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”