Pràctica 1 de Xarxes:
Disseny d'aplicacions de xarxa

Marc Montes Valls

23 d'Abril de 2019

Índex

Índex de figures

Capítol 1: Introducció

En aquesta pràctica es presenta un escenari per posar a prova els nostres coneixements sobre com comunicar aplicacions a través de sockets, requerint una implementació formal dels components d'aquest escenari i explicant en el present document el procés de desenvolupament de la pràctica.

En aquest cas es presenta una situació en que un servidor emmagatzema la informació de la configuració de xarxa de diversos clients, els quals hauran de registrar-se prèviament en el servidor, un cop registrats podran enviar la seva configuració de xarxa al servidor, o bé obtindre-la d'aquest.

Per les diferents fases de la pràctica hem utilitzat els tipus de comunicació vists a classe, en la fase de registre utilitzem el protocol UDP, emprat també per al manteniment de la comunicació. En la fase d'enviament de la configuració s'emprarà, en canvi, el protocol TCP.

La implementació de la pràctica presentada no contempla la fase de enviament o recepció de la configuració, per tant, un cop finalitzada la fase de registre només accepta comandes i manté la comunicació amb el servidor.

El document s'estructura seguint les següents parts: presentar els diferents paràmetres modificables tant del client com del servidor, les comandes i paràmetres admesos per ambdues implementacions, una explicació del funcionament tant del client com del servidor, ajudant-se de diagrames i esquemes durant aquesta, i les dificultats i solucions aportades a aquestes, seguit d'una conclusió personal sobre la pràctica.

Capítol 2: Parametrització i comandes

2.1 Paràmetres estàtics

Alguns dels paràmetres de la pràctica han estat inserits directament en el codi, donat que no es habitual canviar-los, aquests es poden consultar en la part superior del document on es troba el codi, tant del client com del servidor. Alguns d'aquests exemples poden ser la IP, el número màxim que poden conformar una comanda, etc. Altres paràmetres, com el port pel qual enviara el client, no es poden modificar, donat que el client utilitzarà el primer port disponible lliurat pel pròpi sistema.

Altres paràmetres es poden editar per mitjà d'un arxiu, els quals es troben en els fitxers de configuració que llegiran tant el client com el servidor, per defecte aquests arxius son:

- client.cfg: Conté el nom i l'adreça MAC del client, així com els ports TCP i UDP del servidor.
- server.cfg: Conté el nom, l'adreça MAC i els ports TCP i UDP del servidor.
- equips.dat: En aquest fitxer es troben el nom i la mac dels equips autoritzats en el servidor.

Aquests arxius es poden canviar mitjançant els paràmetres que veurem en el següent apartat, en el qual veurem els paràmetres acceptats per comanda.

2.2 Paràmetres especificats en l'execució

Tant el client com el servidor accepten els següents paràmetres d'entrada:

- -c {fitxer}: canvia el arxiu de configuració del qual es llegirà la configuració del servidor o bé del client, no es comprova si el fitxer existeix o no, per tant després del paràmetre -c"ha d'anar el nom del fitxer correctament escrit, si no s'especifica ens mostrara per pantalla que falta el paràmetre del nom de l'arxiu i finalitzara el programa.
- -d: executa el client o bé el servidor en mode debug, el qual mostrarà la informació de cada paquet enviat i rebut, a més dels canvis d'estat que es mostraríen en cas contrari.

A més a més el servidor també accepta el paràmetre següent:

- -u {fitxer}: funciona igual que el paràmetre -c", però, en aquest cas es modifica el arxiu on es troben els equips autoritzats pel servidor. El control d'errors es tracta igual que en el paràmetre -c.

Per tant una possible execució dels dos programes utilitzant tots els paràmetres podria ser la següent:

```
$./client -d -c client1.cfg

$python server.py -d -u equips2.dat -c server1.cfg
```

2.3 Comandes acceptades

Tant el servidor com el client accepten la comanda quit, la qual finalitza el programa, tanca tots els fils i els sockets actius, d'aquesta forma no queden ports ocupats ni fils ocupant capacitat de processament degut a una mala sortida del programa, tancar els programes de qualsevol altra manera pot deixar els fils executant-se, o bé els ports ocupats pels sockets sense tancar.

