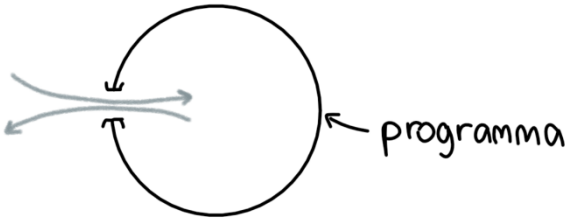
---------- seti 11/10 ----------

API **POSIX** **SOCKET**

SOCKET -> INTERFACCIA DI PROGRAMMAZIONE USATO DAL POSIX PER ACCEDERE ALLE FUNZIONALITÀ DELLA RETE

UN SOCKET È UN FILE DESCRIPTOR DAL PUNTO DI VISTA DEL SISTEMA OPERATIVO

SOCKET -> ATTRAVERSO UNA SYSTEM CALL IL NOSTRO PROCESSO CHIEDE AL SISTEMA OPERATIVO DI POTER INTERAGIRE CON L'ESTERNO (COME SE CREASSIMO UN BUCO NELL'INVOLUCRO DEL PROCESSO, DA CUI POSSO INVIARE E RICEVERE MESSAGGI)



QUANDO SI CREA UN SOCKET SI DEVE DIRE A COSA FA RIFERIMENTO; QUINDI CI SONO DIVERSI TIPI DI SOCKET (QUANDO VIENE ISTANZIATO SI DEVE DICHIARARE DI CHE TIPO È):

- **SOCK_DGRAM** -> CON DATAGRAMMA UDP
- **SOCK_STREAM** -> CON PROTOCOLLO TCP } principali
- SOCK_RAW

LE MODALITÀ DI FUNZIONAMENTO DEL SOCKET DI TIPO DATAGRAM:

PASSIAMO COME PARAMATRO DELLA SYSTEM CALL IL SOCKET CHE VOGLIAMO OTTENERE E SE TUTTO VA BENE NELLA VARIABILE DI TIPO INT ABBIAMO IL FILE DESCRIPTOR DEL SOCKET; SE QUALCOSA VA STORTO RESTITUISCO VALORE -1

```
int fd = socket (dgram);
```

DOPO AVER CREATO IL SOCKET DECIDO SE VOGLIO USARLO PER INVIARE O PER RICEVERE MESSAGGI

- SE VOGLIO INVIARE UN MESSAGGIO;

```
send_to (fd (socket su cui voglio operare),
        ind (indirizzo destinatario -> indirizzo ip e numero di
        porta),
        buffer (con dati da inviare),
        lunghezza (numero di byte del buffer))
```

ESEMPIO:

```
char * b = "ciao"
send_to (ind, b, 4)
```

VIENE INVIATO MA NON È DETTO CHE VIENE RICEVUTO IN QUANTO PROTOCOLLO DATAGRAM

COME FACCIO A SAPERE L'INDIRIZZO IP E IL NUMERO DI PORTA DEL DESTINATARIO? IL NOSTRO PROGRAMMA DEVE AVERE UNA SUA MODALITÀ CHE TI PERMETTE DI SAPERLI

- GLI INDIRIZZI IP VENGONO ASSEGNATI DALL'AMMINISTRATORE DEL SISTEMA -> DIVISO IN DUE PARTI: INDIRIZZO DELLA SOTTORETE E NUMERO DI HOST -> SEMPRE QUELLO
- IL NUMERO DI PORTA VIENE ASSEGNATO DINAMICAMENTE, NON È PREDEFINITO -> PUÒ ESSERE OTTENUTO CON UNA SYSTEM CALL `bind (fd, int (numero di porta))`

