uClinux 的信號 signal 處理 -1

(uClinux-3)

在 uClinux 作業系統下的驅動程式，可以使用中斷來處理比較緊急的事件，或是在兩個獨立的編程中傳遞訊息；但是在應用程式中要如何處理這些事情呢？-- signal，為應用程式编程中非常重要的部分，本文將詳細介紹信號機制的基本概念、及大致的使用方法。

信號機制是行程之間相互傳遞消息的一種方法，信號全稱為軟體中斷信號；從它的命名可以看出，它的實質和使用上就是中斷；只是使用者模式所使用的中斷。

一、信號的基本概念

- 信號（signal）用來通知行程發生了非同步事件。行程之間可以互相通過系統呼叫--- kill 函式發送軟體中斷信號。核心也可以因為內部事件而發送信號給行程，通知行程發生了某個事件。
- 信號只是用來通知某進程發生了什麼事件，並不給該行程傳遞任何資料。
- 收到信號的行程對各種信號有三種不同的處理方法：
- 對於需要處理的信號，行程可以指定處理函數，由該函數來處理。
- 忽略某個信號，對該信號不做任何處理。
- 對該信號的處理保留系統的預設值。

二、信號的類型

- 與行程終止相關的信號。當行程退出，或者子行程終止時，發出這類信號。
- 與行程例外事件相關的信號。如行程越界，或企圖寫一個唯讀的記憶體區域（如程式正文區），或執行一個特權指令及其他各種硬體錯誤。
- 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。
- 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。
- 在使用者模式下的行程發出的信號。如行程調用系統調用 kill 向其他進程發送信號。
- 與終端交互相關的信號。如使用者關閉一個終端，或按下 break 鍵等情況。
- 跟蹤進程執行的信號。
- uClinux 支持的信號清單放於 signal.h 檔案中，由於信號非常多，筆者也

無法全部瞭解，以下列出所知之信號代表意義，其它有待使用者自行挖掘：

信號	代表數值	發出信號原因
SIGHUP	1	終端掛起或者控制行程終止
SIGINT	2	鍵盤中斷（Ctrl+c 鍵被按下）
SIGQUIT	3	鍵盤中斷(Ctrl + 4 鍵被按下)
SIGILL	4	非法指令
SIGTRAP	5	跟蹤/中斷點捕獲
SIGABRT, SIGIOT	6	由 abort(3)發出的退出指令
SIGBUS	7	匯流排錯誤(錯誤的記憶體訪問)
SIGFPE	8	浮點異常
SIGKILL	9	kill 信號
SIGUSR1	10	使用者自訂信號 1
SIGSEGV	11	無效的記憶體引用
SIGUSR2	12	使用者自訂信號 2
SIGPIPE	13	管道破裂: 寫一個沒有定義埠的管道
SIGALRM	14	由 alarm(2)發出的信號
SIGTERM	15	終止信號
SIGVTALRM	26	實際時間報警時鐘信號

- 系統行程在執行時，若按下 Ctrl+C，終端機上會出現如 pid 17: failed 2 的訊息，這就是系統核心接收到鍵盤 Ctrl+C 時對應用程式發出 SIGINT 中斷所出現的訊息，其中 17 為行程的編號(PID)，2 即為 SIGINT 的代號。

三、信號的使用

- signal 及 kill 函式形成信號的基本操作；signal 是行程用來設定某個信號的處理方法，kill 是用來發送信號給指定的行程。
- 函式 pause 和 alarm 是通過信號實現的行程暫停和計時器到時。
- signal 函式格式如 void (*signal(int signum, void (*handler)(int)))(int));

含入檔： signal.h

參數說明： int signum 定義信號產生條件，如上表。

*handler 信號處理函式，或是以下三種定義：

SIG_DFL 信號處理預設值

SIG_IGN 忽略此信號處理

SIG_ERR 信號處理錯誤時傳回錯誤代碼

- alarm(int seconds)函式，用來設置信號 SIGALRM 在經過參數 seconds 指定的秒數後傳送給目前的進程。

四、鬧鐘計時器及信號範例：以下範例將會啟動鬧鐘信號 SIGALRM，且在 3 秒後產生 SIGALARM 信號，且透過信號處理函式，將所產生的信號代碼印出。

➢ 主函式：

hellosignal.c	
行號	內容
1	`int main(void)`
2	`{`
3	`int i;`
4	`printf("ESD Signal Test Process ID = %d\n",getpid());`
5	`signal(SIGALRM,sig_handle);`
6	`signal(SIGINT,sig_handle);`
7	`signal(SIGQUIT,sig_handle);`
8	`alarm(3);`
9	`for(i=0;;i++){`
10	`printf("Sleep %d.....\n",i);`
11	`sleep(1);`
12	`}`
13	}

- 行號 4 利用 getpid()讀取此行程之代碼，並印出。
- 行號 5-7 分別利用 signal 信號函式，設定 SIGALRM、SIGINT 及 SIGQUIT 信號產生之信號處理函式為 sig_handle。
- 行號 8 使用 alarm 函式定義三秒鐘後將產生一 SIGALRM 的信號。
- 行號 9-12 將每隔一秒列印計數器 i 的數值，此時只等待信號的產生。

➢ 信號處理函式：

hellosignal.c	
行號	內容
1	`void sig_handle(int sig)`
2	`{`
3	`switch(sig){`
4	`case SIGINT:`
5	`printf("Hello Signal %d = INT\n",sig);`
6	`break;`
7	`case SIGQUIT:`
8	`printf("Hello Signal %d = QUIT\n",sig);`
9	`break;`

```
10              case SIGALRM:
11                   printf("Hello Signal %d = ALARM\n",sig);
12                   break;
13          }
14  }
```

- 核心產生信號將會將信號代碼傳至信號處理函式。
- 行號 3-13 將所產生的信號代碼印出。

➢ 程式執行

- 執行 hellosignal 後將在終端機上每隔一秒列印以下訊息

```
Sleep 1 ……..
Sleep 2 ……..
Hello Signal 14 = ALARM      第三秒產生 ALARM 信號
Sleep 3 ……..
Hello Signal 3 = QUIT        按下 Ctrl+4 產生之信號
Sleep 4 ……..
Hello Signal 2 = INT         按下 Ctrl+4 產生之信號
```

本文章介紹信號的處理方式及實例，但是細心的讀者，應該在使用實例時發現了一個重大的缺陷，就是 ALARM 信號產生一次後就不再產生了，當然 QUIT 信號也是第二次產生時就跳出行程了，如果需要定時產生信號該如何設計呢？下回分曉。

Victor 於加拿大