LAB2 文档

本机测试环境：ubuntu 18.04　目录中 1.txt 不可删除

编译：在目录下执行 make 即可

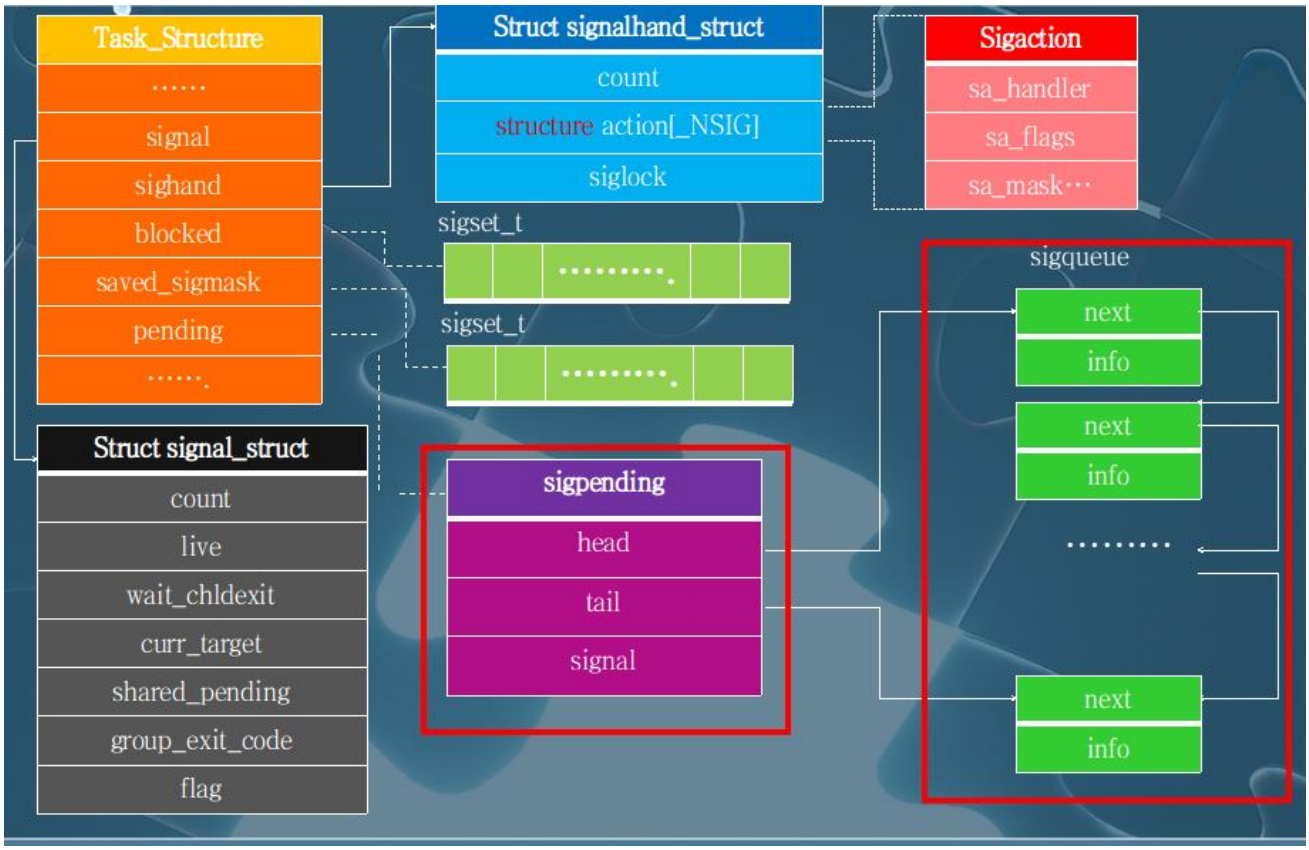
运行：./aaa <信号，如 SIGFPE>

./bbb

./ccc

1 Linux kernel 处理信号的机制描述

1.1　信号在 kernel 中的数据结构



信号在内核的数据结构如上图所示

在进程描述符 task_structure 中，有几个和信号相关的变量：

```
struct task_struct {
    ...
    struct signal_struct    *signal;
    struct sighand_struct    *sighand;
    sigset_t      blocked;
    struct sigpending    pending;
    ...
}
```

