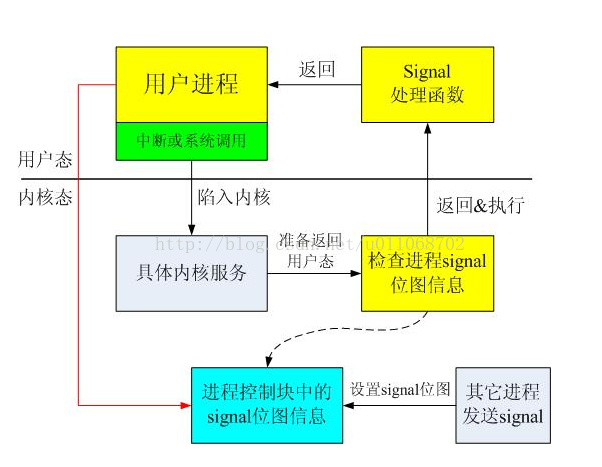
<https://blog.csdn.net/u011068702/article/details/54602566>

最近同事的程序设计过程中用到了[**Linux**](http://lib.csdn.net/base/linux)的signal机制，从而引发了我对Linux中signal机制的思考。Signal机制在Linux中是一个非常常用的进程间通信机制，很多人在使用的时候不会考虑该机制是具体如何实现的。signal机制可以被理解成进程的软中断，因此，在实时性方面还是相对比较高的。Linux中signal机制的模型可以采用下图进行描述。



个进程都会采用一个进程控制块对其进行描述，进程控制块中设计了一个signal的位图信息，其中的每位与具体的signal相对应，这与中断机制是保持一致的。当系统中一个进程A通过signal系统调用向进程B发送signal时，设置进程B的对应signal位图，类似于触发了signal对应中断。发送signal只是“中断”触发的一个过程，具体执行会在两个阶段发生：

1、  system call返回。进程B由于调用了system call后，从内核返回用户态时需要检查他拥有的signal位图信息表，此时是一个执行点。

2、  中断返回。进程被系统中断打断之后，系统将CPU交给进程时，需要检查即将执行进程所拥有的signal位图信息表，此时也是一个执行点。

综上所述，signal的执行点可以理解成从内核态返回用户态时，在返回时，如果发现待执行进程存在被触发的signal，那么在离开内核态之后（也就是将CPU切换到用户模式），执行用户进程为该signal绑定的signal处理函数，从这一点上看，signal处理函数是在用户进程上下文中执行的。当执行完signal处理函数之后，再返回到用户进程被中断或者system call（软中断或者指令陷阱）打断的地方。

       Signal机制实现的比较灵活，用户进程由于中断或者system call陷入内核之后，将断点信息都保存到了堆栈中，在内核返回用户态时，如果存在被触发的signal，那么直接将待执行的signal处理函数push到堆栈中，在CPU切换到用户模式之后，直接pop堆栈就可以执行signal处理函数并且返回到用户进程了。Signal处理函数应用了进程上下文，并且应用实际的中断模拟了进程的软中断过程。

