linux系統編程之信號（二）：信號處理流程（產生、註冊、註銷、執行）

對於一個完整的信號生命週期(從信號發送到相應的處理函數執行完畢)來說，可以分為三個階段：

- 信號誕生
- 信號在進程中註冊
- 信號在進程中的註銷
- 信號處理函數執行

1 信號誕生

信號事件的發生有兩個來源：硬件來源(比如我們按下了鍵盤或者其它硬件故障)；軟件來源，最常用發送信號的系統函數是kill, raise, alarm和setitimer以及sigqueue函數，軟件來源還包括一些非法運算等操作。

這裡按發出信號的原因簡單分類，以了解各種信號：

（1） 與進程終止相關的信號。當進程退出，或者子進程終止時，發出這類信號。

（2） 與進程例外事件相關的信號。如進程越界，或企圖寫一個只讀的內存區域（如程序正文區），或執行一個特權指令及其他各種硬件錯誤。

（3） 與在系統調用期間遇到不可恢復條件相關的信號。如執行系統調用exec時，原有資源已經釋放，而目前系統資源又已經耗盡。

（4） 與執行系統調用時遇到非預測錯誤條件相關的信號。如執行一個並不存在的系統調用。

（5） 在用戶態下的進程發出的信號。如進程調用系統調用kill向其他進程發送信號。

（6） 與終端交互相關的信號。如用戶關閉一個終端，或按下break鍵等情況。

（7） 跟踪進程執行的信號。

Linux支持的信號列表如下。很多信號是與機器的體系結構相關的

信號值默認處理動作發出信號的原因

SIGHUP 1 A 終端掛起或者控制進程終止

SIGINT 2 A 鍵盤中斷（如break鍵被按下）

SIGQUIT 3 C 鍵盤的退出鍵被按下

SIGILL 4 C 非法指令

SIGABRT 6 C 由abort(3)發出的退出指令

SIGFPE 8 C 浮點異常

SIGKILL 9 AEF Kill信號

SIGSEGV 11 C 無效的內存引用

SIGPIPE 13 A 管道破裂: 寫一個沒有讀端口的管道

SIGALRM 14 A 由alarm(2)發出的信號

SIGTERM 15 A 終止信號

SIGUSR1 30,10,16 A 用戶自定義信號1

SIGUSR2 31,12,17 A 用戶自定義信號2

SIGCHLD 20,17,18 B 子進程結束信號

SIGCONT 19,18,25 進程繼續（曾被停止的進程）

SIGSTOP 17,19,23 DEF 終止進程

SIGTSTP 18,20,24 D 控制終端（tty）上按下停止鍵

SIGTTIN 21,21,26 D 後台進程企圖從控制終端讀

SIGTTOU 22,22,27 D 後台進程企圖從控制終端寫

處理動作一項中的字母含義如下

A 缺省的動作是終止進程

B 缺省的動作是忽略此信號，將該信號丟棄，不做處理

C 缺省的動作是終止進程並進行內核映像轉儲（dump core），內核映像轉儲是指將進程數據在內存的映像和進程在內核結構中的部分內容以一定格式轉儲到文件系統，並且進程退出執行，這樣做的好處是為程序員提供了方便，使得他們可以得到進程當時執行時的數據值，允許他們確定轉儲的原因，並且可以調試他們的程序。

D 缺省的動作是停止進程，進入停止狀況以後還能重新進行下去，一般是在調試的過程中（例如ptrace系統調用）

E 信號不能被捕獲

F 信號不能被忽略

2 信號在目標進程中註冊

在進程表的表項中有一個軟中斷信號域，該域中每一位對應一個信號。內核給一個進程發送軟中斷信號的方法，是在進程所在的進程表項的信號域設置對應於該信號的位。如果信號發送給一個正在睡眠的進程，如果進程睡眠在可被中斷的優先級上，則喚醒進程；否則僅設置進程表中信號域相應的位，而不喚醒進程。如果發送給一個處於可運行狀態的進程，則只置相應的域即可。

進程的task_struct結構中有關於本進程中未決信號的數據成員： struct sigpending pending：

```
struct sigpending{
    struct sigqueue *head, *tail;
    sigset_t signal;
};
```

