Porta seriale e parallela

Porta seriale e parallela

La porta seriale e la posta parallela sono semplici dispositivi di connessione dei computer con dispositivi esterni. Vengono principalmente utilizzate per stampanti, mouse, tastiere. Sono tuttavia spesso usate anche per il collegamento di dispositivi di misura e controllo (ad es. multimetri).

La porta seriale consiste essenzialmente di due linee digitali, una dal computer al dispositivo ed una dal dispositivo al computer (più un certo numero di linee di controllo); può essere trasmesso un solo bit alla volta.

La porta parallela consiste invece di un certo numero di linee dati e di controllo [tre registri di memoria a 8-bit: DATA (8 bit disponibili), CONTROL (bit 0-3 disponibili) e STATUS (bit 3-7 disponibili)], che consentono quindi il trasferimento di più bit per volta permettendo quindi la realizzazione di semplici circuiti "pilotatibili" tramite PC.

Comunicazione con i devices

La comunicazione con i "devices" è particolarmente "naturale" in ambiente Linux secondo la filosofia

"Everything is a file"

open

```
#include <fcntl.h>
int open(const char *pathname, int flags);
```

pathname: `/dev/ttySX` (seriale) `/dev/parportX` (parallela)
($X = 0,..,N_{porte} - 1$)
flags: `O_RDWR`, `O_RDONLY`, `O_WRONLY`,...

Il valore di ritorno è un intero (il più basso disponibile) che identifica univocamente il file (file descriptor) se il file (o device) esiste ed è accessibile, -1 altrimenti.

close

```
#include <unistd.h>
int close(int fd);
```

Funzioni di utilizzo generale

ioctl

```
#include <sys/ioctl.h>
int ioctl(int fd, int command, ...);
```

La funzione `ioctl` manipola i parametri dei "devices"; in particolare molte loro caratteristiche operative possono essere controllate specificando `command` (dipendente dal device); `fd` è il file descriptor. Il terzo argomento (se esistente) è un puntatore a memoria di tipo non specificato (tipicamente un `char *argp`). La caratteristiche (e la presenza !) del terzo parametro dipendono dal tipo di comando. Valore di ritorno: 0 in caso di successo, -1 in caso di errore.

Per informazioni dettagliate su tutte queste funzioni di sistema date il comando, da linea di terminale:

```
man funzione
```

Funzioni di utilizzo generale

write

```
#include <unistd.h>
int write(int fd, const void *buf, int n);
```

Scrive `n` bytes nel file `fd` copiandoli dalla memoria a partire dal puntatore `buf`.
In caso di successo il numero di bytes scritti è restituito come valore di ritorno; in caso di errore il valore di ritorno è -1.

open

```
#include <unistd.h>
int read(int fd, void *buf, int n);
```

Legge `n` bytes dal file `fd` in memoria a partire dal puntatore `buf`.
In caso di successo il numero di bytes letti è restituito come valore di ritorno; in caso di errore il valore di ritorno è -1.

Porta seriale: configurazione

Il driver per la porta seriale è caricato automaticamente, per poter accedervi è necessario abilitare i permessi necessari con

```
chmod utente+permessi nomefile
```

come per un qualsiasi file (l'utente può essere `u` (user), `g` (group), `a` (all), i permessi `r` (read) `w` (write) `x` execute)

Dopo aver aperto le comunicazioni con la porta (`open`) è necessario specificare le modalità di comunicazione (velocità, numero di bit, etc...). Ciò viene fatto assegnando il valore dei membri della struttura `termios`

struttura termios

```
#include <termios.h>
structure termios{
  unsigned int c_iflag;      /* input modes */
  unsigned int c_oflag;      /* output modes */
  unsigned int c_cflag;      /* control modes */
  unsigned int c_lflag;      /* local modes */
  char         c_cc[NCCS];   /* control chars */
}
```

Porta seriale: configurazione (II)

(Alcune) opzioni per la modalità di input `c_iflag`:
`IGNBRK` ignora condizione di break (segnale di condizioni anomale)
`IGNPAR` ignora errori di parità

(Alcune) opzioni per la modalità di controllo `c_cflag`:
`CREAD` abilita la ricezione
`CLOCAL` ignora le linee del modem
`CS5,CS6,CS7,CS8` setta il numero di bit da trasmettere
`PARODD` seleziona parità dispari
`CSTOPB` setta a due gli stop bits (default uno)

Ogni modo può essere settato componendo, con operazioni bit a bit, varie opzioni (la maggior parte delle label corrisponde ad un intero con un solo bit a 1).

