Systems Programming

Updated : 1-sept-2019

System Calls

Overview

Last Time

- Outback

Readings for today

- Text Chapter 2, 3, 18.8, 15.1
- /class/csce510-001/Code/TLPI

Prologue

- Fundamental Unix/systems programming (Chap 2) continued

Epilogue

- System Calls
- Directories
- Stat system call for information on files

Prepare Your VM

For the .ova, use this instead :

https://drive.google.com/open?id=1d-Ml_DDupAuJhVk-gWuqKydteNJhC5OF

Follow the instruction here

https://projects.ui.ac.id/projects/kuliah-sysprog/wiki/Import_Virtual_Appliance

user : user

pass : sysprog2019

Overview Continued - Chapter 2

- Shell revisited
- 2.6 Programs
- 2.7 Processes
- 2.8 Memory Mappings
- 2.9 Static and Shared Libraries
- 2.10 Interprocess Communication and Synchronization
- 2.11 Signals
- 2.12 - 2.19 Other Topics

Shell revisited

Print prompt
→ Read command
→ Substitutions
→ Fork/Exec
→Wait - get return (exit) status

- Substitutions
  - Wildcards, Filename completion, alias subs., history, cmd substitution
- I/O redirection
  - Filter a program that reads stdin writes stdout
  - ls –l > listing
  - grep Unix | wc

File I/O Model

- One of the distinguishing features of the I/O model on UNIX systems is the concept of universality of I/O.
- This means that the same system calls:
  - open(), read(), write(), close(),
  - used to perform I/ O on all types of files, including devices.
  - File descriptors 0 – stdin, 1 – stdout, 2 – stderr unless remapped
- Stdio Library
  - FILE *fp = fopen(“filename, “w+”)

Programs

- Source files
- .c, .cpp, .y, .py, .rb …

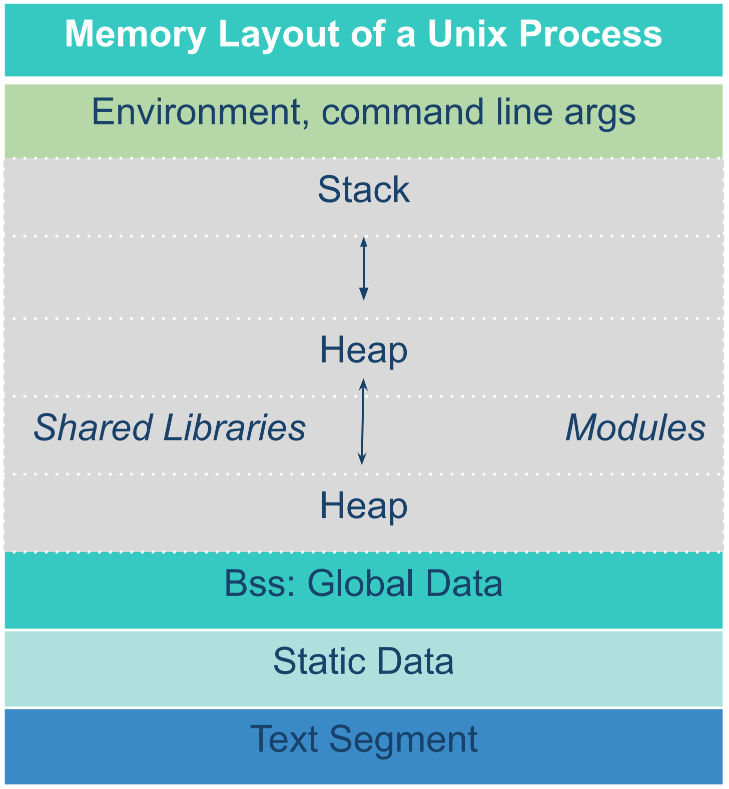
- gcc -c prog.c -lm
- gcc -S prog.c
- gcc prog.c

- Object modules
- Executables
- Ar archives please access http://en.wikipedia.org/wiki/Ar_(Unix)
- Scripts
  - Shell scripts, perl, python, ruby,

Processes

- fork()- creates processes
  - Parent, Child
- Termination : _exit()/ exit(), wait()
- Start : init
- ps, kill, env, export

- Memory Layout



Static and Shared Libraries

- Static Libraries : *.a
  - Static link
- Shared Libraries : *.so
  - Dynamic link

- Try using ldd to know what libraries they use
- What is the difference between static and shared lib? Point out the + and -
- Hunt it on `/usr/lib/`

Interprocess Communication and Synchronization

- Signals : SIGINT, SIGTERM, etc
- Pipes : `ls -al | grep [something]`
- Sockets :
- File locks : apt
- IPC: semaphores, shared memory, msgqueues

