The Linux Serial Programming HOWTO

by Peter H. Baumann, *Peter.Baumann@dlr.de*
譯者: 曾元佑 *yytseng@ms16.hinet.net*
v1.0, 22 一月 1998

本文件將敘述如何在 Linux 環境下撰寫序列埠的通訊程式.

目錄

1 簡介

本文是 Linux 序列埠程式撰寫的 HOWTO. 全篇都在討論如何在 Linux 環境下, 以序列埠與其他 裝置/電腦 通訊的程式寫法. 所解釋的技術包含: 標準的 I/O (只具備 傳送/接收 線的), 非同步 I/O, 及 等待來自多信號源的輸入訊號 的寫法.

本文不會敘述如何設定序列埠, 因爲這在 Greg Hankins 的 Serial-HOWTO 已經有說明了.

我必需強調我並非此領域中的專家, 而是在專案中曾遇到過這類的通訊問題. 在這所提到的範例程式是衍生自 miniterm 的程式碼. 可在 LDP 程式設計師指南取得 (ftp://sunsite.unc.edu/pub/Linux/docs/LDP/programmers-guide/lpg-0.4.tar.gz 及其他映射站) 在範例那個目錄下.

我開始寫這份文件是在 1997 年 六月, 現在我已經移轉到 WinNT 以滿足客户的需求, 以致於我沒能學得更深入的知識. 如果任何人有什麼意見, 我很樂意把它擺進這份文件中 (參考 回饋 那一節). 如果有人能接手這份工作並加以改進, 請 e-mail 給我.

所有的範例都在 i386 Linux Kernel 2.0.29 下測試過.

1.1 版權

1.2 本文最新的版本

Serial-Programming-HOWTO 最新的版本將放在 *ftp://sunsite.unc.edu:/pub/Linux/docs/HOWTO/Serial-Programming-HOWTO* 及其他映設站台. 有許多的格式, 如 PostScript 及 DVI 的版本放在 other-formats 目錄下. Serial-Programming-HOWTO 也放在 *http://sunsite.unc.edu/LDP/HOWTO/Serial-Programming-HOWTO.html* 並會每個月擺一份到 *comp.os.linux.answers*.

1.3 回饋

請把任何修正, 問題, 意見, 建議, 或其它附加的題材傳送給我. 以讓我改進這份 HOWTO! 並詳細告訴我哪個部份是您不能了解, 或不夠清楚的. 你可以用 email 連絡我 *Peter.Baumann@dlr.de*. 請把 Serial-Programming-HOWTO 的版本號碼附上, 本文版本號碼是 0.3.

2 開始

2.1 偵錯

最好的偵錯你程式碼的方法是建構另一台 Linux box, 並把兩台電腦用 null-modem 纜線連接. 用 miniterm (可在 LDP 程式設計師指南取得

(ftp://sunsite.unc.edu/pub/Linux/docs/LDP/programmers-guide/lpg-0.4.tar.gz 在範例那個目錄下) 以傳送字元到你的 Linux box. Miniterm 很容易編譯而它會把所有輸入到鍵盤的字元透過序列埠傳送. 只有這個宣告定義會被檢查 #define MODEMDEVICE "/dev/ttyS0". 如果是 COM1 設定爲 ttyS0, 如果是 COM2 設定爲 ttyS1 等等.. 先前的測試是必要的, 所有的 字元都將以 raw 方式 (不經任何處理) 直接傳送. 測試是否連接正確, 在兩台電腦上都啓動 miniterm 然後隨便在鍵盤上亂按. 在其中一台上輸入的字元應該會顯示在另一台電腦上反之亦同. 但輸入的字元不會回應到與之相連的螢幕上.

要自製 null-modem 的電纜, 你必需要把 TxD (傳送) 及 RxD (接收) 兩線對調. 詳細的說明在 Serial-HOWTO 的第 7 段.

