PROGRAMACIÓ D'APLICACIONS DE XARXA

Pràctica 1

Enric Tobeña Casanovas

DNI: 48059124A

Curs: 2021-2022

Data d'entrega: 17/04/2022

Índex de continguts

Índex de figures

1. Introducció

L'objectiu d'aquesta pràctica era aconseguir una comunicació periòdica mitjançant l'enviament continu de paquets a través de "sockets" que operen amb el protocol UDP.

El codi font del client està implementat amb ANSI C, mentre que pel del servidor s'ha utilitzat Python 3.9, amb les diferències sintàctiques existents entre els dos llenguatges a l'hora de gestionar el format de les dades.

El resultat final és el d'un o més clients que es comuniquen amb un mateix servidor enviant-li paquets de dades i un servidor que gestiona els les dades de cadascun dels clients, rebudes a través d'aquests paquets.

Aquest document està estructurat tenint en compte les característiques essencials perquè el lector pugui comprendre de la manera més fàcil possible aspectes de la pràctica com l'estructura, la estratègia emprada per mantenir la comunicació client-servidor i el protocol implementat sobre UDP. Per definir aquests conceptes, s'han utilitzat estratègies com els diagrames de blocs i els diagrames d'estats, ambdues acompanyades de textos que donen més detalls de cada apartat.

2. Client

2.1. Diagrama d'estats

En la figura 1 hi ha representat el diagrama d'estats del client, amb les seves corresponents variacions segons en quina fase d'intercanvi de paquets es trobi. L'estat final és quan s'arriba al SEND_ALIVE i hi va havent intercanvis de paquets del tipus ALIVE correctes entre client i servidor.

Tots els estats previs a l'inici de la comunicació periòdica indiquen que el client està esperant a que el servidor li accepti o rebutgi les dades que aquest li ha enviat prèviament i procedir segons la resposta rebuda.

Qualsevol paquet rebut en un estat que no li correspon suposarà la nul·litat del procés de registre/comunicació vigent i es reiniciarà el procés de registre i l'estat del client tornarà a NOT_REGISTERED.

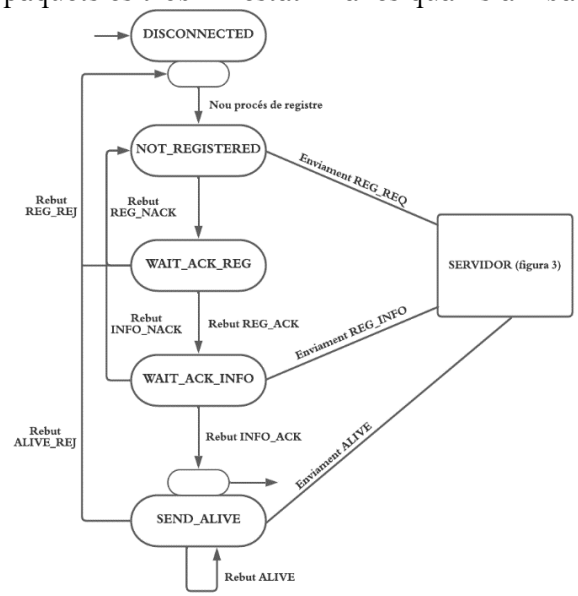

Figura 1. Diagrama d'estats del client

2.2 Diagrama de blocs

A la figura 2, utilitzant un diagrama de blocs, s'hi reflexa l'estructura del client, amb tots els seus mètodes principals i les crides que hi ha entre ells. Aquest entramat és el que garanteix que el client sempre estigui en un estat "correcte". És a dir, que compleixi el diagrama d'estats de la figura 1.

