The Linux Programming Interfaces

陈辉

Contents

1 History and Standards

2 Fundamental Concepts

2.1 The Kernel

2.2 The Shell

2.3 Users and Groups

Users

- *login name*
- *user ID(UID)*
- *Group ID*
- *Home directory*
- *Login shell*:the name of the program to be executed to interpret user commands.

These information of each user resides in *password files*

Groups

Superuser

userID = 0.

2.4 Single Directory Hierarchy, Directories, Links, and Files

Figure of single directory hierarchy.(P71)

FIle Types

The other file types:
devices, pipes, sockets, directories, and symbolic links.

Directories and links

The links between directories establish the directory hierarchy.

Symbolic links

Pathnames

- *absolute pathname*
- *relative pathname*

2.5 File I/O Model

universality of I/O:The same system calls(*open,read,write,close*) are used to perform I/O on **all types of files**.

File descriptors

A nonnegative integer obtained by a call to *open()*.
Often 0 for input, 1 for output and 2 for errors or other abnormal messages.

2.6 Programs

Filters

Command-line arguments

2.7 Processes

Process memory layout

segments:

- *Text*
- *Data*
- *Heap*
- *Stack*

Process creation and execution

fork():
The kernel creates the child process by making a duplicate of the parent process which inherits copies of parent's **data, stack, heap**. The text is placed in memory marked read-only shared by them.
execve():
child call execve() system calls to replace the origin segments with new target program.

Process ID and parent process ID

Process termination and termination status

child call *_exit()* or be killed by a signal.
parent *wait()* for child's *termination status*.

Process user and group identifiers

- *Real user ID and real group ID*
- *Effective user ID and effective group ID*
- *Supplementary group IDs*

The *init* process

The *init* is the parent of all processes with a constant PID = 1.

Daemon processes

background

Environment list

2.8 Memory Mappings

mmap():

2.9 Static and Shared Libraries

Static libraries

A static library is essentially a structured bundle of compiled object modules. To use functions from it we specify that library in the **link** command used to **build** a program.

Shared libraries

Shared libraries were designed to address the wasting problems with static libraries.

- While building: the linker writes a record into the executable.
- While runtime: *dynamic linker* ensures the required shared libraries are found and loaded to the memory.

2.10 Interprocess Communication(IPC) and Synchronization

The set of mechanisms for interprocess communication(IPC):

- *signals*
- *pipes*
- *sockets*

- *file locking*
- *message queues*
- *semaphores*
- *shared memory*

2.11 Signals

Signals are often described as “software interrupts”

2.12 Threads

Threads **share**

- Text(for program codes).
- Data.
- Heap.
- virtual memory.
- global variables.(for communication)

Threads **differ in**

- Stack.
- local variables.
- function call linkage information(why?).

Threads’ **advantages**

- easy to share data rather than multiprocesses.
- good for parallel processing

2.13 Process Groups and Shell Job Control

2.14 Sessions, Controlling Terminals, and Controlling Processes

2.15 Pseudoterminals

A pseudoterminal is a **pair of connected virtual devices** known as **master and slave**.
Master drives the user program and slave drives terminal-oriented program. This connection is like a bridge. e.g. *telnet and ssh*.

2.16 Date and Time

- *Realtime.*
- *Process time.*

2.17 Client-Server Architecture

2.18 Realtime

POSIX.1b:

2.19 The /proc File System

A virtual file system that provide an **interface to kernel data structures**.

3 System Programming Concepts

3.1 System Calls

(P87)

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

(P95)

3.6 Portability Issues

4 File I/O

Keypoints:

- System call APIs to perform file I/O.
- File descriptor.

4.1 Overview

Key word:*file descriptor*

41 Fundamentals of Shared Libraries

41.1 Object Libraries

(P833) **Object Library**: A set of object files.

41.2 Static Libraries(archives)

Creating and maintaining a static library

Syntax:

```
\$ ar \textit{options archive object-files}
```

Conventional form of static libraries: lib*name*.a.

```
        \$ gcc -g -c mod1.c mod2.c
        \$ ar r libdemo.a mod1.o mod2.o
        \$ rm mod1.o mod2.o
```

Delete modules from the archive:

```
        \$ ar d libdemo.a mod2.o
```

Using a static library

Basic way:

```
\$ gcc -g -c prog.c
\$ gcc -g -o prog prog.o libdemo.a
```

Alternative way:

```
\$ gcc -g -o prog prog.o -ldemo
```

'-ldemo' means '-l' and archive without lib prefix and .a suffix which resides in one of the standard directories(e.g. /usr/lib)

41.3 Overview of Shared Libraries

41.4 Creating and Using Shared Libraries

ELF(Executable and Linking Format shared libraries):ELF is the format employed for executables and shared libraries on modern linux.

41.4.1 Creating a Shared Library

```
\$ gcc -g -c -fPIC -Wall mod1.c mod2.c mod3.c
\$ gcc -g -o -shared -o libfoo.so mod1.o mod2.o mod3.o
```

Remark. Unlike static libraries, object modules cannot be add or deleted from previously built shared library(why?)

41.4.2 Position-Independent Code

-fPIC specifies that the compiler generate position-independent code. How to check whether an object file has been compiled with *-fPIC*(P882)

41.4.3 Using a Shared Library

Two main steps to utilize shared library:

- Embedding the name of the shared library inside the executable during linking.
- Resolving the embedding library name: performed by *dynamic linker*, named **/lib/ld-linux.so.2**.

The LD_LIBRARY_PATH enveironment variable. (P840)

41.4.4 The Shared Library Soname

(P884)

6 Processes

6.1 Processes and Programs

Executable and Linking Format

6.2 Process ID and Parent Process ID

6.3 Memory Layout of a Process

segments:
The memory allocated to each process. Including:

- *text segment*: Machine-language instructions. Read-only, shared.
- *Initialized data segment*: global and static variables explicitly initialized.

- *Uninitialized data segment.*

- *Stack.* One stack frame for each current called function.

- *Heap*: dynamically allocated data.

6.4 Virtual Memory Management

Figure: Typical memory layout of a process.(P163)
pages:
A virtual memory scheme splits the memory used by each program into small fixed-size units called pages.
page table: (P164)
Processes can use *shmget()* and *mmap()* to explicitly request sharing of memory regions with other processes, for the purpose of interprocess communication.