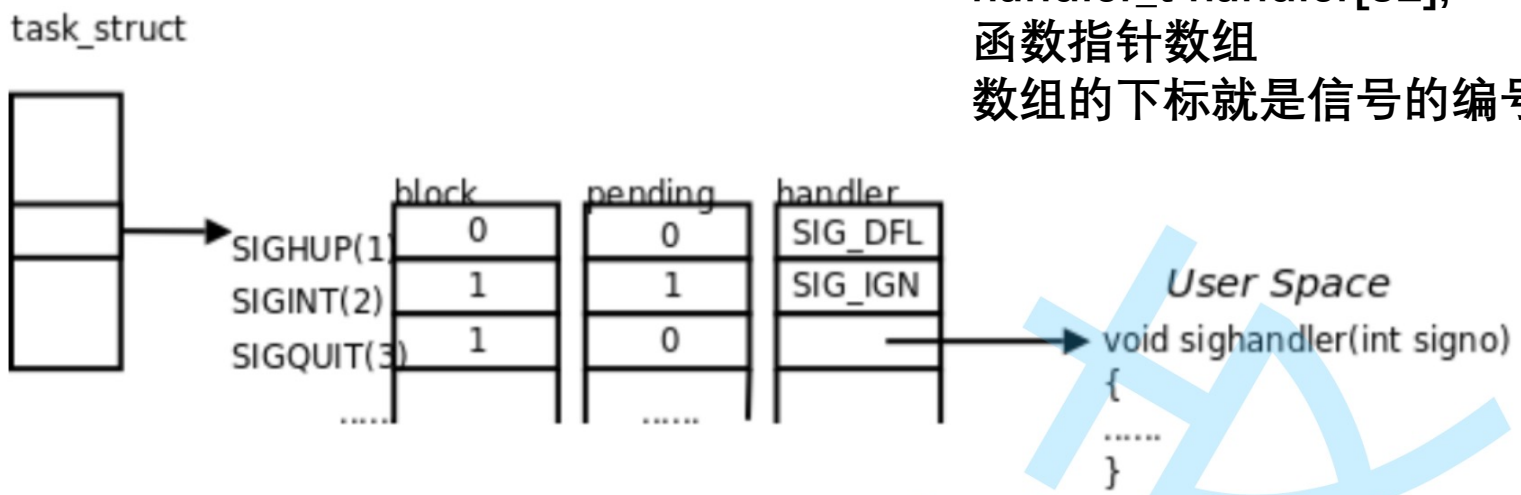
0105信号

1. 信号其他相关常见概念

- 实际执行信号的处理动作称为信号递达(Delivery)
- 信号从产生到递达之间的状态,称为信号未决(Pending)。
- 进程可以选择阻塞 (Block)某个信号。
- 被阻塞的信号产生时将保持在未决状态,直到进程解除对此信号的阻塞,才执行递达的动作.
- 注意,阻塞和忽略是不同的,只要信号被阻塞就不会递达,而忽略是在递达之后可选的一种处理动作。

1. 如果一个信号被阻塞，那么它永远都不会递达，除非被解除阻塞
2. 忽略是递达之后的一种动作



handler_t handler[32];
函数指针数组
数组的下标就是信号的编号

递达有三种处理方式：
1. **默认**
2. **忽略**
3. **自定义**

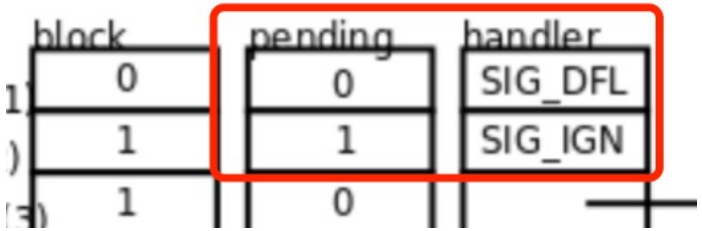
自定义我们上节课已经讲过了
其实我们调用signal函数
其实就是把pending表填上我们传过去的信号编号，然后handler表填上处理方法。
这，就是信号递达的自定义处理方法。

那么
忽略和默认怎么去理解呢？

```c
int main()
{
    //处理信号
    //signal(信号编号,方法)
    signal(2,SIG_IGN);
    signal(2,SIG_DFL);
    return 0;
}
```

