Programació d'aplicacions de Xarxa

Autor: Aleix Segura Paz

Data: 16 d'abril de 2023

Pràctica 1

Xarxes (102015-2223)

Universitat de Lleida

Resum

L'arquitectura client-servidor és un dels principals models utilitzats en l'Internet i el podem trobar present en protocols i en una gran quantitat de serveis. És tracta d'una comunicació centralitzada, on els usuaris en qüestió realitzen peticions de serveis al servidor i aquest s'encarrega de satisfer-les.

El present document explicarà l'implementació d'aquest model donades una sèrie de requisits i configuracions que s'han establert en el guió de la pràctica.

Paraules clau: arquitectura, client, servidor, Internet, comunicació centralitzada.

Abstract

The client-server architecture is one of the main models that run on multiple services and protocols of the Internet. It's a centralized communication, on which the users make requests to the server and it satisfies their petitions.

The present document will explain the implementation of this model given certain requirements and configurations that have been established in the work script.

Keywords: architecture, client, server, Internet, centralized communication.

Índex

Índex de Figures

1 Introducció

La tasca d'implementar un client escrit en el lleguatge de programació **ANSI C** i un servidor escrit en el lleguatge de programació **Python** no ha estat pas senzilla. Durant el camí han anat sorgint diversos tipus de complicacions a nivell de codi que han necessitat d'una bona estona de *debugging*. A més a més, un cop semblava el problema solucionat i s'avançava al següent pas, resultava que el disseny emprat per a implementar una funció o un grup de funcions no era el millor. Fet que significava tornar un o dos passos enrere i pensar una solució més òptima.

De qualsevol forma, realitzar la pràctica ha estat un repte i els objectius d'apendre a programar aplicacions de xarxa, entendre el model client-servidor i apendre a programar i dissenyar un protocol de comunicacions s'han, en la mesura del possible, assolit.

A continuació es comencen a explicar alguns dels pilars clau en la implementació de la pràctica.

2 Manteniment de comunicació: estratègia

El primer pas per a poder mantenir comunicació client-servidor és establir-la. Aquest procés comença a la fase de registre del client al servidor. Si un client determinat abans d'esgotar el seus intents de procés de registre ha enviat una PDU (**P**rotocol **D**ata **U**nit) correcta al servidor llavors haurà aconseguit registrar-se i per tant, establir la connexió. Ara es qüestió de mantenir-la.

L'estratègia emprada per a mantenir la comunicació ha estat la següent: per par del client, es crea un fil anomenat *alives_thread* que s'encarregarà d'executar la funció *send_alives*. Aquesta funció es basa en l'execució d'un bucle infinit on es van enviant paquets UDP (**U**ser **D**atagram **P**rotocol) de tipus *ALIVE_INF* al servidor i el client rep la resposta. Aquest procés és el que s'executa continuament si tot funciona correctament, és a dir, el client envia *ALIVE_INF* correct i el servidor respon amb un *ALIVE_ACK* correcte. D'altra banda, el client pot rebre altres tipus de paquet del servidor o també pot no rebre'n cap. A continuació és detalla com es gestionen:

- És rep *ALIVE_ACK* però les dades no són vàlides: no es fa cas del paquet rebut.
- És rep *ALIVE_NACK*: s'incrementa la variable *no_recept_confirm* que fa referencia a que no s'ha rebut resposta per part del servidor.
- És rep *ALIVE_REJ* i l'estat del client és *ALIVE*: s'interpreta com a frau d'indentitat, es canvia l'estat del client a *DISCONNECTED* i s'inicia un nou procés de registre.
- És rep *ALIVE_REJ* i l'estat del client no és *ALIVE*: s'interpreta com a que s'està forçant una resposta del client per tal d'inserir informació, es canvia l'estat del client a *DISCONNECTED* i s'inicia un nou procés de registre.

A més a més, si no s'ha rebut resposta a 3 *ALIVES* consecutius s'interpreta com a que no hi han bones comunicacions amb el servidor, es canvia l'estat del client a *DISCONNECTED* i s'inicia un nou procés de registre.

Per part del servidor, la funció *make_channels* és executada en el fil *server_thread* crida dos funcions, una d'elles: *udp* que s'encarrega, tal com el seu nom indica, de gestionar la part UDP del model client-servidor. La part UDP és més complexa en el servidor que en el client. La funció *udp* crea dos fils d'execució: *concurrency*, que executa *alivesloop*, un bucle infinit per comprobar per als clients en estat *ALIVE* si n'hi ha algun que hagi deixat d'enviar tres *ALIVES* consecuitius i l'altre fil de la funció *udp* és *checkfirst* que executa *checkfirst*, un bucle infinit per comprobar si s'ha rebut o no el primer *ALIVE_INF* en 4 segons ($J * R$).

