上面我们讲了信号的发送场景，接下来我们讲一讲信号是怎么发送到进程的，即信号的投递。我们先讲一下进程的接收装置和信号来了放哪里、怎么放。

🌟每个进程都有一个信号队列，每个线程也有一个信号队列。信号队列的数据结构如右边的代码所示。

可以看到信号队列非常简单，sigset是个bit flag，代表当前队列里有哪些信号，list是信号列表的头指针。

🌟下面我们来看一下信号队列里的条目。每发送一次信号都会生成一个sigqueue，sigqueue里面包含了很多和信号相关的信息。

🌟我们前面说了很多发送信号的方法，总体上可以分为两类，普通发送和强制发送。异常处理发送信号都是用的强制发送。这两类方法最终都会调用同一个函数来发送信号，即send-signal。

🌟send\_signal先做了一些简单的处理，然后直接调用另一个\_\_send\_signal。这个\_\_send\_signal先调用prepare\_signal，prepare\_signal对暂停恢复类的信号先做了一下预处理，然后查看信号是否被忽略。然后send-signal根据PID类型决定是把信号放到进程队列里还是线程队列里。然后会判断信号是不是标准信号,对于标准信号，如果信号队列里已经有一个了，就不再接收了，这么做是为了兼容过去。然后调用\_\_sigqueue\_alloc分配一个信号条目sigqueue，分配好之后填充各种数据，把它加入到队列中去。最后调用complete\_signal，这个函数会选择一个合适的线程来唤醒，一般会唤醒当前线程。

🌟然后我们来看强制发送：强制发送的入口函数是force\_sig\_info\_to\_task，它会先把信号的阻塞和忽略取消掉，然后再调用函数send\_signal进行发送。内核还封装了几个函数来辅助强制发送。

🌟接着我们来看普通发送。

它的主要发送函数是kill和tgkill:其中Kill是发送信号给进程，只能向整个进程发送信号

而Tgkill是发送信号给线程。允许向指定线程组中的特定线程发送信号

给进程发信号的接口函数最终都是调用group\_send\_sig\_info。

给线程发信号的接口函数最终都是调用do\_send\_specific。

因而kill调用前面这个函数，tgkill调用后面那个函数。

然后Kill函数是根据pid的值来决定向谁发送信号。

🌟 接下来我们看信号的处理

🌟信号的处理是在线程中进行的，因为线程是代码执行的单元。信号处理方式有三种：如果程序什么也没设置的话，走默认处理方式。默认处理又有五种情况，分别是ignore、term、core、stop、cont。还有两种方式是提前通过接口函数signal或者sigaction设置了处理方式，或者设置一个信号处理函数handler来处理信号。这里要注意默认处理中的忽略和进程主动设置的忽略，两者的逻辑是不同的。

🌟在信号处理之前我们要先看看信号的阻塞，阻塞的信号是暂时不被处理的。

当一个线程暂时不想接收信号的时候，比如说正在处理一个关键的任务，就可以暂时屏蔽信号，等任务完成之后就可以解决屏蔽了。信号屏蔽是线程局部的，每个线程都可以有自己私有信号屏蔽设置。屏蔽信号的接口函数有两个，如下所示。前者是单线程时代的接口函数，后者是多线程时代POSIX规定的接口函数。两者实际上没有区别，下面我们以后者为例来讲解一下。

sigset\_t是一个bit flag，参数set可以用来指定想屏蔽的信号，oldset用来返回线程之前的数据。how用来指定如何设置屏蔽参数，有三个值。

如果你要阻塞SIGKILL和SIGSTOP，此函数实现并不会返回错误值，而是会默默地忽略，返回设置成功。

🌟信号的忽略与捕获的设置方法是一样，忽略可以看成是一种特殊的捕获，相当于是信号处理函数是空函数。

设置信号处理方式的接口函数有两个，signal和sigaction。signal是早期的设置函数，适用于标准信号，比较简单。sigaction是后来新增的接口函数，功能比较强大。

我们先来看signal函数接口：

signal有两个参数，第一个是信号数值，第二个是信号处理函数。信号处理函数的接口如下所示：

这里的第二个参数可以传递特殊值SIG\_IGN，代表忽略这个信号，还可以传递特殊值SIG\_DFL，代表恢复信号的默认处理方式。

下面再来看sigaction函数的接口：

有三个参数，第一个参数是信号数值，第二个参数是要设置的情况，第三个参数会返回旧的设置情况，可以为NULL。

🌟信号处理是在线程从内核空间返回用户空间的时候处理的。下面我们以x86为例讲解一下。通过代码，可以看出线程在返回到用户空间之前不断地检查有没有信号要处理。如果有的话就使用函数get\_signal取出一个信号，然后在函数handle\_signal里面去执行。

🌟接着我们来看一下get-signal函数的流程

它首先看有没有STOP相关的信号，如果有的话执行处理。

然后去取一个信号出来，先取同步信号，如果没有的话就去取其它信号，还没有的话就再去进程的信号里面去取。

如果取到的信号的处理设置是忽略，或者默认处理方式是忽略，则继续取下一个信号。

如果取到的信号没有设置信号处理函数，则在这里执行其默认处理，终结进程或者coredump之后再终结进程。

最后，如果没有取到信号则返回值为0。如果取到了信号，且信号设置了信号处理函数则返回值为1。然后把ksig传递给函数handle\_signal来处理。

🌟接着我们来看一下handle\_signal函数，代码如下

🌟信号处理程序的代码是在用户态的，而从系统调用返回到用户态前还是属于内核态，CPU是禁止内核态执行用户态代码的，那么怎么办？答案是先返回到用户态执行信号处理程序; 然后再返回到内核态，在内核态完成收尾工作。

这里又有两个问题：

第一个是怎么从内核态返回到用户态：

CPU要从内核栈中找到返回到用户态的地址，我们把这个返回地址修改为信号处理程序的入口(即修改内核态eip)，这样当从系统调用返回到用户态时，就可以执行信号处理程序了。

这个过程是通过调用setup\_frame函数来实现的

第二个问题是怎么返回到内核态并恢复原来内核栈的内容：在用户态栈空间构建一个 Frame（帧）；setup\_frame() 函数把原来内核栈的内容保存到帧中。然后修改用户栈eip的值，修改后跳转到用户栈的某一处，在这里进行sigreturn() 系统调用：sigreturn再调用 restore\_sigcontext()，这个函数可以从帧中读取原来内核栈的数据，然后恢复。

🌟最后，这是我们讲解的信号机制框架图。

🌟谢谢观看