SYSTEM PROGRAMMING

From the book by STEWART WEISS

Chapter 01

Introduction to System Programming

Concepts Covered

- The kernel and kernel API,
- System calls and libraries,
- Processes, logins and shells,
- Environments, man pages,
- Users, the root, and groups,
- Authentication,
- File system, file hierarchy,
- Files and directories,
- Device special files,
- UNIX standards, POSIX,
- System programming,
- Terminals and ANSI escape sequences,
- *syscall, getpid, ioctl*

Modern Computer Systems

- Multiuser Environment
- Multithreading
- Multiple Software

Operating Systems

Managing System Resources:

- **Disk:** Files, Directories, Links, etc.
- **Memory:** Paging, Segmenting, Virtualization
- **CPU:** Processes, Multithreading
- **Network:** HTTP, NFS, SSH, TCP/IP, etc.
- **Screen, Keyboard, Mice, Printers, etc.**

Cornerstones of UNIX

- Files and File Hierarchy
- Processes
- Users and Groups
- Privileged and Non-Privileged Instructions
- Environments
- Shells
- Documentation: Man Pages, Texinfo

The UNIX Kernel:

- Defines the **A**pplication **P**rogramming **I**nterface
- Provides all of UNIX's services

The UNIX Kernel

- A program can make requests for services:
  - By making a system call to a function built directly into the kernel
  - By calling a higher-level library routine that makes use of this call

The UNIX Kernel API: **System Resources**

- Process scheduling and management
- I/O handling
- Physical and virtual memory management
- Device management
- File management
- Signaling and inter-process communication
- Multi-threading
- Multi-tasking
- Real-time signaling and scheduling
- Networking services.

The UNIX Kernel API: **System Calls**

- System Calls => The Kernel
- System Calls + System Libraries + System Utilities => The Kernel API

```
printf("Thread id %ld \n", syscall(SYS_gettid));
```

The UNIX Kernel API: **System Libraries**

- UNIX designed to keep the kernel as small as possible
- Single kernel function performs input operations
  - *read* operation reads large blocks of data from a device to system buffers
  - No other version in kernel
- The API enriched with an extensive set of higher-level routines kept in system libraries.

UNIX and Related Standards: **The Problem**

- Dozens of different UNIX distributions, each with its own different behavior.
- Standards have been developed in order to define UNIX.
- OSs branded as conforming to one standard or another.

UNIX and Related Standards: **Solution**

- **POSIX:** Portable Operating System Interface, IEEE 1003, ISO/IEC 9945:2003
- Standardizes all **system calls**, **system libraries**, and **utility programs** such as *grep, awk*, and *sed*
- POSIX standard relies on C standard, since the API is coded in C

Learning System Programming by Example

Pick an existing program that uses the API, e.g. a shell command:

- Using man pages, investigate system calls and kernel data structures this program uses in its implementation

- Write a new version of the program, iteratively improve it until it behaves like the actual command

The UNIX File Hierarchy

Directory	Purpose
bin	All essential binary executables
boot	Static files of the boot loader
dev	Essential device files
etc	Host conguration files. Something like the registry of the Windows.
home	All user home directories
lib	Essential shared libraries and kernel modules
media	Mount point for removable media
mnt	Mount point for mounting a file system temporarily
opt	Add-on application software packages
sbin	Essential system binaries
srv	Data for services provided by this system
tmp	Temporary files
usr	Non-essential binaries, libraries, and sources: **/usr/bin & /usr/sbin**: containing binaries, **/usr/lib**, containing library files, and **/usr/local**: "local" programs and data
var	Variable les (les containing data that can change)

Pathnames and Directories

- Wildcards
  - Root Directory : **/**
  - Current Directory : **.**
  - Parent Directory : **..**
  - User Home Directory: **~**
- Commands
  - *pwd, ls, cd, mkdir, rmdir, rm, mv*

Files and Filenames

- Regular files
- Device files (character or block)
- FIFOs
- Sockets
- Symbolic links

Working with Files

Command	
cp	Copy File(s)
ln	Create Link
rm	Remove
mv	Move / Rename
cat	Print screen file(s)
more, less, pg	Print page by page
head	Print first n lines
tail	Print last n lines

