



武汉纺织大学
WUHAN TEXTILE UNIVERSITY

崇真尚美



Linux系统编程

数学与计算机学院

教师：朱萍

本课程的主要内容

- **Linux**基础知识（4学时）
- **Linux**文件系统（6学时）
- **Linux**系统管理（4学时）
- **Linux**网络管理及应用（4学时）
- **Linux Shell**编程（6学时）
- **Linux**下C编程（6学时）
- **Linux**下进程控制（4学时）
- **Linux**文件IO（4学时）
- **Linux IPC**（6学时）
- **Linux**线程控制（4学时）

Linux进程控制

❖ 程序vs 进程

- ❖ Linux进程创建函数: `fork`
- ❖ Linux获得进程标识符函数: `getpid, getppid`
- ❖ Linux进程执行函数: `exec`函数族
- ❖ Linux终止进程函数: `exit`
- ❖ Linux等待进程函数: `wait waitpid`
- ❖ 进程间信号通信

程序 vs 进程

- ❖ 程序是静态的概念，进程是动态概念
- ❖ Linux为每个进程分配一个ID号，简称PID。
- ❖ 分为运行、阻塞、就绪三个基本状态。
- ❖ Linux中与进程相关的系统调用有十几个，只介绍常用的几个。

Linux进程控制

❖ 程序vs 进程

❖ Linux进程创建函数： `fork`

❖ Linux获得进程标识符函数： `getpid, getppid`

❖ Linux进程执行函数： `exec`函数族

❖ Linux终止进程函数： `exit`

❖ Linux等待进程函数： `wait waitpid`

进程创建函数调用：fork

❖ fork() 函数用于从已存在的进程中创建一个新进程。新进程称为子进程，而原进程称为父进程。

❖ 语法：

所需头文件↵	<code>#include <sys/types.h> // 提供类型 <code>pid_t</code> 的定义↵ #include <unistd.h>↵</code>
函数原型↵	<code>pid_t fork(void)↵</code>
函数返回值↵	0: 子进程↵
	子进程 ID (大于 0 的整数): 父进程↵
	-1: 出错↵

pid_t表示有符号整型

forktest.c

```
int main(void)
{   pid_t pid; int n=0; char * msg;
    pid = fork();
    if(pid == -1){
        msg="fork fail!!!"; printf(msg);
        return 1;}
    else if(pid == 0){
        msg=" I AM CHILD\n" ; n=6;
    } else{
        msg="I am father\n";  n=3;}
    for(;n>0;n--){
        printf("%s",msg);
        usleep(1000);}
}
```

❖ gcc编译运行，多运行几次，比较结果

```
jcx@ubuntu:~/work/chapt7$ gcc -o forktest forktest.c
jcx@ubuntu:~/work/chapt7$ ./forktest
i am father
I AM CHILD
i am father
i am father
I AM CHILD
jcx@ubuntu:~/work/chapt7$ I AM CHILD
I AM CHILD
I AM CHILD
I AM CHILD
```

???

进程的动态
性

```
jcx@ubuntu:~/work/chapt7$ ./forktest
i am father
i am father
I AM CHILD
i am father
jcx@ubuntu:~/work/chapt7$ I AM CHILD
I AM CHILD
I AM CHILD
I AM CHILD
I AM CHILD
```