A més a més de crear els dos fils que estan monitoritzant continuament, la funció *udp* crida *handle_udp* la funció que gestiona els paquets que rep del client i que envia els corresponents de tornada. En aquest cas, des de el punt de vista del servidor, els paquets rebuts es tracten de la següent manera en la funció *alives* cridada per *handle_udp*:

- És rep *ALIVE_INF* correcte: s'envia el paquet *ALIVE_ACK* de tornada.
- És rep *ALIVE_INF* i l'equip no està autoritzat: s'envia el paquet *ALIVE_REJ* amb missatge "*Client not authorized.*".
- És rep *ALIVE_INF* i el client no està registrat: s'envia el paquet *ALIVE_REJ* amb missatge "*Client not registered.*".
- És rep *ALIVE_INF* i la IP no és vàlida: s'envia el paquet *ALIVE_NACK* amb missatge "*Wrong ip address.*".
- És rep *ALIVE_INF* i el número aleatori no coincideix: s'envia el paquet *ALIVE_NACK* amb missatge "*Wrong random number.*".

Obviament per a mantenir una comunicació periòdica és necessari que tant client com servidor executin un bucle infinit per tal d'estar constantment enviant i rebent paquets.

3 Consideracions de l'implementació

A continuació es detallen alguns aspectes sobre la implementació del client i el servidor.

3.1 Client

Nom de l'arxiu del client: *client.c*.
Nom del binari compilat: *client*.

En la implementació del client hi han els següents aspectes a comentar:

- Si es passen per paràmetre arxius de configuració de *software (-c)* o de configuració de l'equip *(-f)* i algun d'ells no és vàlid el client finalitza.
- Tant en la fase de registre com en la fase de manteniment de la comunicació, el valor de *timeout* que s'utilitza en la funció *select* correspon a l'interval de temps d'enviament del paquet precedent al que es rep. És a dir, si el valor utilitzat en la funció *sleep* per a esperar per a l'enviament d'un paquet és 2 llavors el *timeout* es 2.
- En les funcions *get_server_package*, tant en la seva versió per a comunicació UDP com TCP, si la funció *select* arriba al valor de *timeout* sense tenir informació en el *socket* es retorna un paquet de servidor amb tots els camps a valor "0" i el tipus de paquet és *ERROR* (0x0F).

Això es fa per a poder tractar les (no) respostes del servidor com per exemple en el cas de pèrdua d'*alives*.

- Com s'ha comentat anteriorment, en la fase de manteniment de la comunicació si es rep un paquet del tipus *ALIVE_REJ* i l'estat del client es diferent a *ALIVE* es considera que s'està intentant forçar el client per a inserir informació.

3.2 Servidor

Nom de l'arxiu del servidor: *server.py*.

Per la banda del servidor cal considerar:

- Tots els fils es creen amb l'opció *daemon* a valor *True* per tal de que finalitzin automàticament quan finalitza el programa principal.
- Per al control de la rebuda o no rebuda del primer *ALIVE_INF* abans de 4 segons s'ha creat una funció (*verify_first_alive*) dins la classe *Client* que es executada per a cada client registrat en la funció *checkfirst* que executa un fil continuament.
- Per a controlar si en algun moment s'han perdut 3 *alives* consecutius s'ha creat una funció (*verify_alive_inf*) dins la classe *Client* que es executada per a cada client en estat *ALIVE* en la funció *alivesloop* que executa un fil continuament.
- Es declara una variable anomenada *DELAY_TIME* establerta a 0.5 (segons) que representa el temps simbòlic de sincronització per a les funcions *verify_first_alive* i *verify_alive_inf*. Aquesta variable agrega uns instants insignificants de temps als valors 6 segons (que representa el temps que hauria passat si no s'han rebut 3 *alives* consecutius) i 4 (que representa el temps màxim per a rebre el primer *ALIVE_INF*). Aquesta variable ha resultat necessària ja que s'han executat les probes en una màquina virtual i, en ocasions, sobretot quan s'executa més d'un client a la vegada la terminal no té el rendiment apropiat que tindria si s'hagués instal·lat el sistema operatiu en el *hardware*. Per tant, quan s'està executant des d'un equip amb el sistema operatiu instal·lat en el seu propi *hardware* aquesta variable es pot reduir molt més inclús eliminar.
- En la funció *listen* que crida el *socket* TCP s'estableix el valor de connexions màximes a 3. Aquest valor es pot incrementar o decrementar en funció de quantes connexions màximes volem que accepti el servidor.