File Attributes: **Users and Groups**

- The set of all users partitioned into: **user, group, others**

- File owner: **user**.
- User group associated with file: **group**.
- Anyone else other than the **user** and not member of the **group** called **others**

- *ls –l* shows files with attributes
- *stat* shows provides even more
- *chown user1:group1 file1*
- *chown -R user1:group1 dir1*
- *chgrp group1 file1*
- *touch file1* (updates timestamp)

File Attributes: **Access Modes**

- Read
- Write
- Execute

File Attributes: **Permissions**

- uuu ggg ooo
- wrx wrx wrx

- Example: 110 010 010 = 755 (Octal)

- *chmod 755 file1*
- *chmod –r 755 dir1*
- *chmod +x file1*
- *chmod g+r file1*

File Attributes: Symbolic Links

- *ln -s somefile linkname*

- **Caution:** Circular Reference!

Login Process

- Upon bootup:
  - The kernel initializes the data structures it needs and enables interrupts, and creates init process with pid 1.
- Init process:
  - is the ancestor of all user-level processes in a UNIX system
  - runs with root's privileges
  - monitors the activities of all processes in the outer layers of the OS
  - manages computer shutdown sequence

Login Process

- Init process creates, for each available terminal (i.e., consoles, modems, etc.), a process, **getty**, to listen for activity on that terminal

- **getty** runs the login program, passing it the user-name.

- login prompts the user for the password and tries to validate it.

Login Process

- If the password is valid:
  - login sets the *PWD* to the user's home directory
  - sets the process's user-id to that of the user
  - initializes the user's environment
  - adjusts permissions and ownership of various files
  - starts up the user's login shell.
- If the password is invalid:
  - the login program exits
  - init starts up a new getty for that terminal

Login Process: Network Logins

- *init* creates the process that will listen for the incoming network requests for logins.
- **For SSH as an example:**
- *init* will create a process named *sshd*, the SSH daemon
- *sshd* creates a new process for each remote login
- These processes will create a pseudo-terminal driver, which will spawn the login program
- login process does the above password validation procedure

A First System Program: *more*

- *more* prints file content page by page to standard output and it can be invoked as:

  - *$ more file1 file2 ... fileN*
  - *$ ls -l | more*
  - *$ more < myfile*

A First System Program: **outline**

1. Show P - 1 lines from standard input (last line is for the prompt)
2. Show the [more?] message after the lines.
3. Wait for an input of Enter, Space, or 'q‘
4. If input is Enter, advance one line; go to 2
5. If input is Space, go to 1
6. If input is 'q', e x i t .

A First System Program: *more_v0.1*

```
#include <stdio.h>
#include <stdlib.h>

#define SCREEN_ROWS 23  /*assume 24 lines per screen*/
#define LINELEN 512

int main(int argc,char* argv[]){
	FILE* fp;
	int i = 0;

	if(1 == argc) {
		do_more_of(stdin); //no args, read from standard input
	} else {
		while(++i < argc){
			fp = fopen(argv[i], "r");
			if(NULL != fp){
				do_more_of(fp);
				fclose(fp);
			} else {
				printf("Skipping %s\n", argv[i]);
			}
		}
	}
	return 0;
}
```

```
#define SPACEBAR 1
#define RETURN 2
#define QUIT 3
#define INVALID 4

int get_user_input() {
	int c;

	printf("\033[7m more? \033[m"); /*reverse on a VT100*/
	while((c = getchar()) != EOF) { /* wait for response*/
		switch (c) {
			case 'q': /*'q' pressed*/
				return QUIT;
			case ' ': /*' ' pressed*/
				return SPACEBAR;
			case '\n': /*Enter key pressed*/
				return RETURN;
			default: /*invalid if anything else*/
				return INVALID;
		}
	}
	return INVALID;
}
```