fork工作过程示意图

Parent

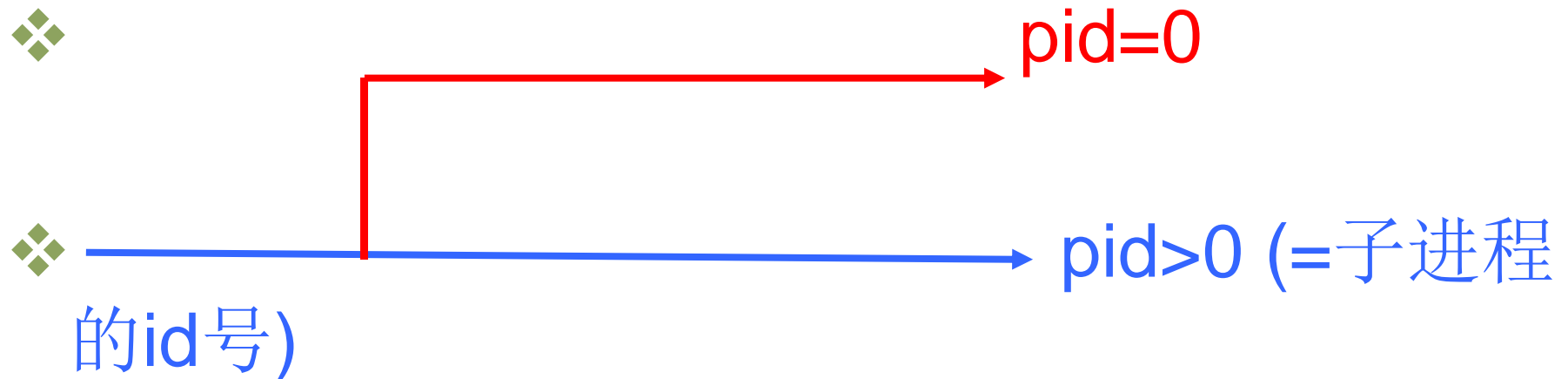
```
int main()
{
    pid_t pid;
    char *message;
    int n;
    pid = fork();
    if (pid < 0) {
        perror("fork failed");
        exit(1);
    }
    if (pid == 0) {
        message = "This is the child\n";
        n = 6;
    } else {
        message = "This is the parent\n";
        n = 3;
    }
    for(; n > 0; n--) {
        printf(message);
        sleep(1);
    }
    return 0;
}
```

Child

```
int main()
{
    pid_t pid;
    char *message;
    int n;
    pid = fork();
    if (pid < 0) {
        perror("fork failed");
        exit(1);
    }
    if (pid == 0) {
        message = "This is the child\n";
        n = 6;
    } else {
        message = "This is the parent\n";
        n = 3;
    }
    for(; n > 0; n--) {
        printf(message);
        sleep(1);
    }
    return 0;
}
```

fork函数特点

❖ 一次调用，两次返回



❖ 像个叉子（fork）一样

Linux进程控制

❖ 程序vs 进程

❖ Linux进程创建函数： fork

❖ Linux获得进程标识符函数： getpid,getppid

❖ Linux进程执行函数： exec函数族

❖ Linux终止进程函数： exit

❖ Linux等待进程函数： wait waitpid

获得进程、父进程 id号

获得本进程id号

✚

所需头文件✚	<code>#include <sys/types.h>✚</code> <code>#include <unistd.h>✚</code>
函数原型✚	<code>pid_t getpid(void)✚</code>
函数返回值✚	该进程 id✚

✚

获得父进程id号

pid_t getppid(void)

forktest_pid.c

```
{
    pid_t pid;

    pid = fork();
    if(pid == -1){
        printf("fork fail!!!");
        return 1;
    }
    else if(pid == 0){
        printf("I AM CHILD, pid=%d, parent pid=%d\n",getpid(),getppid(
    }else{
        sleep(1);
        printf("i am father, pid=%d\n",getpid());
    }

    return 0;
}
```

```
jlx@ubuntu:~/work/chapt7$ gcc -o forktest_pid forktest_pid.c
jlx@ubuntu:~/work/chapt7$ ./forktest_pid
I AM CHILD, pid=3579, parent pid=3578
i am father, pid=3578
jlx@ubuntu:~/work/chapt7$
```

- ❖ 上述程序如果去掉父进程中的sleep语句会有什么结果？

```
jcx@ubuntu:~/work/chapt7$ ./forktest_pid
i am father, pid=3739
jcx@ubuntu:~/work/chapt7$ I AM CHILD, pid=3740, parent pid=1916
```

- ❖ 孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。
- ❖ 孤儿进程将被init进程(进程号为1)所收养，并由init进程对它们完成状态收集工作。
- ❖ 后来的ubuntu，upstart是在图形化界面下的一个后台的守护程序，是图形化界面收养孤儿进程。