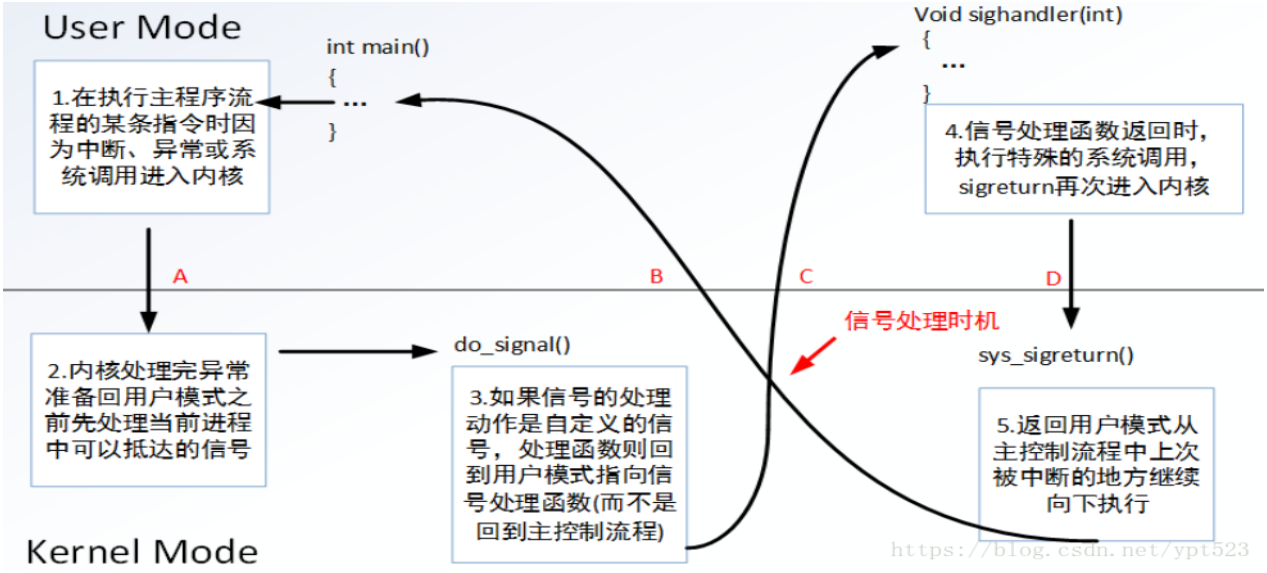
signal 指向结构 signal_struct，是用来跟踪挂起信号的结构，它是线程所在的线程中共享的，其中的 share_pending 是线程组共享的挂起信号队列

sighand 指向结构 sighand_struct，是用来描述每个信号必须怎样被处理的，即描述信号处理函数，其中的 sigaction 结构，是用来记录每个处理函数的信息，如果处理函数指向指针的，这些都是线程组共享的

pending 指向结构 sigpending，这是该线程私有的挂起信号队列，记录该进程的私有信号

blocked 则是阻塞的信号位集

1.2 信号捕获函数的调用机制



如上图所示，程序从内核态转回用户态之前，首先会检查当前进程的信号集，查看是否需要处理的信号，

如果没有需要处理的信号，则返回用户态回到程序主流程

如果存在需要处理的且未被阻塞的信号，则回到用户态，执行处理函数，处理函数执行完毕之后，会以 sigreturn 的方式再次返回内核态，之后再从内核态返回用户态，进行程序的主流程。