A First System Program: *more_v0.1*

```
void do_more_of(FILE*fp){
	char line[LINELEN];		//buffer to store line of input
	int num_of_lines = SCREEN_ROWS;	//# of lines left on screen
	int getmore = 1;		//boolean to signal when to stop
	int reply;			//input from user

	while(getmore && fgets(line, LINELEN, fp)) { //fgets() returns pointer to string reader NULL
		if(num_of_lines == 0){
			reply = get_user_input(); //reached screen capacity so display prompt
			switch(reply){
				case SPACEBAR:
					num_of_lines = SCREEN_ROWS; //allow full screen
				break;
				case RETURN:
					num_of_lines++; //allow one more line
				break;
				case QUIT:
					getmore = 0;
				break;
				default: //in case of invalid input
				break;
			}
		}

		if(fputs(line, stdout) == EOF)
			exit(1);
		num_of_lines--;
	}
}
```

A First System Program: **Keywords & Functions**

- *FILE*: a file stream
- *fopen* opens a file and returns a *FILE**
- *fclose* closes a FILE stream
- *fgets* reads a string from a *FILE* stream
- *fputs* writes a string to a *FILE* stream

A First System Program: **Arguments to Main**

```
int main(int argc, char *argv[])
```

argc: Argument Count

argv: String Array of arguments

A First System Program: **Problems**

- Displays the first 23 lines
- Pressing space-bar or 'q' has no effect until you press the Enter key.
- the more? prompt is not erased.
- Rediretion from stdin causes problems:
  - We have to get the user's input from the keyboard regardless of the source of the standard input stream

Device Special Files

- Every I/O device (disk, printer, modem, etc.) is associated with a device special file
- Special files can be accessed using the same system calls as regular files
- But the system call activates the device driver for that device rather than causing the direct transfer of data
- A user program just connects to a file variable, which may be associated with a disk file, a display device, a printer, or any other device at run time.
- This is the essence of **device-independent I/O**

Device Special Files: **Examples**

- */dev/tty* current terminal
- */dev/console* display device
- */dev/mem* is a character interface to memory
- */dev/null* discards all data sent to it and returns null characters ('\0') when read
- */dev/rd0a* or */dev/hda* hard disk drive partitions
- */dev/cdrom* the cd-rom drive

Device Special Files: *Echo* to Devices

$ tty

/dev/pts/4

$ echo "hello" > /dev/pts/4

hello

A Second Attempt at the *more* Program

```
void do_more_of( FILE * fp) {
    …
    …
    FILE * fp_tty;
    fp_tty = fopen("/dev/tty", "r");
    if (fp_tty == NULL) // if open fails
        exit(1);
    while (getmore && fgets(line, LINELEN, fp)) {
        // fgets() returns pointer to string read
        if (num_of_lines == 0) {
            // reached screen capacity
            reply = get_user_input(fp_tty);
            switch (reply) {
    …
    …
```

```
int get_user_input(FILE * fp)
…
…
    // Now we use getc instead of get
    char.
    // It is the same except
    // that it requires a FILE* argument
    while ((c = getc(fp)) != EOF)
…
…
```

A Second Attempt at the *more* Program

```
int num_lines;
char *endptr;
char *linestr = getenv("LINES");
if (NULL != linestr) {
    num_lines = strtol(linestr, &endptr, 0);
    if (errno != 0) {
        /* handle error and exit */
    }
    if (endptr == linestr) {
        /* not a number so handle error and exit */
    }
}
```

A Second Attempt at the *more* Program

```
#include <sys/ioctl.h>
struct winsize window_arg;
int num_rows, num_cols;
fp_tty = fopen("/dev/tty", "r");
if (fp_tty == NULL)
        exit(1);
if (-1 == ioctl(fileno(fp_tty), TIOCGWINSZ, &window_arg))
        exit(1);
num_rows = window_arg.ws_row;
num_cols = window_arg.ws_col;
```

Still Problems?

- ‘Enter’ needed for user keyboard intervention
- Percentage not shown
- Duplicate ‘more?’ prompts

Where We Go from Here

- Using only the usual high-level I/O libraries, we cannot write a program versatile as *more*
- The objective is to give you the tools for solving this kind of problem, and to expose you to the major components of the kernel's API, while also explaining how the kernel looks "under the hood "
- We are going to look at each important component of the kernel. You will learn how to rummage around the file system and man pages for the resources that you need.

Thanks…