- Se vogliamo ricevere un messaggio:

```
recv_from (fd (socket su cui voglio operare),
           buffer (che metto a disposizione, dove ricevo),
           lunghezza (byte liberi nell'area di memoria buffer),
           &ind (indirizzo in uscita, da dove ho ricevuto))
```

  per ricevere dobbiamo aspettare che qualcuno faccia da send

  Prima ipotesi : prima il mittente invia il messaggio e dopo un po' il destinatario fa la recieve -> in questo caso il messaggio inviato dal mittente deve essere memorizzato all'interno della memoria della macchina dell'host del destinatario -> abbiamo bisogno quindi di buffer di ricezione gestiti dal sistema operativo -> successivamente il processo che esegue la recieve deve specificare un buffer di ricezione diverso da quello in cui è stato contenuto il messaggio da cui è stato appena ricevuto -> il sistema operativo copia il contenuto del buffer di ricezione all'interno del buffer specificato dal ricevente, il buffer originale gestito dal sistema operativo può essere cancellato e riutilizzato per ricevere altri messaggi -> **finche il destinatario non chiama la recieve il messaggio arrivato deve rimanere memorizzato nel buffer del sistema operativo** -> il sistema operativo mette a disposizione una certa quantità di memoria per questi buffer -> se il destinatario non legge i messaggi dal buffer di ricezione, il sistema operativo inizierà a scartare i messaggi per inserire i nuovi messaggi da ricevere -> quindi la recieve non è "ricevo il messaggio dalla rete in questo momento" ma "leggo i messaggi che ho ricevuto e li copio nel mio buffer di memoria"

  Seconda ipotesi : il destinatario esegue la recieve prima che il messaggio sia effettivamente ricevuto -> il mittente ha inviato il messaggio ma è ancora in transito nella rete oppure il mittente non ha proprio ancora inviato il messaggio -> quasi tutte le system call in ambiente posix hanno una **semantica bloccante** -> se non è verificata la condizione necessaria per procedere, il processo viene dissimulato -> se il programma esegue la recieve e il messaggio non è ancora stato ricevuto, il programma viene interrotto, rimane in attesa fin quando non arriva un messaggio -> il processo che è in attesa viene risvegliato e a quel punto completa l'operazione di copia del messaggio dal buffer del sistema operativo al buffer di memoria, poi termina l'esecuzione del programma

  quindi la system call (in una semantica bloccante) inizia l'esecuzione ma non necessariamente procede fino alla fine : se non è ancora arrivato il messaggio l'esecuzione viene bloccata a metà, e solo dopo che è arrivato viene ripresa e completata l'esecuzione

Quando non serve più un socket posso chiuderlo usando **`close (fd)`**

Connessione client server, usando una connessione 3-way handshake:

		stream	
	client		server
int fd = socket (stream)			df = socket ( )
	getaddrinfo		getaddrinfo
	connect (fd, ind)		bind (fd, ind)
	send (fd)	write	listen (fd)
	recv (fd)	read	ff2 = accept (fd
			ff3 = accept (fd,

Nel punto di partenza definiamo un socket di tipo stream, poi chiamo la system call connect (con l'indirizzo a cui devo essere connesso) -> quando termina la connect può ritornarmi un errore o dirmi che la connessione è stata effettuata. Nella send e nella recv non devo aggiungere altro se non fd perché la connessione è già avvenuta.
Posso aggiungere altre system call come write (per scrivere all'interno di un file) o read -> sinonimi di send e recv (?)
Il primo handshake viene gestito dal sistema operativo; per stabilire la connessione dal lato del server si una un'altra system call chiamata **accept** (ossia rispondi con syn ack) accept non mi ritorna semplicemente un si o un no, mi ritorna un altro file descriptor (ff2), diverso dall'iniziale, che server per comunicare con il client che si è connesso quindi il client ha un solo socket, il server ne ha almeno due: uno sul quale avviene l'handshake, e uno secondario sul quale avviene la comunicazione con il client connesso -> il vantaggio è che a questo punto il server può chiamare più volte l'accept, per **accettare la connessione da più di un client** -> quindi più client possono collegarsi allo stesso server
listen -> da questo punto in avanti mi metto in ascolto