忽略

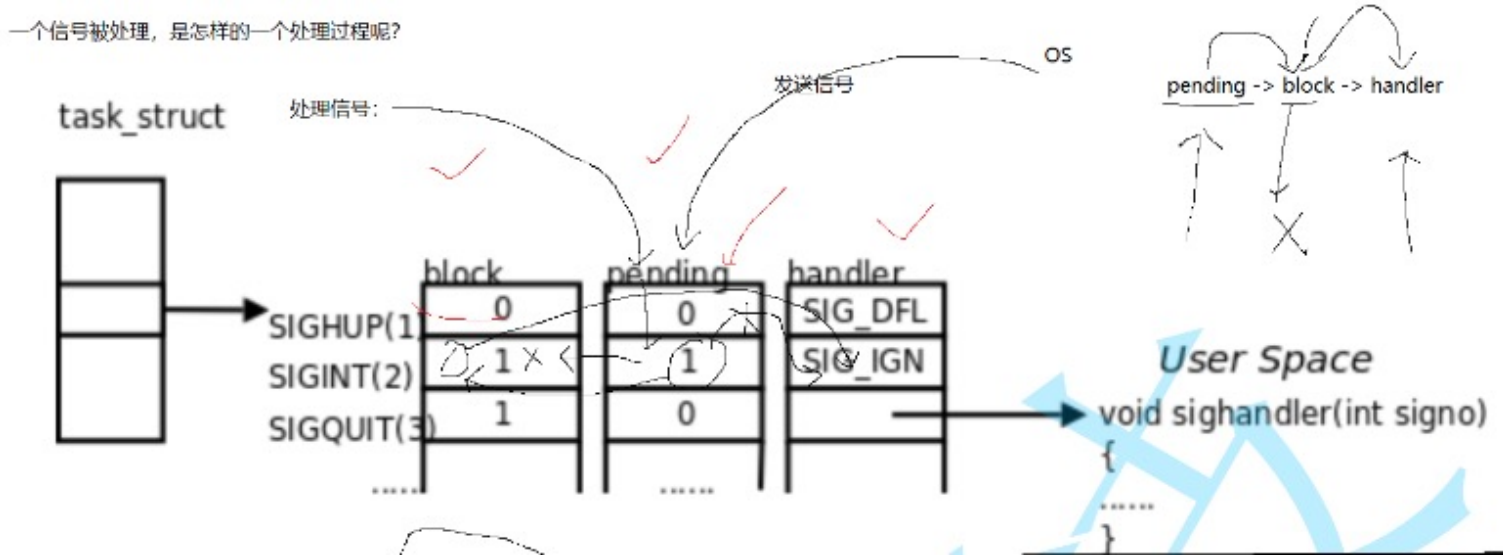
默认



所以，再重复一次：
调用signal不是调用handler方法
而是把东西填到pcb里面的这些表中而已。

```c
/* Fake signal functions.  */
#define SIG_ERR   ((__sighandler_t) -1)      /* Error return.  */
#define SIG_DFL   ((__sighandler_t) 0)    /* Default action.  */
#define SIG_IGN   ((__sighandler_t) 1)    /* Ignore signal.  */
```

那么block表
也是一个位图！
结构和pending一模一样
位图中的内容代表的含义是对应的信号是否被阻塞。



发送信号，其实就是填pending

处理信号：
首先在pending里面找
找到1了，说明有这个信号，
但是此时，不是直接调用对应的handler，而是先检查一下，是否有被block

3. sigset_t

从上图来看,每个信号只有一个bit的未决标志,非0即1,不记录该信号产生了多少次,阻塞标志也是这样表示的。因此,未决和阻塞标志可以用相同的数据类型sigset_t来存储,sigset_t称为信号集,这个类型可以表示每个信号的"有效"或"无效"状态,在阻塞信号集中"有效"和"无效"的含义是该信号是否被阻塞,而在未决信号集中"有效"和"无效"的含义是该信号是否处于未决状态。下一节将详细介绍信号集的各种操作。 阻塞信号集也叫做当前进程的信号屏蔽字(Signal Mask),这里的"屏蔽"应该理解为阻塞而不是忽略。