1.2 实验观察

```
#测试函数代码
#测试流程：编译 gcc -g main.c  -o main
#以此编译，允许产生 core 文件，采用命令 ./main 运行程序
#在另一个终端采用 kill 命令给该进程发送不同的信号查看行为
#include <stdio.h>

int main() {
    while(1);
}
```

1.3.1. 异常终止 SIGABRT 6

接收到信号行为：

```
chen@chen:~/chen/systempg/lab2code$ ./main
已放弃 (核心已转储)
chen@chen:~/chen/systempg/lab2code$ gdb main core
```

查看产生的 core 文件

```
Program terminated with signal SIGABRT, Aborted.
#0  main () at main.c:4
4           while(1);
```

可知是在 while 处接收到 SIGABRT 信号，与实验一致，

说明程序接收到 SIGABRT 的默认行为是终止生成 core 文件

1.3.2. 使暂停进程继续 SIGCONT 18

```
chen@chen:~/chen/systempg/lab2code$ ps -as | grep ./main
 1000  14887 0000000000000000 0000000000000000 0000000000000000 0000000000000000 R+   pts/0      0:10 ./main
 1000  14889 0000000000000000 0000000000000000 0000000000000000 0000000180000000 S+   pts/1      0:00 grep --
chen@chen:~/chen/systempg/lab2code$ kill -19 14887
chen@chen:~/chen/systempg/lab2code$ ps -as | grep ./main
 1000  14887 0000000000000000 0000000000000000 0000000000000000 0000000000000000 T    pts/0      0:22 ./main
 1000  14892 0000000000000000 0000000000000000 0000000000000000 0000000180000000 S+   pts/1      0:00 grep --
chen@chen:~/chen/systempg/lab2code$ kill -18 14887
chen@chen:~/chen/systempg/lab2code$ ps -as | grep ./main
 1000  14887 0000000000000000 0000000000000000 0000000000000000 0000000000000000 R    pts/0      0:25 ./main
 1000  14894 0000000000000000 0000000000000000 0000000000000000 0000000180000000 S+   pts/1      0:00 grep --
chen@chen:~/chen/systempg/lab2code$
```

运行 main，采用 ps -as | grep ./main 查看状态，第一次发现 main 是运行状态 R

之后发送 SIGSTOP 信号给 main，查看状态，发现 main 的状态变成暂停状

态 T

接着发送 SIGCONT 信号给 main，打印状态，发现进程继续运行，状态为 R

```
chen@chen:~/chen/systempg/lab2code$ ps -as | grep ./main
 1000  14887 0000000000000000 0000000000000000 0000000000000000 0000000000000000 R    pts/0      8:26 ./main
 1000  14922 0000000000000000 0000000000000000 0000000000000000 0000000180000000 S+   pts/1      0:00 grep --
chen@chen:~/chen/systempg/lab2code$ kill -18 14887
chen@chen:~/chen/systempg/lab2code$ ps -as | grep ./main
 1000  14887 0000000000000000 0000000000000000 0000000000000000 0000000000000000 R    pts/0      8:44 ./main
 1000  14924 0000000000000000 0000000000000000 0000000000000000 0000000180000000 S+   pts/1      0:00 grep --
```

运行状态下再发送信号，查看状态，发现状态还是 R

说明程序接收到 SIGCONT 的默认行为是继续运行（暂停状态下）

或者忽略（运行状态下）

1.3.3. 算数异常 SIGFPE 8

```
chen@chen:~/chen/systempg/lab2code$ ./main
[2]+  已杀死               ./main
浮点数例外 (核心已转储)
chen@chen:~/chen/systempg/lab2code$ gdb main core
```

```
Core was generated by `./main'.
Program terminated with signal SIGFPE, Arithmetic exception.
#0  main () at main.c:4
4           while(1);
```

如图所示，main 接收到 SIGFPE 信号，终止并且生成 core 文件，gdb 调试 core，发现在 whil 出发送 SIGFPE 信号

说明进程接收到 SIGFPE 信号的默认终止，并且生成 core 文件

1.3.4. 键盘状态请求 SIGINFO

该信号系统内没有，无法测试

1.3.5. 终端中断符 SIGINO 2

```
chen@chen:~/chen/systempg/lab2code$ ./main
^C
chen@chen:~/chen/systempg/lab2code$
```