Per settare la velocità di trasmissione (in input e in output)

```
int cfsetispeed(struct termios *termios_p, int speed);
int cfsetospeed(struct termios *termios_p, int speed);
```

`speed=BXXXX` (dove `XXXX` è il valore della velocità in baud)

Porta seriale: configurazione (III)

Una volta assegnati i valori desiderati alla struttura termios questa viene usata per configurare il device utilizzando la funzione

```
int tcsetattr(int fd, int opt, struct termios *term);
```

`opt`: `TCSANOW`: la modifica ha effetto immediato

...

Metex MXD-4660

Configurazione

- tipo di segnale: 7n2
- velocità di comunicazione: 9600 baud
- bit di controllo di flusso RTS settato a zero (non standard, in realtà l'apparecchio non utilizza il controllo di flusso ma la tensione corrispondente serve per alimentare le linee di trasmissione).

Comando di misura

Scrittura della stringa "d\n" (2 caratteri) nel device

Lettura

Il device fornisce in output 4 array di 14 caratteri nel formato

```
01234567890123
TT xxxxxx  uu
```

Es.:

```
DC 0.0121  mV
   0.0122
   0.0123
   0.0124
```

Metex MXD-4660: esempio di programma per l'acquisizione (I)


```
void init_metex(int fd){
  int status;
  struct termios term;          // crea variabile di tipo termios

  term.c_oflag = 0;             // ...ne assegna i membri
  term.c_lflag = 0;
  term.c_iflag = IGNBRK | IGNPAR;
  term.c_cflag = CREAD | CLOCAL | ...;

  cfsetispeed(&term,....);
  cfsetospeed(&term,....);

  tcsetattr(fd,TCSANOW,&term); // assegna la struttura al file

  ioctl(fd,TIOCMGET,&status);  // chiede stato linee di controllo
  status &= ~TIOCM_RTS;        // setta  RTS a 0
  ioctl(fd,TIOCMSET,&status);  // setta  stato linee di controllo
}
```

Metex MXD-4660: esempio di programma per l'acquisizione (II)


```
int main(){
  int fd;
  fd = open("/dev/ttyS0",O_RDWR); // apertura
  if (fd<0){
    ... // messaggio di errore
    return 1;
  }
  init_metex(fd);                 // set della modalita' di comunicazione
  string command="d\n";           // definizione del comando di misura
  write(fd,...,...);              // invio del comando di misura
  char buf[14];
  for (int i=0;i<4;i++){
    read(fd,...,...);             // lettura dei dati
    if (i==0){
      string sbuf(buf,14);
      //... lettura del risultato...
    }
  }
  close(fd);
  ...
}
```

N.B. Mancano gli `#include` opportuni, individuateli con l'ausilio di `man`.

Classe stringa e utilizzo delle classi di I/O per le stringhe (I)

Utili metodi della classe stringa

```
string substr(int firstchar, int nchar);
int find(const string&str);
```

Es.:

```
string str = "bbbaa";
cout << str.substr(0,3) << endl;
int pos    = str.find("aa");
cout << pos << endl;
pos    = str.find("c");
if (pos == string::npos) cout << "Not found" << endl;
```

Classe stringa e utilizzo delle classi di I/O per le stringhe (I)

Utili metodi della classe stringa

```
string substr(int firstchar, int nchar);
int find(const string&str);
```

Es.:

```
string str = "bbbaa";
cout << str.substr(0,3) << endl;
int pos    = str.find("aa");
cout << pos << endl;
pos    = str.find("c");
if (pos == string::npos) cout << "Not found" << endl;
```

bbb

Classe stringa e utilizzo delle classi di I/O per le stringhe (I)

Utili metodi della classe stringa

```
string substr(int firstchar, int nchar);
int find(const string&str);
```

