

实验五：进程管理（二）

实验四

Hollow Man

实验名称：

进程管理（二）

实验目的：

1. 进一步学习进程的属性
2. 学习进程管理的系统调用
3. 掌握使用系统调用获取进程的属性、创建进程、实现进程控制等
4. 掌握进程管理的基本原理

实验时间

6 学时

实验要求：

4. 阅读下列程序：

```
/*  usage of kill,signal,wait  */  
  
#include<unistd.h>  
  
#include<stdio.h>  
  
#include<sys/types.h>  
  
#include<signal.h>  
  
#include<stdlib.h>  
  
  
int flag;  
  
void stop();  
  
int main(void)  
{  
  
    int pid1,pid2;  
  
    signal(3,stop);
```

```

while((pid1=fork()) == -1);

if(pid1>0){
    while((pid2=fork()) == -1);

    if(pid2>0){//父进程
        flag=1;
        sleep(5);

        kill(pid1,16);
        kill(pid2,17);
        wait(0);
        wait(0);
        printf("\n parent is killed\n");
        exit(EXIT_SUCCESS);
    }else{//子进程 2
        flag=1;
        signal(17,stop);
        printf("\n child2 is killed by parent\n");
        exit(EXIT_SUCCESS);
    }
}else{//子进程 1
    flag=1;
    signal(16,stop);
    printf("\n child1 is killed by parent\n");
    exit(EXIT_SUCCESS);
}
}

void stop(){
    flag = 0;
}

```

下面时用到的函数 `signal` 的解析:

`signal` (设置信号处理方式)

表头文件 `#include<signal.h>`

定义函数 `void (*signal(int signum,void(* handler)(int)));`

函数说明 `signal()`会依参数 `signum` 指定的信号编号来设置该信号的处理函数。当指定的信号到达时就会跳转到参数 `handler` 指定的函数执行。如果参数 `handler` 不是函数指针, 则必须是下列两个常数之一:

`SIG_IGN` 忽略参数 `signum` 指定的信号。

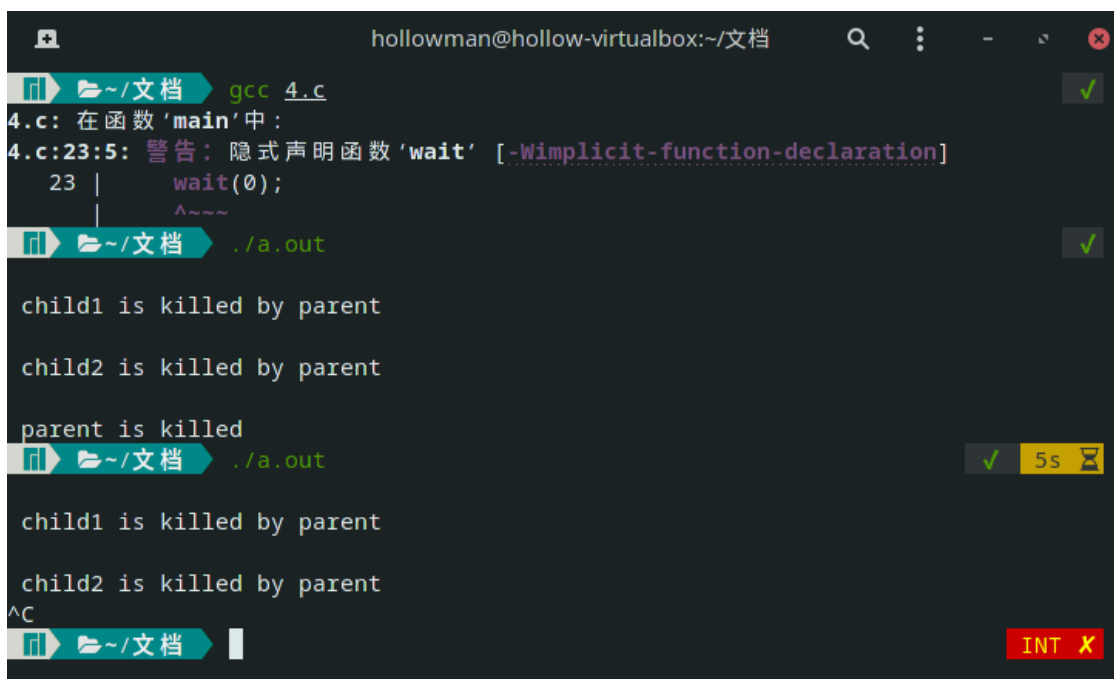
`SIG_DFL` 将参数 `signum` 指定的信号重设为核心预设的信号处理方式。