Linux进程控制

- ❖ 程序vs 进程
- ❖ Linux进程创建函数： fork
- ❖ Linux获得进程标识符函数： getpid,getppid
- ❖ Linux进程执行函数： exec函数族
- ❖ Linux终止进程函数： exit
- ❖ Linux等待进程函数： wait waitpid

exec函数族

- ❖ **exec**函数族就提供了一个在进程中启动另一个程序执行的方法。
- ❖ 用**fork**创建子进程后，子进程和父进程执行不同的代码分支，子进程往往在自己的分支要调用**exec**函数族来执行另一个程序。
- ❖ **exec**函数族可以根据指定的文件名或目录名找到可执行文件，并用它来取代原调用进程的数据段、代码段和堆栈段，在执行完之后，原调用进程的内容除了进程号外，**其他全部被新的进程替换了**。
- ❖ 这里的可执行文件既可以是二进制文件，也可以是Linux下任何可执行的脚本文件。

exec函数族

所需头文件	<code>#include <unistd.h></code>
函数原型	<code>int execl(const char *path, const char *arg, ...)</code>
	<code>int execv(const char *path, char *const argv[])</code>
	<code>int execl(const char *path, const char *arg, ..., char *const envp[])</code>
	<code>int execve(const char *path, char *const argv[], char *const envp[])</code>
	<code>int execlp(const char *file, const char *arg, ...)</code>
	<code>int execvp(const char *file, char *const argv[])</code>
函数返回值	-1: 出错

前 4 位	统一为: <code>exec</code>	
第 5 位	l: 参数传递为逐个列举方式	<code>execl</code> 、 <code>execl</code> 、 <code>execlp</code>
	v: 参数传递为构造指针数组方式	<code>execv</code> 、 <code>execve</code> 、 <code>execvp</code>
第 6 位	e: 可传递新进程环境变量	<code>execl</code> 、 <code>execve</code>
	p: 会从PATH环境变量中查找可执行文件	<code>execlp</code> 、 <code>execvp</code>

execdemo.c

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    pid_t pid;
    pid = fork();
    if(pid == 0){
        printf("I AM CHILD 111111,pid=%d\n",getpid());
        //sleep(10);
        execl("/home/jkx/work/chapt7/mytest","mytest",NULL);
        printf("I AM CHILD 22222\n");
    }
    if (pid>0){
        printf("In father process\n" );
    }
    return 0;
}
```

```
#include <stdio.h>
#include <unistd.h>
void main()
{
    printf("I am here, I am in mytest.c !!!\n");
    sleep(20);
    return;
}
```

运行execdemo

```
jcx@ubuntu:~/work/chapt7$ ./execdemo
In father process
jcx@ubuntu:~/work/chapt7$ I AM CHILD 111111,pid=2956
I am here, I am in mytest.c !!!
```

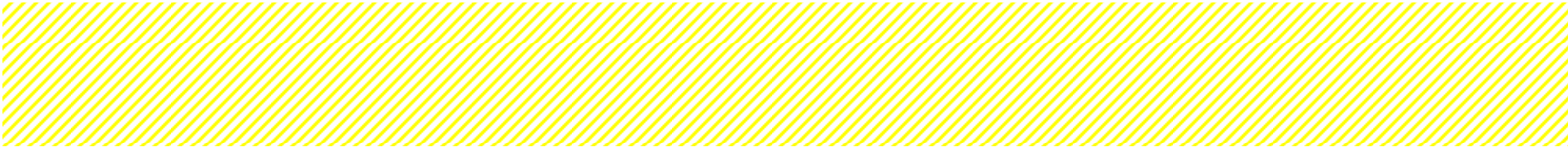
❖ 将上述代码sleep打开，另外的termianl的标签观察，同一个进程pid的进程名变化了。

```
jcx@ubuntu:~/work/chapt7$ ps aux|grep 2956
jcx      2956  0.0  0.0   2200    60 pts/3    S   22:24   0:00 ./execdemo
jcx      2958  0.0  0.0   5108   876 pts/6    S+  22:24   0:00 grep --color=au
to 2956
jcx@ubuntu:~/work/chapt7$ ps aux|grep 2956
jcx      2956  0.0  0.0   2200   512 pts/3    S   22:24   0:00 mytest
jcx      2963  0.0  0.0   5108   792 pts/6    S+  22:24   0:00 grep --color=au
to 2956
```

exec test.c

