**进程间的通讯之信号**

1. **信号简介**

**<1>什么是信号?**

**用过windows的我们应该都知道，当我们无法正常结束一个程序时，可以用任务管理器强制结束这个进程。我们看起来只是鼠标的轻轻一点点击。但是我们的操作系统是如何实现的呢?其实这个过程就是利用了我们信号的过程。我们的操作系统给当前进程发送了一个终止的信息。让当前进程终止。我们的linux**

**系统也同样如此，它是通过生成信号和捕捉信号来实现的。运行中的进程捕获到这个信号然后作出一定的操作并最终被终止。**

**<2>信号深度讲解**

**信号是在软件层次上对中断的一种模拟机制。他是进程通信机制中唯一的异步通信机制（异步可理解为，我们正在做某件事情的时候，有人打断了我们，说5分钟后我们去吃饭，然后我继续做我们的事情，5分钟我们一起去吃饭）这写通信方式可以来自系统的外部，例如：ctrl + c系统外部产生了一个中断符号。信号作为一种进程间通信的基本形式，它其实用于内核通知进程发生了异步事件。**

**也可以用于当前进程给另一个进程发送一个信号以实现进程间的通信。**

**<3>生成和捕捉概念了解**

**信号是由于我们的linux系统响应（或者外部触）而产生的一个事件，接收到信号的进程由一个错误产生的。它可以明确的由一个进程发送给另一个进程。**

**一个信号的产生叫做生成，接收到一个信号的行为叫做捕捉。**

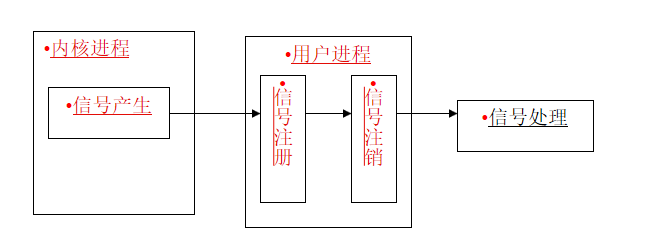
**二．信号的生命周期 SIGKILL**

**OK，我们了解了我们的信号的基本特点之后。我们来了解一下一个信号的生命周期（从出生到到死亡的过程）。它可以划分为我们的三个阶段：**

**<1>产生信号**

**<2>发送信号给进程**

**<3>信号的处理和注销**



**如上图所示，首先，我们的信号的产生是指触发信号的事件（例如，硬件异常，定时器超时，或这外部中断等待）。然后进入我们用户空间，找到目标进程的pid号，进行注册操作。注册是指将信号值加入到目标进程的未处理的信号集合中。**

**表明进程已经知道这些信号的存在，但还没来得及处理。当CPU处理其他事情完毕，准备处理该信号的时候，CPU会把该信号的指令取出来。信号注销机制会把当前信号从目标进程的未处理的信号集合中注销，防止多次CPU多次调度的处理。信号注销之后，立即执行信号处理函数，执行完毕之后，信号本次的发送对当前进程的影响就彻底的结束了。**