D'altra banda, el servidor també implementa la comanda list, la qual llista la informació pertinent dels diferents equips autoritzats, en el següent ordre:

- Estat: mostra l'estat actual del client, per defecte el seu valor és "DISCONNECTED".
- Nom: nom del client, rebut per l'arxiu d'equips autoritzats.
- MAC: adreça mac del client, llegida també de l'arxiu d'equips autoritzats.
- Número aleatori: Numero aleatori inicialitzat a "000000\0".
- IP: adreça IP des de la qual s'han rebut els missatges dels clients, s'actualitza en cada nou registre.

Capítol 3: Execució del client

Per tal de seguir l'execució del client i entendre i explicar correctament el seu principal funcionament ens recolzarem en la figura 4.1, la qual ens mostra el diagrama de flux del client. Donat que aquesta imatge pot ser massa gran per aquest document, es troba adjuntada en el fitxer d'entrega de la pràctica la imatge corresponent a aquest, així com el diagrama de flux del servidor i el diagrama d'estats del protocol.

El principal funcionament del client en una execució correcta segueix els següents passos:

- Inicialment llegeix els arxius de configuració i inicialitza totes les variables necessàries en l'inici del programa.
- Tot seguit envia un paquet de petició de registre i passa a l'estat d'espera de registre.
- Si es rep un paquet de confirmació de registre l'equip passa a l'estat connectat i envia un paquet de manteniment de la comunicació.
- S'espera tres segons a rebre un paquet de confirmació de manteniment de la comunicació, si es rep passa a l'estat de manteniment de la comunicació.
- A partir d'aquest punt es mantindrà la comunicació amb el servidor fins que s'introdueixi la comanda quit.

En els casos en que no es doni una situació correcta es pot observar el diagrama de flux per tal de seguir l'execució del programa. aquests casos poden ser:

- No rebre confirmació de registre o bé rebre un paquet de registre no acceptat, el qual farà que es torni a intentar la subscripció mentre el numero d'intents de subscripció sigui més petit que tres, aquest valor es pot modificar en el programa.
- Rebre un paquet de registre rebutjat per part del servidor, el qual farà que el client es desconnecti.
- No rebre contestació al paquet de manteniment de la comunicació o be rebre un paquet de manteniment de la comunicació no acceptat, la qual cosa desconnectara el client i iniciarà un nou intent de registre.
- Rebre un paquet de manteniment de la comunicació rebutjat, el qual provocarà que el client passi a estat desconnectat i torni a iniciar una petició de registre.
- Rebre qualsevol tipus de paquet erroni o fora de seqüència es considerarà com no haver rebut resposta, tot i que es mostrarà el tipus de paquet rebut si es troba entre els contemplats en el protocol.

