Process and Signal

张海宁

贵州大学

hnzhang1@gzu.edu.cn

May 8, 2018

Overview

1. Process
    - What is a Process
    - Process Structure
    - Starting New Process

2. Appendix

Process

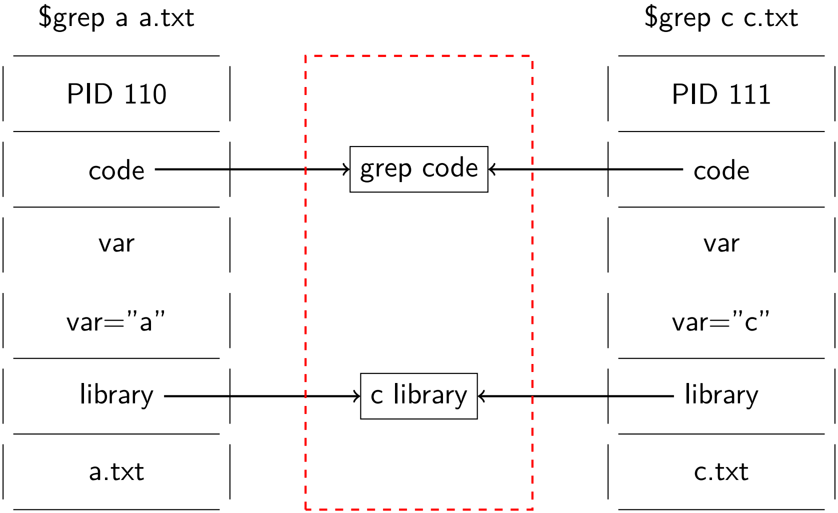
What is a Process

Process

- A process is a program that is running.
- A process is running consists of program code, data, variables (occupying system memory), open files (file descriptors), and an environment.

Typically, a Linux system will share code and system libraries among processes so that there' s only one copy of the code in memory at any one time.

Process Structure



$grep a a.txt

PID 110

code

var

var=”a”

library

a.txt

grep code

c library

$grep c c.txt

PID 111

code

var

var=”c”

library

c.txt

Figure: process structure

分析

Figure 1 显示了操作系统如何安排两个同时运行的 grep 程序（一种可能的状态）。从中我们可以看出：

1. Each process is allocated a unique number, called a process identifier or PID(This is usually a positive integer between 2 and 32,768, The number 0/1 is typically reserved for the special init process, which manages other processes).
2. The code of grep was loaded into memory as read-only and was shared by the two processes. The system library worked the same way as the grep code.
3. A process has its own stack space, used for local variables in functions and for controlling function calls and returns. It also has its own environment space, containing environment variables.

The Process Table

The Linux process table is like a data structure describing all of the processes that are currently loaded with, for example, their PID, status, and command string, the sort of information output by ps.

ps I

The ps utility displays a header line, followed by lines containing information about all of your processes.

list processes

```
$ ps -f -u zhanghaining
UID        PID  PPID  C STIME TTY        TIME CMD
zhangha+ 25020 25015  0 13:18 ?     00:00:00 sshd: zhn@pts/3
zhangha+ 25021 25020  0 13:18 pts/3  00:00:00 -bash
zhangha+ 25201 25021  0 13:45 pts/3  00:00:00 ps -f -u zhn
```

分析

- The PPID field of ps output indicates the parent process ID, the PID of either the process that caused this process to start or, if that process is no longer running, init (PID 1).
- TTY column shows which terminal the process was started from
- TIME gives the CPU time used so far
- the CMD column shows the command used to start the process

ps II

we will show how to view the status of a process.

status of process

```
$ ps ax
  PID TTY      STAT   TIME COMMAND
25020 ?        S      0:00 sshd: zhanghaining@pts/3
25021 pts/3    Ss     0:00 -bash
25109 ?        S      0:00 [kworker/5:2]
25114 ?        S      0:00 [kworker/2:0]
25155 ?        S      0:00 [kworker/5:0]
25179 pts/3    R+     0:00 ps ax
```

ps III

status

STAT Code	Description
S	Sleeping. Usually waiting for an event to occur, such as a signal or input to become available.
R	Running. Strictly speaking, “runnable,” that is, on the run queue either executing or about to run.
Z	Defunct or “zombie” process.
+	Process is in the foreground process group.
<	High priority task.

