The Linux Serial Programming HOWTO
리눅스 시리얼 프로그래밍 하우투

Peter H. Baumann, Peter.Baumann@dir.de v1.0, 22 January 1998

전성민, schun@crypto.pe.kr 2000년 2월 24일

이 문서는 리눅스 시스템에서 시리얼 통신을 어떻게 프로그래밍하는가를 설명하는 하우투 문서이다.

목 록

1 소개

이 문서는 리눅스 시스템에서 시리얼 통신을 어떻게 프로그래밍하는가를 설명하는 하우투 문서이다. 다양한 시리얼 통신 프로그램 기법들(Canonical I/O, 비동기 I/O, I/O 멀티플렉싱)을 설명한다.

시리얼 포트를 셋업하는 방법에 대해서는 Greg Hankins의 *Serial-HOWTO* 문서를 읽어보기 바란다.

나는 이 분야의 전문가가 아님을 먼저 밝혀둔다. 그러나 프로젝트를 하면서 많은 문제들에 부딪혔다. 이 문서에 소개된 코드 예제들은 LDP(Linux Document Project) 프로그래머 가이드(`ftp://sunsite.unc.edu/pub/Linux/docs/LDP/programmers-guide/lpg-0.4.tar.gz` 및 미러 사이트들)에서 구할 수 있는 miniterm code에 근거하여 만들었다.

1997년 6월에 이 문서를 작성한 뒤에 나는 고객의 요구에 의해 Win NT 환경으로 옮기게 되어서 더 이상 깊은 지식을 얻을 수 없었다. 누구든지 커맨트가 있으면 이 문서를 관리하는데에 기꺼이 활용하겠다.(Feedback 부분을 볼 것) 또한 이 문서의 유지 및 관리를 맡고 싶거나 version UP을 원한다면 e-mail을 주기 바란다.

이 문서의 모든 예제는 i386 Linux Kernel 2.0.29에서 테스트를 하였다.

1.1 저작권

1.2 이 문서의 최신 버전

이 문서의 최신 버전은 `ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/Serial-Programming-HOWTO` 와 미러 사이트에서 찾을 수 있다. PostScript 및 DVI 버전도 다른 포맷들을 갖고 있는 디렉토리에 있다. `http://sunsite.unc.edu/LDP/HOWTO/Serial-Programming-HOWTO.html` 에서도 구할 수 있으며, `comp.os.linux.answers`에 매달 포스팅 될 것이다.

1.3 Feedback

오류, 질문, 커맨트, 건의사항이나 그 외 추가할 사항이 있다면 알려주기 바란다. 이 HOWTO를 계속적으로 업데이트 하기를 원한다. 당신이 정확히 이해하지 못하는 부분이 있거나 더 명확해야 할 부분이 있다면 알려줘라. 내 e-mail 주소는 `Peter.Baumann@dir.de` 이다. 이 문서에 관한 e-mail을 보낼 때는 문서의 버전도 같이 알려줘라. 이 문서 버전은 1.0이다.

2 시작하기

2.1 디버깅

코드를 디버깅하는 가장 좋은 방법은 또 하나의 리눅스 박스를 셋업하고 두 리눅스 박스를 null-modem 케이블로 연결하는 것이다. Miniterm 프로그램을 이용하여 문자들을 전송해보라. Miniterm은 컴파일하기도 쉽고, 키보드에서 입력되는 문자들(특수문자 포함)을 시리얼 포트로 전송할 수 있다. 컴파일할 때 체크해야 할 것은

`#define MODEMDEVICE "/dev/ttyS0"`

문장이 제대로 설정되어 있는가 하는 것이다. COM1으로 맞추려면 /dev/ttyS0, COM2로 하려면 /dev/ttyS1으로 수정한다. 테스팅을 할 때 가장 중요한 것은 문자가 시리얼 포트로 출력될 때 데이터가 출력 데이터 처리(output processing)을 하지 않고 그대로(raw) 전송이 되는가를 확인하는 것이다. 테스트 과정은 다음과 같다. 두 대의 리눅스 박스에서 각각 miniterm 프로그램을 실행시키고 키보드를 쳐본다. 한 곳에서 타이핑한 문자가 다른 곳에서 그대로 나타나는 지 확인한다.