第三個成員是進程中所有未決信號集，第一、第二個成員分別指向一個sigqueue類型的結構鏈（稱之為"未決信號信息鏈"）的首尾，信息鏈中的每個sigqueue結構刻畫一個特定信號所攜帶的信息，並指向下一個sigqueue結構:

```
struct sigqueue{
    struct sigqueue *next;
    siginfo_t info;
}
```

信號在進程中註冊指的就是信號值加入到進程的未決信號集sigset_t signal（每個信號佔用一位）中，並且信號所攜帶的信息被保留到未決信號信息鏈的某個sigqueue結構中。只要信號在進程的未決信號集中，表明進程已經知道這些信號的存在，但還沒來得及處理，或者該信號被進程阻塞。

當一個實時信號發送給一個進程時，不管該信號是否已經在進程中註冊，都會被再註冊一次，因此，信號不會丟失，因此，實時信號又叫做"可靠信號"。這意味著同一個實時信號可以在同一個進程的未決信號信息鏈中佔有多個sigqueue結構（進程每收到一個實時信號，都會為它分配一個結構來登記該信號信息，並把該結構添加在未決信號鏈尾，即所有誕生的實時信號都會在目

標進程中註冊）。

當一個非實時信號發送給一個進程時，如果該信號已經在進程中註冊（通過sigset_t signal指示），則該信號將被丟棄，造成信號丟失。因此，非實時信號又叫做"不可靠信號"。這意味著同一個非實時信號在進程的未決信號信息鏈中，至多佔有一個sigqueue結構。

總之信號註冊與否，與發送信號的函數（如kill()或sigqueue()等）以及信號安裝函數（signal()及sigaction()）無關，只與信號值有關（信號值小於SIGRTMIN的信號最多只註冊一次，信號值在SIGRTMIN及SIGRTMAX之間的信號，只要被進程接收到就被註冊）

3 信號的執行和註銷

內核處理一個進程收到的軟中斷信號是在該進程的上下文中，因此，進程必須處於運行狀態。當其由於被信號喚醒或者正常調度重新獲得CPU時，在其從內核空間返回到用戶空間時會檢測是否有信號等待處理。如果存在未決信號等待處理且該信號沒有被進程阻塞，則在運行相應的信號處理函數前，進程會把信號在未決信號鏈中佔有的結構卸掉。

對於非實時信號來說，由於在未決信號信息鏈中最多只佔用一個sigqueue結構，因此該結構被釋放後，應該把信號在進程未決信號集中刪除（信號註銷完畢）；而對於實時信號來說，可能在未決信號信息鏈中佔用多個sigqueue結構，因此應該針對占用sigqueue結構的數目區別對待：如果只佔用一個sigqueue結構（進程只收到該信號一次），則執行完相應的處理函數後應該把信號在進程的未決信號集中刪除（信號註銷完畢）。否則待該信號的所有sigqueue處理完畢後再在進程的未決信號集中刪除該信號。

當所有未被屏蔽的信號都處理完畢後，即可返回用戶空間。對於被屏蔽的信號，當取消屏蔽後，在返回到用戶空間時會再次執行上述檢查處理的一套流程。

內核處理一個進程收到的信號的時機是在一個進程從內核態返回用戶態時。所以，當一個進程在內核態下運行時，軟中斷信號並不立即起作用，要等到將返回用戶態時才處理。進程只有處理完信號才會返回用戶態，進程在用戶態下不會有未處理完的信號。

處理信號有三種類型：進程接收到信號後退出；進程忽略該信號；進程收到信號後執行用戶設定用系統調用signal的函數。當進程接收到一個它忽略的信號時，進程丟棄該信號，就像沒有收到該信號似的繼續運行。如果進程收到一個要捕捉的信號，那麼進程從內核態返回用戶態時執行用戶定義的函數。而且執行用戶定義的函數的方法很巧妙，內核是在用戶棧上創建一個新的層，該層中將返回地址的值設置成用戶定義的處理函數的地址，這樣進程從內核返回彈出棧頂時就返回到用戶定義的函數處，從函數返回再彈出棧頂時，才返回原先進入內核的地方。這樣做的原因是用戶定義的處理函數不能且不允許在內核態下執行（如果用戶定義的函數在內核態下運行的話，用戶就可以獲得任何權限）。