```
char *envp1[ ]={"PATH=/tmp",NULL};
char *envp2[ ]={"PATH=/home/jkx",NULL};
char *argv_execv[ ]={"echo", "executed by execv", NULL};
char *argv_execvp[ ]={"echo", "executed by execvp", NULL};
char *argv_execve[ ]={"env", NULL};

if(fork()==0)
    if(execl("/bin/echo", "echo", "executed by execl", NULL)<0)
        perror("Err on execl");
if(fork()==0)
    if(execvp("echo", "echo", "executed by execlp", NULL)<0)
        perror("Err on execlp");
if(fork()==0)
    if(execle("/usr/bin/env", "env", NULL, envp1)<0)
        perror("Err on execl");
if(fork()==0)
    if(execv("/bin/echo", argv_execv)<0)
        perror("Err on execv");
if(fork()==0)
    if(execvp("echo", argv_execvp)<0)
        perror("Err on execvp");
if(fork()==0)
    if(execve("/usr/bin/env", argv_execve, envp2)<0)
        perror("Err on execve");
```



```
jkg@ubuntu:~/work/chapt7$ gcc -o exectest exectest.c
jkg@ubuntu:~/work/chapt7$ ./exectest
jkg@ubuntu:~/work/chapt7$ PATH=/tmp
executed by execlp
executed by execl
PATH=/home/jkg
executed by execv
executed by execvp
```

Linux进程控制

- ❖ 程序vs 进程
- ❖ Linux进程创建函数: `fork`
- ❖ Linux获得进程标识符函数: `getpid, getppid`
- ❖ Linux进程执行函数: `exec`函数族
- ❖ Linux终止进程函数: `exit`
- ❖ Linux等待进程函数: `wait waitpid`

终止进程函数 `exit`

❖ 格式

- `#include <stdlib.h>`
- `void exit(int status)`

❖ 说明：该函数是进程的自我终止，使其进入僵死状态，等待父进程进行善后处理。**status**是返回给父进程的一个整数。

exittest.c

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    printf("I will exit\n");
    exit(0);
    printf("I have exited\n");
    return 0;
}
```

```
jkx@ubuntu:~/work/chapt7$ gcc -o exittest exittest.c
jkx@ubuntu:~/work/chapt7$ ./exittest
I will exit
jkx@ubuntu:~/work/chapt7$
```


Linux进程控制

- ❖ 程序vs 进程
- ❖ Linux进程创建函数: `fork`
- ❖ Linux获得进程标识符函数: `getpid, getppid`
- ❖ Linux进程执行函数: `exec`函数族
- ❖ Linux终止进程函数: `exit`
- ❖ Linux等待进程函数: `wait waitpid`

wait函数

❖ **wait()**函数是用于使父进程（也就是调用**wait()**的进程）**阻塞**，直到有一个子进程结束或者该进程收到了一个指定的信号为止。如果该父进程没有子进程或者该子进程已经结束，则**wait()**就会立即返回。

❖ 格式：

- `#include <sys/types.h>`
- `#include <sys/wait.h>`
- `pid_t wait(int *status)`

- **status**: 保存子进程状态，后面会详细说明，如果不关心子进程状态可以写成**NULL**。

死等，
可怜呀！

waittest.c

- ❖ 修改forktest_pid.c，让父进程等子进程退出后才退出。

```
jkx@ubuntu:~/work/chapt7$ ./waittest
i am father, pid=3369
I AM CHILD, pid=3370, parent pid=3369
```

waitpid()

- ❖ waitpid()的作用和wait()一样，更灵活。它可以指定等待哪个子进程，还可以非阻塞方式。
- ❖ 实际上wait()函数只是waitpid()函数的一个特例，在Linux内部实现wait()函数时直接调用的就是waitpid()函数。
- ❖ waitpid灵活，所以使用起来麻烦些

看一眼就
跑，
嘿嘿

waitpid函数说明

所需头文件	#include <sys/types.h> #include <sys/wait.h>	
函数原型	pid_t waitpid(pid_t pid, int *status, int options)	
函数传入值	Pid	pid > 0: 只等待进程 ID 等于 pid 的子进程, 不管已经有其他子进程运行结束退出了, 只要指定的子进程还没有结束, waitpid()就会一直等下去。
		pid = -1: 等待任何一个子进程退出, 此时和 wait()作用一样。
		pid = 0: 等待其组 ID 等于调用进程的组 ID 的任一子进程。
		pid < -1: 等待其组 ID 等于 pid 的绝对值的任一子进程。
函数传入值	status	同 wait()。
	options	WNOHANG: 若由 pid 指定的子进程不立即可用, 则 waitpid()不阻塞, 此时返回值为 0。
		WUNTRACED: 若实现某支持作业控制, 则由 pid 指定的任一子进程状态已暂停, 且其状态自暂停以来还未报告过, 则返回其状态。
函数返回值	0: 同 wait(), 阻塞父进程, 等待子进程退出。	
	正常: 已经结束运行的子进程的进程号。	
	使用选项 WNOHANG 且没有子进程退出: 0。	
	调用出错: -1。	

