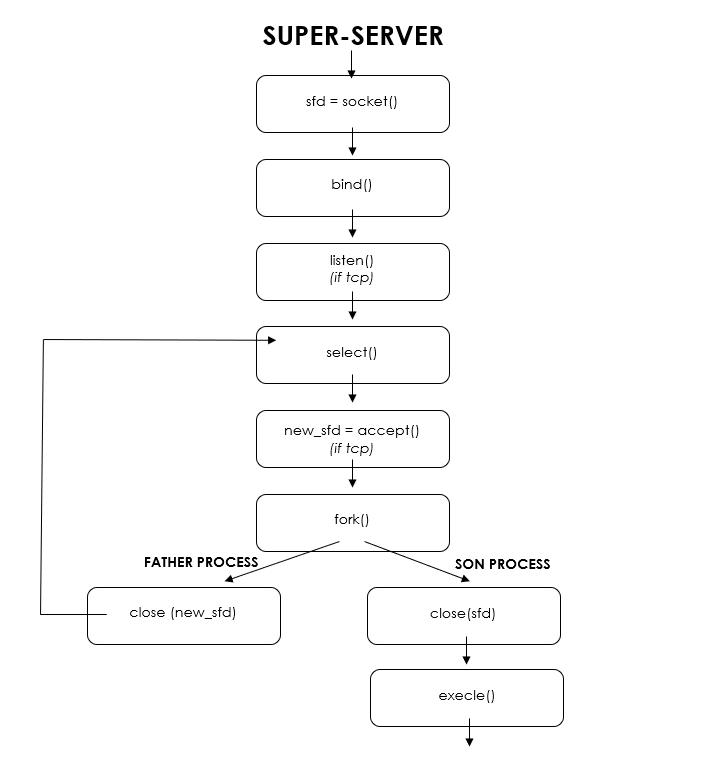
Report Assignment 2

Baiardi Martina, Lombardini Alessandro

TASK 1

Il superserver crea una socket per ogni servizio che mette a disposizione e la associa, con la chiamata della funzione *bind(),* con la porta pubblica associata al servizio. A questo punto, se il servizio è TCP, viene invocata la funzione *listen()* che mette in ascolto la socket creata di eventuali richieste di connessione.

Viene invocata la funzione *select()* che permette di monitorare più socket contemporaneamente in attesa che queste vengano contattate da client per delle richieste.

Una volta giunta una nuova richiesta, se la connessione è TCP viene invocata la *accept()* che apre una socket di connessione con il client richiedente.

Viene effettuata una *fork()*, cioè viene copiata l’immagine del processo del superserver in un nuovo processo, chiamato processo figlio, che viene poi opportunamente sostituito con l’eseguibile del server richiesto per il servizio, attraverso la funzione *execle()*. Il processo padre, nonché il superserver, torna in ascolto di nuove richieste sulla select.

TASK 2

**Questions:**

1. Which is behavior of udpServer in wait mode? Which one in nowait mode?

2. Which is behavior of tcpServer in wait mode? Which one in nowait mode?

For **Task2**, report **must** contain:

• A description of how you designed your Super-server

• Instructions of how you compiled your superserver.c source code

• Report the tests you carried out to verify the correctness of your source

code:

– Describe how you carried out the test;

– Describe the experienced behavior and answer questions at slide 16;

– Support your report with screenshot of the Terminal and considerations about the

experienced behavior.

Il superserver, per prima cosa, effettua una lettura del file di configurazione *inetd.conf* e salva le informazioni contenute al suo interno dentro la struttura dati preventivamente creata, la quale contiene per ogni servizio:

1. *Protocollo di trasporto*, il quale può essere TCP o UDP
2. *Tipo di servizio*, il quale può essere *wait* o *nowait*
3. *Porta del servizio*, che viene richiesta da parte dei client richiedenti
4. *Indirizzo del servizio*, il quale non è altro che la directory in cui si trova il processo server che deve essere eseguito per quel servizio
5. *Nome del servizio*, il quale rappresenta il nome dell’applicativo del server richiesto
6. *Socket File Descriptor*, il quale mantiene le informazioni sulla socket alla quale è stato associato il servizio descritto
7. *Process Identifier*, il quale mantiene il codice univoco dei processi server che sono stati creati. Questo parametro serve per gestire i servizi di tipo *wait*.