/proc (Proc File System)

- "an interface to kernel data structures in a form that looks like files and directories"

- /proc/<pid> - directory of info about running process
  - /proc/1 – information on init

- /proc/cpuinfo
- /proc/sys/fs/file-max

Other Topics from Chapter 2

- 2.12 Threads
- 2.13 Process Groups and Shell Job Control
- 2.14 Sessions, Controlling Terminals, and Controlling Processes
- 2.15 Pseudoterminals
- 2.16 Date and Time
- 2.17 Client-Server Architecture
- 2.18 Realtime

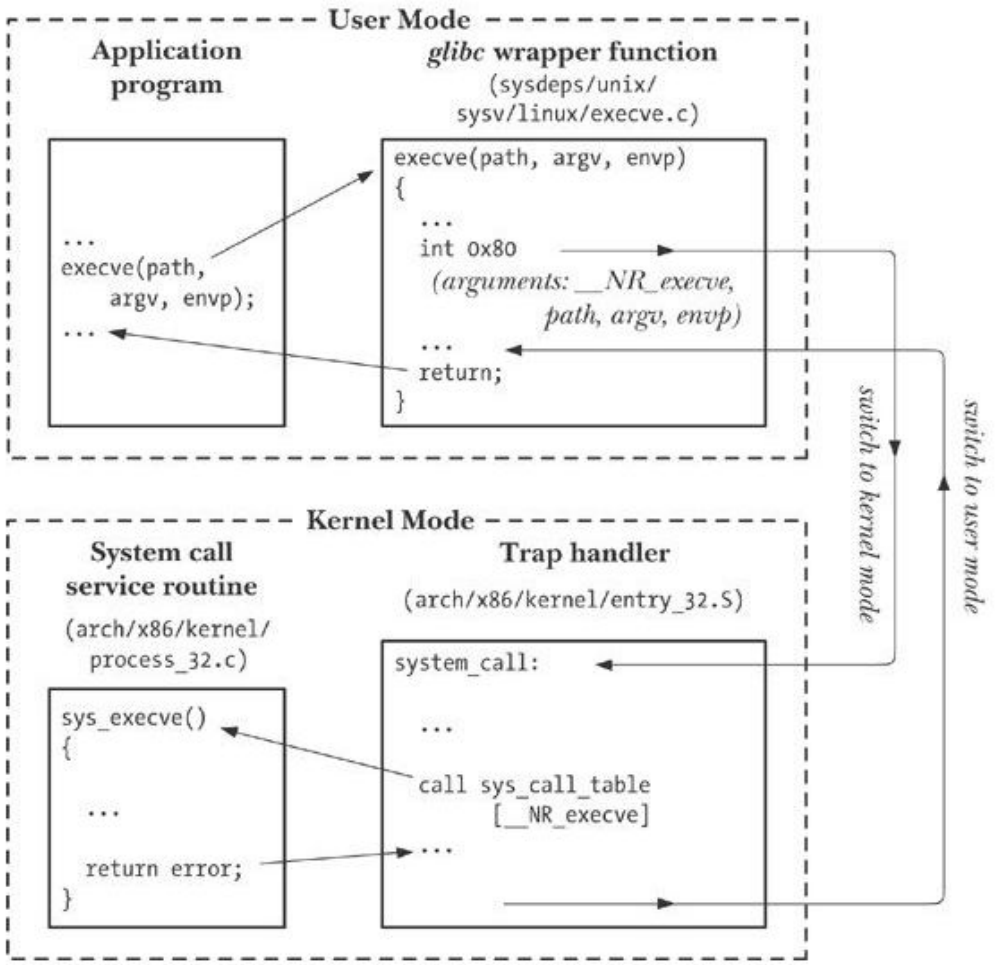
2nd Half

- 3.1 System Calls
- 3.2 Library Functions
- 3.3 The Standard C Library; The GNU C Library (glibc)
- 3.4 Handling Errors from System Calls and Library Functions
- 3.5 Notes on the Example Programs in This Book
- 3.5.1 Command-Line Options and Arguments
- 3.5.2 Common Functions and Header Files
- 3.6 Portability Issues
- 18.8 Directories
- 15.1 Stat system call to retrieve file information

3.1 System Calls

- A controlled entry point into the kernel
- Allows process to request the kernel to do something
- Switches from user mode to kernel mode

Fig 3.1 Execution of System Call



User Mode
Application program
...
execve(path, argv, envp);
...
glibc wrapper function
(sysdeps/unix/sysv/linux/execve.c)
execve(path, argv, envp)
{
...
int 0x80
(arguments: __NR_execve, path, argv, envp)
...
return;
}
switch to kernel mode
switch to user mode
Kernel Mode
System call service routine
(arch/x86/kernel/process_32.c)
sys_execve()
{
...
return error;
}
Trap handler
(arch/x86/kernel/entry_32.S)
system_call:
...
call sys_call_table [__NR_execve]
...

