Programmare utilizzando porte di I/O sotto Linux

Questa estate, assieme a due compagni di università, abbiamo sviluppato un programma che trasferisce file attraverso la porta parallela sotto linux, come progetto di Sistemi 1.
La porta parallela, come tutti ben sapete, è stata inserita nei personal computer per rendere più veloce l'invio di documenti alle stampanti, e per questo motivo ancora oggi i pin che compongono il cavo sono chiamati secondo le funzioni originali. Questo è lo schema per costruirsi da soli il cavo per il trasferimento, chiamato laplink (ogni riga è un filo del cavo):

Connettore 25 PIN D-SUB MALE al pc host		**Connettore 25 PIN D-SUB MALE al pc client**	
Nome	**Pin**	**Pin**	**Nome**
Data Bit 0	2	15	Error
Data Bit 1	3	13	Select
Data Bit 2	4	12	Paper Out
Data Bit 3	5	10	Acknowledge
Data Bit 4	6	11	Busy
Acknowledge	10	5	Data Bit 3
Busy	11	6	Data Bit 4
Paper Out	12	4	Data Bit 2
Select	13	3	Data Bit 1

Error	15	2	Data Bit 0
Reset	16	16	Reset
Select	17	17	Select
Signal Ground	25	25	Signal Ground

Qui si notano i nomi dati ai pin ed il fatto che questi sono incrociati, per permettere l'invio e la ricezione dei dati in entrambi i computer. Questo cavo prevede solamente 5 pin di ricezione dei dati, contro i 12 di invio disponibili nella porta: per questo motivo non possiamo utilizzare la stessa banda disponibile quando utilizziamo le stampanti.
L'utilizzo della parallela al posto della seriale, molto più semplice da gestire, è motivato dal maggior numero di cavi che trasferiscono i dati, cosa che la rende molto più veloce.
In questi ultimi anni lo standard USB sta prendendo il posto delle vecchie porte dei pc, ma comprendere il funzionamento di una porta abbastanza semplice serve per imparare a programmare qualsiasi porta di input/output, e quindi a gestire qualsiasi tipo di periferica esterna al pc (modem, scanner, webcam...).
In questo articolo tenterò di spiegare come programmare una porta nel linguaggio C compilato per il sistema operativo GNU/Linux con il gcc.
Linux ci fornisce un'area di memoria per ogni porta disponibile nel nostro pc, sulla quale possiamo scrivere e leggere i dati, ignorando quasi completamente l'architettura che esegue fisicamente il trasferimento. Ad ogni area di memoria di questo tipo, chiamato registro della porta, corrisponde un indirizzo che di solito è lo stesso per ogni personal computer, tramite il quale possiamo inviare i nostri dati, scrivendoli con una funzione particolare del C che vedremo più avanti.
La dimensione del registro è sempre di un byte, qualsiasi tipo di porta si stia utilizzando, ma il numero dei registri disponibili per ognuna è diverso in base alle sue specifiche fisiche.
Nel caso della parallela, i registri sono 3, corrispondenti ai diversi pin del cavo, che sono suddivisi in tre categorie di funzionamento:
Dati : gruppo composto da 8 pin, utilizzati per inviare i bit dei dati
Stato: composto da 8 pin, serve per ricevere segnali di stato dalla stampante
Controllo: composto da 6 pin, serve per mandare segnali alla stampante
Il sistema quindi ci fornisce tre registri, dagli indirizzi consecutivi, che ci permettono di gestire tutti e tre i gruppi di segnali. La porta LPT1 standard ha l'indirizzo esadecimale di memoria che parte da 378h, quindi lo schema dei registri sarà il seguente:

Indirizzo	Rapporto con l'indirizzo di base		Nome registro
378h	base	=	Data Register
379h	base+1	=	Status Register
37ah	base+2	=	Control Register

In base alla configurazione del bios, o alla porta che si utilizza (LPT1, LPT2, ecc.) il valore di partenza degli indirizzi cambia: i possibili valori di base sono 3BCh, 378h, 278h.
Proviamo quindi un primo stupido programma per inviare qualcosa alla porta:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/io.h>
#define indirizzodibase 0x378 /* lpt1 */

int main()
{
  int lettura;
  /* Prendo accesso alla porta, chiedendone i permessi per aprirla */
  if (ioperm(indirizzodibase, 3, 1)) {perror("ioperm"); exit(1);}

  /* Faccio in modo che tutti i pin di dati vadano a 0 */
  outb(0,indirizzodibase);

  /* Leggo dal registro di stato (BASE+1) e mostro il risultato */
  lettura = inb(indirizzodibase + 1); //leggo la porta
  printf("Stato: %d\n",lettura); //scrivo quello che ho letto su                    schermo

  /*Non abbiamo più bisogno della porta, la chiudiamo*/
  if (ioperm(indirizzodibase, 3, 0)) {perror("ioperm"); exit(1);}

  exit(0);
}
```

Per compilare il programma di esempio, salvatelo con il nome esempio.c e da shell di testo eseguite il comando:

```
#>gcc esempio.c -o esempio
```

Per eseguirlo come al solito eseguite:

```
#> ./esempio
```

NB: il "./" iniziale è fondamentale, perchè dobbiamo sempre esplicitare il path relativo dell'eseguibile, per farlo correre.
Abbiamo introdotto, con questo semplice esempio, la funzione che chiede il permesso di gestire la porta: ioperm. La sintassi è ioperm(indirizzo_della_porta, numero_indirizzi, accedi), dove il numero degli indirizzi è quello dei registri consecutivi che si vogliono aprire (nel nostro caso, come ricorderete, sono 3: dati, stato e controllo), ed accedi è 1 se si vuole aprire la porta, 0 se si vuole rilasciare. Nel sistema operativo Linux, non tutti gli utenti hanno il controllo totale del sistema, e la gestione dei permessi è di vitale importanza per la sicurezza dei servizi di rete. Per utilizzare un programma che fa uso della funzione ioperm, che fornisce accesso a tutte le periferiche del sistema, dobbiamo essere utenti "root", o avere gli stessi diritti. Un modo per evitare questo problema è utilizzare il programma "sudo", oppure dividere il programma in due parti, una delle quali fornirà accesso alla periferica, e sarà eseguita dall'utente root, mentre l'altra conterrà il resto del codice.

Un'altra cosa fondamentale che possiamo capire da questo programma sono le funzioni di scrittura e lettura su una porta. Per scrivere usiamo outb(dato_da_scrivere, indirizzo), per leggere usiamo inb(indirizzo_da_leggere). Fate qualche prova per acquisire familiarità: potete leggere e scrivere nei registri, anche se non avete un cavo collegato alla parallela!
Per spedire un dato da un pc all'altro, collegati tramite il cavo laplink, bisogna scrivere nel registro di dati nel primo pc, e leggere nel registro di stato nel secondo (controllate la prima tabella dell'articolo per la corrispondenza dei pin)
Il problema difficile che abbiamo dovuto risolvere per spedire un file, era capire come veniva trasportato il byte scritto nel registro, e leggere nel registro di stato del secondo pc solamente i bit che venivano ricevuti (in rete non esiste documentazione a riguardo). Se diamo un'occhiata di nuovo allo schema del cavo, vediamo che solo i 5 bit meno significativi del byte verranno veramente spediti al secondo computer: dobbiamo quindi leggere solamente questa porzione del bite del registro. Questo schema riassume la situazione che si viene a creare quando scriviamo un byte nel registro dei dati, e nell'altro pc lo leggiamo nel registro di stato:

Pc che invia (data register)	7	6	5	4	3	2	1	0	
Cavo laplink				o	|	|	|	|	
Pc che riceve (status register)	***Il 7 è negato dall'hardware***			7	6	5	4	3	2	1	0

Per scrivere un byte nel registro dei dati, lo trasformiamo in base binaria e tramite questo schema sappiamo il valore di ogni pin (che può essere ovviamente solo 1 o 0). Stessa cosa accade in lettura, stando attenti che i valori che ci interessano sono solamente quelli dei pin 4,3,2,1,0.

Esempio: leggendo il numero 9 dal registro, sappiamo che i pin 0,3 sono a 1, mentre gli altri sono a 0. Se leggiamo 73, sappiamo che i pin 0,3,6 sono a 1, ma a noi interessano solo i pin 0,3.

Per trasferire un file, quindi, lo abbiamo letto byte per byte, diviso il byte in due parti (chiamati nibble), e trasferito un nibble alla volta utilizzando gli ultimi 4 bit del registro di dati.
Per sapere quando spedire un nibble successivo, abbiamo deciso di utilizzare il 5° bit a nostra disposizione, tramite il quale il pc che spedisce il file comunica all'altro di aver terminato il ciclo.
Il ricevente, leggendo continuamente questo pin, sa quando deve leggere il registro, e dopo aver terminato manda un segnale scrivendo nel suo registro di dati (che corrisponde al registro di stato del pc server) un nibble predefinito.
Un'altro problema da affrontare quando si trasferiscono file di qualsiasi tipo è il controllo che quello che abbiamo inviato sia stato ricevuto correttamente. Per risolvere questo problema abbiamo utilizzato un algoritmo chiamato CRC32 (Cyclical Redundancy Check algorithm), che legge una parte del file, e restituisce un codice di 32 bit. Questo codice è strettamente legato ai dati che vengono passati all'algoritmo, perchè il risultato è sensibile anche a differenze di un bit, grazie ad una propagazione ridondante dell'errore, e questo permette il controllo che due copie di una stessa porzione di file siano identiche.
Il pc che spedisce i dati, spedisce in coda anche il suo codice crc calcolato sui dati spediti.
Il pc cliente calcola a sua volta il codice e lo confronta con quello ricevuto: se i due codici sono uguali, allora il trasferimento prosegue con la prossima porzione di file.
Finisce qui questo viaggio introduttivo alla programmazione delle periferiche esterne del pc. Sotto DOS e Windows valgono gran parte delle cose che abbiamo esposto, tranne le funzioni che si utilizzano per scrivere e leggere sui registri, che sono leggermente

diverse.
Per ulteriori informazioni potete visitare il sito http://lptransfer.sourceforge.net∞, dove abbiamo inserito la relazione del progetto ed il codice sorgente del nostro programma, che nel frattempo continua ad essere sviluppato.