waitpidtest.c

```
int main(void)
{
    pid_t pid;
    pid=fork();
    if(pid<0){
        perror("fork failed");
        exit(1);
    }
    if (pid == 0){
        int i;
        for(i=3;i>0;i--){
            printf("this is the child\n");
            sleep(1);
        }
        exit(3);
    }else{
        int stat_val;
        waitpid(pid,&stat_val,0);
        printf("this is father!\n");
        if( WIFEXITED(stat_val))
            printf("Child exited with code %d\n", WEXITSTATUS
(stat_val));
    }
    return 0;
}
```

两个宏

- ❖ **WIFEXITED(status)** 这个宏用来指出子进程是否为正常退出的，如果是，它会返回一个非零值。
 - **WEXITSTATUS(status)** 当**WIFEXITED**返回非零值时，我们可以用这个宏来提取子进程的返回值，如果子进程调用**exit(5)**退出，**WEXITSTATUS(status)**就会返回**5**；如果子进程调用**exit(7)**，**WEXITSTATUS(status)**就会返回**7**。请注意，如果进程不是正常退出的，也就是说，**WIFEXITED**返回**0**，这个值就毫无意义。

waitpidtest.c输出结果

```
jkx@ubuntu:~/work/chapt7$ ./waitpidtest
this is the child
this is the child
this is the child
this is father!
Child exited with code 3
```

❖ 父进程阻塞方式等待子进程完毕后才打印

waitpidtest_no.c

❖ 父进程非阻塞方式等待

```
int main(void)
{
    pid_t pid;
    pid=fork();
    if(pid<0){
        perror("fork failed");
        exit(1);
    }
    if (pid == 0){
        int i;
        for(i=3;i>0;i--){
            printf("this is the child\n");
            sleep(1);
        }
        exit(3);
    }else{
        int stat_val;
        waitpid(pid,&stat_val,WNOHANG);
        printf("this is father!\n");
        if( WIFEXITED(stat_val))
            printf("Child exited with code %d\n", WEXITSTATUS
(stat_val));
    }
    return 0;
}
```

课堂作业

❖ 编程实现以下功能：父进程使用fork函数创建2个子进程。

❖ 要求：

- 父进程阻塞方式等子进程1，非阻塞方式等子进程2。
- 父进程输出自己pid，子进程输出自己和父亲的pid
- 子进程2内，休息5s
- 父进程等待子进程后输出孩子的退出码。

```
jkx@ubuntu:~/work/chapt7$ ./wait2child
I am child2,pid=4270,parent pid=4268,I will sleep for 5 seconds!
I am child1,pid=4269,parent pid=4268
I am father,pid=4268
child1 exit with code 1
jkx@ubuntu:~/work/chapt7$ I am child2.I have awaked and I will exit!
```

Linux进程控制

- ❖ 程序vs 进程
- ❖ Linux进程创建函数: `fork`
- ❖ Linux获得进程标识符函数: `getpid, getppid`
- ❖ Linux进程执行函数: `exec`函数族
- ❖ Linux终止进程函数: `exit`
- ❖ Linux等待进程函数: `wait waitpid`
- ❖ 进程间信号通信

进程间通信 (Inter-Process Communication, IPC)

- ✓ 信号
- ✓ 管道（匿名/有名）
- ✓ 信号量
- ✓ 共享内存
- ✓ 消息队列
- ✓ 套接字 socket

信号 (Signal)

- ❖ 信号是在软件层次上对中断机制的一种模拟，用于通知进程有某事件发生，一个进程收到一个信号与处理器收到一个中断请求效果上可以说是一样的。
- ❖ 进程之间可以互相通过系统调用kill发送软中断信号。内核也可以因为内部事件而给进程发送信号，通知进程发生了某个事件。
- ❖ 命令kill -l可以查看系统包含哪些信号

