Practica 1

Xarxes

Dídac Cayuela Dolcet
2n GEI
16/04/2023

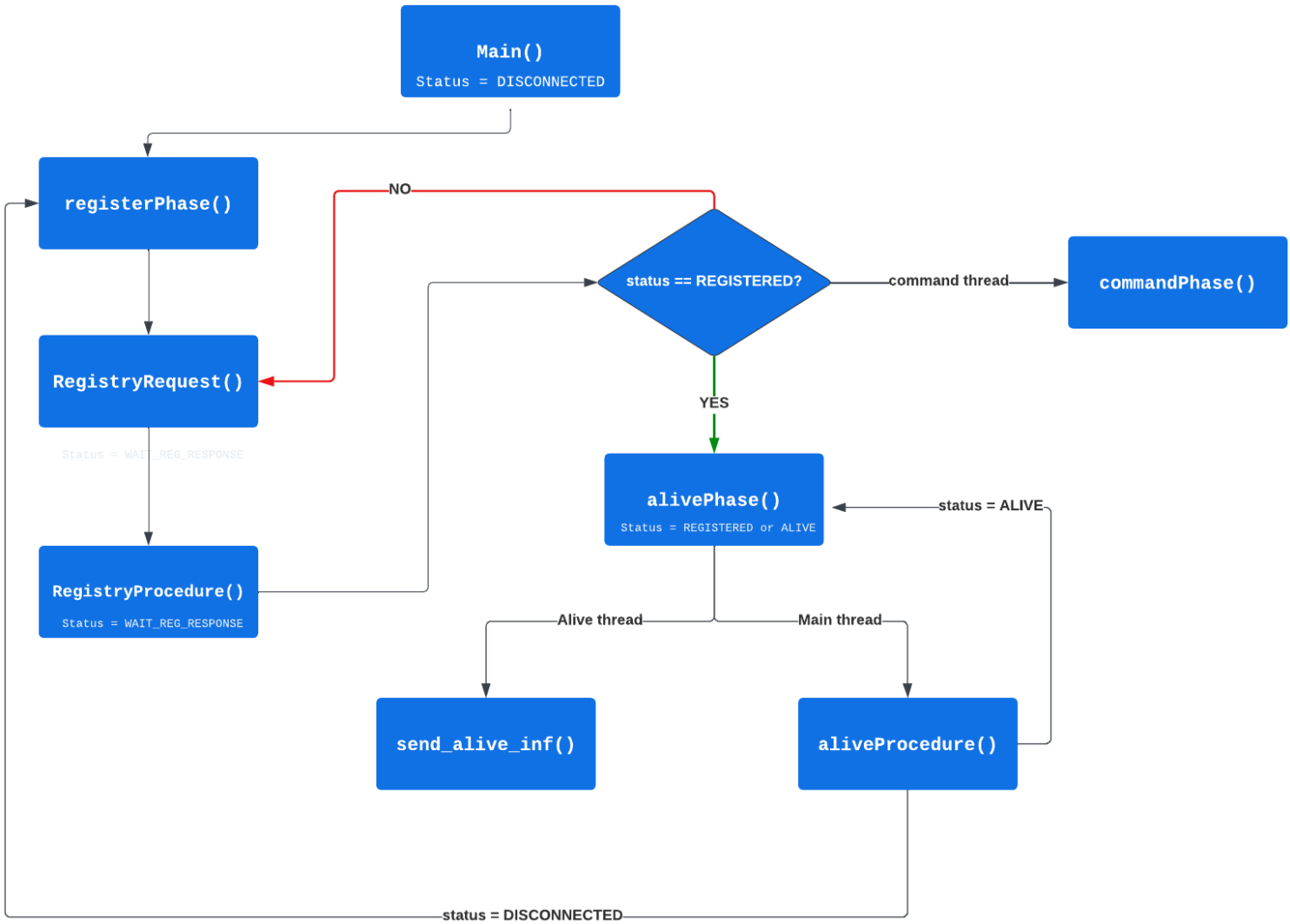
Índex

Introducció

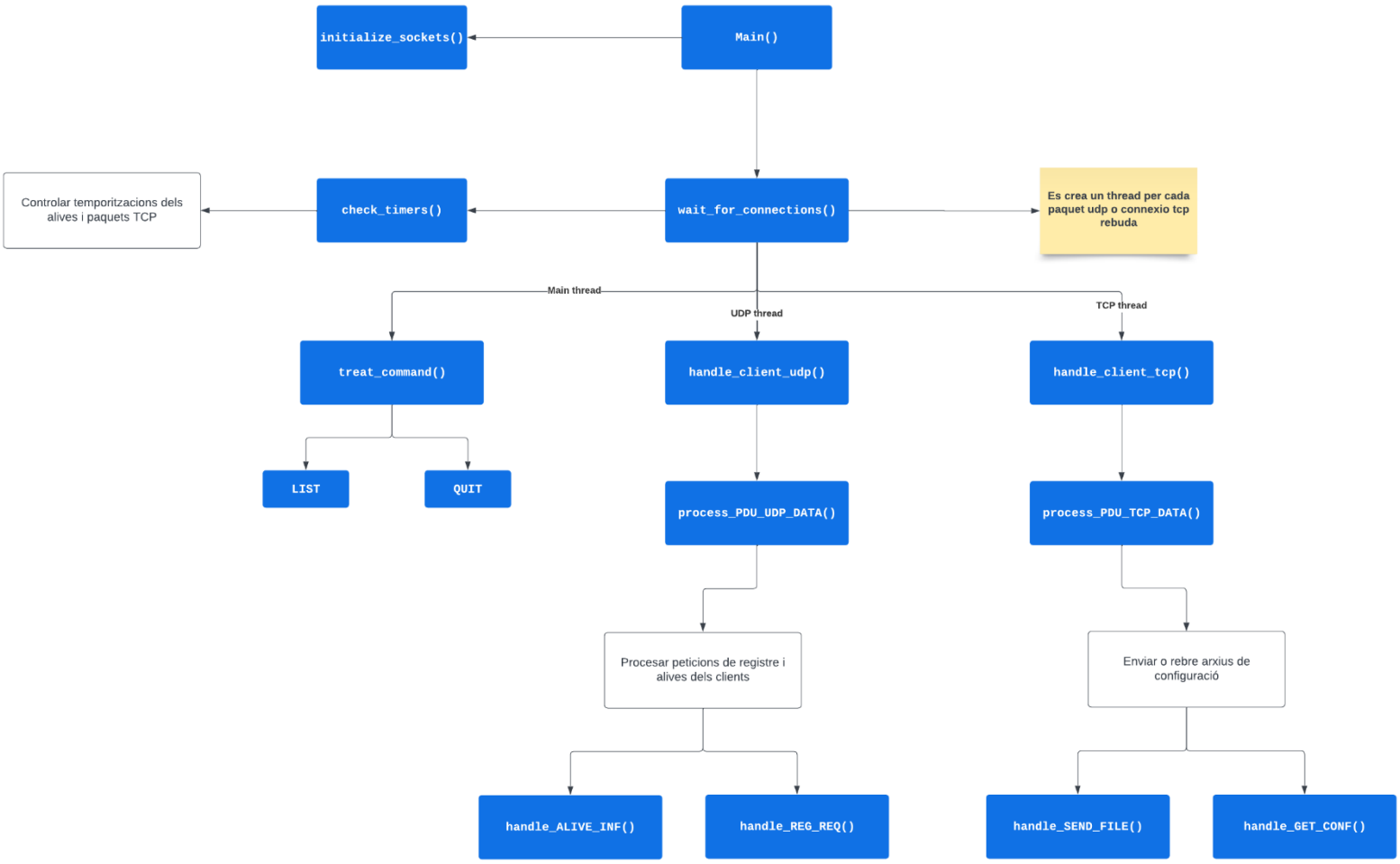
En aquest informe s'explicarà part del funcionament i desenvolupament d'un model de comunicació client-servidor. S'explicarà l'estratègia emprada per al manteniment de la comunicació entre aquests dos sistemes i es mostraran uns diagrames de blocs per a una millor comprensió de l'estructura. Tot seguit s'afegiran un parell de conclusions sobre aspectes rellevants de la implementació i finalment es reflexionarà sobre la feina feta en les conclusions.

L'estructura del client i del servidor

- Diagrama de blocs del client



- Diagrama de blocs del servidor



Estratègia emprada per al manteniment de la comunicació

- Estratègia emprada en el client

Per a poder mantenir la comunicació entre el client i el servidor amb els alives, primer el client ha d'haver estat registrat de manera correcta.

Un cop el client es troba en el estat *[REGISTERED]*, cridarem a la funció **alivePhase()** que s'ocuparà d'emmagatzemar una copia de la PDU rebuda durant la fase de registre, per així més endavant poder comprovar que es reben els paquets alive amb dades correctes.

A continuació es crea un fil que crida a la funció **send_ALIVE_INF()** que s'ocupa d'enviar un paquet *[ALIVE_INF]* cada *r* segons, i de portar el compte dels paquets enviats al servidor per així passar al estat *DISCONNECTED* en cas que no es rebi confirmació de *s* paquets consecutius, com s'especifica a l'enunciat.