**=================================================**

**注:以上就是我们信号处理的过程，可能比较的复杂，多看几遍就明白了。熟能生巧。**

1. **信号的处理机制**

**OK，了解了我们的信号生命周期之后，最后一步是我们linux系统对信号的处理。我们的linux系统中对信号的处理一般分为以下三种：**

**<1>忽略信号**

**对这里系统不做任何处理。当作没有接收到它们一样。默认有两个信号不能那个忽略。即SIGKILL (杀死当前进程)和 SIGSTOP（停止当前进程）。**

**<2>捕捉信号**

**定义并注册信号处理函数，当信号发生时，执行相应的处理函数。【重点】**

**<3>执行省缺操作 interrupt**

**我们的操作系统，对每一个信号对做了相应的默认处理。这里我们就是默认执行操作系统的规定。例如：SIGCON 继续当前进程。**

**我们的linux中信号的种类很多，如下图。我们所有的信号都是SIG开头。**

**四．信号处理函数**

**<1>导读**

**由上图其实已经知道了一些我们常用的信号。我们一些快捷键其实可以产生我们对于那个的信号。例如：ctrl + c 产生SIGINT[signal interrupt],**

**ctrl + \ 产生SIGQUIT[signal quit]信号，ctrl + z产生SIGTSTP[SIG STOP]信等。**

**以上的信号可以说都是由我们的键盘触发来产生的，那么对这些信号该如何处理呢？我们来看看下面-----------注册信号处理函数。**

**<2>设置进程对信号的处理方式-------注册信号处理函数**

**Ok,首先回顾，我们以前学习的内容？==>回忆函数指针和回调函数。**

**void fun (int ) ;//函数的声明**

**void \*fun1 (int ); //函数指针fun1的定义。保存void (int )类型的函数**

**typedef int a; //a<==>int**

**typedef void (\*funp)(int );//给函数指针取一个别名.funp<==>void (\*funp)(int)**

**funp m; // ===》void (\*m)(int); ==>void (\*m)(int )**

**=================================================**

**#include <signal.h>**

**//信号处理函数的类型pthread**

**typedef void (\*sighandler\_t)(int);**

**sighandler\_t signal(int signum, sighandler\_t handler);**

**// void (\*signal(int signum,void ( \*handler )(int)) )(int);**

**功能：注册信号处理函数**

**参数：**

**@signum 捕捉的信号名称。 例如：SIGINT,SIGCHLD**

**@handler 信号处理的方式函数，他是一个函数指针。保存**

**例如：void function(int m)这里类型的地址。**

**返回值：成功返回信号原来处理方式对应的函数指针值。**

**失败返回函数指针SIG\_ERR。**

**注意：**

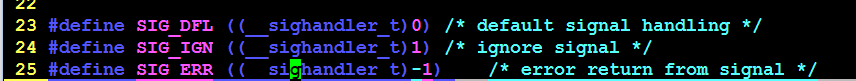
**<1>sighandler\_t类型说明，确定信号处理函数的原型为,void (\*sighandler\_t)(int); 信号处理函数有一个int类型的参数，当被调用的时候，保存时的信号的编号之。**

**<2> signum常用信号: SIGINT ctrl + c发送的中断信息。**

**SIGCHLD 子进程状态改变的时候，父进程会收到**

**信号。[常用于回收僵尸态子进程。]**

**SIGTSTP ctlr + z发送的暂停信号**



**<3> handler的使用方法：**

1. **设置SIG\_IGN，采用忽略的方式来处理信号。**
2. **设置SIG\_DFL,采用系统默认的方式来处理信号**
3. **利用回调函数，设置信号处理函数来处理。信号处理函数格式如下：**