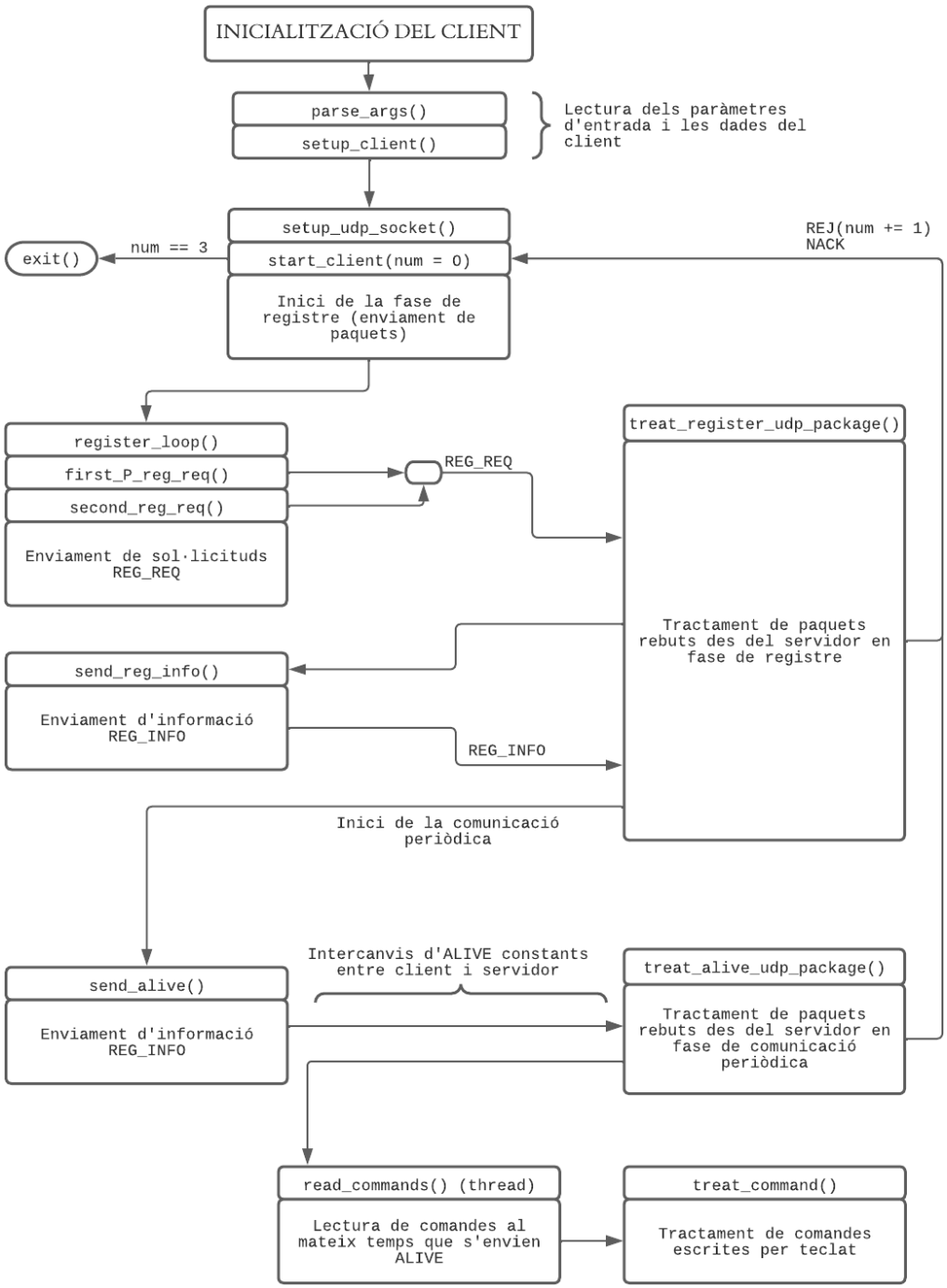

Figura 2. Diagrama de blocs del client

3. Servidor

3.1. Diagrama d'estats

El servidor com a tal no es troba en un estat concret, sinó que el que fa és definir i gestionar l'estat que tenen cadascun dels dispositius autoritzats dins del mateix servidor, tenint en compte si es troben en el procés de registre o en la comunicació periòdica. En la figura 3 es representa el funcionament d'aquest procés.

D'igual manera que en el cas del client, la recepció d'un paquet d'un tipus determinat en un estat que no li pertoca suposarà que el servidor marqui el client transmissor d'aquesta trama errònia com a DISCONNECTED.