实验输入 ctrl+C,进程直接退出，也没有生成 core 文件

说明 SIGINT 的默认行为是终止

1.3.6. 异步 IO SIGIO 29

```
chen@chen:~/chen/systempg/lab2code$ ./main
I/O 可行
chen@chen:~/chen/systempg/lab2code$
```

实验结果是进程终止，并打印 I/O 可行，也没有 core 文件

说明 SIGIO 的默认行为是终止（书本所说的 忽略 行为没有测试到）

1.3.7. 终止 SIGKILL 9

```
chen@chen:~/chen/systempg/lab2code$ ./main
已杀死
chen@chen:~/chen/systempg/lab2code$
```

实验结果是进程终止，并打印已杀死，没有产生 core 文件

说明 SIGKILL 的默认行为是终止进程

1.3.8. 资源丢失 SIGLOST

该信号系统内没有，无法测试

1.3.9. 写至无毒进程的管道 SIGPIPE 13

```
chen@chen:~/chen/systempg/lab2code$ ./main
chen@chen:~/chen/systempg/lab2code$
```

运行后，给进程发送 SIGPIPE 信号，进程直接退出，不打印任何内容，不产生 core 文件

说明 SIGPIPE 的默认行为是终止

1.3.10. 终端退出符 SIGQUIT 3

```
chen@chen:~/chen/systempg/lab2code$ ./main
退出 (核心已转储)
chen@chen:~/chen/systempg/lab2code$ gdb main core
Core was generated by `./main'.
Program terminated with signal SIGQUIT, Quit.
#0  main () at main.c:6
6           while(1);
```

进程收到 SIGQUIT 信号，退出，并打印“退出（核心已转储）”，并产生 core 文件，查看 core 文件，发现接收到 SIGQUIT 退出，与实验相符

说明 SIGQUIT 的默认行为是终止并产生 core 文件

1.3.11. 后台读控制 tty SIGTTIN 21

```
chen@chen:~/chen/systempg/lab2code$ ./main

[1]+  已停止                 ./main
```

```
chen@chen:~/chen/systempg/lab2code$ ps -as|grep ./main
 1000   3380 0000000000000000 0000000000000000 0000000000000000 0000000000000000 T    pts/0      0:13 ./main
 1000   3383 0000000000000000 0000000000000000 0000000000000000 0000000180000000 S+   pts/1      0:00 grep --
```

进程收到 SIGTTIN 信号，打印 “已停止”，查看进程，发现 main 进程处于 T 暂停状态

说明 SIGTTIN 的默认行为是暂停进程。

1.3.12. 后台写向控制 tty SIGTTOU 22

```
chen@chen:~/chen/systempg/lab2code$ ./main

[1]+  已停止                 ./main
```

```
chen@chen:~/chen/systempg/lab2code$ kill -SIGTTOU 3388
chen@chen:~/chen/systempg/lab2code$ ps -as | grep ./main
 1000   3388 0000000000000000 0000000000000000 0000000000000000 0000000000000000 T    pts/0      0:20 ./main
 1000   3393 0000000000000000 0000000000000000 0000000000000000 0000000180000000 S+   pts/1      0:00 grep --
```

进程收到 SIGTTOU 信号，打印 “已停止”，查看进程，发现 main 进程处于 T 暂停状态

说明 SIGTTOU 的默认行为是暂停进程。

1.3.13 无效内存引用 SIGSEGV 11

```
chen@chen:~/chen/systempg/lab2code$ ./main
段错误 (核心已转储)
chen@chen:~/chen/systempg/lab2code$ gdb main core
```

```
Core was generated by `./main'.
Program terminated with signal SIGSEGV, Segmentation fault.
#0  main () at main.c:6
6           while(1);
```