Null-modem 케이블의 TxD와 RxD가 서로 cross 연결이 되어야 한다. 잘 모르겠으면 *Serial-HOWTO* 문서의 7장을 본다.

위의 테스트는 한 대의 컴퓨터만 갖고도 가능하다. 사용할 수 있는 시리얼 포트가 두 개 있다면 케이블을 각각의 시리얼 포트에 연결하고 miniterm을 두 개 실행하여 테스트하면 된다.

2.2 포트 세팅

장치 파일인 /dev/ttyS*는 리눅스에서 터미널을 연결하기 위한 목적으로 사용된다. 이 사실은 시리얼 통신 프로그래밍을 하는데 반드시 기억해야 할 사항이다. 예를 들어, 시리얼 포트도 문자 에코를 하도록 설정되어 있다. 이 설정은 보통 데이터 전송시에 바꿔야 할 사항이다. (역자 주: 시리얼 장치 파일도 터미널 장치로 분류되기 때문에 초기 설정은 일반 터미널에서 사용되는 에코가 되도록 설정되어 있는 것이다.)

모든 파라미터들은 프로그램 코드에서 쉽게 설정할 수 있다. 파라미터들은 <asm/termbits.h>에 정의되어 있는 struct termios 구조체에 저장되어 있다.

```
#define NCCS 19
struct termios {
        tcflag_t c_iflag;               /* input mode flags */
        tcflag_t c_oflag;               /* output mode flags */
        tcflag_t c_cflag;               /* control mode flags */
        tcflag_t c_lflag;               /* local mode flags */
        cc_t c_line;                    /* line discipline */
        cc_t c_cc[NCCS];                /* control characters */
};
```

이 파일은 모든 flag들을 정의하고 있다. c_iflag(입력 모드 flag)는 모든 입력 처리(input processing)를 정의한다. 입력 처리란 read() 함수에 의해 시리얼 포트로 들어온 데이터를 read에 의해 읽기 전에 데이터들을 c_iflag에 정의한 대로 처리하는 것을 의마한다. c_oflag(출력 모드 flag)는 출력 처리(output processing) 하는 방법을 정의한다. c_cflag(제어 모드 flag)는 baudrate, data bits, stop bits 등의 포트 세팅을 정의한다. c_lflag(local 모드 flag)는 echo를 할 것인지 등을 결정한다. 마지막으로 c_cc(제어 문자) 배열은 EOF(End of File), STOP 등의 제어 동작들을 어떤 문자로 정의할 것인가를 설정한다. 제어 문자의 디폴트 문자는 <asm/termios.h>에 정의되어 있다. 위 flag들에 관한 설명은 termios(3) man page에 나와있다. termios 구조체의 c_line 항목은 POSIX 호환 시스템에서 사용되지 않는다.

2.3 시리얼 장치의 입력 방법

이 섹션에서는 세 가지의 입력 방법을 기술하기로 한다. 응용 분야에 따라서 알맞은 방법을 사용해야 한다. 한 문자씩 읽는 루프를 돌려서 전체 문자열을 받는 방법은 가능하다면 피해야 한다. 내가 이런 방법으로 했을 때, 문자를 잃어버리는 경우가 생긴 반면, 전체 문자열을 한번에 읽을 때는 에러가 발생하지 않았다.

2.3.1 Canonical 입력 처리(Canonical Input Processing)

Canonical 입력 처리는 터미널의 기본 처리 방법이다. 이 방법은 한 줄 단위로 처리하는 다른 프로그램과 통신하는데에 사용할 수 있다. 한 줄은 디폴트로 NL(New Line, ASCII는 LF) 문자, EOF(End of File) 문자, 혹은 EOL(End of Line)에 의해 종료되는 문자열을 의미한다. CR(Carriage Return, DOS/Windows의 디폴트 EOL 문자임) 문자는 디폴트 세팅에서 한 줄의 종료 문자로 인식되지 않는다.

또한 Canonical 입력 처리 모드에서는 ERASE, DELETE WORD, REPRINT CHARACTERS 문자들을 처리할 수 있고, CR 문자를 NL 문자로 변환 처리를 할 수 있다.