Figure 3-1. Steps in the execution of a system call

Overhead with 10 millions call

getppid() -> 2.2 secs

C func -> 0.11 secs

Errors on System Calls

```
fd = open( pathname, flags, mode); /* system call to open a file */

if (fd == -1) {
   /* Code to handle the error */
}
...
if (close(fd) == -1) {
   /* Code to handle the error */
}
```

- **-1 -> errno -> perror()**

The GNU C Library (glibc)

- http:// www.gnu.org/software/libc/

- Determining the version of glibc on the system
  - `/lib/libc.so.6`

- ldd - list dynamic dependencies

```
#include <gnu/libc-version.h >
const char *gnu_get_libc_version(void);
```

18.8 Reading Directories

- `opendir(), readdir()`

Check soure code at:

https://projects.ui.ac.id/attachments/7247/linux-programming-interface-exercises-master.zip or http://s.id/sysprogtlpi

tlpi-dist/dirs_links/list_files.c

see the file content and go to listFiles function

tlpi-dist/list_files.c

```
DIR *dirp;
struct dirent *dp;
Boolean isCurrent; /* True if 'dirpath' is "." */
isCurrent = strcmp(dirpath, ".") == 0;
dirp = opendir(dirpath);
if (dirp  == NULL) {
    errMsg("opendir failed on '%s'", dirpath);
    return;
}
```

tlpi-dist/list_files.c

```
   /* For each entry in this directory, print directory +
filename */
   for (;;) {
       errno = 0; // To distinguish error from end-of-directory
       dp = readdir(dirp);
       if (dp == NULL)
           break;

       if (strcmp(dp->d_name, ".") == 0 || strcmp(dp->d_name,
"..") == 0)
           continue;               /* Skip . and .. */

       if (!isCurrent)
           printf("%s/", dirpath);
       printf("%s\n", dp->d_name);
   }
```

tlpi-dist/list_files.c

```
if (errno != 0)
    errExit("readdir");
if (closedir(dirp) == -1)
    errMsg("closedir");
```

TLPI Source Code for practice

Please Download the book’s source code:

https://github.com/posborne/linux-programming-interface-exercises

or

https://projects.ui.ac.id/attachments/download/7181/linux-programming-interface-exercises-master.zip

Contents in “tlpi-dist” directory:

```
root> ls
acl                 getopt          pmsg      svmsg
…
daemons   Makefile.inc.MacOSX      pshm      tty README
dirs_links   Makefile.inc.Solaris  pty       users_groups
…
files               pgsjc           sockets
filesys             pipes           svipc
```

NB: Please Compile the source codes using “make”

TLPI/dirs_links

- **`cd dir_links`**
- **`ls`**

```
bad_symlink.c
list_files_readdir_r.c
t_dirbasename.c
file_type_stats.c
Makefile
t_unlink.c
list_files.c
nftw_dir_tree.c
view_symlink.c
```

QA

Stat structure

```
struct stat {
        dev_t     st_dev;       /* ID of device containing file */
        ino_t     st_ino;        /* inode number */
        mode_t    st_mode;       /* protection */
        nlink_t   st_nlink;     /* number of hard links */
        uid_t     st_uid;             /* user ID of owner */
        gid_t     st_gid;             /* group ID of owner */
        dev_t     st_rdev;      /* device ID (if special file) */
        off_t     st_size;            /* total size, in bytes */
        blksize_t st_blksize;   /* blocksize for file system I/O */
        blkcnt_t  st_blocks;    /* number of 512B blocks allocated*/
        time_t    st_atime;     /* time of last access */
        time_t    st_mtime;     /* time of last modification */
        time_t    st_ctime;     /* time of last status change */
      };
```

st_mode macros

S_ISREG(m) is it a regular file?

S_ISDIR(m) directory?

S_ISCHR(m) character device?

S_ISBLK(m) block device?

S_ISFIFO(m) FIFO (named pipe)?

S_ISLNK(m) symbolic link? (Not in POSIX.1-1996.)

S_ISSOCK(m) socket? (Not in POSIX.1-1996.)

st_mode Flags

- ls /usr/include/sys

...

- less   /usr/include/sys/stat.h

... pages of stuff

```
#define S_IRUSR __S_IREAD       /* Read by owner.   */
#define S_IWUSR __S_IWRITE   /* Write by owner.   */
#define S_IXUSR __S_IEXEC       /* Execute by owner.   */
...
#define S_IRGRP (S_IRUSR >> 3)  /* Read by group.   */
```