最近写程序，各种bug各种错，有一回程序莫名退出，没报错，也没产生日志和core文件，貌似正常退出一样。

但又不是在程序全部走完后退出，中途莫名退出，这就叫我想到了signal，应该是某些函数错误后发送kill信号给主进程，然后退出。

现在总结下signal各种类型：

|  |  |
| --- | --- |
| **Signal** | **Description** |
| SIGABRT | 由调用abort函数产生，进程非正常退出 |
| SIGALRM | 用alarm函数设置的timer超时或setitimer函数设置的interval timer超时 |
| SIGBUS | 某种特定的硬件异常，通常由内存访问引起 |
| SIGCANCEL | 由Solaris Thread Library内部使用，通常不会使用 |
| SIGCHLD | 进程Terminate或Stop的时候，SIGCHLD会发送给它的父进程。缺省情况下该Signal会被忽略 |
| SIGCONT | 当被stop的进程恢复运行的时候，自动发送 |
| SIGEMT | 和实现相关的硬件异常 |
| SIGFPE | 数学相关的异常，如被0除，浮点溢出，等等 |
| SIGFREEZE | Solaris专用，Hiberate或者Suspended时候发送 |
| SIGHUP | 发送给具有Terminal的Controlling Process，当terminal被disconnect时候发送 |
| SIGILL | 非法指令异常 |
| SIGINFO | BSD signal。由Status Key产生，通常是CTRL+T。发送给所有Foreground Group的进程 |
| SIGINT | 由Interrupt Key产生，通常是CTRL+C或者DELETE。发送给所有ForeGround Group的进程 |
| SIGIO | 异步IO事件 |
| SIGIOT | 实现相关的硬件异常，一般对应SIGABRT |
| SIGKILL | 无法处理和忽略。中止某个进程 |
| SIGLWP | 由Solaris Thread Libray内部使用 |
| SIGPIPE | 在reader中止之后写Pipe的时候发送 |
| SIGPOLL | 当某个事件发送给Pollable Device的时候发送 |
| SIGPROF | Setitimer指定的Profiling Interval Timer所产生 |
| SIGPWR | 和系统相关。和UPS相关。 |
| SIGQUIT | 输入Quit Key的时候（CTRL+\）发送给所有Foreground Group的进程 |
| SIGSEGV | 非法内存访问 |
| SIGSTKFLT | Linux专用，数学协处理器的栈异常 |
| SIGSTOP | 中止进程。无法处理和忽略。 |
| SIGSYS | 非法系统调用 |
| SIGTERM | 请求中止进程，kill命令缺省发送 |
| SIGTHAW | Solaris专用，从Suspend恢复时候发送 |
| SIGTRAP | 实现相关的硬件异常。一般是调试异常 |
| SIGTSTP | Suspend Key，一般是Ctrl+Z。发送给所有Foreground Group的进程 |
| SIGTTIN | 当Background Group的进程尝试读取Terminal的时候发送 |
| SIGTTOU | 当Background Group的进程尝试写Terminal的时候发送 |
| SIGURG | 当out-of-band data接收的时候可能发送 |
| SIGUSR1 | 用户自定义signal 1 |
| SIGUSR2 | 用户自定义signal 2 |
| SIGVTALRM | setitimer函数设置的Virtual Interval Timer超时的时候 |
| SIGWAITING | Solaris Thread Library内部实现专用 |
| SIGWINCH | 当Terminal的窗口大小改变的时候，发送给Foreground Group的所有进程 |
| SIGXCPU | 当CPU时间限制超时的时候 |
| SIGXFSZ | 进程超过文件大小限制 |
| SIGXRES | Solaris专用，进程超过资源限制的时候发送 |

**signal对应的值：**

**POSIX.1中列出的信号：**

SIGHUP 1 A 终端挂起或者控制进程终止   
SIGINT 2 A 键盘中断（如break键被按下）   
SIGQUIT 3 C 键盘的退出键被按下   
SIGILL 4 C 非法指令   
SIGABRT 6 C 由abort(3)发出的退出指令   
SIGFPE 8 C 浮点异常   
SIGKILL 9 AEF Kill信号   
SIGSEGV 11 C 无效的内存引用   
SIGPIPE 13 A 管道破裂: 写一个没有读端口的管道   
SIGALRM 14 A 由alarm(2)发出的信号   
SIGTERM 15 A 终止信号   
SIGUSR1 30,10,16 A 用户自定义信号1   
SIGUSR2 31,12,17 A 用户自定义信号2   
SIGCHLD 20,17,18 B 子进程结束信号   
SIGCONT 19,18,25 进程继续（曾被停止的进程）   
SIGSTOP 17,19,23 DEF 终止进程   
SIGTSTP 18,20,24 D 控制终端（tty）上按下停止键   
SIGTTIN 21,21,26 D 后台进程企图从控制终端读   
SIGTTOU 22,22,27 D 后台进程企图从控制终端写 

**没在POSIX.1中列出，而在SUSv2列出**

SIGBUS 10,7,10 C 总线错误(错误的内存访问)   
SIGPOLL A Sys V定义的Pollable事件，与SIGIO同义   
SIGPROF 27,27,29 A Profiling定时器到   
SIGSYS 12,-,12 C 无效的系统调用 (SVID)   
SIGTRAP 5 C 跟踪/断点捕获   
SIGURG 16,23,21 B Socket出现紧急条件(4.2 BSD)   
SIGVTALRM 26,26,28 A 实际时间报警时钟信号(4.2 BSD)   
SIGXCPU 24,24,30 C 超出设定的CPU时间限制(4.2 BSD)   
SIGXFSZ 25,25,31 C 超出设定的文件大小限制(4.2 BSD) 

（对于SIGSYS，SIGXCPU，SIGXFSZ，以及某些机器体系结构下的SIGBUS，Linux缺省的动作是A (terminate)，SUSv2 是C (terminate and dump core)）。