进程接收到 SIGSEGV 信号，进程退出，打印 “段错误（核心已转储）”，并产生 core 文件，查看 core 文件，发现其收到 SIGSEGV 信号退出，与实验一致

说明 SIGSEGV 的默认行为是终止并产生 core 文件

1.3.14. 无效系统调用 SIGSYS 31

```
chen@chen:~/chen/systempg/lab2code$ ./main
错误的系统调用（核心已转储）
chen@chen:~/chen/systempg/lab2code$ gdb main core
```

```
Core was generated by `./main'.
Program terminated with signal SIGSYS, Bad system call.
#0  main () at main.c:6
6           while(1);
```

进程接收到 SIGSYS 信号，退出并打印 “错误的系统调用（核心已转储）”，并产生 core 文件，查看 core 文件，发现其收到 SIGSYS 信号退出，与实验一致

说明 SIGSYS 的默认行为是终止并产生 core 文件

1.3.15. 用户定义信号 SIGUSR1/2 10/12

```
chen@chen:~/chen/systempg/lab2code$ ./main
用户定义信号 1
chen@chen:~/chen/systempg/lab2code$ ./main
用户定义信号 2
chen@chen:~/chen/systempg/lab2code$
```

运行两次进程，分别发送 SIGUSR1 和 SIGUSR2，两次实验结果都是直接退出，并打印出 用户定义信号 1 和 用户定义信号 2，没有产生 core 文件

说明 SIGUSR1/2 的默认行为是退出

1.3.16. 超过文件长度限制 SIGXFSZ 25

```
chen@chen:~/chen/systempg/lab2code$ ./main
文件大小超出限制（核心已转储）
chen@chen:~/chen/systempg/lab2code$ gdb main core
```

```
Core was generated by `./main'.
Program terminated with signal SIGXFSZ, File size limit exceeded.
#0  main () at main.c:6
6           while(1);
```

进程接收到 SIGXFSZ 信号终止，打印“文件大小超出限制（核心已转储）”，产生 core 文件，查看 core，发现其接收到 SIGXFSZ 信号退出，与实验一致

说明 SIGXFSZ 的默认行为是终止并产生 core 文件

2 捕获程序错误信号 aaa.c 程序设计

```
chen@chen:~/chen/systempg/lab2code$ gcc aaa.c -o aaa
aaa.c: In function ‘sigfpe’:
aaa.c:38:16: warning: division by zero [-Wdiv-by-zero]
     int a = 10 / 0;
               ^
chen@chen:~/chen/systempg/lab2code$ ./aaa SIGFPE
捕捉到SIGFPE信号，程序发生错误
chen@chen:~/chen/systempg/lab2code$ ./aaa SIGSEGV
捕捉到SIGSEGV信号，程序发生错误
chen@chen:~/chen/systempg/lab2code$ ./aaa SIGILL
捕捉到SIGILL信号，程序发生错误
chen@chen:~/chen/systempg/lab2code$ ./aaa SIGABRT
捕捉到SIGABRT信号，程序发生错误
chen@chen:~/chen/systempg/lab2code$ ./aaa SIGBUS
捕捉到SIGBUS信号，程序发生错误
chen@chen:~/chen/systempg/lab2code$ ./aaa SIGPIPE
客户端启动成功
hello
hello
捕捉到SIGPIPE信号，程序发生错误
chen@chen:~/chen/systempg/lab2code$
```

aaa.c 程序运行结果如上所示。

aaa.c 程序编译命令：gcc aaa.c -o aaa

运行命令 ./aaa <信号类别 如果 SIGFPE>

在 aaa.c 中，实现了产生与捕获 6 种信号,主体逻辑是

1、 为信号绑定信号处理事件，然后根据输入的参数，执行对应产生信号的函数

2、 信号函数执行不同的代码以产生对应函数

3、 产生信号后，捕获信号，处理函数判断信号类别，输出内容，之后退出程序。

程序中的六种信号与产生方式：

1、SIGFPE 除 0 产生

2、SIGSEGV 访问空指针的数据

3、SIGILL 查遍资料，不知道如何产生，只能通过 raise 函数产生

4、SIGABRT 调用 abort 函数产生

5、SIGBUS 进行两个 mmap，之后进行 memcpy，copy 时 des 的大小小于 src 的大小，产生 SIGBUS 信号，注意，文件夹底下有个 1.txt,用于 mmap 时使用，不可删除