A questo punto il superserver procede alla creazione delle socket che vengono associate ai servizi:

* Viene invocata la funzione ***socket()***
  + Se la socket è richiesta di tipo TCP, la funzione conterrà i parametri: AF\_INET (Famiglia di indirizzi IPv4), SOCK\_STREAM (Socket con connessione di controllo), IPPROTO\_TCP (Protocollo TCP).
  + Se la socket è richiesta di tipo UDP, la funzione conterrà i parametri: AF\_INET (Famiglia di indirizzi IPv4), SOCK\_DGRAM (Socket senza connessione), IPPROTO\_UDP (Protocollo UDP)
* Viene inizializzata la struttura dati contenente le informazioni sul server, di tipo ***sockaddr\_in***.
* Viene chiamata la funzione ***bind()*** che associa alla nuova socket l’indirizzo del servizio descritto nella struttura dati.
* Se la socket generata è di tipo TCP, allora viene invocata anche la funzione ***listen()***, che permette di mettere una certa socket di connessione in ascolto di richieste, se invece è di tipo UDP non viene chiamato nulla.

Una volta istanziate le socket, dopo aver lanciato la funzione signal, che nel momento dell’arrivo del segnale SIGCHLD manda in esecuzione la funzione ***handle\_signal()*** che gestisce l’arrivo del segnale, il superserver entra in un loop infinito in cui gestisce le richieste dei client sulle sue porte.

All’interno del ciclo vengono inizializzati i tempi di timeout per la select e viene riempita la struttura ***fd\_set*** con tutti i socket file descriptor che si vuole vengano controllati per l’arrivo di richieste. Una volta chiamata la select, che si mette in attesa di richieste in arrivo sulle porte dei servizi, si attende di avere una richiesta da parte di un client. Quando arriva la richiesta per un servizio, per prima cosa si controlla se questo è di tipo TCP, perché in tal caso viene invocata la funzione ***accept()*** che apre una nuova socket con il client, sulla quale poi verranno trasmessi i dati.

A questo punto viene invocata la ***fork()*** che crea un processo figlio con la stessa immagine di quello corrente. A questo processo figlio verrano chiusi i file descriptor 0,1,2, che corrispondono rispettivamente a input, output e error e verranno sostituiti con il socket file descriptor della connessione tra client e server se tcp o della socket creata per il servizio concesso se udp. Una volta chiusi, viene sostituita l’immagine del procesos figlio con quella del processo server richiesto, che va in esecuzione con il client. Il processo padre ha il compito di rimuovere la socket del servizio aperto solo nel caso in cui questo è wait.

Infine, per gestire i segnali SIGCHLD inviati dai processi figli quando vengono terminati, è stata disposta la procedura *handle\_signal()* che controlla che il segnale arrivato sia effettivamente un SIGCHLD, e in tal caso controlla che se il servizio richiesto è di tipo wait allora riaggiunge la socket alla lista fd\_set per permette alla *select()* di controllarla per le richieste di connessione.

Le istruzioni utilizzate per compilare il superserver sono le seguenti:

all: superserver.exe udpClient.exe udpServer.exe tcpClient.exe tcpServer.exe

superserver.exe: superserver.o

gcc ${CFLAGS} -o superserver.exe superserver.o

superserver.o: superserver.c

gcc -c ${CFLAGS} superserver.c

udpClient.exe: udpClient.o

gcc ${CFLAGS} -o udpClient.exe udpClient.o

udpClient.o: udpClient.c

gcc -c ${CFLAGS} udpClient.c

udpServer.exe: udpServer.o

gcc ${CFLAGS} -o udpServer.exe udpServer.o

udpServer.o: udpServer.c

gcc -c ${CFLAGS} udpServer.c

tcpClient.exe: tcpClient.o

gcc ${CFLAGS} -o tcpClient.exe tcpClient.o

tcpClient.o: tcpClient.c

gcc -c ${CFLAGS} tcpClient.c

tcpServer.exe: tcpServer.o

gcc ${CFLAGS} -o tcpServer.exe tcpServer.o

tcpServer.o: tcpServer.c

gcc -c ${CFLAGS} tcpServer.c

.PHONY: clean

clean:

-rm -f \*.exe \*.o