Qualche informazione sui tipi di socket

- I socket (o “le socket”?) rappresentano uno strumento di comunicazione estremamente versatile
- Vedremo solo alcuni aspetti generali

Esistono due principali modi per comunicare tramite socket:

- il connection oriented model
- il connectionless oriented model

In corrispondenza a questi due paradigmi di comunicazione si possono usare i seguenti tipi di socket:

- Stream socket: forniscono stream di dati affidabili, ordinati. Nel dominio Internet sono supportati dal protocollo TCP (Transmission Control Protocol).
- Socket datagram: trasferiscono messaggi di dimensione variabile, preservando i confini ma senza garantire né l’ordine né l’arrivo dei pacchetti. Supportate nel dominio Internet dal protocollo UDP (User Datagram Protocol).

Il dominio Internet: indirizzi IP

- Un indirizzo IP (Internet Protocol) è costituito da quattro numeri decimali interi (con valori da 0 a 255) separati da punti (es. 128.110.3.7)
- consente di individuare univocamente la posizione di una sottorete e di un host all’interno di quest’ultima.
- Per utilizzare gli indirizzi IP in programmi C, bisogna prima convertirli nel tipo apposito tramite la seguente system call:

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int inet_aton(const char *ip_add, struct in_addr *inp);
```

Il dominio Internet: indirizzi IP

- Esempio:

```
struct sockaddr_in indirizzo;

if (inet_aton("128.110.3.7", &indirizzo.sin_addr) == 0) {
    perror("errore in inet_aton");
    exit(1);
}
```

- `inet_aton()` converte un indirizzo, dato in forma di stringa come in `"128.110.3.7"`, e lo memorizza nella struttura puntata da `inp`.
- ritorna 0 se l’indirizzo non è valido

Il dominio Internet: indirizzi IP

Nel file header `netinet/in.h` sono definiti:

- Il tipo:

```
typedef uint32_t in_addr_t;

struct in_addr {
      in_addr_t s_addr;
};
```

(quindi `in_addr_t` rappresenta un indirizzo nello spazio occupato da un intero senza segno a 4 byte)

- La costante simbolica

  `INADDR_ANY`

che (semplificando!) rappresenta l’indirizzo/i dell’host su cui è in esecuzione il programma.

Il dominio Internet: porte

- Oltre a conoscere l'indirizzo IP dell'host a cui connettersi, bisogna disporre dell'informazione sufficiente per collegarsi al processo server corretto.
- Per questo motivo esistono i numeri di porta (port number).
  Ogni servizio viene associato ad una precisa porta.
- Le connessioni avvengono sempre specificando sia un indirizzo IP che un numero di porta.
  - Le porte da 0 a 1023 sono solitamente riservate per servizi standard (e.g., FTP, HTTP, SSH, ecc.),
  - mentre le porte da 1024 a 65535 sono solitamente disponibili per i processi utente.

Strutture dati

L'indirizzo ed il numero di porta devono essere memorizzati in apposite strutture dati (definite negli header). La struttura dipende dal tipo di socket. In particolare:

- Per descrivere un socket generico ("raw"):

```
struct sockaddr {
  short sa_family;    /* Address family */
  char sa_data[];     /* Address data. */
};
```

- Per indicare un socket nel dominio `UNIX` (socket "locali")

```
#define UNIX_PATH_MAX  108

struct sockaddr_un {
  sa_family_t   sun_family;     /* Costante AF_UNIX */
  char sun_path[UNIX_PATH_MAX]; /* Path name */
};
```

in questo caso la comunicazione si "appoggia" sul file specificato

Strutture dati

Esempio nel dominio `UNIX`

```
  ...
struct sockaddr_un mio_indirizzo;

mio_indirizzo.sun_family = AF_UNIX;

strcpy(mio_indirizzo.sun_path, "/tmp/mio_socket");
  ...
```

(si veda anche `man 7 unix`)

Strutture dati

- Per indicare un socket nel dominio `INET`

```
struct sockaddr_in {
  sa_family_t    sin_family; /* Costante AF_INET */
  u_int16_t      sin_port;   /* Porta (ushort) */
  struct in_addr sin_addr;   /* indir. IP */
};
```

dove, come prima:

```
struct in_addr {
  u_int32_t  s_addr;  /* ulong */
};
```

(si veda `man 7 ip`)

Strutture dati

Esempio nel dominio `INET`

```
  ...
struct sockaddr_in mio_indirizzo;

mio_indirizzo.sin_family = AF_INET;
mio_indirizzo.sin_port   = htons(5200);
inet_aton("128.110.3.7", &mio_indirizzo.sin_addr)
  ...
```

oppure:

