Lesson 15

1.通过 fork 创建的父子进程对于 fork 之前打开的 fd 共享文件偏移量。

2.目前我们碰到的共享文件偏移量的情况有 2 种：

a) 通过 dup 等手段复制 fd，此时两个 fd 共享文件偏移量

b) fork 父子进程，二者共享文件偏移量。

3.close 的关闭采用的是引用计数，当执行 close 时，是把该 fd 指向的内核中的文件表项引用计数减 1，仅当引用计数为 0 时才真正销毁该结构。

4.shell 的工作原理当我们在键盘上敲入”ls”的时候

a) shell（bash、zsh）先 fork 一个子进程

b) 将子进程的代码使用 exec 替换为“ls”

c) shell 负责该子进程的回收

5.对于经典的 fork+exec 的组合模式，fork 出子进程再进行替换，那么复制完整的子进程的地址空间是无意义的。所以提出两种解决方案：

a) vfork：vfork 的目的就是为了 exec。

b) 对于 fork 采用写时复制技术。

6.fork 的写时复制技术：

a) fork 子进程时，仅仅复制页表项，而不是具体的进程空间。同时将地址空间设为只读。

b) 每当任何一方试图修改地址空间时，就自己复制一份。

7.写时复制（COW）使得父子进程，在物理上共享地址空间的，但是在逻辑上地址空间是相互独立的。

8.处理僵尸进程的手段：

a) 处理 SIGCHLD 信号

b) 采用 wait、waitpid

9.如果没有任何子进程，那么执行 wait 时，会立刻返回-1，同时 errno 为 ECHLD。否则阻塞，使用 WNOHANG 可以避免阻塞。

10.waitpid 不是按照顺序回收子进程。

11.system 与 exec 区别：

a) exec 替换的是当前进程

b) system 则是创建子进程，然后调用 exec 替换

12.system 的实现：

a) 创建子进程

b) 子进程采用 exec 进行进程替换

c) 父进程回收子进程，注意 EINTR

13.守护进程与普通进程的区别：

a) 守护进程不属于 shell 所在的会话组

b) 当 shell 退出的时候，守护进程不受影响

14.中断分为两类：

a) 硬中断，就是通常所说的中断，中断处理程序运行在

内核态，需要一定的硬件支持

b) 软中断是在软件层次上对中断的一种模拟，就是常说的信号，它的处理程序运行在用户态。它是软件级别的，不需要特定的硬件支持。

15.常见的信号：

a) SIGINT： ^C

b) SIGQUIT: ^\

c) SIGKILL: kill -9 pid

d) SIGPIPE: 与关闭的 TCP 连接有关

e) SIGCHLD： 子进程消亡，出现僵尸进程

f) SIGALRM：alarm 函数

16.关于信号的术语：

a) 发送信号

b) 接收信号

17.对信号的处理方式有三种：

a) 自行编写 handler

b) 忽略

c) 采用系统的默认行为

18.发送信号：

a) kill -n pid：给 pid 进程发送 n 信号

b) 使用系统调用 kill

c) alarm 函数，给自己发 SIGALRM 信号

19.已经发送但是还未被接收的信号称为待处理信号。系统采用一个向量表示待处理信号的集合，所以每种类型的信号最多只有一个待处理信号。一个待处理信号最多只能被接收一次。

20.信号处理带来的一些问题：

a) 待处理信号被阻塞：如果程序正在处理 SIGINT，那么此时到达一个 SIGINT，会被阻塞，成为待处理信号，一直到上一个 SIGINT 处理程序返回。

b) 待处理信号不会排队等待：每种类型的信号最后只有一个待处理信号，如果程序中已经有一个 SIGINT 待处理信号，此时到达一个 SIGINT，会被丢弃。

c) 系统调用可以被 handler 打断，即 EINTR，例如 read(fd, ...)是一个慢速的系统调用，此时如果有信号到来，会切换到该信号的 handler 中，当从该 handler 返回的时候，read 也返回-1，errno 置为 EINTR。

21.07signal 的分析：

a) 子进程 sleep2s 后退出，父进程收到 10 个 SIGCHLD 信号（这 10 个信号几乎是同一时间发出）。

b) 父进程开始处理第一个 SIGCHLD，里面 sleep2s。

c) 第二个信号到来，被阻塞为待处理信号。

d) 其他所有的信号都被简单的丢弃。

22.解决以上问题的关键思想：

a) 信号对应的向量置位为 1，说明至少存在一个该信号。

b) 当处理 SIGCHLD 信号时，应该尽可能多处理子进程。

23.08signal 的分析：

a) 10 个子进程几乎同时消亡，发出 10 个 SIGCHLD 信号

b) 父进程接收一个 SIGCHLD，执行 waitpid，回收第一个子进程，然后睡眠 2s。

c) 在这 2s 内：第二个 SIGCHLD 变成待处理，其余丢失。

d) 睡眠结束，此时存在 9 个僵尸进程，然后执行 9+1 次 while 循环，全部处理完毕。

e) 以上都属于第一个 SIGCHLD 的 handler

f) 处理第二个 SIGCHLD，此时没有子进程，waitpid 失败，退出。

24.处理僵尸进程：

a) 在父进程中编写同步代码，可以按照顺序或者异步的方式逐个回收子进程。这种方法的缺陷是需要准确把握子进程的消亡时机。

b) 处理 SIGCHLD 信号，这个方法提供了一种异步处理能力。其中要注意：处理 SIGCHLD 信号时，应该使用 while 循环尽可能多去处理子进程，这里主要是防止信号的阻塞和丢失问题。例如：

```
void handler(int signum)
{
    pid_t pid;
    //每次尽可能多去处理子进程
```

```
        while((pid = waitpid(-1, NULL, WNOHANG)) > 0)
        {
            printf("Handler process child %d\n", pid);
            sleep(2);     //故意去阻塞别的 SIGCHLD 信号
        }
}
```