返回值 返回先前的信号处理函数指针, 如果有错误则返回 `SIG_ERR(-1)`。

附加说明 在信号发生跳转到自定的 `handler` 处理函数执行后, 系统会自动将此处理函数换回原来系统预设的处理方式, 如果要改变此操作请改用 `sigaction()`。

编译并运行, 等待或者按`^C`, 分别观察执行结果并分析, 注释程序主要语句。

`flag` 有什么作用? 通过实验说明。



```
hollowman@hollow-virtualbox:~/文档
~/文档 gcc 4.c
4.c: 在函数 'main' 中:
4.c:23:5: 警告: 隐式声明函数 'wait' [-Wimplicit-function-declaration]
    23 |     wait(0);
        |     ^~~~
~/文档 ./a.out
child1 is killed by parent
child2 is killed by parent
parent is killed
~/文档 ./a.out
child1 is killed by parent
child2 is killed by parent
^C
~/文档
```

每个进程都有一个 `flag`

`Flag=1` 时标志进程再运行,

`Flag=0` 时标志进程结束。

但是, 由于子进程运行速度过快,

在父进程睡眠之前就已经执行结束了，

所以子进程未能接收到信号。

所以实际上子进程的 flag 值一直为 1。

父进程开头的 signal(3,stop);

代表着若手动终止进程，则会调用 stop 将 flag 置 0.

5. 编写程序，要求父进程创建一个子进程，使父进程和个子进程各自在屏幕上输出一些信息，但父进程的信息总在子进程的信息之后出现。

程序代码如下：

```
#include <stdio.h>

#include <unistd.h>

#include <wait.h>

int main(void)
{
    int p;

    while((p = fork()) == -1); //创建一个子进程

    if(p > 0)                //父进程
    {
        wait(0);

        printf("parent process:\n");

        printf("\tpid: %d\n", p);
    }

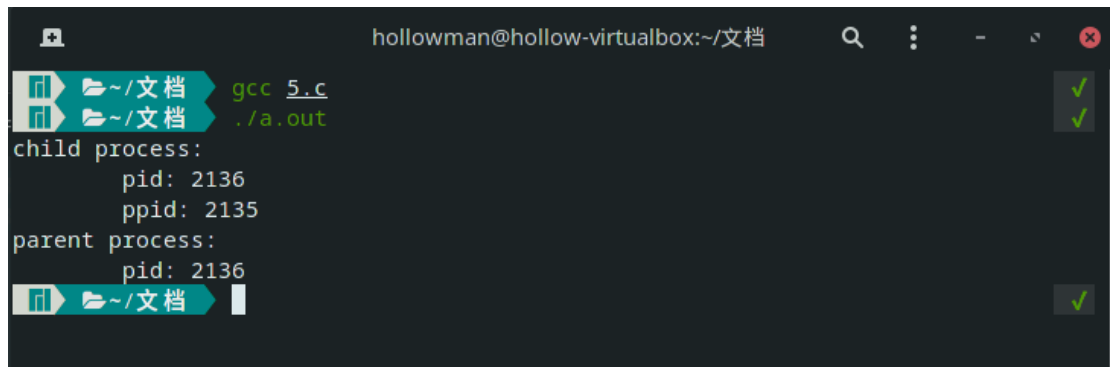
    else                      //子进程
    {
        printf("child process:\n");

        printf("\tpid: %d\n", getpid());

        printf("\tppid: %d\n", getppid());
    }

    return 0;
}
```

该程序执行的结果如下：

A terminal window titled 'hollowman@hollow-virtualbox:~/文档'. It shows the compilation of a C program: 'gcc 5.c' and './a.out'. Both commands are preceded by a folder icon and '~/.文档', and each has a green checkmark in a box to its right. Below the compilation, the program's output is displayed: 'child process:' followed by 'pid: 2136' and 'ppid: 2135', then 'parent process:' followed by 'pid: 2136'. The output is preceded by a folder icon and '~/.文档', and has a green checkmark in a box to its right.

```
hollowman@hollow-virtualbox:~/文档
gcc 5.c
./a.out
child process:
  pid: 2136
  ppid: 2135
parent process:
  pid: 2136
```