當然也可以只用一台電腦來作相同的測試, 只要電腦上有兩個未使用的序列埠. 當然你也就要執行兩個 miniterm 來當虛擬控制台. 如果你是藉由拔去滑鼠來取得另一個序列埠, 記得要把 /dev/mouse 裝置重新導向, 如果它存在的話. 如果你使用多埠的序列埠控制卡, 請確定它已設定正確. 當我在我的電腦上測試時也曾經因爲設定錯誤而出過槌. 當我連到另一台電腦, 通訊埠開始傳送字元. 就因爲剛好這不是完整的非同步式傳輸, 所以可在同一台電腦上執行兩個程式.

2.2 連接埠設定

/dev/ttyS* 裝置會被當成連接到你的 Linux box 的終端機, 並且在啓動後就設定好了. 這個觀念在你寫 raw 裝置的通訊程式時必需記住. 也就是說這個連接埠被設定爲回應所有自這個裝置送出的字元, 而用在資料傳輸時通常這種要改變這種工作模式.

所有的參數可以由一個小程式簡單的完成. 設定參數被放在一個結構體內 struct termios, 他的定義檔在 <asm/termbits.h>:

```
#define NCCS 19
struct termios {
        tcflag_t c_iflag;               /* 輸入模式旗標 */
        tcflag_t c_oflag;               /* 輸出模式旗標 */
        tcflag_t c_cflag;               /* 控制模式旗標 */
        tcflag_t c_lflag;               /* 區域模式旗標 */
        cc_t c_line;                    /* 行控制 (line discipline) */
        cc_t c_cc[NCCS];                /* 控制特性 */
};
```

這個檔案也包含所有的旗標定義. 輸入模式旗標在 c_iflag 掌管所有的輸入處理, 這就意謂著由裝置上傳來的字元在還沒用 read 功能讀取前可以先處理過. 同理 c_oflag 掌管所有的輸出處理. c_cflag 包含連接埠的設定, 如 鮑率, 每字元多少位元, 停止位元, 等等.. 區域模式旗標放在 c_lflag 用來偵測字元是否回應, 而訊號會送到你的程式, 等等.. 最後 c_cc 陣列定義了檔案終了的控制字元, 停止, 等等.. 預設的控制字元值放在 <asm/termios.h>. 有關旗標的細節擺在使用手冊 termios(3). termios 結構體內的 c_line 行控制 (line discipline) 元素, 不能在 POSIX 相容的系統下使用 譯者註:這裡所說的 line discipline 雖然我翻成 行控制 但還是很難說出那是舍. 如果想知道請看看 kernel :(.

2.3 序列裝置的輸入觀念

有三個輸入的觀念要說明. 按照所要寫的應用程式選用適合的觀念. 儘量避免使用迴圈來讀取單一的字元再組成字串. 我曾這樣做過, 會掉字元, 且對 read 而言不會顯示任何錯誤.

2.3.1 標準輸入程序

這是終端機的標準處理程序, 但用來與其他 dl 型式的以行爲單位的輸入通訊也很有用, 也就是 `read` 會傳回一整行完整的輸入資料. 行預設的終止字元是 `NL` (ASCII `LF`), 檔案結束符, 或行終止字元. 預設環境下, `CR` (是 DOS/Windows 預設的行終止符) 不會終止一行的敘述.

標準的輸入處理程序還可以處理 清除, 刪除字, 重印字元, 及轉換 `CR` 爲 `NL` 等等功能..

2.3.2 非標準輸入程序

非標準輸入程序可以用在需要每次讀取固定數量字元的情況, 並允許使用字元輸入時間的計時器. 這種模式可以用在讀取固定字元數量的應用程式, 或者所連接的裝置會突然送出大量字元的狀況.

2.3.3 非同步式輸入

以上所敘述的兩種模式都可以用在非同步與同步的傳輸模式. 預設是在同步的模式下工作, 也就是在尚未讀取完之前, `read` 的狀態會被阻斷. 而非同步模式下 `read` 的狀態會直接返回並送出訊號到所叫用的程式直到完成工作. 這個訊號可以由訊號的處理程式 handler...來接收.

2.3.4 等待來自多個訊號來源的輸入

這並不是一個不一樣的輸入模式. 如果你要透過序列埠連接並處理多個裝置的話, 它是滿有用的. 在我的應用程式中我必需在幾乎同一時間內, 透過 TCP/IP socket 及序列埠處理來自其他電腦的輸入訊號. 下面這個範例程式將等待來自兩個不同輸入源的訊號. 如果其中一個信號源出現, 他就會被處理, 而程式會繼續等待新的輸入訊號.