Mentre el fil s'ocupa d'enviar els paquets *[ALIVE_INF]* corresponents, al fil principal s'executa al funció **aliveProcedure()** en un bucle *while* tan sols si l'estatus del client és *REGISTERED* o *SEND_ALIVE*.

La funció **aliveProcedure(),** que es troba al fil principal, té un *Select* que es queda en espera de rebre paquets UDP com a màxim *r* segons, i s'encarrega de tractar els paquets de tipus *[ALIVE_ACK], [ALIVE_REJ] i [ALIVE_NACK].*

En el cas de que el client passi al estat *DISCONNECTED* es tornarà a la fase de registre.

- Estratègia emprada en el servidor

Degut que el servidor ha de ser capaç de atendre múltiples clients a l'hora, l'estratègia emprada per mantenir la comunicació canvia força.

El servidor, en el fil principal, es queda constantment a la espera de rebre alguna cosa per el ***stdin,*** el ***socket udp,*** o bé el ***socket tcp***. Per explicar el manteniment de la comunicació tan sols ens centrarem en el socket udp, que es l'encarregat d'enviar i rebre alives.

El servidor crearà un fil per cada paquet udp rebut, i un cop aquest paquet hagi estat tractat el fil finalitzarà. En el cas de rebre paquets *[ALIVE_INF],* aquest es comprovarà que estigui conformat de manera correcta i es respondrà amb *[ALIVE_ACK], [ALIVE_REJ] o [ALIVE_NACK]* segons correspongui.

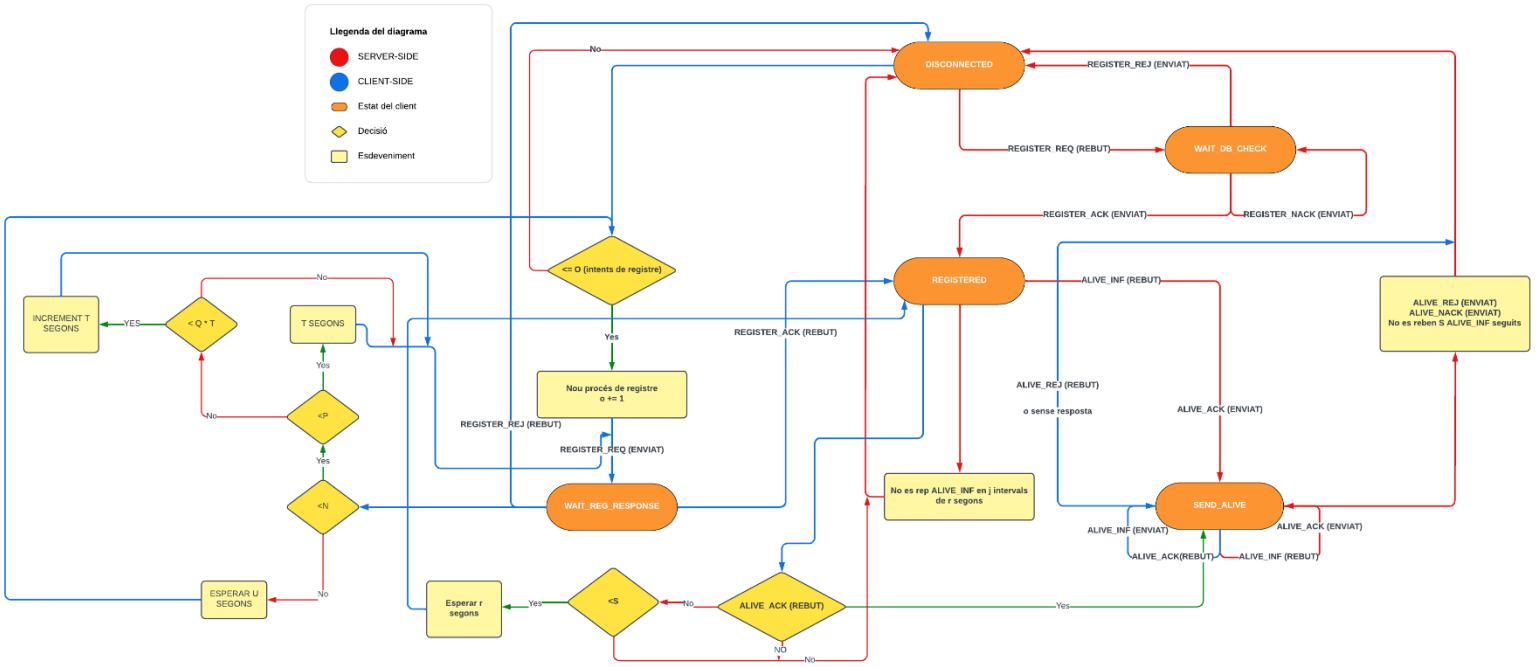
Per tal de portar un control sobre la temporització del primer alive i la pèrdua d'alives, s'ha creat una funció anomenada **check_timers()**. Aquesta es cridada des del fil principal dins del mateix bucle, que s'encarrega d'esperar les connexions entrants.