2.3.2 Non-Canonical 입력 처리(Non-Canonical Input Processing)

Non-Canonical 입력 처리 모드에서는 한 번 읽을 때마다 정해진 크기의 문자만을 읽어낼 수 있다. 또한 타이머를 두어서 일정 시간까지 `read()`가 리턴하지 않는 경우 강제 리턴을 할 수 있다. 이 모드는 항상 정해진 크기의 문자들만을 읽어내거나 대량의 문자들을 전송하고자 할 때 사용한다.

2.3.3 비동기 입력

위에서 설명한 두 가지 모드는 동기 방식이나 비동기 방식으로 사용될 수 있다. 동기 방식은 `read`의 조건이 만족될 때까지 block되는 방식으로서 디폴트로 설정되어 있다. 비동기 방식에서는 `read()` 함수가 바로 리턴되며, 호출한 프로그램에게 signal을 보낸다. 이 signal은 signal handler(시그널 처리 함수)로 보내진다.

2.3.4 입력장치 멀티플렉싱

위에서 설명한 입력 모드에 해당하진 않지만 여러 개의 장치들을 다루고자 할 때 유용하다. 예를 들어 내 응용프로그램에서 TCP/IP 소켓과 시리얼 통신에서 동시에 입력을 받아야 했다. 아래 3.4의 예제는 두 개의 서로 다른 장치로부터 동시에 입력을 기다리는 코드이다. 둘 중 한 개의 장치에서 입력이 들어오면 처리를 하고 또 다시 새로운 입력이 올 때까지 기다린다.

아래 3.4의 예제는 복잡해 보일 수 있지만, 리눅스가 multi-processing OS임을 알고 있기에 매우 중요하다. `select()` 시스템 호출 함수는 입력을 기다리는 동안 CPU에 부하를 주지 않는다. 반면 입력이 들어왔는지 루프를 돌면서 체크하는 polling 방식은 시스템에 부하를 주게 되어 다른 프로세스의 수행 속도를 저하시키게 된다.

3 프로그램 예제

여기의 모든 예제는 `miniterm.c`에서 따왔다. Canonical 입력 처리에서 처리할 수 있는 최대 길이의 문자는 255개(<linux/limits.h> 혹은 <posix1_lim.h>에 정의됨) 로서 버퍼의 최대 길이는 255로 제한된다.

여러 입력 처리 모드의 사용법에 대한 설명을 원하면 코드 내의 comment를 참조하라. 코드가 이해하기 쉽기를 바란다. Canonical 입력 처리 모드의 예제는 comment를 가장 잘 해놓았다. 다른 예제는 canonical 모드 예제와 다른 부분에만 comment를 달았다.

설명이 완벽하진 않지만, 이 예제로 직접 테스트를 해보면 당신의 프로그램에 적용할 때 최적의 방법을 찾을 수 있을 것이다.

시리얼 포트 장치 파일의 선택을 제대로 했는가를 다시 한 번 확인하고, 파일 접근 허가는 제대로 되어 있는지 보기를 바란다. (예: `chmod a+rw /dev/ttyS1`)

3.1 Canonical 입력 처리(Canonical Input Processing)

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

/* Baudrate 설정은 <asm/termbits.h>에 정의되어 있다.
/* <asm/termbits.h>는 <termios.h>에서 include된다. */
#define BAUDRATE B38400
/* 여기의 포트 장치 파일을 바꾼다. COM1="/dev/ttyS1, COM2="/dev/ttyS2 */
#define MODEMDEVICE "/dev/ttyS1"
#define _POSIX_SOURCE 1 /* POSIX 호환 소스 */

#define FALSE 0
#define TRUE 1

volatile int STOP=FALSE;

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

/* 읽기/쓰기 모드로 모뎀 장치를 연다.(O_RDWR)
   데이터 전송 시에 <CTRL>-C 문자가 오면 프로그램이 종료되지 않도록
   하기 위해 controlling tty가 안되도록 한다.(O_NOCTTY)
*/
 fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY );
 if (fd <0) {perror(MODEMDEVICE); exit(-1); }

 tcgetattr(fd,&oldtio); /* save current serial port settings */
 bzero(&newtio, sizeof(newtio)); /* clear struct for new port settings */