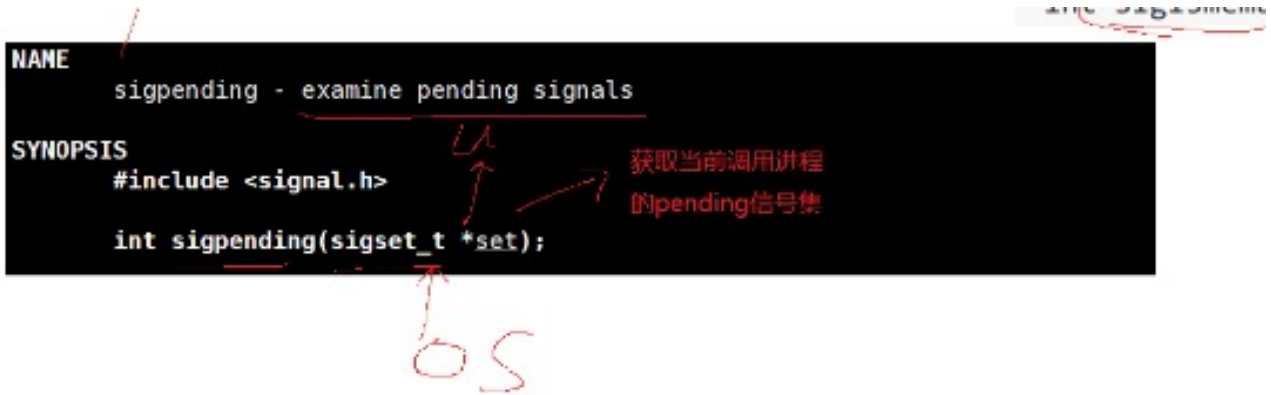
三张表，以后我们这样叫：

1. **信号屏蔽字**
2. **pending信号集**
3. **handler处理方法表**

sigset_t --- 不允许用户自己进行位操作 --- OS给我们提供了对应的操作位图方法
sigset_t --- user是可以自己使用该类型 --- 和内置类型 && 自定义类型 没有任何差别
sigset_t --- 一定需要对应的系统接口，来完成对应的功能，其中系统接口需要的参数，可能就包含了sigset_t 定义的变量或者对象

```
#include <signal.h>
int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset (sigset_t *set, int signo);
int sigdelset(sigset_t *set, int signo);
int sigismember (const sigset_t *set, int signo);
```

- 函数sigemptyset初始化set所指向的信号集,使其中所有信号的对应bit清零,表示该信号集不包含 任何有效信号。
- 函数sigfillset初始化set所指向的信号集,使其中所有信号的对应bit置位,表示 该信号集的有效信号包括系统支持的所有信号。
- 注意,在使用sigset_ t类型的变量之前,一定要调 用sigemptyset或sigfillset做初始化,使信号集处于确定的状态。初始化sigset_t变量之后就可以在调用sigaddset和sigdelset在该信号集中添加或删除某种有效信



SIG_BLOCK	set包含了我们希望添加到当前信号屏蔽字的信号，相当于mask=mask|set
SIG_UNBLOCK	set包含了我们希望从当前信号屏蔽字中解除阻塞的信号，相当于mask=mask&~set
SIG_SETMASK	设置当前信号屏蔽字为set所指向的值，相当于mask=set

写代码：

1. 如果我们对所有的信号都进行了自定义捕捉 --- 我们是不是就写了一个不会被异常或者用户杀掉的进程？？可以吗？？
2. 如果我们将2号信号block，并且不断的获取当前并打印当前进程的pending信号集，如果我们发送一个2号信号，我们就应该肉眼看到pending信号集中，有一个比特位0->1
3. 如果我们对所有的信号都进行block，那是不是和1一样？

```
void catchSig(int signum)
{
    std::cout << "获得了一个信号: " << signum << std::endl;
}
int main()
{
    // 处理信号
    // signal(信号编号,方法)
#if 0
    signal(2,SIG_IGN);
    signal(2,SIG_DFL);
#endif
    for (int i = 0; i <= 31; i++)
    {
        signal(i, catchSig);
    }
    while (true)
    {
        std::cout << "pid: " << getpid() << std::endl;
        sleep(1);
    }
    return 0;
}
```