**Void signal\_deal(int num)**

**{**

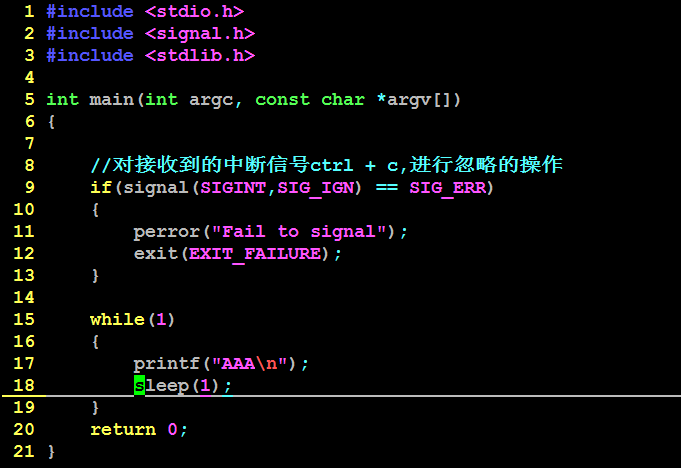
**代码；**

**}**

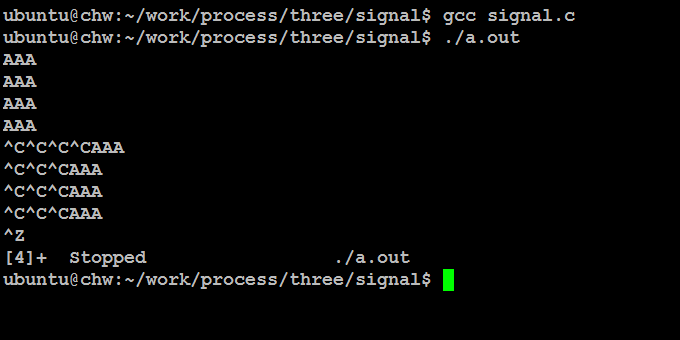
**注意：上面的num表示接收到的信号的编号。例如SIGKILL 为9**

**实例演练：**

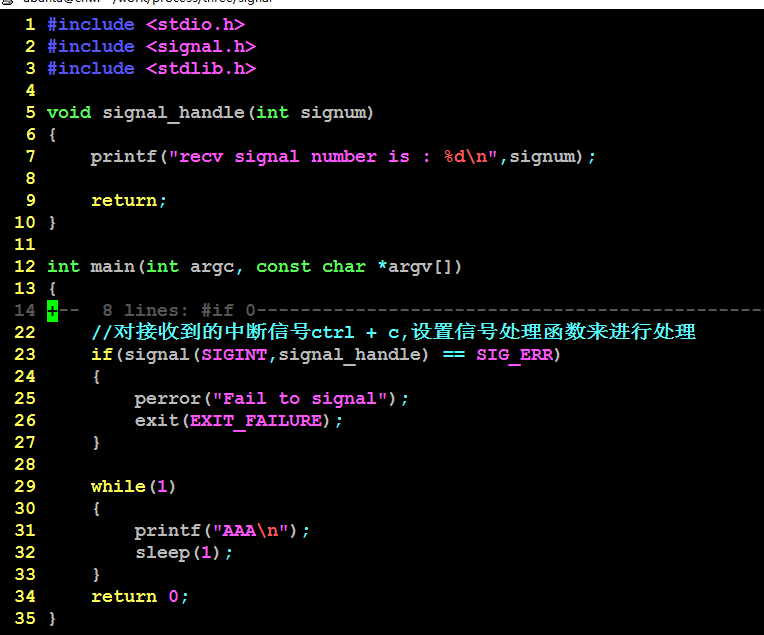
**//忽略系统信号**



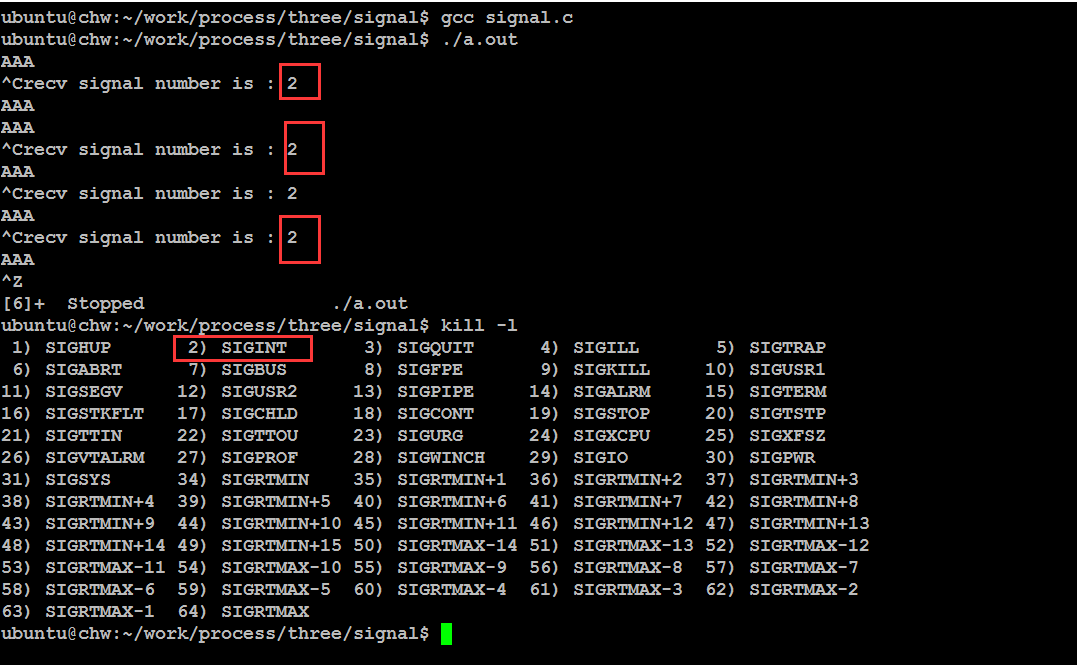
**运行结果：**



**Signal1.c**



**运行结果:**



**练习：**

**fork前采用signal信号处理函数不阻塞，不轮询的方式回收僵尸态子进程[waitpid()函数]。 在信号处理函数signal\_handler()中对信号进行收尸操作。**

**然后利用fork函数创建一个子进程。休眠10s后退出。**

**父进程是一个死循环，每秒输出****"father do something…"的字符串。**

**提示：**

**子进程在终止时会给父进程发SIGCHLD，该信号的默认处理动作是忽略，父进程可以自定义SIGCHLD信号的处理函数。我们这里调用waitpid非阻塞的回收僵尸态子进程。这样父进程只需要专心处理自己的工作，不必关心子进程了，子进程终止时会通知父进程，父进程在信号处理函数中调用waitpid函数清理子进程即可。**

