02: Introduction, UNIX history, UNIX Programming Basics
- 2008-09-09: File I/O, File Sharing
- 2008-09-16: Files and Directories, Filesystems
- 2008-09-23: System Data Files, Time & Date, Process Environment
- 2008-09-30: Process Control, Signals
- 2008-10-07: Signals
- 2008-10-14: Midterm Project due; no class (Monday Class Schedule)
- 2008-10-21: Interprocess Communication
- 2008-10-28: Advanced I/O: Nonblocking I/O, Polling, and Record Locking
- 2008-11-04: Daemon Processes, final project discussion
- 2008-11-11: UNIX tools: make(1), cvs(1), gdb(1), etc.
- 2008-11-18 – 2005-11-25: *TBD*
- 2008-12-02: Final Exam
- 2008-12-16: Final Project due

UNIX History

UNIX history

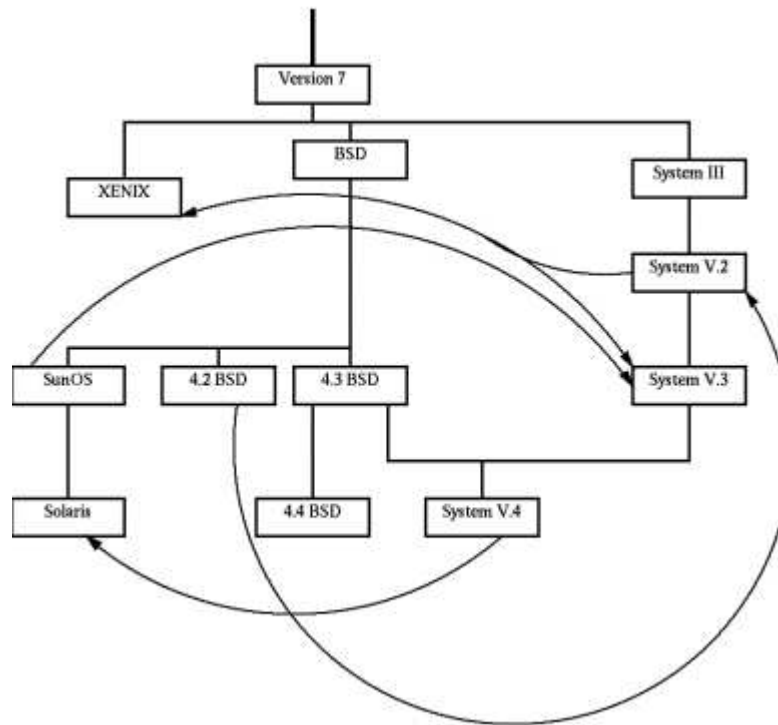
- Originally developed in 1969 at Bell Labs by Ken Thompson and Dennis Ritchie.
- Underlying philosophy: Write programs that do one thing and do it well. Write programs to work together.



- 1974: Thompson, Joy, Haley and students at Berkeley develop the Berkeley Software Distribution (BSD) of UNIX

UNIX history

- 1978: UNIX Version 7 released
- two main directions emerge: BSD and what was to become “System V”



UNIX history: legal issues

- BSD re-written almost entirely from scratch to allow redistribution without ATT license as Networking Release 2
- USL sues BSDI and UC Berkely
- Settlement leads to 4.4BSD-Lite

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

The basics: login in, files and directories

- `login in`: name and password looked up in the system password file (`/etc/passwd`). Entries appear as:
`login_name:password:uid:gid:comment:home_dir:shell`
Passwords entered in the login program are encrypted and compared against the password field in the system password file.
- next: a `shell` (command interpreter) loads

The basics: login in, files and directories

- **login in:** name and password looked up in the system password file (/etc/passwd). Entries appear as:
`login_name:password:uid:gid:comment:home_dir:shell`
Passwords entered in the login program are encrypted and compared against the password field in the system password file.
- **next:** a shell (command interpreter) loads
- **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.

The basics: login in, files and directories

- **login in:** name and password looked up in the system password file (/etc/passwd). Entries appear as:
`login_name:password:uid:gid:comment:home_dir:shell`
Passwords entered in the login program are encrypted and compared against the password field in the system password file.
- **next:** a shell (command interpreter) loads
- **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.
All processes have a current working directory from which all relative paths are specified. (Absolute paths begin with a slash, relative paths do not.)

The basics: User Identification, Unix Time Values

- *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*.

The basics: User Identification, Unix Time Values

- *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*.

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

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)
- IEEE POSIX (1003.1-2001) and SUSv3 (ISO/IEC 9945-2003)
- X/Open XPG3 and FIPS

The basics: Standard I/O, Processes, Signals

- 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`

The basics: Standard I/O, Processes, Signals

- 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`
- Programs executing in memory are called *processes*.
 - Programs are brought into memory via one of the six `exec` 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` system call.
 - process control is performed mainly by the `fork`, `exec` and `waitpid` functions.

The basics: Standard I/O, Processes, Signals

- 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`
- Programs executing in memory are called *processes*.
 - Programs are brought into memory via one of the six `exec` 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` system call.
 - process control is performed mainly by the `fork`, `exec` and `waitpid` functions.
- 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

The basics: Important ANSI C Features, Error Handling

- Important ANSI C Features:

- Function Prototypes
- Generic Pointers (`void *`)
- System 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)
- visit and bookmark these websites:
 - <http://www.cs.stevens.edu/~jschauma/810-APUE/>
 - http://www.unix-systems.org/single_unix_specification/
- generate an ssh-key pair and send me the public key