Es.:

```
string str = "bbbaa";
cout << str.substr(0,3) << endl;
int pos    = str.find("aa");
cout << pos << endl;
pos    = str.find("c");
if (pos == string::npos) cout << "Not found" << endl;
```

bbb
3

Classe stringa e utilizzo delle classi di I/O per le stringhe (I)

Utili metodi della classe stringa

```
string substr(int firstchar, int nchar);
int find(const string&str);
```

Es.:

```
string str = "bbbaa";
cout << str.substr(0,3) << endl;
int pos    = str.find("aa");
cout << pos << endl;
pos    = str.find("c");
if (pos == string::npos) cout << "Not found" << endl;
```

bbb
3
Not found

Classe stringa e utilizzo delle classi di I/O per le stringhe (II)

Classi di I/O per le stringhe

istringstream (analogo a ifstream)

```
#include <sstream>
  string str;
  ...
  istringstream inputstring(str);
  inputstring >> ...;
```

ostringstream (analogo a ofstream)

```
#include <sstream>
  ostringstream outputstring;
  outputstring << ...;
  string str=outputstring.str();
```

Es.:

```
  double val;
  string tipo,unit,buf = "DC 0.0121  mV";
  istringstream is(buf);
  is >> tipo >> val >> unit; cout << val << endl;
  ostringstream os;
  os << tipo << " " << val << " " << unit; cout <<  os.str() << endl;
```

Classe stringa e utilizzo delle classi di I/O per le stringhe (II)

Classi di I/O per le stringhe

istringstream (analogo a ifstream)

```
#include <sstream>
  string str;
  ...
  istringstream inputstring(str);
  inputstring >> ...;
```

ostringstream (analogo a ofstream)

```
#include <sstream>
  ostringstream outputstring;
  outputstring << ...;
  string str=outputstring.str();
```

Es.:

```
  double val;
  string tipo,unit,buf = "DC 0.0121  mV";
  istringstream is(buf);
  is >> tipo >> val >> unit; cout << val << endl;
  ostringstream os;
  os << tipo << " " << val << " " << unit; cout <<  os.str() << endl;
```

0.0121

Classe stringa e utilizzo delle classi di I/O per le stringhe (II)

Classi di I/O per le stringhe

istringstream (analogo a ifstream)

```
#include <sstream>
  string str;
  ...
  istringstream inputstring(str);
  inputstring >> ...;
```

ostringstream (analogo a ofstream)

```
#include <sstream>
  ostringstream outputstring;
  outputstring << ...;
  string str=outputstring.str();
```

Es.:

```
  double val;
  string tipo,unit,buf = "DC 0.0121  mV";
  istringstream is(buf);
  is >> tipo >> val >> unit; cout << val << endl;
  ostringstream os;
  os << tipo << " " << val << " " << unit; cout <<  os.str() << endl;
```

0.0121
DC 0.0121 mV

Esercitazione in laboratorio: porta seriale

Predisporre le seguenti funzioni per l'acquisizione del multimetro tramite seriale

```
void init_metex(int fd);
double get_metex(int fd, string tipo);
```

o facoltativamente

```
void get_metex(int fd, string tipo, double &mis, double& err);
```

ed utilizzatele per misurare il valore di una resistenza, di una tensione, etc....

La funzione `get_metex` deve controllare che il multimetro sia effettivamente configurato per fare la misura richiesta (`tipo`) e deve convertire il risultato in unità S.I. (per l'eventuale calcolo dell'errore fate riferimento al manuale dello strumento).
Per sicurezza (in assenza di controllo di flusso) è bene far intercorrere 0.2s tra scrittura e lettura Utilizzate le funzioni:
`void sleep(int secondi)` e `void usleep(int microsecondi)`.

Porta parallela (I)

Il driver `ppdev` per la porta parallela `/dev/parport0` non è solitamente caricato automaticamente.

Per caricare in memoria (attuando alcuni test per la configurazione automatica) un modulo aggiuntivo al kernel (nucleo) di linux occorre (da utente root) dare il comando

```
modprobe nome_modulo
```

ed in questo caso il comando è

```
modprobe parport_pc &&  modprobe ppdev
```

I comandi `lsmod` e rmmod listano e rimuovono rispettivamente i moduli "aggiunti" al kernel.