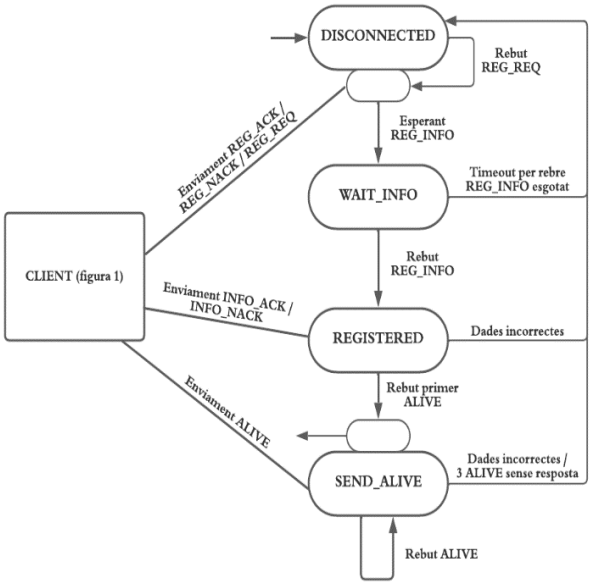

Figura 3. Diagrama d'estats del servidor

3.2. Diagrama de blocs

A la figura 4 (situada a la pàgina següent) s'hi representa el diagrama de blocs del codi del servidor, amb tots els mètodes i les respectives crides que hi ha entre ells per tal de que el es puguin processar els registres i mantenir les comunicacions periòdiques de tots els clients autoritzats de manera simultània.

Pel que fa a les comprovacions per veure si els paquets són correctes, he utilitzat "getters" i "setters" que iteren sobre una llista en la qual s'hi guarden instàncies de la classe "client", que conté com a atributs l' identificador de dispositiu, identificador de comunicació, adreça IP, elements i temporitzadors. Aquesta llista conté tantes instàncies com dispositius autoritzats a connectar-se al servidor.