/*
  BAUDRATE: 전송 속도. cfsetispeed() 및 cfsetospeed() 함수로도 세팅 가능
  CRTSCTS : 하드웨어 흐름 제어. (시리얼 케이블이 모든 핀에 연결되어 있는
            경우만 사용하도록 한다. Serial-HOWTO의 7장을 참조할 것.)
  CS8     : 8N1 (8bit, no parity, 1 stopbit)
  CLOCAL  : Local connection. 모뎀 제어를 하지 않는다.
  CREAD   : 문자 수신을 가능하게 한다.
*/
 newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;

/*
 IGNPAR   : Parity 에러가 있는 문자 바이트를 무시한다.
 ICRNL    : CR 문자를 NL 문자로 변환 처리한다. (이 설정을 안하면 다른
            컴퓨터는 CR 문자를 한 줄의 종료문자로 인식하지 않을 수 있다.)
  otherwise make device raw (no other input processing)
*/
```

```
 newtio.c_iflag = IGNPAR | ICRNL;

/*
 Raw output.
*/
 newtio.c_oflag = 0;

 ICANON   : canonical 입력을 가능하게 한다.
  disable all echo functionality, and don't send signals to calling program
*/
 newtio.c_lflag = ICANON;

/*
  모든 제어 문자들을 초기화한다.
  디폴트 값은 <termios.h> 헤어 파일에서 찾을 수 있다. 여기 comment에도
  추가로 달아놓았다.
*/
 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;     /* inter-character timer unused */
 newtio.c_cc[VMIN]     = 1;     /* blocking read until 1 character arrives */
 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' */

/*
  이제 modem 라인을 초기화하고 포트 세팅을 마친다.
*/
 tcflush(fd, TCIFLUSH);
 tcsetattr(fd,TCSANOW,&newtio);

/*
  터미널 세팅이 끝났고, 이제는 입력을 처리한다.
  이 예제에서는 한 줄의 맨 첫 문자를 'z'로 했을 때 프로그램을
  종료한다.
*/

 while (STOP==FALSE) {     /* 종료 조건(STOP==TRUE)가 될 때까지 루프 */
 /* read()는 라인 종료 문자가 나올 때까지 255 문자를 넘어가더라도
    block 된다. read 하고자 하는 문자 개수가 입력 가능한 문자 개수보다
```

```
    적은 경우에는 또 한번의 read를 하여 나머지를 읽어낼 수 있다.
    res는 read에 의해서 실제로 읽혀진 문자의 개수를 갖게 된다. */

    res = read(fd,buf,255);
    buf[res]=0;             /* set end of string, so we can printf */
    printf(":%s:%d\n", buf, res);
    if (buf[0]=='z') STOP=TRUE;
  }
  /* restore the old port settings */
  tcsetattr(fd,TCSANOW,&oldtio);
 }
```

3.2 Non-Canonical 입력 처리(Non-Canonical Input Processing)

Non-Canonical 입력 처리 모드에서는 입력이 한 줄 단위로 처리되지 않는다. erase, kill, delete 등의 입력 처리도 수행되지 않는다. 이 모드에서 설정하는 파라미터는 `c_cc[VTIME]`과 `c_cc[VMIN]` 두 가지이다. `c_cc[VTIME]`은 타이머의 시간을 설정하고, `c_cc[VMIN]`은 `read`할 때 리턴되기 위한 최소의 문자 개수를 지정한다.

MIN > 0, TIME = 0

MIN은 `read`가 리턴되기 위한 최소한의 문자 개수. TIME이 0이면 타이머는 사용되지 않는다.(무한대로 기다린다.)

MIN = 0, TIME > 0

TIME은 time-out 값으로 사용된다. Time-out 값은 TIME * 0.1 초이다. Time-out이 일어나기 전에 한 문자라도 들어오면 `read`는 리턴된다.

MIN > 0, TIME > 0

TIME은 time-out이 아닌 inter-character 타이머로 동작한다. 최소 MIN 개의 문자가 들어오거나 두 문자 사이의 시간이 TIME 값을 넘으면 리턴된다. 문자가 처음 들어올 때 타이머는 동작을 시작하고 이후 문자가 들어올 때마다 재시작된다.

MIN = 0, TIME = 0

`read`는 즉시 리턴된다. 현재 읽을 수 있는 문자의 개수나 요청한 문자 개수가 반환된다. Antonino씨에 의하면 `read`하기 전에 `fcntl(fd, F_SETFL, FNDELAY);` 를 호출하면 똑같은 결과를 얻을 수 있다.

`newtio.c_cc[VTIME]`과 `newtio.c_cc[VMIN]`을 수정하여 위 네 가지 방식을 테스트 할 수 있다.