xx@ubuntu:~/work/chapt7\$ kill -l

1) SIGHUP	2) SIGINT	3) SIGQUIT	4) SIGILL	5)
6) SIGABRT	7) SIGBUS	8) SIGFPE	9) SIGKILL	10)
11) SIGSEGV	12) SIGUSR2	13) SIGPIPE	14) SIGALRM	15)
16) SIGSTKFLT	17) SIGCHLD	18) SIGCONT	19) SIGSTOP	20)
21) SIGTTIN	22) SIGTTOU	23) SIGURG	24) SIGXCPU	25)
26) SIGVTALRM	27) SIGPROF	28) SIGWINCH	29) SIGIO	30)
31) SIGSYS	34) SIGRTMIN	35) SIGRTMIN+1	36) SIGRTMIN+2	37)
38) SIGRTMIN+4	39) SIGRTMIN+5	40) SIGRTMIN+6	41) SIGRTMIN+7	42)
43) SIGRTMIN+9	44) SIGRTMIN+10	45) SIGRTMIN+11	46) SIGRTMIN+12	47)
48) SIGRTMIN+14	49) SIGRTMIN+15	50) SIGRTMAX-14	51) SIGRTMAX-13	52)
53) SIGRTMAX-11	54) SIGRTMAX-10	55) SIGRTMAX-9	56) SIGRTMAX-8	57)
58) SIGRTMAX-6	59) SIGRTMAX-5	60) SIGRTMAX-4	61) SIGRTMAX-3	62)
63) SIGRTMAX-1	64) SIGRTMAX			

常用的信号

❖ 几个常用的信号含义如下所示：

- **SIGHUP**: 终端上发出的结束信号。
- **SIGINT**: 来自键盘的中断信号 (**CTRL+C**)。
- **SIGQUIT**: 来自键盘的退出信号 (**CTRL +**)。
- **SIGFPE**: 浮点异常信号。
- **SIGALRM**: 进程的定时器到期，发送该信号。
- **SIGTERM**: **kill**发送出的信号。
- **SIGCHLD**: 标识子进程停止或结束的信号。
- **SIGKILL**: 该信号结束接收信号的进程。
- **SIGSTOP**: 来自键盘 (**CTRL+Z**) 或调试程序的停止信号。

信号发送函数：kill

❖ 格式：

```
#include <sys/types.h>
```

```
#include <signal.h>
```

```
int kill(pid_t pid,int signo)
```

❖ 说明：**kill**向**pid**进程发送信号**signo**。调用成功返回**0**；否则，返回**-1**。其中**pid**参数值不同，接收进程不同。

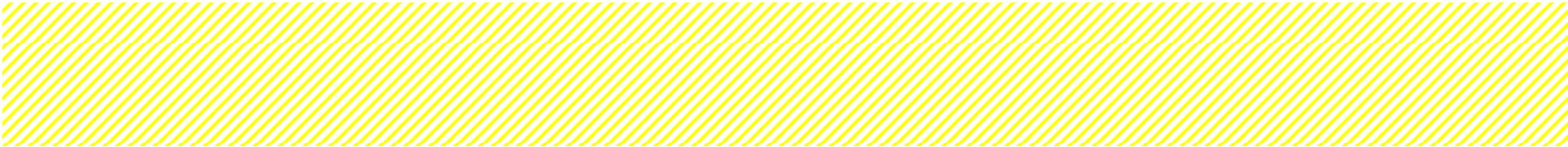
表 7.7 系统调用 kill 中参数 pid 的取值及含义

参数 pid 的值	信号的接收进程
pid>0	进程 id 为 pid 的进程
pid=0	同一个进程组的进程
pid<0 且 pid!= -1	进程组 id 为 -pid 的所有进程
pid=-1	除发送进程自身外，所有进程 id 大于 1 的进程

killtest.c

```
int main(void)
{
    pid_t pid;
    int retval;

    pid=fork();
    if(pid<0){
        perror("fork failed");
        exit(1);
    }
    if (pid == 0){
        printf("I AM CHILD 111111\n");
        sleep(3);
        printf("I AM CHILD 222222\n");
        exit(0);
    }else{
        printf("i am father \n");
        sleep(1);
        if ( 0 == waitpid(pid,NULL,WNOHANG)){
            retval = kill(pid, SIGKILL);
            if (0 == retval)
                printf("%d is killed !\n", pid);
            else
                printf("kill fail\n");
        }
    }
}
```



```
gcc -o killtest killtest.c
jkx@ubuntu:~/work/chapt7$ gcc -o killtest killtest.c
jkx@ubuntu:~/work/chapt7$ ./killtest
i am father
I AM CHILD 111111
6704 is killed !
```

