8-进程控制

8.3 fork

- strlen使用需要一次系统调用；对于字符串使用sizeof是编译时计算长度
- 父进程的所有打开的文件描述符共享给子进程
- 父进程和子进程共享一个偏移量

8.4 vfork

- vfork保证子进程先运行；子进程调度后才会运行父进程
- exec, exit执行前，vfork在原来父进程的空间进行执行
- exit时，会把父进程的缓冲区也冲刷掉

8.5 wait和waitpid

- 获取子进程的终止状态
- wait阻塞获取，waitpid可以设置成非阻塞获取
- 避免僵死状态
  - wait, waitpid
  - 使用两次fork，让init进程接管(**第一个子进程直接退出；第二个子进程变为孤儿进程，由init进程接管**)

```c
#include "apue.h"
#include <sys/wait.h>

int
main(void)
{
  pid_t pid;

  if ((pid = fork()) < 0) {
    err_sys("fork error");
  } else if (pid == 0) {    /* first child */
    if ((pid = fork()) < 0)
      err_sys("fork error");
    else if (pid > 0)
      exit(0);  /* parent from second fork == first child */

    /*
     * We're the second child; our parent becomes init as soon
     * as our real parent calls exit() in the statement above.
     * Here's where we'd continue executing, knowing that when
     * we're done, init will reap our status.
     */
    sleep(2);
    printf("second child, parent pid = %ld\n", (long)getppid());
    exit(0);
  }

  if (waitpid(pid, NULL, 0) != pid) /* wait for first child */
    err_sys("waitpid error");

  /*
   * We're the parent (the original process); we continue executing,
   * knowing that we're not the parent of the second child.
   */
  exit(0);
}
```

8.6 竞争条件

- 子进程等待父进程；使用轮询；浪费大量CPU资源

```c
while(getppid()!=1)
    sleep(1);
```

- 使用信号等IPC机制

8.7 exec函数

- exec族函数**只有execve是系统调用，其他都是库函数**

8.8 更改用户ID和更改组ID

用户设置ID位：将有效用户ID设置为实际用户ID

- 只有超级用户进程可以更改实际用户ID
- 只有**设置用户设置ID位，exec才设置有效用户ID**
- **保存的设置用户ID是有效用户ID副本**

```
#include <unistd.h>
int setuid(uid_t uid);
int setgid(gid_t gid);
```

- 进程root，setuid将实际用户ID，有效用户ID和保存设置用户ID设置为uid
- uid为实际用户ID或保存设置ID，setuid将有效用户ID设置为uid
- 否则返回-1，errno设置EPERM

8.12 解释器文件

- 文本文件，起始行

```
#!pathname[optional-argument]

#!/bin/sh
```

- 对这种文件的识别由内核的exec来负责；内核执行的是pathname指向的文件
- arg[0]是解释器的pathname，arg[1]是可选参数，arg[2]是pathname
- 解释器参数-f
  - 如果路径在/usr/local/bin下，直接命令行输出awkexample
  - 否则/bin/awk -f /usr/local/bin/awkexample，-f告诉awk在何处找到awk程序

8.13 system函数

- 可以执行shell命令

- 从命令行取参数后，正常执行

```c
int
system(const char *cmdstring) /* version without signal handling */
{
  pid_t pid;
  int   status;

  if (cmdstring == NULL)
    return(1);    /* always a command processor with UNIX */

  if ((pid = fork()) < 0) {
    status = -1;  /* probably out of processes */
  } else if (pid == 0) {        /* child */
    execl("/bin/sh", "sh", "-c", cmdstring, (char *)0);
    _exit(127);   /* execl error */
  } else {              /* parent */
    while (waitpid(pid, &status, 0) < 0) {
      if (errno != EINTR) {
        status = -1; /* error other than EINTR from waitpid() */
        break;
      }
    }
  }

  return(status);
}
```

- 可能出现切换超级用户权限后造成安全性漏洞的情况；可以考虑当有效用户ID与实际用户ID不匹配时，将有效用户ID设置为实际用户ID

8.14 进程会计

- init进程和守护进程不会进行会计
- 会计文件中记录的顺序对应于进程终止的顺序
- accton on开启记录，accton off关闭记录，Linux记录在/var/log/account/pacct中

8.15 进程调度

- 友好值调度，友好值越小，优先级越高
- nice获取或更改友好值，需要使用前清除errno

```c
#include <unistd.h>
int nice(int incr);
```

- getpriority可以获取一组相关进程的友好值
- setpriority可用于为进程、进程组和属于特定用户ID的所有进程设置优先级

```c
#include <sys/resource.h>
int getpriority(int which, id_t who);
int setpriority(int which, id_t who, int value);
// which可取进程、进程组、用户ID；who参数选择感兴趣的一个或多个进程
// value增加到NZERO上，变成新的友好值
```

8.16 进程时间

```c
#include <sys/times.h>
clock_t time(struct tms *buf);
// 成功返回流逝的墙上流逝时间；出错返回-1

struct tms {
    clock_t tms_utime;    // user cpu time
    clock_t tms_stime;    // system cpu time
    clock_t tms_cutime;   // user cpu time, terminated children
    clock_t tms_cstime;   // system cpu time, terminated children
};
```