6. 编写程序, 要求父进程创建一个子进程, 子进程执行 shell 命令 `find / -name hda*` 的功能, 子进程结束时由父进程打印子进程结束的信息。执行中父进程改变子进程的优先级。

程序代码如下:

```
#include <stdio.h>

#include <unistd.h>

#include <wait.h>

#include <sys/resource.h>

int main(void)
{
    int p;

    while((p = fork()) == -1); //创建一个子进程

    if(p > 0)                //父进程
    {
        setpriority(PRIO_PROCESS, p, 1);

        printf("child process priority: %d.\n", getpriority(PRIO_PROCESS, p));

        wait(0);             //等待子进程结束

        printf("child process terminated.\n");
    }

    else                    //子进程
    {
        execlp("find", "find", "/", "-name", "hda*", NULL);
    }

    return 0;
}
```

}

程序的执行结果如下：

```
hollowman@hollow-virtualbox:~/文档
gcc 6.c
sudo ./a.out
[sudo] hollowman 的密码：
child process priority: 1.
find: '/run/user/1000/gvfs': 权限不够
/usr/share/alsa/topology/hda-dsp
/usr/share/alsa/ucm2/hda-dsp
/usr/share/alsa/ucm2/hda-dsp/hda-dsp.conf
/usr/share/alsa/ucm2/codecs/hda
/usr/share/alsa/init/hda
/usr/share/icons/Papirus/24x24/apps/hdajackretask.svg
/usr/share/icons/Papirus/22x22/apps/hdajackretask.svg
/usr/share/icons/Papirus/64x64/apps/hdajackretask.svg
/usr/share/icons/Papirus/16x16/apps/hdajackretask.svg
/usr/share/icons/Papirus/32x32/apps/hdajackretask.svg
/usr/share/icons/Papirus/48x48/apps/hdajackretask.svg
/usr/lib/modules/5.10.30-1-MANJARO/kernel/drivers/platform/x86/hdaps.ko.xz
/usr/lib/modules/5.10.30-1-MANJARO/kernel/sound/hda
/usr/lib/modules/5.10.30-1-MANJARO/kernel/sound/pci/hda
/usr/lib/modules/5.10.30-1-MANJARO/build/sound/hda
/usr/lib/modules/5.10.30-1-MANJARO/build/sound/pci/hda
/usr/lib/modules/5.10.30-1-MANJARO/build/include/config/sensors/hdaps.h
/usr/lib/modules/5.10.30-1-MANJARO/build/include/config/snd/hda.h
/usr/lib/modules/5.10.30-1-MANJARO/build/include/config/snd/soc/hdac
```

```
hollowman@hollow-virtualbox:~/文档
/usr/lib/modules/5.10.30-1-MANJARO/build/sound/hda
/usr/lib/modules/5.10.30-1-MANJARO/build/sound/pci/hda
/usr/lib/modules/5.10.30-1-MANJARO/build/include/config/sensors/hdaps.h
/usr/lib/modules/5.10.30-1-MANJARO/build/include/config/snd/hda.h
/usr/lib/modules/5.10.30-1-MANJARO/build/include/config/snd/soc/hdac
/usr/lib/modules/5.10.30-1-MANJARO/build/include/config/snd/soc/hdac/hda.h
/usr/lib/modules/5.10.30-1-MANJARO/build/include/config/snd/soc/intel/skylake/hda
audio
/usr/lib/modules/5.10.30-1-MANJARO/build/include/config/snd/soc/intel/skl/hda
/usr/lib/modules/5.10.30-1-MANJARO/build/include/config/snd/soc/sof/hda.h
/usr/lib/modules/5.10.30-1-MANJARO/build/include/config/snd/soc/sof/hda
/usr/lib/modules/5.10.30-1-MANJARO/build/include/config/snd/hda
/usr/lib/modules/5.10.30-1-MANJARO/build/include/sound/hda_codec.h
/usr/lib/modules/5.10.30-1-MANJARO/build/include/sound/hda_regmap.h
/usr/lib/modules/5.10.30-1-MANJARO/build/include/sound/hda_register.h
/usr/lib/modules/5.10.30-1-MANJARO/build/include/sound/hda_chmap.h
/usr/lib/modules/5.10.30-1-MANJARO/build/include/sound/hdaudio_ext.h
/usr/lib/modules/5.10.30-1-MANJARO/build/include/sound/hda_component.h
/usr/lib/modules/5.10.30-1-MANJARO/build/include/sound/hdaudio.h
/usr/lib/modules/5.10.30-1-MANJARO/build/include/sound/hda_hwdep.h
/usr/lib/modules/5.10.30-1-MANJARO/build/include/sound/hda_verbs.h
/usr/lib/modules/5.10.30-1-MANJARO/build/include/sound/hda_i915.h
child process terminated.
```