```
#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 compliant source */
#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); /* 현재 설정을 oldtio에 저장 */

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

 /* set input mode (non-canonical, no echo,...) */
 newtio.c_lflag = 0;

 newtio.c_cc[VTIME]    = 0;   /* 문자 사이의 timer를 disable */
 newtio.c_cc[VMIN]     = 5;   /* 최소 5 문자 받을 때까진 blocking */

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


 while (STOP==FALSE) {       /* loop for input */
   res = read(fd,buf,255);   /* 최소 5 문자를 받으면 리턴 */
   buf[res]=0;               /* '\0' 종료 문자열(printf를 하기 위해) */
   printf(":%s:%d\n", buf, res);
   if (buf[0]=='z') STOP=TRUE;
 }
 tcsetattr(fd,TCSANOW,&oldtio);
}
```

3.3 비동기 입력

```
#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 compliant source */
#define FALSE 0
```

```
#define TRUE 1

volatile int STOP=FALSE;

void signal_handler_IO (int status);   /* signal handler 함수 정의 */
int wait_flag=TRUE;                    /* signal을 받지 않은 동안은 TRUE */

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

  /* Non-blocking 모드로 시리얼 장치를 연다(read 함수 호출 후 즉각 리턴) */
  fd = open(MODEMDEVICE, O_RDWR | O_NOCTTY | O_NONBLOCK);
  if (fd <0) {perror(MODEMDEVICE); exit(-1); }

  /* install the signal handler before making the device asynchronous */
  /* 장치를 비동기 모드로 만들기 전에 signal handler */
  saio.sa_handler = signal_handler_IO;
  saio.sa_mask = 0;
  saio.sa_flags = 0;
  saio.sa_restorer = NULL;
  sigaction(SIGIO,&saio,NULL);

  /* SIGIO signal을 받을 수 있도록 한다. */
  fcntl(fd, F_SETOWN, getpid());
  /* file descriptor를 비동기로 만든다. (manual page를 보면
     O_APPEND 와 O_NONBLOCK만이 F_SETFL에 사용할 수 있다고 되어 있다.) */
  fcntl(fd, F_SETFL, FASYNC);

  tcgetattr(fd,&oldtio); /* save current port settings */
  /* canonical 입력처리를 위한 포트 세팅 */
  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);

  /* loop while waiting for input. normally we would do something
     useful here */
  while (STOP==FALSE) {
    printf(".\n");usleep(100000);
    /* after receiving SIGIO, wait_flag = FALSE, input is available
       and can be read */
    if (wait_flag==FALSE) {
```

```
      res = read(fd,buf,255);
      buf[res]=0;
      printf(":%s:%d\n", buf, res);
      if (res==1) STOP=TRUE; /* stop loop if only a CR was input */
      wait_flag = TRUE;      /* wait for new input */
    }
  }
  /* restore old port settings */
  tcsetattr(fd,TCSANOW,&oldtio);
}

/***************************************************************************
* signal handler. sets wait_flag to FALSE, to indicate above loop that     *
* characters have been received.                                           *
***************************************************************************/

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

3.4 입력 장치 멀티플렉싱

이 섹션은 간략한 설명만을 하겠다. 어떻게 하는 지에 대한 간단한 힌트만을 주기 위한 것이므로 짧은 예제 코드만을 담았다. 이 방법은 시리얼 포트에만 적용되는 것이 아니라 file descriptor를 사용하는 모든 입출력에 사용할 수 있다.