捕捉所有的信号

好像信号都被捕捉了
但是！
9号信号
是不可被捕捉的！

```
OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

pid: 31463
pid: 31463
获得了一个信号：2
pid: 31463
pid: 31463
pid: 31463
pid: 31463
^C获得了一个信号：2
pid: 31463
pid: 31463
pid: 31463
pid: 31463
pid: 31463
pid: 31463
pid: 31463
pid: 31463
pid: 31463
pid: 31463
Killed
(base) [yufc@VM-12-12-centos:~/Files/BitCodeField/0105]$
```

```
(base) [yufc@VM-12-12-centos:~/Files]$ kill -1 31464
bash: kill: (31464) - No such process
(base) [yufc@VM-12-12-centos:~/Files]$ kill -1 31463
(base) [yufc@VM-12-12-centos:~/Files]$ kill -2 31463
(base) [yufc@VM-12-12-centos:~/Files]$ kill -9 31463
(base) [yufc@VM-12-12-centos:~/Files]$
```

```cpp
// test2
static void showPending(sigset_t &pending)
{
    for (int sig = 1; sig <= 31; sig++)
    {
        if (sigismember(&pending, sig))
        {
            std::cout << "1";
        }
        else
            std::cout << "0";
    }
    std::cout << std::endl;
}
```

```
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
^C01000000000000000000000000000000
01000000000000000000000000000000
01000000000000000000000000000000
^C01000000000000000000000000000000
01000000000000000000000000000000
```

```cpp
int main()
{
    // 1. 定义信号集对象
    sigset_t bset, obset;
    sigset_t pending;
    // 2. 初始化
    sigemptyset(&bset);
    sigemptyset(&obset);
    sigemptyset(&pending);
    // 3. 添加我们要进行屏蔽的信号
    sigaddset(&bset, 2);
    // 4. 设置set到内核对应的进程当中 [默认情况：进程不会对任何信号进行block]
    int n = sigprocmask(SIG_BLOCK, &bset, &obset);
    assert(!n);
    (void)n;
    std::cout << "block 2号信号成功..." << std::endl;
    // 5. 重复打印当前进程的pending信号集
    while (true)
    {
        // 获取当前进程的pending信号集
        sigpending(&pending);
        // 现实pending信号集中没有被递达的信号
        showPending(pending);
        sleep(1);
    }
    return 0;
}
```

此时我们可以肉眼看到
发送2号信号之后
有一个比特位的0变成1了

如果我们恢复之后，1也会变成0

```cpp
int count = 0;
while (true)
{
    // 获取当前进程的pending信号集
    sigpending(&pending);
    // 现实pending信号集中没有被递达的信号
    showPending(pending);
    sleep(1);
    count++;
    if (count == 10)
    {
        // 恢复
        int n = sigprocmask(SIG_SETMASK, &obset, nullptr);
        //因为前面我们得到了老的block，即obset
        //我们直接用obset去设置就行了，用SIG_SETMASK选项
        //当然，用SIG_UNBLOCK也是可以的
        assert(n == 0);
        (void)n;
        std::cout << "解除对于2号信号的block" << std::endl;
    }
}
return 0;
```

恢复

我们期望看到的是
有一个比特位
从1变成0

```
g++ -o mysignal mysignal.cc -std=c++11
(base) [yufc@VM-12-12-centos:~/Files/BitCodeField/0105]$ ./mysignal
block 2号信号成功...
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
^C01000000000000000000000000000000
01000000000000000000000000000000
01000000000000000000000000000000
01000000000000000000000000000000
01000000000000000000000000000000
01000000000000000000000000000000

(base) [yufc@VM-12-12-centos:~/Files/BitCodeField/0105]$
```

为什么程序终止了?
没有看到我们期望的现象

因为2号信号默认是终止进程

解除block之后，2号信号递达
进程就被终止了

所以我们来捕捉一下二号信号

```
void catchSig(int signum)
{
    std::cout << "获得了一个信号: " << signum << std::endl;
}
int main()
{
    // 0. 为了方便测试，捕捉2号信号
    signal(2, catchSig);
    // 1. 定义信号集对象
```

为什么没有设置
pending位图的接口？
没必要
我们每一个发信号的动作就是在修改pending位图

```
(base) [yufc@VM-12-12-centos:~/Files/BitCodeField/0105]$ make
g++ -o mysignal mysignal.cc -std=c++11
(base) [yufc@VM-12-12-centos:~/Files/BitCodeField/0105]$ ./mysignal
block 2号信号成功...
00000000000000000000000000000000
00000000000000000000000000000000
^C01000000000000000000000000000000
01000000000000000000000000000000
01000000000000000000000000000000
01000000000000000000000000000000
01000000000000000000000000000000
01000000000000000000000000000000
01000000000000000000000000000000
01000000000000000000000000000000
获得了一个信号: 2
解除对于2号信号的block
00000000000000000000000000000000
00000000000000000000000000000000
```

那如果我们屏蔽所有信号呢？

```c
static void blockSig(int sig)
{
    // 这个接口的作用就是屏蔽sig信号
    sigset_t bset;
    sigemptyset(&bset);
    sigaddset(&bset, sig);
    int n = sigprocmask(SIG_BLOCK, &bset, nullptr);
    assert(!n);
    (void)n;
}
int main()
{
    // blockSig(2);
    for (int sig = 1; sig <= 32; sig++)
    {
        blockSig(sig);
    }
    sigset_t pending;
    while(true)
    {
        sigpending(&pending);
        showPending(pending);
        sleep(1);
    }
    return 0;
}
```

