recvfrom () richiede di specificare il numero di byte da leggere: la quantità richiesta è limitativa, se ci sono più byte questi non verranno letti in quella specifica lettura.

I byte letti sono contenuti all’interno di un *buffer di livello di trasporto* (definito proprio così, sicuro al 100%) gestito dal sistema operativo, non legato all’applicativo direttamente. Il buffer che noi utilizziamo per memorizzare il messaggio è invece legato solo e soltanto alla nostra applicazione.

Se voglio leggere tutti contenuti sul buffer, e sono maggiori della dimensione da me passata, i dati devo eventualmente effettuare più letture.

MI viene anche fornito, in client\_addr, l’identificativo del client. E l’unico modo che ho per ottenerlo ed eventualmente rispondergli.

La lettura può anche fallire, eventualmente posso stampare l’errore legato alla lettura stessa.

Sendto () vuole:

* Socket
* Dati
* Dimensione dati
* Informazioni client (ottenuti dalla recvfrom ())
* Dimensione informazioni client

Possibili errori potrebbero essere legati al fatto che il buffer di trasporto è pieno oppure è quasi pieno, con uno spazio non sufficiente per mantenere tutti i dati da inviare. È quindi necessario controllare che i byte inviati e quelli che si sarebbero dovuti inviare è uguale.

A livello di trasporto abbiamo due buffer: uno di trasmissione e uno di ricezione.

Noi mandiamo man a mano dati sul buffer di trasmissione che poi, l’entità di protocollo che si occupa di questo buffer, deciderà quando inviare rispettando le sue regole private.

Il codice non fornito non funziona su windows in quanto le librerie utilizzate fanno riferimento al file system di linux. Il codice inoltre presenta l’indirizzo 127.0.0.1, ovvero può funzionare solo se i due programmi vengono eseguiti sulla stessa macchina.

UDP è un protocollo che funziona a messaggi mentre TCP è un protocollo orientato al byte in quanto tratta un flusso di byte per comunicare. UDP invia messaggi singoli, fini a sé stessi. TCP effettua l’handshake al fine di scambiarsi un flusso di byte.

L’operazione di close è importante per il TCP in quanto permette di chiudere un flusso di comunicazione (uno solo in quando è full duplex, dunque è possibile fare la close da entrambi i lati). La close ha importanza anche nell’UDP in particolar modo in quando consente di liberare delle risorse bloccate.

Dup: permette di associare stdin, stout e stderror (solo dopo averli sganciati tutti) alla socket. Va chiamata 3 volte, e associa gli FD alla socket in ordine di grandezza: prima il numero 0, poi 1 e poi 2. Li associa però solo se non sono associati a nulla (quindi SGANCIARE da stdint, stdout e stderror prima di effettuare le tre chiamate).

execle: consente di mettere in esecuzione il servizio richiesto dal client. Ha un numero di parametri variabile, dipende dai primi parametri inseriti e dal tipo di programma che vogliamo mettere in esecuzione. Se non ci sono stati errori non ritorna nulla, mette in esecuzione l’immagine del processo richiesto. Parametri:

* Full path del programma (senza nome dell’applicazione)
* Nome del programma da eseguire
* Da terzo parametro in poi abbiamo la possibilità di passare un numero variabile di parametri al programma di cui è richiesta l’esecuzione. La lista viene terminata dal passaggio un parametro NULL.
* Infine dobbiamo passare una variabile che descrive l’ambiente (noi usiamo ‘env’)

Execl sostituisce il programma corrente in esecuzione sul processo con quello specificato.

E’ importante discriminare l’attivazione della select che può avvenire per due motivi diversi:

* Un processo figlio è terminato, quindi parte la signal con SIGCLD per segnalare l’evento
* Oppure se un file descriptor ha rilevato che bisogna, per esempio, leggere qualcosa da esso

Se handle\_signal viene chiamata siamo certi che l’evento che si sta verificando la chiusura del processo. Quindi posso usare la wait su quel segnale per ottenere il pid del processo morto, e risalire nella struttura dati per ottenere informazioni sul servizio appena terminato. Se il servizio era di tipo wait (ovvero sequenziale), dobbiamo reinserire il servizio nella select per renderlo nuovamente disponibile.

Descrizione task1: cosa è il diagramma e cosa descrive

Descrizione task2: Riportare nella relazione come compilare il codice (abbiamo fornito un make file, oppure..).

Quando si consegna verificare bene se compila e se funziona.

3\* ASSIGNAMENT

Se il messaggio arrivato dal client non va bene (numeri <0, ecc.., qualsiasi errore), viene inviato 404 error.

Nel campo payload nel messaggio di probe ci può essere un qualunque messaggio, a patto che la dimensione sia compresa nel campo dimensione precedentemente specificato nella fase di hello.

Se il massaggio di probe arrivato è valido, viene rispedito al mittente. Se il messaggio non è valido (ovvero, per esempio il numero di sequenza non è valido), invia un 404 error message e chiude la connessione TCP.

Quando arriva anche l’ultimo messagio di probe dal server al client(come copia di quello precedentemente inviato dal client), il client invia un messaggio di chiusura che deve ricevere conferma dal server. Se il server specifica tutto ok, allora il client chiude la connessione TCP.

Attenzione alle unità di misura quando si fanno i conti con throughput e rtt. HANNO UNITA DI MISURA DIVERSE.

E’ importante variare la dimensione del message size ongi venti prob inviati, in quando viene variato la dimensione del pyload ( e nel caso del throuput la idmensione del messaggio).

Possiamo stabilire noi come far ottenere i parametri usati a seguito del’instaurazione del messaggio TCP. Possono essere letti da file oppure passati tramite promt in in input.

Opzionale: possiamo variare il valore di server-delay, compreso di conclusioni sul perché variare questi campi ha un impatto.