Table: some status

Starting New Process

system I

We can cause a program to run from inside another program and thereby create a new process by using the system library function.

system

```
#include <stdlib.h>
int system (const char *string);
```

man 3 system

system – pass a command to the shell

system returns 127 if a shell can’t be started to run the command and -1 if another error occurs. Otherwise, system returns the exit code of the command.

system II

use system to write a program to run ps.

system

```
$ cat system1.c
#include<stdlib.h>
#include<stdio.h>

int main(){
 printf("running ps with my program.\n");
 int i = system("ps x");
 //int i = system("ps x &");
 printf("Done. And the return code is:%d.\n",i);
 exit(0);
}
```

sytem III

In general, using system is a far from ideal way to start other processes, because it invokes the desired program using a shell.

1. inefficient

   a shell must be started before the program is stared.

2. between shells

   quite dependent on the installation for the shell and environment that are used.

exec

An exec function replaces the current process with a new process specified by the path or file argument.

The exec functions are more efficient than system because the original program will no longer be running after the new one is started.

exec family

execl, execle, execlp, execv, execvp – execute a file

1. execl, execle,execlp
   参数可变，长度不固定
2. execv, execvp
   参数不可变，长度固定

execl, execle,execlp

```
#include <unistd.h>
int
execl(const char *path, const char *arg0, ... ,
  (char *)0);

int
execle(const char *path, const char *arg0, ... ,
  (char *)0, char *const envp[] );

int
execlp(const char *file, const char *arg0, ... ,
  (char *)0);
```

execv, execvp

```
#include <unistd.h>
int
execv(const char *path, char *const argv[]);

int
execvp(const char *file, char *const argv[]);
```

参数说明

对于 exec 函数来说，第一个参数是要执行程序的路径名；其次的参数是要执行程序的文件名；然后是其他参数，有一点要注意的是，参数列表要以一个空指针结尾。

例子

```
$ cat execlp.c
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
int main(){
printf("Running ps with execlp:\n");
//execlp("ps","ps","ax",0);
execlp("/Users/hainingzhang/linux/system1","system1",0);
printf("Done.\n");
exit(0);
}
```

exec 启动程序的特点

- exec 启动的新程序会替换掉旧的程序，即原程序会消失，原程序 exec 之后的语句也不会再执行。
- exec 启动的新程序会继承原有程序的许多特征，包括文件描述符等。

复制一个进程镜像

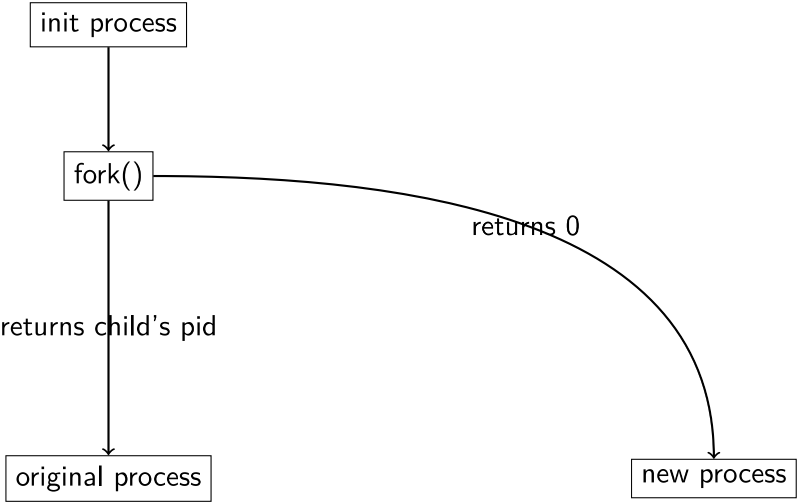
如果想使用一个进程同时运行多个程序，使用 exec 是不可以的，因为它会替换掉原有进程。
为了达到这个目的，可以使用多线程，或者 fork 系统调用 +exec 函数。
fork 会创建一个跟原始进程几乎一样的进程，但是这个新进程拥有自己独立的 pid 和地址空间，可以认为是两个不同的进程，然后在新的进程里执行其他操作。

fork

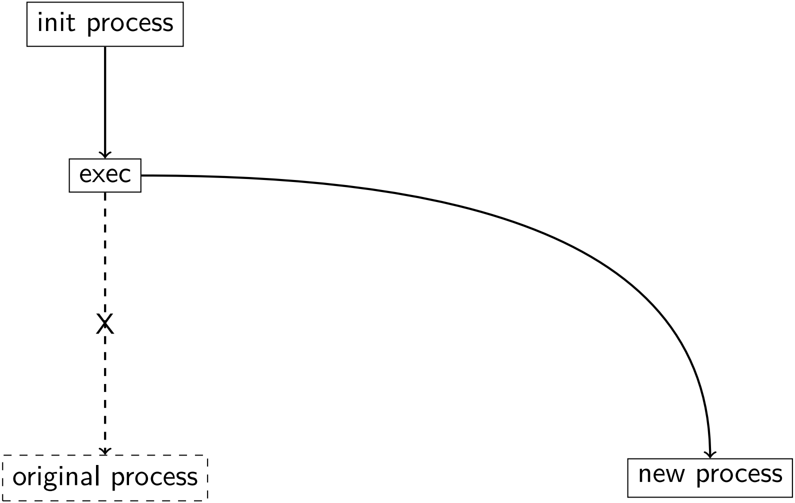
fork – create a new process
#include <unistd.h>
pid_t fork(void);