以下這個方法看起來相當覆雜, 但請記住 Linux 是一個多工的作業系統. `select` 這個系統呼叫並不會在等待輸入訊號時把 CPU 負載加重, 而如果你用迴圈方式來等待輸入訊號將使得其它同時執行的行程被拖慢.

3 程式範例

所有的範例來源自 `miniterm.c`. The type ahead 暫存器被限制在 255 個字元, 就跟標準輸入程序的最大字串長度相同 (`<linux/limits.h>` 或 `<posix1_lim.h>`).

參考程式碼中的註解它會解釋不同輸入模式的使用. 我希望這些程式碼都能被了解. 標準輸入程序的程式範例的註解寫得最好, 其它的範例都只在不同於其它範例的地方做註解.

敘述不是很完整, 但可以激勵你對這範例做實驗, 以延生出合於你所需應用程式的最佳解.

別忘記要把序列埠的權限設定正確 (也就是: `chmod a+rw /dev/ttyS1`)!

3.1 標準輸入程序

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
```

```
#include <stdio.h>

/* 鮑率設定被定義在 <asm/termbits.h>, 這在 <termios.h> 被引入 */
#define BAUDRATE B38400
/* 定義正確的序列埠 */
#define MODEMDEVICE "/dev/ttyS1"
#define _POSIX_SOURCE 1 /* POSIX 系統相容 */

#define FALSE 0
#define TRUE 1

volatile int STOP=FALSE;

main()
{
  int fd,c, res;
  struct termios oldtio,newtio;
  char buf[255];
/*
  開啓數據機裝置以讀取並寫入而不以控制 tty 的模式
  因為我們不想程式在送出 CTRL-C 後就被殺掉.
*/
 fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
 if (fd <0) {perror(MODEMDEVICE); exit(-1); }

 tcgetattr(fd,&oldtio); /* 儲存目前的序列埠設定 */
 bzero(&newtio, sizeof(newtio)); /* 清除結構體以放入新的序列埠設定值 */

/*
  BAUDRATE: 設定 bps 的速度. 你也可以用 cfsetispeed 及 cfsetospeed 來設定.
  CRTSCTS : 輸出資料的硬體流量控制 (只能在具完整線路的纜線下工作
            參考 Serial-HOWTO 第七節)
  CS8     : 8n1 (8 位元, 不做同位元檢查,1 個終止位元)
  CLOCAL  : 本地連線, 不具數據機控制功能
  CREAD   : 致能接收字元
*/
 newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;

/*
  IGNPAR  : 忽略經同位元檢查後, 錯誤的位元組
  ICRNL   : 比 CR 對應成 NL (否則當輸入訊號有 CR 時不會終止輸入)
            在不然把裝置設定成 raw 模式(沒有其它的輸入處理)
*/
 newtio.c_iflag = IGNPAR | ICRNL;

/*
 Raw 模式輸出.
*/
 newtio.c_oflag = 0;

/*
```

```
  ICANON  : 致能標準輸入, 使所有回應機能停用, 並不送出信號以叫用程式
*/
 newtio.c_lflag = ICANON;

/*
  初始化所有的控制特性
  預設值可以在 /usr/include/termios.h 找到, 在註解中也有,
  但我們在這不需要看它們
*/
 newtio.c_cc[VINTR]    = 0;     /* Ctrl-c */
 newtio.c_cc[VQUIT]    = 0;     /* Ctrl-\ */
 newtio.c_cc[VERASE]   = 0;     /* del */
 newtio.c_cc[VKILL]    = 0;     /* @ */
 newtio.c_cc[VEOF]     = 4;     /* Ctrl-d */
 newtio.c_cc[VTIME]    = 0;     /* 不使用分割字元組的計時器 */
 newtio.c_cc[VMIN]     = 1;     /* 在讀取到 1 個字元前先停止 */
 newtio.c_cc[VSWTC]    = 0;     /* '\0' */
 newtio.c_cc[VSTART]   = 0;     /* Ctrl-q */
 newtio.c_cc[VSTOP]    = 0;     /* Ctrl-s */
 newtio.c_cc[VSUSP]    = 0;     /* Ctrl-z */
 newtio.c_cc[VEOL]     = 0;     /* '\0' */
 newtio.c_cc[VREPRINT] = 0;     /* Ctrl-r */
 newtio.c_cc[VDISCARD] = 0;     /* Ctrl-u */
 newtio.c_cc[VWERASE]  = 0;     /* Ctrl-w */
 newtio.c_cc[VLNEXT]   = 0;     /* Ctrl-v */
 newtio.c_cc[VEOL2]    = 0;     /* '\0' */