`select()` 시스템 호출 함수와 해당하는 매크로 함수들은 `fd_set`을 사용한다. `fd_set`는 bit array로서 file descriptor의 bit entry로 작용한다. `select()`는 해당하는 file descriptor의 bit들을 세팅한 `fd_set`을 입력 파라미터로 받아서 입력, 출력 혹은 예외 발생이 일어난 경우 리턴한다. `fd_set`은 FD_로 시작하는 매크로 함수들을 통해서 관리한다. `select(2)`의 man page를 참조하라.

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

main()
{
   int    fd1, fd2;  /* input sources 1 and 2 */
   fd_set readfs;    /* file descriptor set */
   int    maxfd;     /* maximum file desciptor used */
   int    loop=1;    /* loop while TRUE */

   /* open_input_source opens a device, sets the port correctly, and
      returns a file descriptor */
   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;  /* maximum bit entry (fd) to test */

    /* loop for input */
    while (loop) {
      FD_SET(fd1, &readfs);  /* set testing for source 1 */
      FD_SET(fd2, &readfs);  /* set testing for source 2 */
      /* block until input becomes available */
      select(maxfd, &readfs, NULL, NULL, NULL);
      if (FD_ISSET(fd1, &readfs))     /* input from source 1 available */
        handle_input_from_source1();
      if (FD_ISSET(fd2, &readfs))     /* input from source 2 available */
        handle_input_from_source2();
    }

}
```

위의 예제 코드는 입력이 들어올 때까지 계속 block 되는 동작을 보여준다. Time-out이 필요하다면, 다음과 같이 바꾼다.

```
int res;
struct timeval Timeout;

/* set timeout value within input loop */
Timeout.tv_usec = 0;  /* milliseconds */
Timeout.tv_sec  = 1;  /* seconds */
res = select(maxfd, &readfs, NULL, NULL, &Timeout);
if (res==0)
/* number of file descriptors with input = 0, timeout occurred. */
```

이 예제는 1초 후에 time-out이 되는 것을 보여준다. Time-out이 일어나면 `select()`는 0을 반환한다. 여기서 주의할 점은, 설정한 `Timeout` 값은 `select()`에 의해서 감소하기 때문에 다시 `select()`를 호출한다면 `Timeout.tv_usec`과 `Timeout.tv_sec` 값을 다시 설정해야 한다. `Timeout` 값이 0이 되면 time-out이 발생하고 `select()`는 즉시 리턴된다.

4 다른 유용한 정보

- *Serial-HOWTO* 문서는 시리얼 포트를 어떻게 셋업하는지를 설명하고 하드웨어 정보를 제공한다.
- Michael Sweet의 *Serial Programming Guide for POSIX Compliant Operating Systems* `<http://www.easysw.com/~mike/serial>`. 이 링크는 옛날 것이고 최신 위치를 찾을 수가 없다. 이거 찾아줄 분 누구 없수? 이 문서는 매우 잘 정리된 것이다. (역자 주: 다시 이 위치가 부활했다. 예전에 잠시 폐쇄된 적이 있었다.)
- `termios(3)` man page는 `termios` 구조체의 모든 flag에 대해 정의되어 있다.

5 기여한 사람들

1장의 소개 부분에서 언급했듯이, 나는 이 분야의 전문가가 아니다. 그러나 여러 문제에 부딪혔었고, 다른 이들의 도움을 받아서 문제들을 해결했다. Strudthoff 씨, Michael Carter(`mcarter@rocke.electro.swri.edu`)씨

및 Peter Walternberg(p.waltenberg@karaka.chch.cri.nz) 씨의 도움에 감사한다.

Antonino Ianella(antonino@usa.net)씨는 내가 이 문서를 작성할 때 *Serial-Port-Programming Mini HOWTO*를 썼다. Greg Hankins씨는 Antonino의 Mini-HOWTO를 이 문서에 넣으라고 권했다.

이 문서의 구조와 SGML 포맷은 Greg Hankins씨의 *Serial-HOWTO*에서 따왔다. 이 글에서 오류를 수정할 수 있도록 도와주신 많은 분들이 있다. Dave Pfaltzgraff(Dave_Pfaltzgraff@patapsco.com), Sean Lincolne(slincol@tpgi.com.au), Michael Wiedmann(mw@miwie.in-berlin.de), Adrey Bonar(andy@tipas.lt) 씨에게 감사한다.