**下面是其它的一些信号**  
  
信号 值 处理动作 发出信号的原因   
----------------------------------------------------------------------   
SIGIOT 6 C IO捕获指令，与SIGABRT同义   
SIGEMT 7,-,7   
SIGSTKFLT -,16,- A 协处理器堆栈错误   
SIGIO 23,29,22 A 某I/O操作现在可以进行了(4.2 BSD)   
SIGCLD -,-,18 A 与SIGCHLD同义   
SIGPWR 29,30,19 A 电源故障(System V)   
SIGINFO 29,-,- A 与SIGPWR同义   
SIGLOST -,-,- A 文件锁丢失   
SIGWINCH 28,28,20 B 窗口大小改变(4.3 BSD, Sun)   
SIGUNUSED -,31,- A 未使用的信号(will be SIGSYS)   
  
（在这里，- 表示信号没有实现；有三个值给出的含义为，第一个值通常在Alpha和Sparc上有效，中间的值对应i386和ppc以及sh，最后一个值对应mips。信号29在Alpha上为SIGINFO / SIGPWR ，在Sparc上为SIGLOST。）   
  
**处理动作一项中的字母含义如下**  
A 缺省的动作是终止进程   
B 缺省的动作是忽略此信号   
C 缺省的动作是终止进程并进行内核映像转储（dump core）   
D 缺省的动作是停止进程   
E 信号不能被捕获   
F 信号不能被忽略

代码测试

**[html]** [view plain](https://blog.csdn.net/u011068702/article/details/54602566) [copy](https://blog.csdn.net/u011068702/article/details/54602566)

1. #include**<stdio.h>**
2. #include**<signal.h>**
3. #include**<unistd.h>**
4. #include**<stdlib.h>**
5. void when\_alarm();
6. void when\_sigint();
7. void when\_sigchld(int);
8. void when\_sigusr1();
9. void when\_sigio();
10. int main()
11. {
12. int childpid;//子程序进程ID号
13. printf("程序已经开始运行，5秒钟后将接收到时钟信号。/n");
14. if ((childpid=fork())**>**0)//父进程
15. {
16. signal(SIGALRM,when\_alarm);  //当接收到SIGALRM信号时，调用when\_alarm函数
17. signal(SIGINT,when\_sigint);  //当接收到SIGINT信号时，调用when\_sigint函数
18. signal(SIGCHLD,when\_sigchld);//当接收到SIGCHLD信号时，调用when\_sigchld函数
19. signal(SIGUSR1,when\_sigusr1);//当接收到SIGUSR1信号时，调用when\_sigusr1函数
20. signal(SIGIO,when\_sigio);//当接收到SIGIO信号时，调用when\_sigio函数
21. alarm(5);     //5秒钟之后产生SIGALRM信号
22. raise(SIGIO); //向自己发送一个SIGIO信号
23. pause(); //将父进程暂停下来，等待SIGALRM信号到来
24. pause(); //将父进程暂停下来，等待SIGUSR1信号到来
25. pause(); //将父进程暂停下来，等待SIGCHLD信号到来
26. printf("------此时程序会停下来等待，请按下ctrl+c送出SIGINT信号-------/n");
27. pause(); //将父进程暂停下来，等待SIGINT信号到来
28. }
29. else if(childpid==0) //子进程
30. {
31. int timer;
32. for(timer=7;timer**>**=0;timer--) //时钟计时5秒产生SIGALRM信号，再过2秒子进程退出，产生SIGCHLD信号
33. {
34. if(timer**>**2)
35. printf("距离SIGALRM信号到来还有%d秒。/n",timer-2);
36. if(timer==4)
37. kill(getppid(),SIGUSR1); //向父进程发送一个SIGUSR1信号
38. if((timer**<**=2)&&(timer**>**0))
39. printf("子进程还剩%d秒退出，届时会产生SIGCHLD信号。/n",timer);
40. if(timer==0) //子进程退出，产生SIGCHLD信号
41. raise(SIGKILL); //子进程给自己发一个结束信号
42. sleep(1); //每个循环延时1秒钟
43. }
44. }
45. else
46. printf("fork()函数调用出现错误！/n");
47. return 0;
48. }
49. void when\_alarm()
50. {
51. printf("5秒钟时间已到，系统接收到了SIGALRM信号！/n");
52. }
53. void when\_sigint()
54. {
55. printf("已经接收到了SIGINT信号，程序将退出！/n");
56. exit(0);
57. }
58. void when\_sigchld(int SIGCHLD\_num)
59. {
60. printf("收到SIGCHLD信号，表明我的子进程已经中止，SIGCHLD信号的数值是：%d。/n",SIGCHLD\_num);
61. }
62. void when\_sigusr1()
63. {
64. printf("系统接收到了用户自定义信号SIGUSR1。/n");
65. }
66. void when\_sigio()
67. {
68. printf("系统接收到了SIGIO信号。/n");
69. }