/*
  現在清除數據機線並啓動序列埠的設定
*/
 tcflush(fd, TCIFLUSH);
 tcsetattr(fd,TCSANOW,&newtio);

/*
  終端機設定完成, 現在處理輸入訊號
  在這個範例, 在一行的開始處輸入 'z' 會退出此程式.
*/
 while (STOP==FALSE) {     /* 迴圈會在我們發出終止的訊號後跳出 */
 /* 即使輸入超過 255 個字元, 讀取的程式段還是會一直等到行終結符出現才停止.
    如果讀到的字元組低於正確存在的字元組, 則所剩的字元會在下一次讀取時取得.
    res 用來存放眞正讀到的字元組個數 */
    res = read(fd,buf,255);
    buf[res]=0;             /* 設定字串終止字元, 所以我們能用 printf */
    printf(":%s:%d\n", buf, res);
    if (buf[0]=='z') STOP=TRUE;
 }
 /* 回存舊的序列埠設定值 */
 tcsetattr(fd,TCSANOW,&oldtio);
}
```

3.2 非標準輸入程序

在非標準的輸入程序模式下, 輸入的資料不會被組合成一行而輸入後的處理功能 (清除, 殺掉, 刪除, 等等.) 都不能使用. 這個模式有兩個功能控制參數: `c_cc[VTIME]` 設定字元輸入時間計時器, 及 `c_cc[VMIN]` 設定滿足讀取功能的最低字元接收個數.

如果 MIN > 0 且 TIME = 0, MIN 設定爲滿足讀取功能的最低字元接收個數. 由於 TIME 是 零, 所以計時器將不被使用.

如果 MIN = 0 且 TIME > 0, TIME 將被當做逾時設定值. 滿足讀取功能的情況爲讀取到單一字元, 或者超過 TIME 所定義的時間 (t = TIME *0.1 s). 如果超過 TIME 所定義的時間, 則不會傳回任何字元.

如果 MIN > 0 且 TIME > 0, TIME 將被當做一個分割字元組的計時器. 滿足讀取功能的條件爲 接收到 MIN 個數的字元, 或兩個字元的間隔時間超過 TIME 所定義的值. 計時器會在每讀到一個字元後重新計時, 且只會在第一個字元收到後才會啓動.

如果 MIN = 0 且 TIME = 0, 讀取功能就馬上被滿足. 目前所存在的字元組個數, 或者 將回傳的字元組個數. 根據 Antonino (參考 貢獻) 所說, 你可以用 `fcntl(fd, F_SETFL, FNDELAY);` 在讀取前得到相同的結果.

藉由修改 `newtio.c_cc[VTIME]` 及 `newtio.c_cc[VMIN]` 上述的模式就可以測試了.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>

#define BAUDRATE B38400
#define MODEMDEVICE "/dev/ttyS1"
#define _POSIX_SOURCE 1 /* POSIX 系統相容 */
#define FALSE 0
#define TRUE 1

volatile int STOP=FALSE;

main()
{
  int fd,c, res;
  struct termios oldtio,newtio;
  char buf[255];

 fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
 if (fd <0) {perror(MODEMDEVICE); exit(-1); }

 tcgetattr(fd,&oldtio); /* 儲存目前的序列埠設定 */

 bzero(&newtio, sizeof(newtio));
 newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
 newtio.c_iflag = IGNPAR;
 newtio.c_oflag = 0;

 /* 設定輸入模式 (非標準型, 不回應,...) */
```

```
 newtio.c_lflag = 0;

 newtio.c_cc[VTIME]    = 0;   /* 不使用分割字元組計時器 */
 newtio.c_cc[VMIN]     = 5;   /* 在讀取到 5 個字元前先停止 */

 tcflush(fd, TCIFLUSH);
 tcsetattr(fd,TCSANOW,&newtio);


 while (STOP==FALSE) {       /* 輸入迴圈 */
   res = read(fd,buf,255);   /* 在輸入 5 個字元後即返迴 */
   buf[res]=0;               /* 所以我們能用 printf... */
   printf(":%s:%d\n", buf, res);
   if (buf[0]=='z') STOP=TRUE;
 }
 tcsetattr(fd,TCSANOW,&oldtio);
}
```