**一般信号对僵尸态子进程的处理方法：**

**<1>****父进程采用signal(SIGCHLD, hand\_signal),采用信号处理函数，对接收到的SIGCHLD进行进行处理。在接收到SIGCHLD信号的时候，采用waitpid利用非阻塞的方式的释放它们的资源。若是使用wait()函数的话，父进程会阻塞。 [推荐使用]**

**<2>父进程采用signal(SIGCHLD, SIG\_IGN)，忽略SIGCHLD信号，这样子进程结束后，就不需要父进程来wait和释放资源。它会自动被过继给老祖宗init进程，int进程会负责释放他的资源，这样就不会产生僵尸态子进程。注意：此做法只适用于System V和Unix98系统。官方协议没有明确表示会回收。**

**信号处理总结**

1. **一旦安装了信号处理函数，它便一直安装。（较早期的系统是每执行一次将它拆除）**
2. **在一个信号处理函数运行期间，正被递交的信号是阻塞的**
3. **如果一个信号在被阻塞期间产生了一次或多次，那么该信号被解阻塞之后通常只递交一次，也就是说Unix信号默认是不排队的**
4. **可通过sigprocmask函数选择阻塞或解阻塞一组信号**
5. **信号的发送函数**

**Ok，我们刚刚学习了我们的信号处理函数，我们其实可以发现，我们发送的信号，都是通过我们的shell命令来发送的，那么没有什么东西可以通过我们的函数来实现类似的功能呢？下面我们具体的看看。**

**<1>给指定的pid号发送信号**

**int kill(pid\_t pid,int sigum);**

**功能：给指定的pid号发送信号。**

**参数：**

**@pid 指定的pid号**

**@sigum 发送信号的类型的编号。 例如 9 ----SIGKILL**

**返回值：成功返回0，失败返回-1**

**<2>给自己发送信号**

**Int raise(int signum);**

**功能：给当前进程自己发送信号。**

**参数：**

**@ signum发送信号的编号**

**返回值: 成功返回0 ，失败返回 -1**

**==============================》简单函数，自己写程序测试。**

1. **定时闹钟函数[alarm]和挂起函数[pause]**