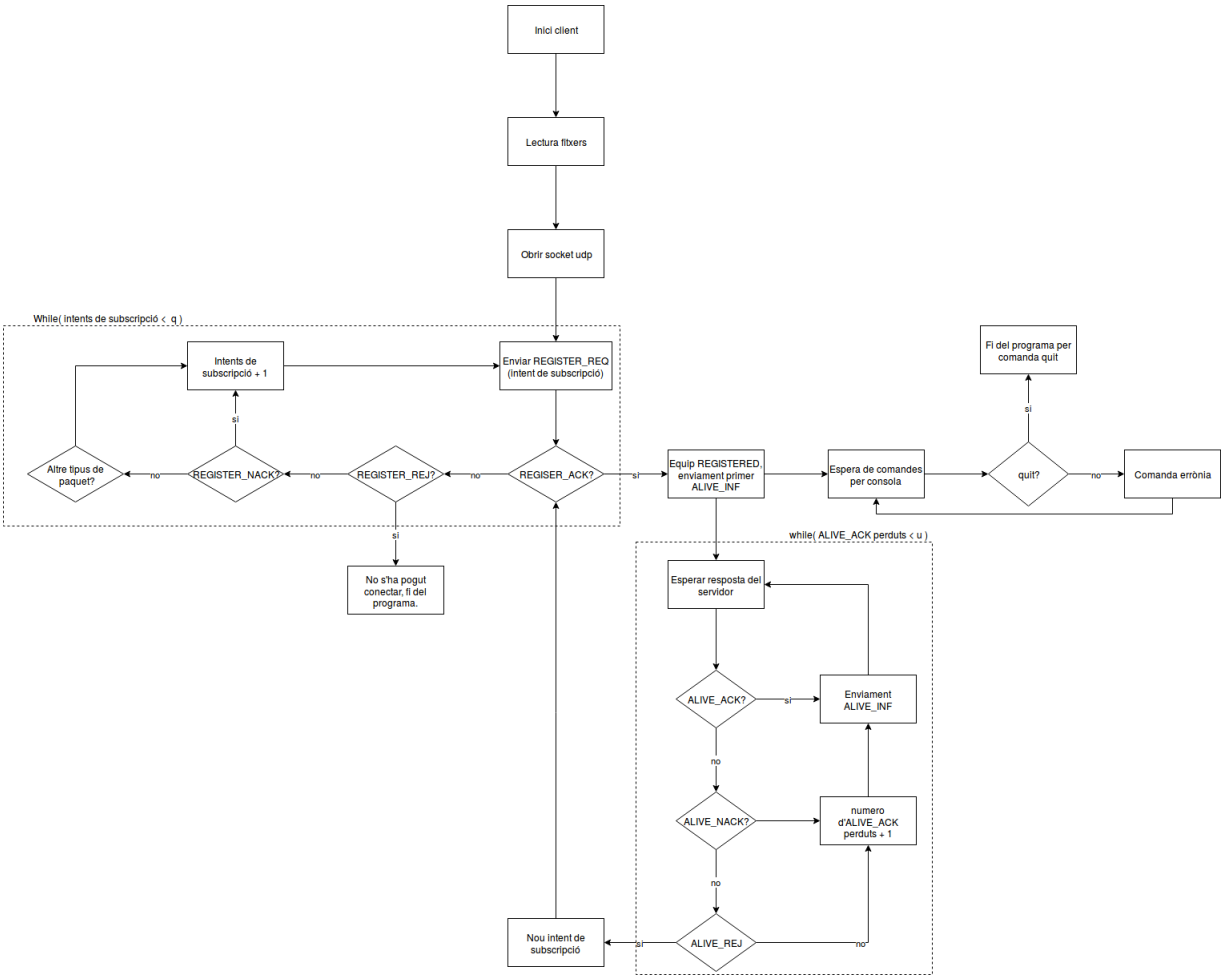

Figura 3.1: Diagrama de flux del client.

3.1 Decisions de disseny del client

Tot seguit veurem diferents implementacions i decisions de disseny que es van prendre durant el desenvolupament del client i perquè es va implementar d'aquesta manera.

Una de les principals dificultats que va impulsar aquesta decisió de disseny va ser rebre un paquet de manteniment de la comunicació cada tres segons, i no enviar un altre paquet de manteniment de la comunicació fins passats aquests tres segons. Per poder mantenir una sincronia entre enviament i recepció de paquets de manteniment de la comunicació vaig utilitzar un select amb un timeout de tres segons. A la documentació del select en c s'explica que el timeout d'aquest atribut, una estructura timeval, s'actualitza amb el temps que triga a arribar quelcom al buffer, i el descompta del temps total del temporitzador, d'aquesta manera si arriba un paquet en 0.013 segons l'estructura timeval del timeout canvia a 2.87 segons, tot i que en milisegons. Per tal d'esperar exactament aquest temps vaig utilitzar la funció nanosleep, la qual rep una estructura timespec, per tal de que s'esperes aquest tems exactament, la unica diferencia entre les dos estructures es que timeval emmagatzema els microsegons restants dels segons, i timespec emmagatzema

els nanosegons, per a això simplement vaig multiplicar els microsegons per mil.