此时我们去给这个进程发信号
期望看到
一个一个比特位从0变成1

所以这里我们写一个脚本吧

```
(base) [yufc@VM-12-12-centos:~/Files]$ ps axj | grep mysignal
30431 14484 14484 30431 pts/74    14484 S+    1001   0:00 ./mysignal
14514 14601 14600 14514 pts/81    14600 S+    1001   0:00 grep --color=auto mysignal
(base) [yufc@VM-12-12-centos:~/Files]$
```

TERMINAL

```
00000000000000000000000000001000
00000000000000000000000000001000
00000000000000000000000000001000
00000000000000000000000000001000
00000000000000000000000000001000
00000000000000000000000000001000
00000000000000000000000000001000
00000000000000000000000000001000
00000000000000000000000000001000
```

mysignal.cc killProc.sh signum.h

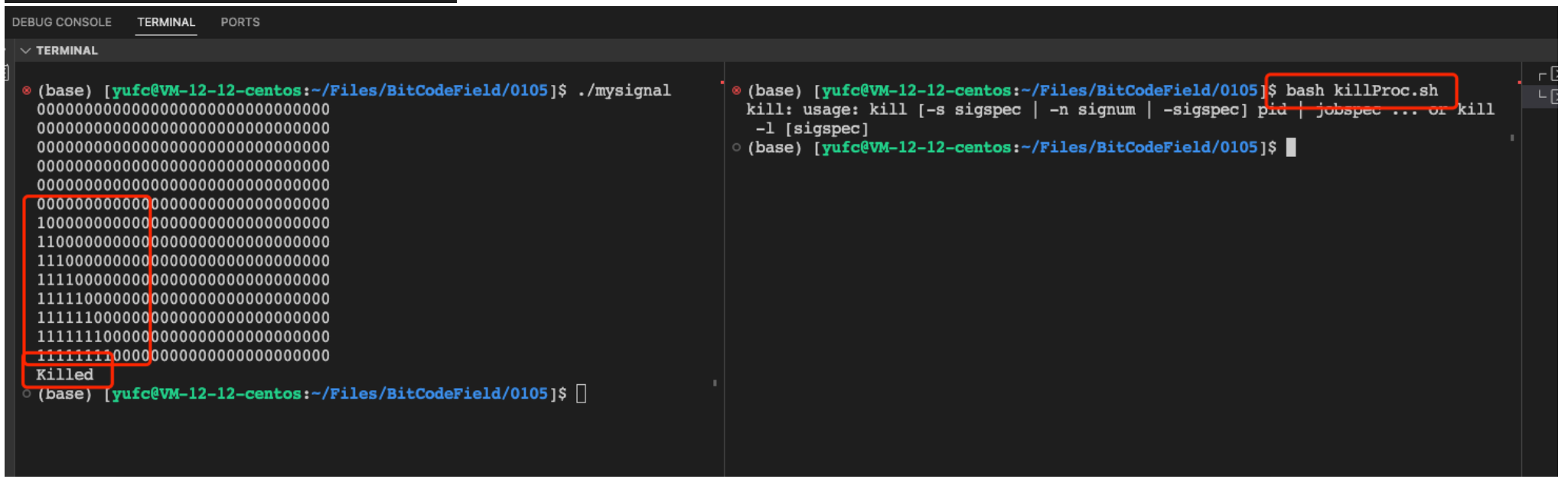
Files > BitCodeField > 0105 > killProc.sh

```
kill -1 $(pidof mysignal)
sleep 1
kill -2 $(pidof mysignal)
sleep 1
kill -3 $(pidof mysignal)
sleep 1
kill -4 $(pidof mysignal)
sleep 1
kill -5 $(pidof mysignal)
sleep 1
kill -6 $(pidof mysignal)
sleep 1
kill -7 $(pidof mysignal)
sleep 1
kill -8 $(pidof mysignal)
sleep 1
kill -9 $(pidof mysignal)
sleep 1
kill -10 $(pidof mysignal)
```

所以，通过这个结果我们可以得到结论：

9号信号不能被屏蔽或者阻塞！

如果我们遇到9的时候跳过
我们会发现，19号信号也屏蔽不了



学到这里，信号保存我们搞定！

下面是，信号处理

信号产生之后，信号可能无法被立即处理
我们前面说：
在合适的时候处理（是什么时候？）

1. 合适的时候（是什么？）
2. 信号处理的整个流程