7. 编写程序，要求父进程创建一个子进程，子进程对一个 50*50 的字符数组赋值，由父进程改变子进程的优先级，观察不同优先级进程使用 CPU 的时间。

代码如下：

```
#include <stdio.h>

#include <unistd.h>

#include <sys/resource.h>

#include <time.h>

#include <stdlib.h>

#include <sys/times.h>

void time_print(char *str,clock_t time){

    long tps=sysconf(_SC_CLK_TCK);

    printf("%s:%6.2f secs",str,(float)time/tps);

}

int main(){

    pid_t pid;

    clock_t start,end;

    struct tms t_start,t_end;

    pid=fork();

    start=times(&t_start);

    if(pid>0){

        //在父进程中设置子进程优先级

        setpriority(PRIO_PROCESS,pid,20);

        //输出修改后的子进程的优先级

        printf("the priority of son process is%d",getpriority(PRIO_PROCESS,pid)); }

    //子进程执行代码

    else{

        int i,j,shu[50][50];

        for(i=0;i<50;i++)

            for(j=0;j<50;j++)

                shu[i][j]=i+j;
```

```

        system("grep the /usr/*/.* >/dev/null 2>/dev/null");
    }

    end=times(&t_end);

    time_print("\nelapsed",end-start);

    printf("\nparent time");

    time_print("\tuser CPU",t_end.tms_utime);

    time_print("\tsys CPU",t_end.tms_stime);

    printf("\nchild time");

    time_print("\tuser CPU",t_end.tms_cutime);

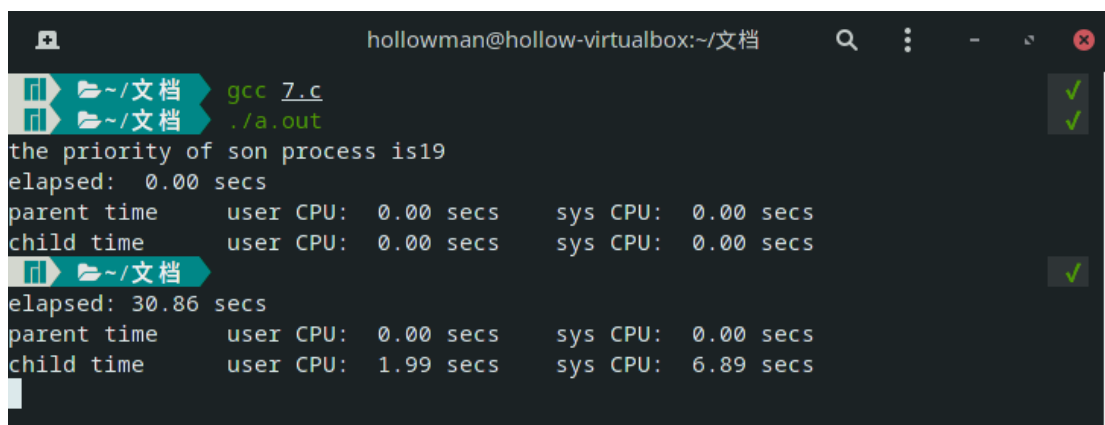
    time_print("\tsys CPU",t_end.tms_cstime);

    printf("\n");

    exit(0);
}

```

运行结果：



The screenshot shows a terminal window with the following output:

```

hollowman@hollow-virtualbox:~/文档
~/文档 gcc 7.c ✓
~/文档 ./a.out ✓
the priority of son process is19
elapsed: 0.00 secs
parent time      user CPU: 0.00 secs    sys CPU: 0.00 secs
child time       user CPU: 0.00 secs    sys CPU: 0.00 secs
~/文档 ✓
elapsed: 30.86 secs
parent time      user CPU: 0.00 secs    sys CPU: 0.00 secs
child time       user CPU: 1.99 secs    sys CPU: 6.89 secs

```