Sockets
A noi interessa saper interagire con TCP e IP.
Per interagire si utilizza l'interfaccia Socket = si tratta di quel tramite tra l'applicazione ed il kernel del sistema, rappresenta il dialogo tra applicazione e sistema operativo.
Compito del socket è di far dialogare processi che non sono nella stessa macchina.

- Socket dal punto di vista figurativo andrebbe visto come un box, dal punto di vista del sistema operativo è come vedere un file = dove si può scrivere, leggere etc. Ma scrivere significa mandare un informazione all'altro socket al quale è collegato il socket.
- Il socket infatti è sempre legato ad altri socket ed è tramite il loro collegamento che si trasmettono le informazioni.
- Esistono due tipi di comunicazione tra socket : con connessione TCP/UDP , senza connessione IP
- I socket rappresentano degli end-point e come tali devono essere riconoscibili : essi infatti hanno un'indirizzo di rete, ma esistono anche altri modi per individuarli.
- Due end-point si inviano informazioni principalmente tramite i protocolli implementati a livello di rete ovvero TCP o UDP
- A livello IP = si parla di macchina, con instradamento da macchina a macchina
- A livello di Socket si parla in end-to-end = Host che riceve infinite informazioni deve sapere indirizzare queste informazioni al destinatario giusto. Il traffico quindi dentro la macchina deve anch'esso essere smistato per essere inoltrato all'opportuno socket. L'identificazione del socket viene fatta tramite un numero di porta (port number).
- Ciò che caratterizza una connessione tra due socket sono gli HOST number (indirizzi IP) e port number

Client-Server Approach : Esistono diversi tipi di server (file system server, mail server) che sono in attesa di operazioni e connessioni e quando ricevono una richiesta si attivano tramite i protocolli per esaudire le richieste del client.
Come faccio a individuare il server a cui mi devo rivolgere?
Poniamo caso di ricevere richiesta HTTP con un url. l'url viene tradotto in DNS in formato numerico.

Quello rappresenta la prima parte dell'indirizzo dell'host , manca però il port number.
Esistono dei servizi che hanno delle porte ben conosciute, note a tutti (email, ftp, snmp(gestione rete), http..) = WELL KNOWN PORT . Esiste un certo range di numeri nel quale è possibile assegnare una WELL KNOWN PORT, ma questo avviene solo quando un servizio diventa molto utilizzato.
I numeri delle porte si dividono in tre classi = 0 - 1023 (riservati per well known port) / 1024-5000 porte effimere che gli utenti possono definire per programmi che creano / >3977 permessi da molti sistemi visto il numero di richieste in crescita .
Alle volte anche a noi come computer viene associata una porta (512-1023) perche fungiamo da server per il server a cui abbiamo richiesto qualcosa = quando ad esempio deve fare una verifica sul nostro sistema. Son comunque porte temporanee.

Comunicazione:
Sono un'applicazione e devo inviare delle informazioni.
Utilizzo quindi il Socket = una scatola.
Il socket comunica con il sistema operativo, comincia ad impacchettare i dati a seconda del tipo di protocollo che viene utilizzato. Quando ha finito chiama la scheda di rete (es. Ethernet) = HARDWARE che lo inserisce nella buffer descriptor list.. Il microprocessore della scheda di rete, prende l'informazione e la trasferisce sulla rete.

Socket Address Structure:
Le system call permettono di creare socket, dargli un nome, stabilire connessione con altri socket.
Le connessioni vengono attivate ma richiedono un passaggio di parametri come ad esempio IP adress e port number.
Questi parametri vengono memorizzati in strutture dati particolari:
sockaddr : lunghezza struttura +tipo di rete(af_inet) + port number + indirizzo ip
Questa struttura viene gestita dall'applicazione e dal sistema operativo come fossero 4 mani. A seconda se si è client o server la struttura viene compilata dall'applicazione o dal sistema operativo.
Dal pdv programmativo prima di usare la struttura si azzera, cioè si crea e si azzera, perchè può capitare che nell'interazione tra sistema operativo e l'applicazione non ci serve di memorizzare certi dati, ma nell'utilizzo un campo non azzerato potrebbe essere interpretato male e dare errore.

Queste strutture vengono poi passate come parametri con le system call. Esistono due tipi di chiamate : da OS a user (accept, recvfrom) e da user a OS (bind, connect, sendto).
Queste funzioni devono essere utilizzate con una sequenza che sarà sempre uguale perchè nel loro utilizzo faranno attivare l'utilizzo di terze parti come i protocolli TCP e UDP.

Esempio connessione TCP Client-Server
Affinchè il client faccia una richiesta, il server deve esistere.