6、SIGPIPE fork 子进程，进行 socket 连接，父进程为 server，子进程为 client，子进程连接完 server 后退出，server 不停的给 client 发送消息，client 退出，pipe 没有读进程，server 再次写的时候就会产生 SIGPIPE 信号。

3 捕获 CTRL-C 信号程序设计

```
chen@chen:~/chen/systempg/lab2$ gcc signal_handler.c -o signal_handler
chen@chen:~/chen/systempg/lab2$ ./signal_handler
^C你确定要退出吗?
^C
chen@chen:~/chen/systempg/lab2$
```

signal_handler.c 运行结果如上图所示

signal_handler.c

编译命令：gcc signal_handler.c -o signal_handler

运行命令：./signal_handler

实现逻辑：

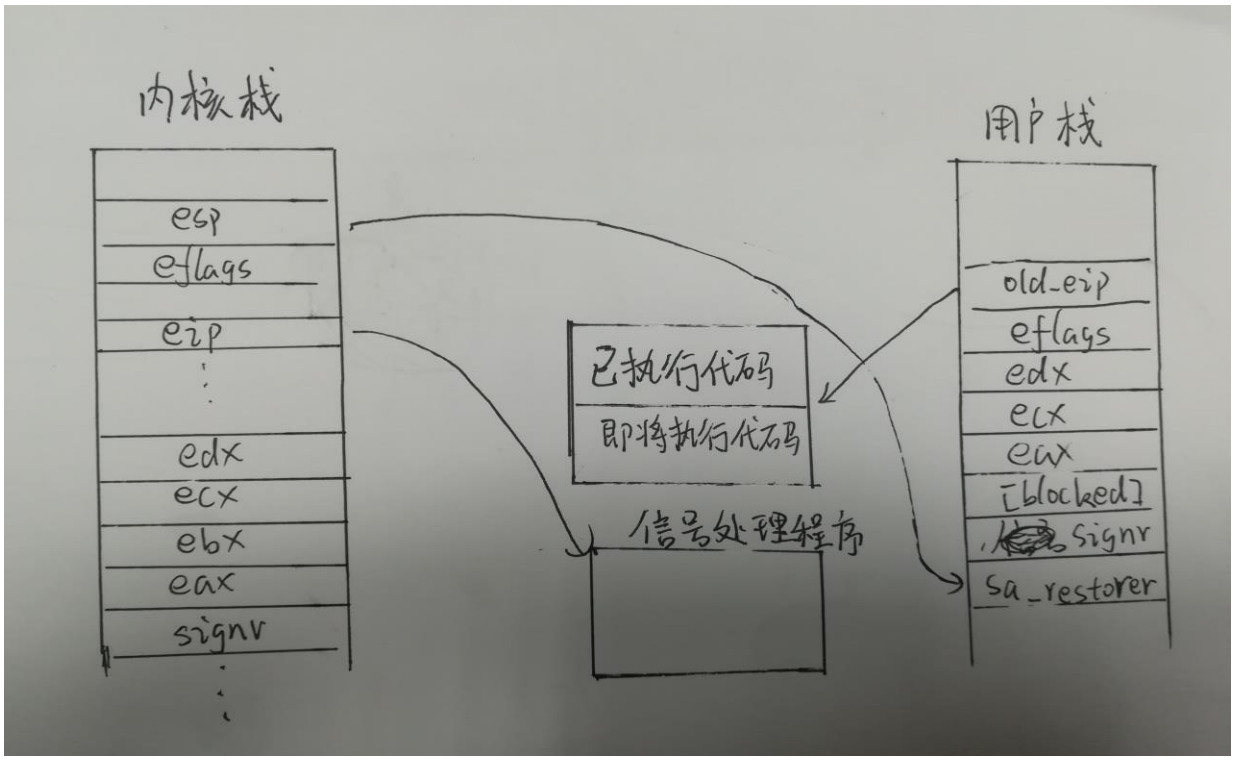
1、 程序运行开始，绑定 SIGINT 的信号处理事件。

2、 进入 while 死循环，等待 sigint 信号

3、 捕捉到 sigint 信号后，执行信号处理函数，信号处理函数会打印问句，并且解绑处理函数，再次进入死循环

4、 再次捕捉到 sigint 则自动退出。

堆栈示意：



在内核调用处理函数是，内核会把 eip 即下次执行代码地址指向信号处理程序，并把必要寄存处存储起来，并把返回地址指向 sa_restore，信号处理函数返回之后，sa_restore 会进行寄存器恢复工作，恢复之后，再次进入主程序，回到断点，执行主程序代码

4 bbb 和 ccc 程序设计

```
chen@chen:~/chen/systempg/lab2code$ gcc bbb.c -o bbb
chen@chen:~/chen/systempg/lab2code$ gcc ccc.c -o ccc
chen@chen:~/chen/systempg/lab2code$ ./bbb
b11111111
c22222222
b33333333
c44444444
b55555555
c66666666
b77777777
c88888888