3.3 Makefile

En quant a l'arxiu *Makefile* que s'utilitza per a la compilació del client s'ha afegit l'opció *-pthread* a les ja predeterminades degut a que s'utilitza la llibreria *pthread.h* per a la creació i gestió de fils d'execució.

4 Conclusions

Com s'ha comentat de forma breu en l'apartat 'Introducció', l'implementació del client i el servidor ha estat un repte. La pràctica ha estat realitzada lentament amb l'objectiu d'avançar, encara que fós poc, dia a dia. Ja que, des de el meu punt de vista i amb el nivell de programació de segon de grau, realitzar-la amb presses hauria estat un gran error. És una gran quantitat d'informació nova que és té que analitzar i entendre bé per tal de saber el que és demana i com s'ha de realitzar.

Un cop realitzada la pràctica es veu de forma clara que el fet de trobar-se amb aquests problemes i, sobretot, aconseguir solucionar-los, ha desembocat en que hagi resultat ser un gran aprenentatge.

Tot i així, és clar que la implementació no es perfecta i segurament és puguin realitzar diferents canvis que ajudarien a millorar alguns defectes o imperfeccions que pugui tenir el client o el servidor. Crec però, que tampoc era l'objectiu principal aconseguir el millor dels codis i per als objectius que es volien assolir en la pràctica definitivament s'han assolit.

A continuació podeu trobar una sèrie de diagrames sobre el disseny del client, el servidor i els estats pels que pot passar un client durant la comunicació UDP.

Diagrames de blocs

Diagrama de blocs client

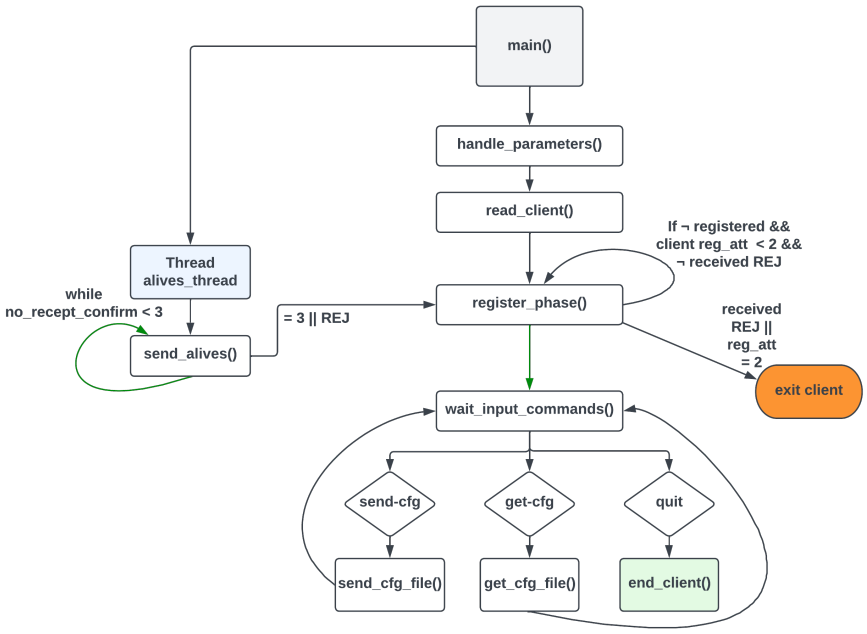

Figura 1: Diagrama de blocs de la implementació del client. Font: Elaboració pròpia vía *Lucidchart*.