3.3 非同步式輸入

```
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/signal.h>
#include <sys/types.h>

#define BAUDRATE B38400
#define MODEMDEVICE "/dev/ttyS1"
#define _POSIX_SOURCE 1 /* POSIX 系統相容 */
#define FALSE 0
#define TRUE 1

volatile int STOP=FALSE;

void signal_handler_IO (int status);   /* 定義訊號處理程序 */
int wait_flag=TRUE;                    /* 沒收到訊號的話就會是 TRUE */

main()
{
  int fd,c, res;
  struct termios oldtio,newtio;
  struct sigaction saio;           /* definition of signal action */
  char buf[255];

  /* 開啓裝置爲 non-blocking (讀取功能會馬上結束返回) */
  fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NONBLOCK);
  if (fd <0) {perror(MODEMDEVICE); exit(-1); }

  /* 在使裝置非同步化前, 安裝訊號處理程序 */
```

```
  saio.sa_handler = signal_handler_IO;
  saio.sa_mask = 0;
  saio.sa_flags = 0;
  saio.sa_restorer = NULL;
  sigaction(SIGIO,&saio,NULL);

  /* 允許行程去接收 SIGIO 訊號*/
  fcntl(fd, F_SETOWN, getpid());
  /* 使檔案ake the file descriptor 非同步 (使用手冊上說只有 O_APPEND 及
  O_NONBLOCK, 而 F_SETFL 也可以用...) */
  fcntl(fd, F_SETFL, FASYNC);

  tcgetattr(fd,&oldtio); /* 儲存目前的序列埠設定值 */
  /* 設定新的序列埠爲標準輸入程序 */
  newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
  newtio.c_iflag = IGNPAR | ICRNL;
  newtio.c_oflag = 0;
  newtio.c_lflag = ICANON;
  newtio.c_cc[VMIN]=1;
  newtio.c_cc[VTIME]=0;
  tcflush(fd, TCIFLUSH);
  tcsetattr(fd,TCSANOW,&newtio);

  /* 等待輸入訊號的迴圈. 很多有用的事我們將在這做 */
  while (STOP==FALSE) {
    printf(".\n");usleep(100000);
    /* 在收到 SIGIO 後, wait_flag = FALSE, 輸入訊號存在則可以被讀取 */
    if (wait_flag==FALSE) {
      res = read(fd,buf,255);
      buf[res]=0;
      printf(":%s:%d\n", buf, res);
      if (res==1) STOP=TRUE; /* 如果只輸入 CR 則停止迴圈 */
      wait_flag = TRUE;      /* 等待新的輸入訊號 */
    }
  }
  /* 回存舊的序列埠設定值 */
  tcsetattr(fd,TCSANOW,&oldtio);
}

/***************************************************************************
* 訊號處理程序. 設定 wait_flag 爲 FALSE, 以使上述的迴圈能接收字元           *
***************************************************************************/

void signal_handler_IO (int status)
{
  printf("received SIGIO signal.\n");
  wait_flag = FALSE;
}
```

3.4 等待來自多個訊號來源的輸入

這一段很短. 它只能被拿來當成寫程式時的提示, 故範例程式也很簡短. 但這個範例不只能用在序列埠上, 還可以用在被當成檔案來使用的裝置上.