```
  ...
mio_indirizzo.sin_family      = AF_INET;
mio_indirizzo.sin_port        = htons(5200);
mio_indirizzo.sin_addr.s_addr = htonl(INADDR_ANY);
  ...
```

dove `htons` e `htonl` convertono tenendo conto della "endian-ness"

System call

Per comunicare attraverso un socket, due processi devono compiere una serie di passi:

- sia server che client devono dapprima definire i rispettivi transport end point (`socket()`)
- il server deve legare l'end point all'indirizzo dell'host (`bind()`)
- il server si mette in uno stato di ascolto "passivo" (`listen()`, alloca le strutture per gestire una coda di client)
- il client richiede la connessione (`connect()`)
- il server accetta la connessione (`accept()`)
- la comunicazione procede (bidirezionalmente) tramite `send()` e `recv()` (ma anche `write()` e `read()`)
- chiusura del socket (`close()` o `unlink()`)

Creazione transport end point

Si effettua tramite la seguente chiamata di sistema:

```
#include <sys/socket.h>

s = socket(int domain, int type, int protocol);
```

dove:

- `domain` indica il dominio del socket (e.g., `PF_INET` o `PF_UNIX`) (più info in `man socket`, in `man 7 ip` e in `man 7 unix`)
- `type` indica se verrà o meno utilizzato il paradigma connection oriented (`SOCK_STREAM`) oppure quello connectionless (`SOCK_DGRAM`);
- `protocol` specifica il protocollo da utilizzare (se impostato a 0, in S.O. sceglie il protocollo in base ai primi due parametri)
- il valore ritorno è un descrittore del socket (int) oppure -1 in caso d'errore

Associazione dell'indirizzo

La seguente system call (eseguita dal server) associa l'end point all'indirizzo

```
#include <sys/types.h>
#include <sys/socket.h>

int bind(int sockfd,
         const struct sockaddr *address,
         size_t add_len);
```

- `sockfd` è il valore restituito da `socket()`
- `sockaddr` è la struct che descrive l'indirizzo (vedi lucidi precedenti) N.B.: a seconda dei casi si userà una struct di tipo `sockaddr_in`, `sockaddr_un`, ... e un cast al tipo `sockaddr *`.
- `add_len` è la dimensione della struct

Esempio:

```
bind(sockfd, (struct sockaddr *) &mio_indirizzo,
                              sizeof(mio_indirizzo));
```

Stato d'ascolto del server

Il server si mette in ascolto per mezzo della seguente system call:

```
#include <sys/socket.h>

int listen(int sockfd, int queue_size);
```

dove `queue_size` indica la dimensione della coda di clienti in attesa della attivazione della connessione

ovvero, il massimo numero di client che possono restare in coda in attesa che il server accetti la/le connessione/i.

NON è il numero di client che possono essere serviti contemporaneamente!

Stabilire una connessione

Il cliente si connette con una `connect` al server in corrispondenza del socket indicata dalla struttura `address`:

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int csockfd,
            const struct sockaddr *server_address,
            size_t add_len);
```

Il server accetta una connessione tramite la chiamata di sistema `accept`:

```
int accept(int sockfd,
           struct sockaddr *client_address,
           size_t *add_len);
```

Accettando una richiesta di connessione, il server crea un nuovo socket (il cui descrittore è il valore di ritorno di `accept()`) che verrà usato per comunicare con il cliente

Il client userà il valore ritornato da `connect()`

La comunicazione

Stabilita la connessione sia server che client possono usare le system call `send()` e `recv()` per trasmettere e ricevere dati

```
#include <sys/types.h>
#include <sys/socket.h>

ssize_t send(int sockfd,
             const void *buffer,
             size_t length, int flags);

ssize_t recv(int sockfd,
             void *buffer,
             size_t length, int flags);
```

se `flags` vale 0, allora `send()` e `recv()` equivalgono, rispettivamente alle system call `write()` e `read()`.
Altrimenti `flags` può essere utilizzato per modificare le modalità di spedizione e ricezione (vedi `man`... o i/il corso/i di reti...)
Ad esempio il flag `MSG_DONTWAIT` può essere usato per realizzare comunicazioni non-bloccanti.

Esercitazione

Esercizio (socket locali):
Si realizzi un server *maiuscolatore* che riceve delle stringhe di testo dai clienti e le restituisce agli stessi dopo averne convertito in maiuscolo tutte le lettere.

Esercizio (socket locali):
Si realizzi un server *maiuscolatore*, che sia in grado di servire più clienti contemporaneamente. [SUGG: combinare `accept()` con `fork()`...]

Esercizio (socket locali):
Completare gli esercizi prevedendo che ci sia un processo "generatore" che crea il server ed i clienti (ad esempio uno ogni *x* secondi).
La creazione dei clienti continua fino a che il processo generatore non riceve un segnale `SIGINT`. A tal punto il generatore invia un particolare messaggio al server che lo gestisce terminando.
Implementare il tutto (le modalità sono a vostra discrezione) in modo che prima della terminazione tutti i clienti ancora esistenti siano serviti.

Esercizio:
Documentarsi sull'errore EADDRINUSE, quando/perchè si verifica?

Esercizi:
Ripetere gli esercizi precedenti nel caso delle socket INET

Esercitazione

Riusare l'indirizzo:

```
#include <sys/types.h>
#include <sys/socket.h>

/* Enable address reuse */
on = 1;
ret = setsockopt( sockdf,
                  SOL_SOCKET,
                  SO_REUSEADDR,
                  &on,
                  sizeof(on) );
```

(vedi `man setsockopt`)