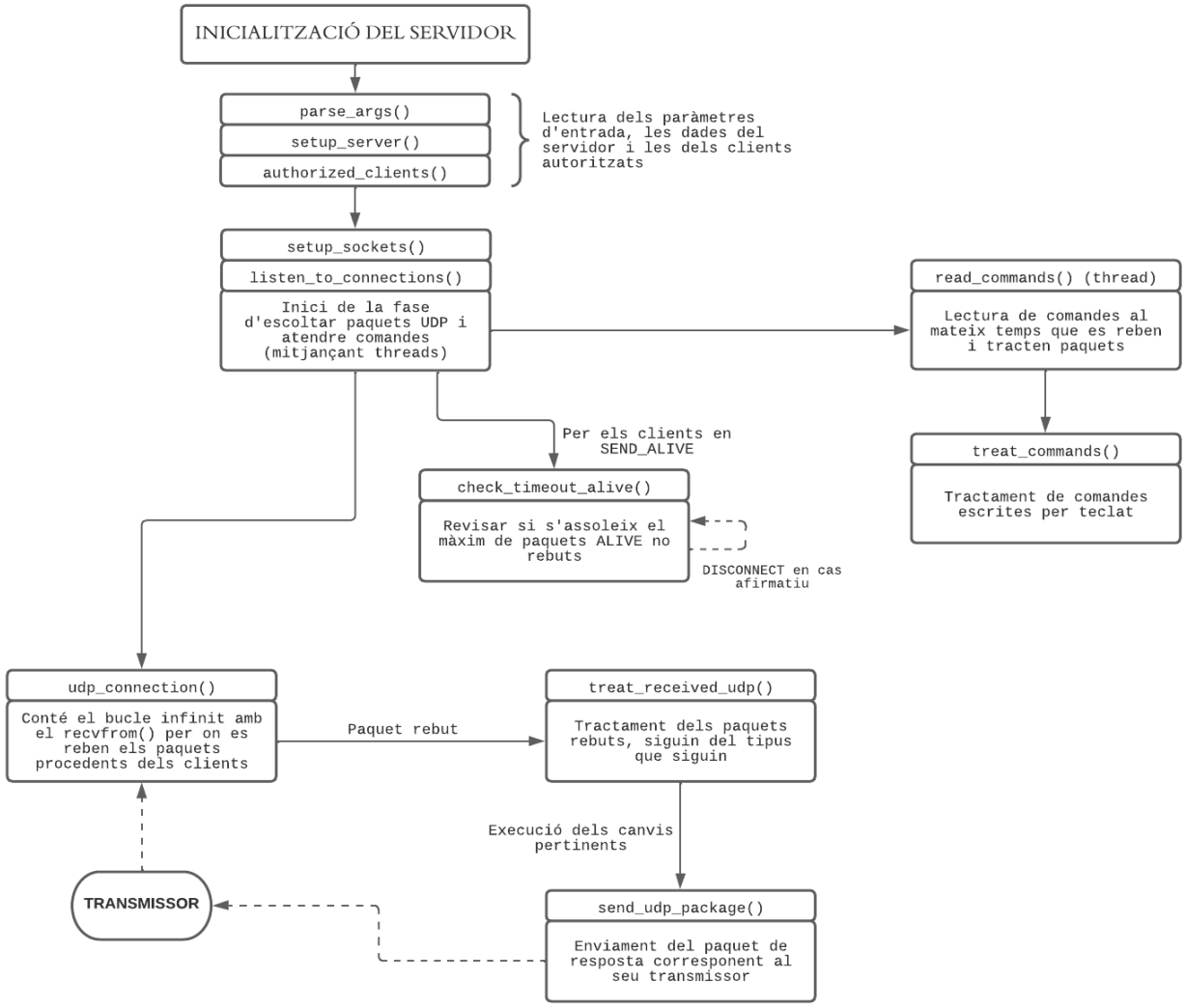

Figura 4. Diagrama de blocs del servidor

4. Estratègia de comunicació

4.1. Comunicació periòdica en el client

Per establir correctament la comunicació he utilitzat la funció *send_alive()*, que està estructurada dins d'un bucle infinit i s'executa mitjançant un "thread" inicialitzat just després de rebre la confirmació de la informació INFO_ACK per part del servidor. Aquesta funció es dedica a enviar els paquets ALIVE al servidor i rebre la resposta per tractar-la, havent fixat un timeout pel socket UDP (funció *setsockopt()*) utilitzant la struct *timeval* immediatament abans d'executar la recepció de l'ALIVE que envia el servidor.

Un cop rebut l'ALIVE, es verifica que sigui correcte i, en cas afirmatiu, s'espera el temps preestablert abans d'enviar el següent paquet amb un *sleep()* i es repeteix el procés en bucle fins que hi hagi una interrupció en la comunicació.

4.2. Comunicació periòdica en el servidor

En el cas del servidor, un cop un client hagi completat la seva fase de registre i, per tant, el servidor el tingui considerat com a SEND_ALIVE, llavors a cada ALIVE que rebi el servidor per part d'aquest client es guardarà el temps en que l'ha rebut.

Per altra banda, aquesta variable servirà per calcular si s'ha arribat al timeout segons el qual s'ha de comptar com ALIVE no rebut per part del servidor. Amb un "thread" que va iterant sobre la llista on hi ha totes les dades dels clients autoritzats i, per cada client en estat SEND_ALIVE, fa la resta entre el temps actual i el temps de l'últim paquet rebut.

En cas de que la diferència sigui superior al temps màxim estipulat, se sumarà 1 a la variable que compta els ALIVE consecutius no rebuts. Si algun client arriba a tenir el valor d'aquesta variable a 3, el servidor el posarà en estat DISCONNECTED.

5. Consideracions

- Tant en el client com en el servidor, és indiferent l'ordre dels paràmetres d'entrada, tot i que sí que s'ha de respectar que després d'un "-c" o "-u" hi ha d'anar el nom de l'arxiu.

- En el client, la part de TCP està implementada, però no és funcional (únicament es pot utilitzar la comanda "set" en TCP). En el cas del servidor no s'ha realitzat la implementació a causa de la repercussió negativa que aquesta tenia sobre el protocol UDP i la connexió periòdica.

-S'ha pressuposat que els fitxers de lectura de dades sempre tindran la mateixa estructura i el mateix ordre.