LATO SERVER. Definire e creare server = 4 istruzioni
***socket()* =** crea interfaccia socket tramite la quale posso gestire connessioni e dialogare con OS.
La chiamata prende come parametro il tipo di famiglia a seconda di con chi deve dialogare il socket (AF_INET , AF_INET6, AFLOCAl (dialoghi nella stessa macchina)). Prende anche come parametro il tipo di connessione (TCP o UDP), l'ultimo campo è solitamente sempre 0. La chiamata ritorna un intero che è il file descriptor.
***bind()* =** collega il socket creato ad un certo indirizzo IP ed ad una certa port number, che è necessario per essere reperibili. La bind utilizza la struttura sockaddr. Gli si passa il sockfd ovvero l'intero di file descriptor ricevuto dalla funzione socket(). Il secondo parametro è un puntatore alla struttura sockaddr. Il port number che si trova in questa struttura può essere WellKnown e quindi si inserisce un numero di porta già riconosciuta. Altrimenti invece si inserisce il numero 0, quando il OS legge numero di porta 0 sa che è lui che deve assegnarli una port number nel range degli effimeri. Terzo parametro è la lunghezza della struttura. Restituisce 0 o -1 a seconda del successo dell'operazione.
***listen()* =** prende l'identificatore del socket e gli crea una coda di connessioni e gli si passa come parametro il numero di connessioni che supporta, solitamente 5.
***accept()* =** funzione lato server, dove il server si blocca in attesa di richieste di connessioni per effettuare certe operazioni. Quando riceve una richiesta, si sblocca e crea un thread o un figlio che gestisce la connessione. Essa è una funzione da OS a processo utente, inserisce parametri cliaddr e addrlen che sono gli indirizzi dei client.

Es. Sono sito elezioni (server) e tutti i client vogliono caricare sito per vedere risultati.

Il sito è bloccato in accept - arriva una richiesta e devo soddisfarlo tramite la port number.
Ma non posso soddisfare la richiesta tramite quella well known port perchè risulterei occupato per gli altri.
Quindi il server ad ogni chiamata crea un nuovo socket figlio o thread i quali gestiscono la connessione che avviene direttamente con l'utente. Di modo che la wellknown port rimane libera e si riblocca sull'accept.

LATO CLIENT. Definire e creare server
Si differenzia però dal server, perchè non chiama il bind ma chiama..
***connect()* =** passa sempre file descriptor, sockaddr relativa alla struttura che voglio chiamare dove IP e port number devono essere compilate e lunghezza della struttura.

Send / Receive = ci si scambiano i dati - con send si manda l'identificatore del socket , il buffer,
Chi chiude la connessione, normalmente il client = chiude il socket.
La chiusura del socket farebbe bloccare il server sulla receive.
Gli manda quindi end of file cosi capisce deve chiudere anche lui la connessione.

Lato Server = Accettazione Richiesta
Lato Server si è in attesa di richieste con accept(). Quando si riceve una richiesta essa viene inserita in campo dati connfd e il padre fa una fork() = crea figlio che mette in connessione con connfd.
Il figlio gestisce la richiesta e dopodichè chiude la connessione. Terminata la gestione anche il padre chiude il connfd.
Il server gestisce la wellknown port = lisfd
Quando poi riceve richiesta, crea connfd ovvero nuova porta e crea il figlio il quale compito è gestire connfd.
Dopodiche chiude immediatamente il lato connfd per non interferire con il figlio.
Il figlio gestisce connfd ma cmq ha anche un riferimento a lisfd che chiude per non interferire con il padre.

Protocollo UDP = no connessione, no handshake, no fork.
recvfrom al posto di listen - ci si blocca in attsa di pacchetti, si viene sbloccati con recvfrom = con il numero di socket ed il puntatore from da chi ho ricevuto richiesta.
ci si blocca su questo con un certo numero di dati e al primo pacchetto

che ricevo mi sblocco

Funzioni che utilizzeremo per compilare sockaddr

- gethostname(char **dovescrivere hostname, int linghezzamassima nome)
- inet_addr (char * address) - converte indirizzo in forma dotted a indirizzo reale socket a 32bit - questo ‘ l’indirizzo che metteremo in sockaddr
- gethostbyname - conosco nome host in formato dotted stile url = mi ritorna indirizzo in AF_INET quindi 4 interi.

Berkley e WinSock - permettono di programmare i socket

IMPLEMENTAZIONE SOCKET

Fattori da minimizzare : Memory-Memory copies, Context Switches che richiedono molto tempo

Process Models.

Esistono due strutture per l’implementazione:

Process-per-protocol = faccio un processo per ogni protocollo e passo il messaggio ai vari processi.

- richiede tantissimo tempo di context switch tra i vari processi = perche tanti processi, c’e’ tanta competizione per cpu, tanti context switch, tanto tempo perso.
- ogni processo ha una coda di dati da elaborare e spedire al prossimo processo, e le code son memoria. Quando devo inoltrare dati da una coda al prossimo processo significa copiare informazioni = tempo perso.

Process-per-messagge = istanzio un processo per ogni richiesta che ricevo, il quale emette i vari protocolli. La richiesta va su e giu attraverso il protocol stack usando procedure abbastanza semplici che son decisamente più economiche di un context switch.

Message Buffer.

Zona di memoria dove porre i dati maneggati da TCP, UDP, IP ed Ethernet.

Quando voglio inviare informazioni, creo un buffer a livello applicativo, lo riempio delle informazioni e lo passo.

Invece che però passare il buffer copiandolo e passandolo a OS, io passo

a OS un puntatore alla zona di memoria senza copiare dati .

Ho 64 descrittori che identificano 64 thread che gestiscono buffer diversi sistemati come bucket list = un messaggio può stare su due bucket, ma su un bucket non può esserci più di un messaggio.
I thread creano i pacchetti inserendo header e trailer, e quando son pronti fanno una chiamata al sistema che li prende e li spedisce.
Tutto viene fatto all'interno dei bucket, senza esser trasferito e copiato, risparmiando tempo e risorse.
Il multi-thread combatto il context switch , risparmiando tempo.
Non posso gestire più di 64 messaggi ma ha comunque numerosi vantaggi- considerando che dopo ci saranno sicuramente altri colli di bottiglia.