**简介：**

**alarm()也称为闹钟函数，它可以在进程中设置一个定时器。当定时器指定的时间到时，内核就向进程发送SIGALARM信号。**

**pause()函数是用于将调用进程挂起直到收到信号为止。**

**<1>设置闹钟**

**unsigned int alarm(unsigned int seconds);**

**功能:给进程启动一个定时器，经过seconds秒后把SIGALRM信号发送给当前进程。**

**参数:**

**@seconds 秒**

**返回值:**

**成功返回0,失败返回 -1**

**注意：一个进程只能有一个闹钟事件，若是多次使用alarm函数，则闹钟时间被刷新。**

**练习:**

**从键盘输入数据，利用alarm设置一个用户超时检测机制，若是用户3s内没有输入数据，就超时1次，当超时3次的时候，提示用户，输入超时，然后推出当前进程。**

**思路：可以设置一个全局变量count = 0，每次超时3s，count就加1，当count ==3的时候，进程退出。**

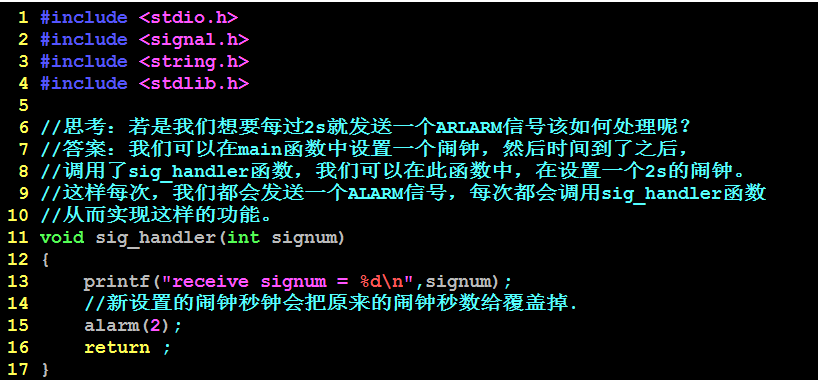
**4.挂起进程**

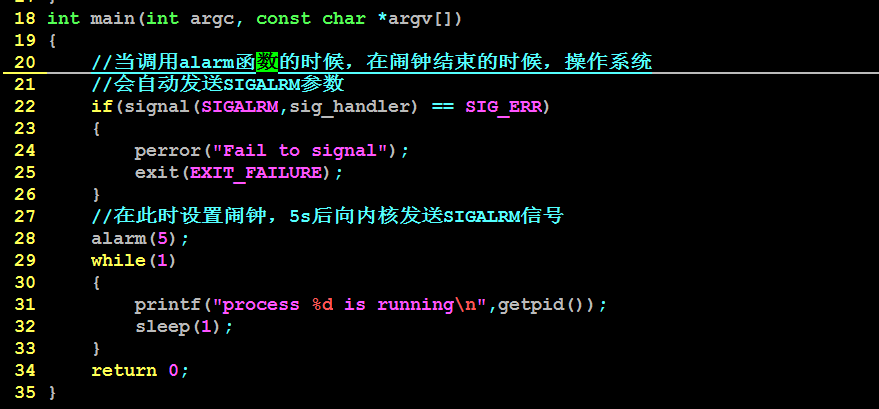
**int pause(void);**

**特点:挂起一个进程，直到进程收到一个信号，进程会继续执行**

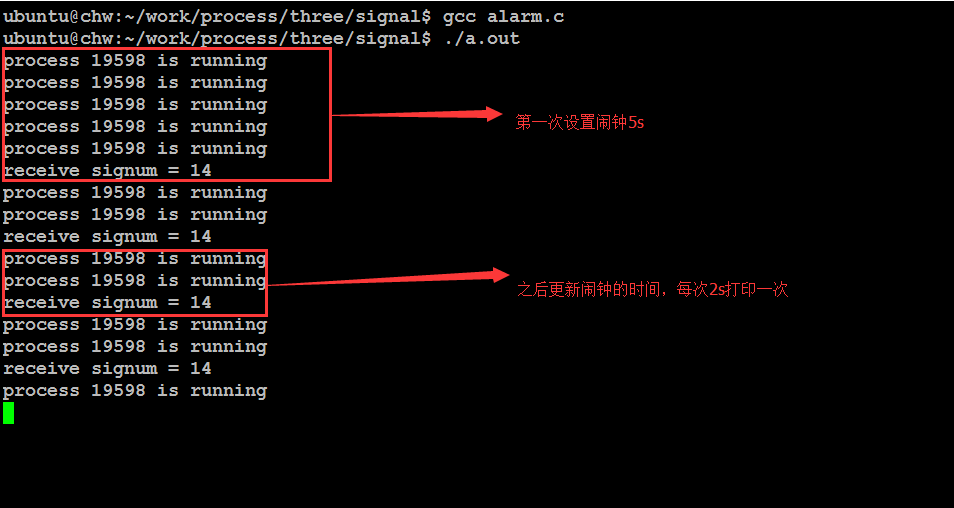
**注意:大部分信号对进程的默认操作都是杀死进程，此时必须设置进程对信号处理方式为捕捉.**

**alarm.c**

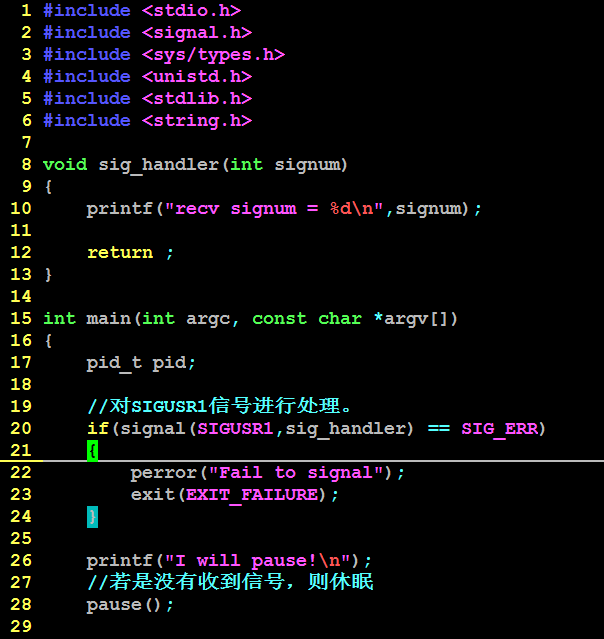


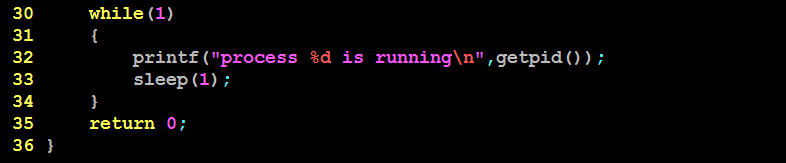


**运行结果：**



**pause.c**





**运行结果:**