Diagrama de blocs servidor

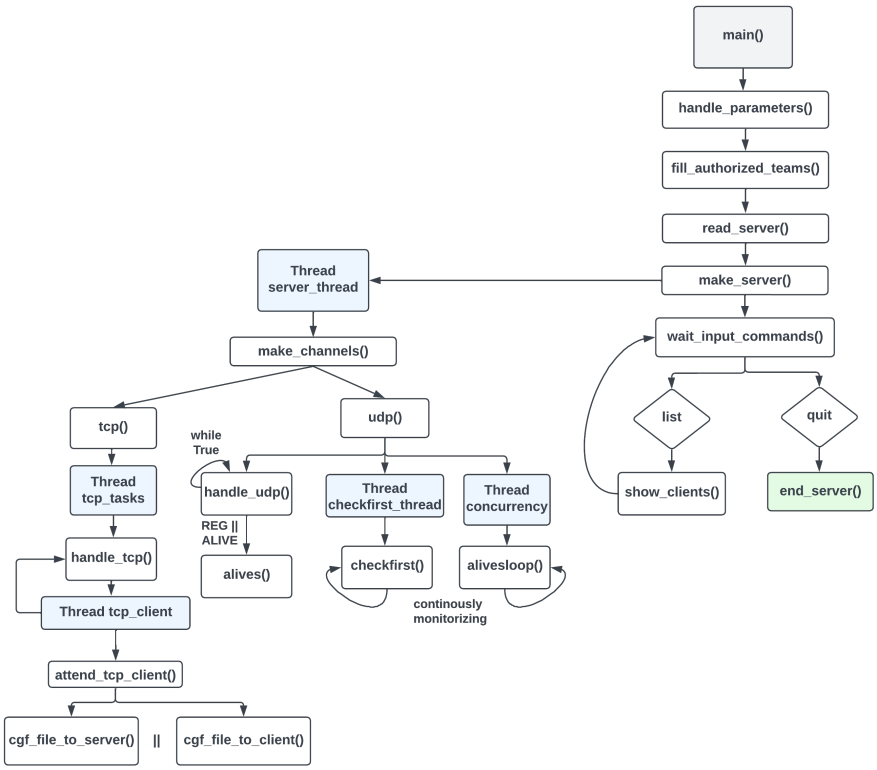

Figura 2: Diagrama de blocs de la implementació del servidor. Font: Elaboració pròpia vía *Lucidchart*.

Diagrama d'estats: Protocol UDP

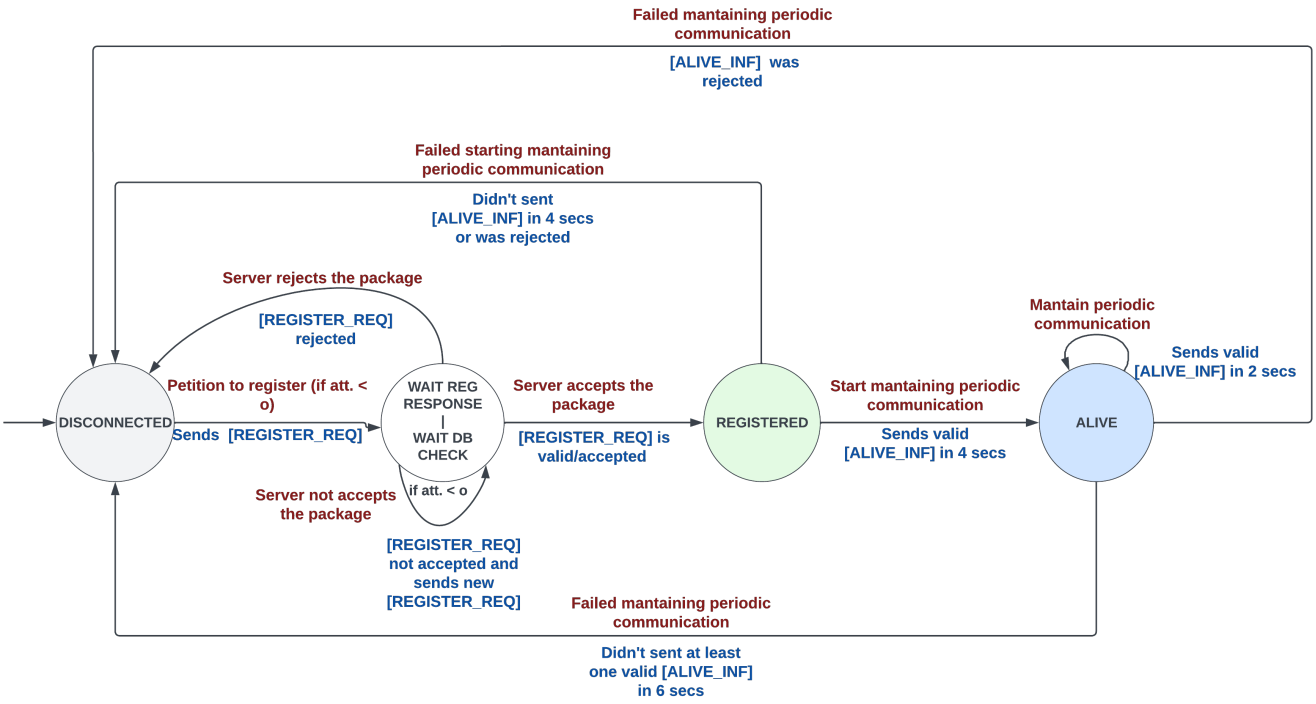

Figura 3: Diagrama d'estats del protocol UDP. Font: Elaboració pròpia vía *Lucidchart*.