信号发送函数: `raise`

❖ 格式:

- **`#include <signal.h>`**
- **`int raise(int signo)`**
- 说明: **`raise`** () 向进程本身发送信号, 参数为即将发送的信号值。调用成功返回**0**; 否则, 返回**-1**。

raisetest.c

```
int main(void)
{
    int i;
    for(i=0; i<100; i++){
        printf("i=%d\n",i);
        if(i==12)
            raise(SIGINT);
    }
    return 0;
}
```

```
jlx@ubuntu:~/work/chapt7$ ./raisetest
i=0
i=1
i=2
i=3
i=4
i=5
i=6
i=7
i=8
i=9
i=10
i=11
i=12
jlx@ubuntu:~/work/chapt7$
```

信号发送函数：alarm

❖ 格式：

- **#include <unistd.h>**
- **unsigned int alarm(unsigned int seconds)**
- 说明：**alarm**（）专门为**SIGALRM**信号而设，在指定的时间**seconds**秒后，将向进程本身发送**SIGALRM**信号，又称为闹钟时间。

```
int main(void)
{
    unsigned int i;
    alarm(1);
    for(i=0;1;i++)
        printf("I=%d\n",i);
    return 0;
}
```

信号接收函数：signal

❖ 格式：

```
#include <signal.h>
```

```
typedef void (*sighandler_t)(int);
```

```
sighandler_t signal(int signum, sighandler_t  
handler));
```

- 说明： **signal**（） 负责接收指定信号**signum**， 并进行相应处理。第一个参数指定信号的值，第二个参数是收到信号后的函数调用。失败返回**SIG_ERR**。

❖ **signal()**可以改变信号和函数的映射关系

signaltest.c

- ❖ 通常情况下接收到键盘的`ctl+c`，就会终止进程。
- ❖ 编写一个程序，该程序当我们输入**Ctrl+C**时输出字符串“**I got signal**”。在其余的时间，该程序只是无限循环，每一秒输出一条“**hello world**”信息。

signaltest.c

```
void myprint(void)
{
    printf("I get signal ctrl+c\n");
}

int main(void)
{
    if (SIG_ERR == signal(SIGINT, myprint)){
        printf("error !!!\n");
        return 1;
    }
    while(1){
        sleep(1);
        printf("hello world!\n");
    }
    return 0;
}
```


信号接收函数: `sigaction`

❖ 格式:

- **`#include <signal.h>`**
- **`int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);`**
- 说明: **`sigaction`** () 是较新的函数, 有三个参数, 支持信号传递信息, **`sigaction`** () 优于 **`signal`** () , 但使用起来更复杂。
- 该函数的第一个参数为信号的值, 可以为除 **`SIGKILL`** 及 **`SIGSTOP`** 外的任何一个特定有效的信号。第二个参数是指向结构 **`sigaction`** 的一个实例的指针, 在结构 **`sigaction`** 的实例中, 指定了对特定信号的处理, 可以为空, 进程会以缺省方式对信号处理; 第三个参数 **`oldact`** 指向的对象用来保存原来对相应信号的处理, 可指定 **`oldact`** 为 **`NULL`**。

课堂练习

❖ 在进程p1中调用系统函数execvp执行程序p2，则：

- A、 p1创建一个子进程执行p2
- B、 p2结束后返回p1继续执行
- C、 p1和p2并行执行
- D、 p2将替换掉p1的代码



❖ 进程调用wait将被阻塞直到:

- A、用户按任意键
- B、收到时钟信号
- C、子进程被创建
- D、子进程结束

作业

❖ P230/7

❖ 附加题1:

❖ 编写一个程序，该程序当输入CTRL+C时输出字符串“capture ctrl+c”，并退出。其余时间无限循环，每隔2秒输出一条“2s elapsed”信息。

附加2

- ✓ 在父进程中创建两个子进程，其中一个子进程运行“ls -l”指令，第二个子进程在暂停5s之后退出。
- ✓ 要求：父进程先用阻塞方式等待第一个子进程的结束，然后用非阻塞方式等待第二个子进程的退出，待收集到第二个子进程结束的信息，父进程就返回。



武汉纺织大学
WUHAN TEXTILE UNIVERSITY

崇真尚美



Thank You !