1. fork() returns a value of 0 to the child process
2. returns the process ID of the child process to the parent process
3. Otherwise, a value of -1 is returned to the parent process.

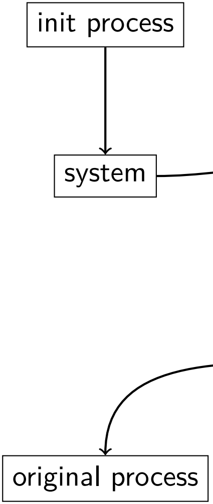
fork 图示


init process
fork()
returns 0
returns child's pid
original process
new process

exec 图示



system 图示



fork 的典型使用方式

fork1.c

```
int main(){
 pid_t pid=fork();
 switch(pid){
  case -1:
   printf("error!");
   break;
  case 0:
   printf("I am the child process.\n");
   execlp("/Users/hainingzhang/linux/system1","system1",0);
   break;
  default:
   printf("I am the parent process.\n");
   printf("I want to do something more...\n");
   break
 } exit(0); }
```

Waiting for a process

fork 了一个新进程之后中，这个新进程就自主的去运行了。但是有些时候父进程想知道子进程的状态，比如，子进程是否结束了？这种情况下可以使用wait来实现。

wait

```
#include <sys/wait.h>
pid_t  wait(int *stat_loc);
```

The wait system call causes a parent process to pause until one of its child processes is stopped.The call returns the PID of the child process.

wait

fork1.c

```c
if(pid!=0){
 int stat;
 pid_t child;
 child=wait(&stat);
 printf("the child(pid=%d) has finished.\n",child);
 if(WIFEXITED(stat)!=0){
  printf("child exit with code:%d\n",WEXITSTATUS(stat));
 }else{
  printf("child terminated abnormally.\n");
 }
}
```

Zombie Process

Using fork to create processes can be very useful, but you must keep track of child processes. When a child process terminates, an association with its parent survives until the parent in turn either terminates normally or calls wait. The child process entry in the process table is therefore not freed up immediately. Although no longer active, the child process is still in the system because its exit code needs to be stored in case the parent subsequently calls wait. It becomes what is known as defunct, or a zombie process.

If the parent then terminates abnormally, the child process automatically gets the process with PID 1 (init) as parent.

zombie example I

zombie.c

```
void myOut(char * msg, int i);
int main(){
char * message;  pid_t child=fork();
switch(child){
 case -1:
  message="Wrong.";
  myOut(message,1); break;
  case 0:
   message="Child.";
   myOut(message,2); break;
 default:
  message="Parent.";
  myOut(message,5); break;
  }
}
```

zombie example II

output

```
$ ./zombie 2>&1 >/dev/null&
[1] 35240
$ ps -al
  UID   PID  PPID     S    TIME CMD
    0 18965   576     Ss  0:00.02 login -pf hainingzhang
  501 18966 18965     S   0:01.62 -bash
  501 35240 18966     S   0:00.01 ./zombie
  501 35241 35240     Z   0:00.00 (zombie)
    0 35243 18966     R+  0:00.00 ps -al
  501 33922 33921     S   0:00.03 -bash
  501 35123 33922     S+  0:00.03 ssh zhanghaining@210.40.16.9
```

作业

编写一个程序，要求如下：

1. 在父进程中 fork 出一个子进程
2. 在父进程中，每 1 秒输出一个大写字母：A∼Z
3. 在子进程中，每 1 秒输出一个小写字母：a∼z

The End

Appendix

本课程相关资源下载

1. ppt
   `https://github.com/gmsft/ppt/tree/master/linux`
2. 实验指导书
   `https://github.com/gmsft/ppt/tree/master/book/linux`

about man page

The manual is generally split into eight numbered sections, organized as follows (on Research Unix, BSD, macOS and Linux):

section	description
1	General commands
2	System calls
3	Library function(C standard library)
4	Special files(devices) and drivers
5	File formats and conventions
6	Games and screensavers
7	Miscellanea
8	System administration commands and daemons

Table: man page

在终端中运行 man read **与** man 2 read **，观察其输出的区别。**