select 呼叫及伴隨它所引發的巨集共用 `fd_set`. `fd_set` 則是一個 位元陣列, 而其中每一個位元代表一個有效的檔案敘述結構. `select` 呼叫接受一個有效的檔案敘述結構並傳回 `fd_set` 位元陣列, 而該位元陣列中若有某一個位元爲 1, 就表示相對映的檔案敘述結構的檔案發生了輸入, 輸出或有例外事件. 而這些巨集提供了所有處理 `fd_set` 的功能. 亦可參考手冊 select(2).

```
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

main()
{
   int    fd1, fd2;  /* 輸入源 1 及 2 */
   fd_set readfs;    /* 檔案敘述結構設定 */
   int    maxfd;     /* 最大可用的檔案敘述結構 */
   int    loop=1;    /* 迴圈在 TRUE 時成立 */

   /* open_input_source 開啓一個裝置, 正確的設定好序列埠,
      並回傳回此檔案敘述結構體 */
   fd1 = open_input_source("/dev/ttyS1");   /* COM2 */
   if (fd1<0) exit(0);
   fd2 = open_input_source("/dev/ttyS2");   /* COM3 */
   if (fd2<0) exit(0);
   maxfd = MAX (fd1, fd2)+1;  /* 測試最大位元輸入 (fd) */

   /* 輸入迴圈 */
   while (loop) {
     FD_SET(fd1, &readfs);  /* 測試輸入源 1 */
     FD_SET(fd2, &readfs);  /* 測試輸入源 2 */
     /* block until input becomes available */
     select(maxfd, &readfs, NULL, NULL, NULL);
     if (FD_ISSET(fd1))         /* 如果輸入源 1 有訊號 */
       handle_input_from_source1();
     if (FD_ISSET(fd2))         /* 如果輸入源 2 有訊號 */
       handle_input_from_source2();
   }

}
```

這個範例程式在等待輸入訊號出現前, 不能確定它會停頓下來. 如果你需要在輸入時加入逾時功能, 只需把 select 呼叫換成:

```
int res;
struct timeval Timeout;

/* 設定輸入迴圈的逾時值 */
Timeout.tv_usec = 0;  /* 毫秒 */
```

```
Timeout.tv_sec  = 1;  /* 秒 */
res = select(maxfd, &readfs, NULL, NULL, &Timeout);
if (res==0)
/* 檔案敘述結構數在 input = 0 時, 會發生輸入逾時. */
```

這個程式會在 1 秒鐘後逾時. 如果超過時間, select 會傳回 0, 但是應該留意 `Timeout` 的時間遞減是由 `select` 所等待輸入訊號的時間爲基準. 如果逾時的值是 0, select 會馬上結束返回.

4 其它資源

- Linux Serial-HOWTO 敘述如何設定序列埠及所有相關的硬體資訊.
- 由 Michael Sweet 所寫的 *Serial Programming Guide for POSIX Compliant Operating Systems* `<http://www.easysw.com/~mike/serial>`. 這個連結已經荒廢了但我找不到它的新位址. 有人知道能在哪找到它嗎? 它是很棒的文件!
- `termios(3)` 的使用手冊. 敘述所有有關 `termios` 結構體的旗標.

5 貢獻

就跟簡介所說的一樣, 我並非在這領域有所專精, 但我自己遇到問題, 並透過他人的幫助找到答案. 感謝來自 European Transonic Windtunnel 的 Strudthoff 先生, Cologne, Michael Carter (`mcarter@rocke.electro.swri.edu`), 及 Peter Waltenberg (`p.waltenberg@karaka.chch.cri.nz`)

與我同時準備這份文件的 Antonino Ianella (`antonino@usa.net` 所纂寫的 Serial-Port-Programming Mini HOWTO. Greg Hankins 要求我把 Antonino's Mini-HOWTO 一併放入這份文件.

這份文件的結構及 SGML 的格式是源自 Greg Hankins 的 Serial-HOWTO. 感謝 Dave Pfaltzgraff (`Dave_Pfaltzgraff@patapsco.com`), Sean Lincolne (`slincol@tpgi.com.au`), Michael Wiedmann (`mw@miwie.in-berlin.de`), 及 Adrey Bonar (`andy@tipas.lt`) 各方面的協助.