chen@chen:~/chen/systempg/lab2code$
```

bbb 与 ccc 程序运行结构如上所示，

bbb.c 编译命令 gcc bbb.c -o bbb

ccc.c 编译命令 gcc ccc.c -o ccc

运行命令 ./bbb

程序逻辑：

1、 bbb 中绑定 SIGUSR1 信号处理事件

2、 bbb fork 出子进程，并 sleep 两秒

3、 子进程进行 execvp 操作，用于运行 ccc

4、 ccc 中绑定信号函数，获取父进程 pid，进行 sigsuspend

5、 bbb 休眠结束，打印出 b11111111，发送信号 SIGUSR1 给 ccc，进入 sigsuspend

6、 ccc 接收信号,执行信号处理函数,发送信号给 bbb,再次进入 sigsuspend

7、 bbb 接收信号，打印信息，发送信号给 ccc，进入 sigsuspend

8、 循环反复，直到 i 到临界值，退出循环

5 shell lab 信号处理分析

经过查看代码，可知 shell lab 中主要的信号处理主要涉及四个信号：SIGINIT、SIGTSTP、SIGCHLD、SIGQUIT

分析代码可知：

1、 对于 SIGINT 的处理是，判断 pid 对应的进程是否是前台进程，如果是的话，则发送 SIGINT 信号给对应 pid 进程，如果 pid 不是前台进程，则忽略。

2、 对于 SIGTSTP 的处理是，判断 pid 对应的进程是否是前台进程，如果是的话，则发送 SIGSTP 信号给对应 pid 进程，如果 pid 不是前台进程，则忽略。

这两个信号都是通过键盘输入发送的，所以针对前台进程，在程序中每个进程都有不同的 pid、不同的 pgid，这样实现确保了后台进程不受到前台键盘输入的影响

3、 对 SIGCHLD 的处理是，因为进程收到 SIGCHLD 的原因有几种：子进程终

止、子进程因为收到 SIGSTOP 或 SIGTSTP 而挂起。所以该进程会根据情况进行区分：

如果子进程正常终止，则删除子进程的 job，回收资源

如果子进程因为信号异常终止，则打印出信号，并删除子进程 job，回收资源

如果子进程是因为收到 SIGSTOP 或 SIGTSTP 而挂起，则修改对应 job 的状态为暂停状态

4、 对 SIGQUIT 的处理是打印出退出信息，然后整个进程退出