Porta parallela (II)

Il driver ppdev supporta (tra le altre) le funzioni standard `open`, `close`, `ioctl`.

In particolare, ricordiamo che `ioctl` prende come parametri il file descriptor, un comando, e (opzionalmente) il puntatore a qualche dato.

Alcuni comandi possibili sono:

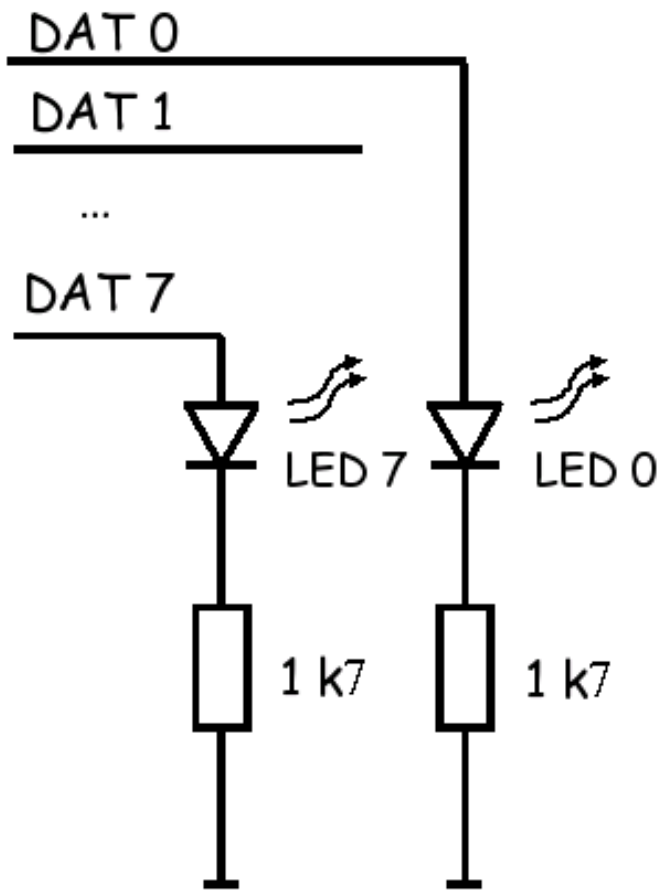
`PPCLAIM` richiede accesso alla porta. Necessario prima di riuscire ad "operare" sulla porta.

`PPWXXXX` setta i bit della linea `XXXX`. Il parametro è un puntatore ad un `unsigned char`.

`PPRXXXX` legge i bit della linea `XXXX`. Il parametro è un puntatore ad un `unsigned char`.

dove `XXXX=(DATA,CONTROL,STATUS)`. Per `STATUS` non c'e' il comando di lettura.

Porta parallela (II)



Nell'esercitazione di laboratorio ogni bit della linea dati (DATA) è collegato ad un led:

- bit a 1 led acceso
- bit a 0 led spento

Il registro dati ha 8 bit tutti disponibili per la scrittura (e la lettura):

`0xff` tutti i led accesi

`0x00` tutti i led spenti

Esempio di programma per pilotare la porta parallela in scrittura


```
int main() {
  int fd;
  fd = open("/dev/parport0",O_WRONLY);  // apertura
  if (fd == -1){
    ... // messaggio di errore
    return 1;
  }
  if (ioctl(fd,PPCLAIM)){               // accesso alla porta
    ... // messaggio di errore
    return 1;
  }

  unsigned char ch=...;
  ioctl(fd,...,...);                    // scrittura

  close(fd)
  ...
}
```

N.B. Mancano gli `#include` opportuni, individuateli con l'ausilio di `man`.

Esercitazione in laboratorio: porta seriale

Modificate il programma in modo da eseguire seguenze di spegnimento/accensione (da destra a sinistra e viceversa). Utilizzate per tale scopo gli operatori bitwise di shift (<< e >>).

È utili utilizzare brevi pause tra una scrittura e l'altra (`sleep` e `usleep`) in modo da avere il tempo di visualizzare l'effetto.

N.B. Per stampare un numero in formato esadecimale

```
unsigned char num;
cout << hex << num << endl;
```

per ritornare in modo decimale

```
cout << dec;
```