**SOCKET**

- Sono file speciali utilizzati per connettere due o più processi con un --canale di comunicazione (anche tra macchine diverse).

Come collegare due processi

- Server

1) Creare un file ricevitore (socket) e allocare un file descriptor.

sfd = socket(...)

2) Associare il socket sfd con un indirizzo (è una stringa).

bind(sfd, "mysocket", ...)

3) Specificare che sul socket siamo disposti ad accettare connessioni da altri processi.

listen(sfd, ...)

listen() dice al kernel di gestire il socket in modalità passiva (ovvero fa sì che su questo socket sia possibile accettare connessioni).

4) Bloccarsi in attesa di richieste di connessioni da parte di altri processi.

sfd1 = accept(sfd, ...)

sfd1 descrittore usato per comunicare tra client e server.

- Client

1) Creare un socket e il relativo file descriptor.

csfd = socket(...)

2) Collegarsi con il socket del server usando il nome esportato con la bind().

connect(csfd, "mysocket", ...)

- Se c'è un match tra accept() e connect() si crea la connessione e la ----accept() che era in attesa ritorna un altro descrittore, in questo caso --sfd1, che può essere usato per comunicare tra client e server usando le --SC write(), read(), close().

- Il server si può rimettere in attesa di connessioni da altri client con --un'altra accept().

Creazione

fd\_skt = socket(AF\_UNIX, SOCK\_STREAM, 0); per noi sempre così

Dare il nome

int bind(

int sock\_fd, file descriptor socket

const struct sockaddr \*sa, indirizzo

socklen\_t sa\_len lunghezza indirizzo

);

#define UNIX\_PATH\_MAX 108

struct sockaddr\_un { andrà fatto un cast

sa\_family\_t sun\_family; per noi AF\_UNIX

char sun\_path[UNIX\_PATH\_MAX]; path del socket

} ;

e.g. bind(fd\_skt, (struct sockaddr \*)&sa, sizeof(sa));

Accettare connessioni

int listen( mette il socket in modalità passiva

int sock\_fd,

int backlog mettiamo cosa ci pare (> 0)

);

---------------------

int accept(

int sock\_fd,

const struct sockaddr \*sa, indirizzo o NULL

socklen\_t sa\_len

);

Ritorna fdc (nuovo file descriptor) in caso di successo, -1 in caso di errore (setta errno).

- Se non ci sono connessioni in attesa si blocca, altrimenti accetta una delle connessioni in coda e crea un nuovo socket.

Gestire più client

- Dopo aver creato la prima connessione, il server deve -------------------contemporaneamente mettersi in attesa sulla accept() per una nuova ------connessione e mettersi in attesa con read() dei messaggi in arrivo dai --client già connessi.

--In altre parole il server vuole gestire tutte le connessioni attive e ---contemporaneamente riuscire ad accettare nuove connessioni senza mai ----bloccarsi.

- Due soluzioni (possono essere combinate).

- Usare un server multi-threaded:

- un thread dispatcher che si mette in attesa ripetutamente su accept() per una nuova connessione.

- ogni volta che la connessione è stata stabilita viene creato un nuovo thread worker che si mette in attesa con read() dei messaggi in arrivo solo dal client relativo ad una particolare connessione.

- Usare un server sequenziale e la SC select() che permette di capire quando un file è pronto.

Aspettare I/O da più file descriptor

int select(

int nfds, numero di fd

fd\_set \*rdset, insieme di lettura o NULL

fd\_set \*wrset, insieme di scrittura o NULL

fd\_set \*errset insieme di errore o NULL

struct timeval \*timeout timeout o NULL

);

Ritorna il numero di bit settati in caso di successo, -1 in caso di errore (setta errno).

- Si blocca finché uno dei descrittori specificati è pronto per --essere usato in una SC senza bloccare il processo:

- un descrittore fd atteso per la lettura (rdset) è considerato pronto quando una read() su fd non si blocca, quindi se ci sono dati o se è stato chiuso (con EOF la read() non si blocca).

- All'uscita della select() i set sono modificati per indicare chi è --pronto.

- rdset, wrset, errset sono maschere di bit.

- FD\_ZERO(&fdset) azzera la maschera.

- FD\_SET(fd, &fdset) mette a 1 il bit corrispondente al file descriptor fd.

- FD\_CLR(fd, &fdset) mette a 0 " " .

- FD\_ISSET(fd, &fdset) vale 1 se il bit corrispondente a fd nella maschera è a 1, 0 altrimenti.