Cada client dins del servidor es troba representat com un objecte de la classe *client.* D'aquesta manera podem emmagatzemar dades rellevants d'aquest, com pot ser el estat en el que es troba o la marca de temps en la que ha enviat l'últim paquet udp que el servidor ha rebut.

Tenint aquest objecte *client* a la nostra disposició fem un càlcul de la diferència de temps entre l'últim enviament i la comprovació actual, així com dels intervals d'enviament per cada un d'ells. Si es dona el cas que no es compleix alguna de les condicions especificades en l'enunciat es prendran les mesures corresponents per canviar l'estat del client a *DISCONNECTED.*

Finalment si algun client es troba en l'estat *DISCONNECTED*, es reinicialitzaran les dades emmagatzemades d'aquest.

Diagrama d'estats UDP



Consideracions

- Timers per a la fase d'alives del client

Tot i que finalment em vaig decantar per l'estratègia comentada anteriorment per a la fase alive en el client, aquest no ha estat el primer enfocament que li vaig donar al problema en un principi.

Vaig començar intentant fer que el fil tan sols s'encarregués d'enviar alives cada r segons, però no de fer la comprovació del nombre de paquets enviats sense resposta que tinc actualment.

A l'hora d'intentar implementar la detecció de que un paquet *[ALIVE_INF]* no rebés resposta, vaig intentar fer anar un seguit de timers en conjunt amb el Select que s'ocupa de tractar els paquets rebuts per així aconseguir el comportament desitjat.

En implementar-ho d'aquesta manera em va quedar un codi molt difícil de llegir i mantenir amb el que no estava satisfet, a més a més hi havia problemes amb la sincronització de temps client-servidor i em saltava el *timeout* en moments que no esperava.

Fins i tot vaig tenir problemes amb l'enviament de *[ALIVE_INF]* ja que els anava enviant segons el que rebia i no cada *r* segons de manera independent als paquets rebuts.

Finalment vaig fer una passa enrere i vaig revaluar les meves opcions, i va estar aleshores quan em vaig adonar que la solució era més senzilla del que m'estava plantejant.

- Funcions per convertir buffer a PDU

Un altre dels obstacles que hem vaig trobar a l'hora de desenvolupar el client va estar a l'hora d'emmagatzemar les dades de la PDU en format de struct i de buffer. Si desitjava enviar havia de col·locar totes les variables de la meva struct en un array de caràcters, i si volia tractar les dades rebudes, havia de desmuntar aquest mateix array per poder emmagatzemar-les en el struct.

En vista d'aquest problema em vaig fer quatre funcions, dos per convertir el struct a un array, tant per TCP com UDP i dos mes per convertir aquest array un altre cop als structs que emmagatzemen els PDU de TCP i UDP.

En un cert punt del desenvolupament del client em vaig adonar de que en comptes de fer anar aquestes funcions es podia passar el struct que es desitja enviar directament, i que el mateix es podia fer per a la recepció de dades.

Tot i que estava molt satisfet amb el disseny de les funcions esmentades, vaig arribar a la conclusió de que el millor era deixar-les de banda per així fer-me la vida més senzilla i a l'hora, deixar el codi més net i compacte.

De totes maneres he deixat aquestes funcions comentades al codi per a que es pugui veure quina era l'idea inicial que havia tingut

Conclusions

Aquesta practica ha estat tot un repte que ha posat a prova tot el que sabia i el que hem vist fins al moment al llarg de la carrera. És una activitat molt completa que m'ha fet posar en practica des de el procés de crear i gestionar fils fins a fer anar tot tipus d'estructures de dades per poder tractar amb els paquets de la manera més neta i eficient possible.

Tot i que ha estat un desenvolupament molt tediós degut a tots els problemes que hem comentat anteriorment, he après molt sobre el funcionament de la dinàmica client-servidor i com es fan anar els sockets.

Considero que ha estat una bona manera de conèixer a fons una capa d'abstracció més d'aquest camp tant ampli, i que moltes vegades donem per fet.