Una altra dificultat que vaig trobar va ser saber si el programa havia de finalitzar per comanda o per pèrdua de paquets, ja que escanejar una comanda de consola es una funció bloquejant i si s'espera una comanda, encara que es perdin paquets i falli l'intent de subscripció el programa no finalitzarà a causa de la naturalesa bloquejant de la recepció de comanda. Per a solucionar aquest problema vaig emprar la funció poll, continguda en la llibreria poll.h, la qual comprova si hi ha algun envent en un descriptor de fitxer dins d'un interval de temps, de forma similar al select. Mitjançant aquesta funció si no es rep res per comanda es passara a comprovar que la resta del client es troba en estat correcte i aquest no ha de finalitzar.

Aquestes son algunes de les dificultats que he trobat a la part del client i la corresponent justificació de la implementació, de totes formes a mesura que he anat desenvolupant la pràctica he trobat altres dificultats però al codi ja es pot veure les solucions proporcionades, en la majoria de casos, sense explicació complementària.

3.2 Jerarquia d'estats del protocol

Per tal d'entendre com evolucionen els estats dins aquest protocol es pot consultar la figura 3.2, la qual representa el comportament natural dels clients seguint els estats i les transicions entre aquests, que hauria de seguir una correcta implementació de la pràctica.

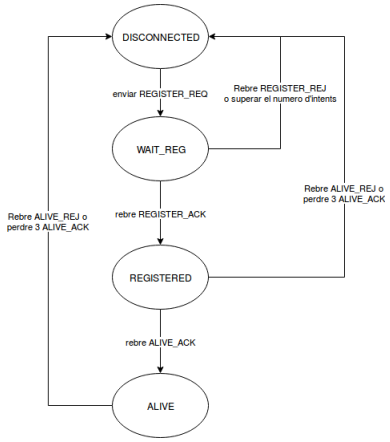

Figura 3.2: Diagrama d'estats del protocol.

Capítol 4: Execució del servidor

Tal com hem vist en el cas anterior ens recolzarem en el diagrama de flux mostrat en la figura, en aquest cas del servidor, adjuntat també amb la resta de material de la pràctica, per tal de seguir el funcionament del servidor, així com les diferents respostes respecte les situacions que es poden donar en rebre missatges dels diferents clients. En aquest aspecte el comportament del servidor es redueix a contestar paquets i controlar temps entre paquets de manteniment de comunicació rebuts, de forma que la implementació es pot reduir en menys passos:

- Inicialment llegeix els arxius de configuració i inicialitza totes les variables necessàries en l'inici del programa.
- Espera a rebre paquets o bé comandes.
- Si es rep un paquet de petició de registre es comprova si el client esta autoritzat i si les dades son correctes, en cas afirmatiu s'envia una acceptació de la petició de registre i s'inicialitza el timeout.
- Si es rep un paquet de manteniment de la comunicació d'un client que es troba en estat registrat o en manteniment de la comunicació es contesta amb un paquet d'acceptació del manteniment de la comunicació.
- Alhora s'escolten les comandes de la consola, les quals poden ser list i quit i s'executa L'acció pertinent.

En els casos en que no es doni una situació correcta es pot observar el diagrama de flux per tal de seguir l'execució del programa. aquests casos poden ser:

- Rebre una petició de registre d'un client no autoritzat, en el qual cas s'enviarà un paquet de rebuig del registre.
- Rebre un paquet de petició de registre amb dades incorrectes, al qual es contestarà amb un paquet de petició de registre no acceptada.
- Rebre un paquet de manteniment de comunicació d'un client no autoritzat, que tindrà com a resposta un paquet de rebuig del manteniment de la comunicació.
- Rebre un paquet de manteniment de comunicació d'un client autoritzat però amb les dades errònies, el qual serà respost amb un paquet de manteniment de la connexió no acceptat.
- Rebre qualsevol tipus de paquet erroni o fora de seqüència es considerarà com no haver rebut cap paquet, tot i que es mostrarà el tipus de paquet rebut si es troba entre els contemplats en el protocol.
- No rebre paquets de manteniment de la comunicació d'un client en el temps d'espera requerit implicarà passar l'estat del client a desconectat.

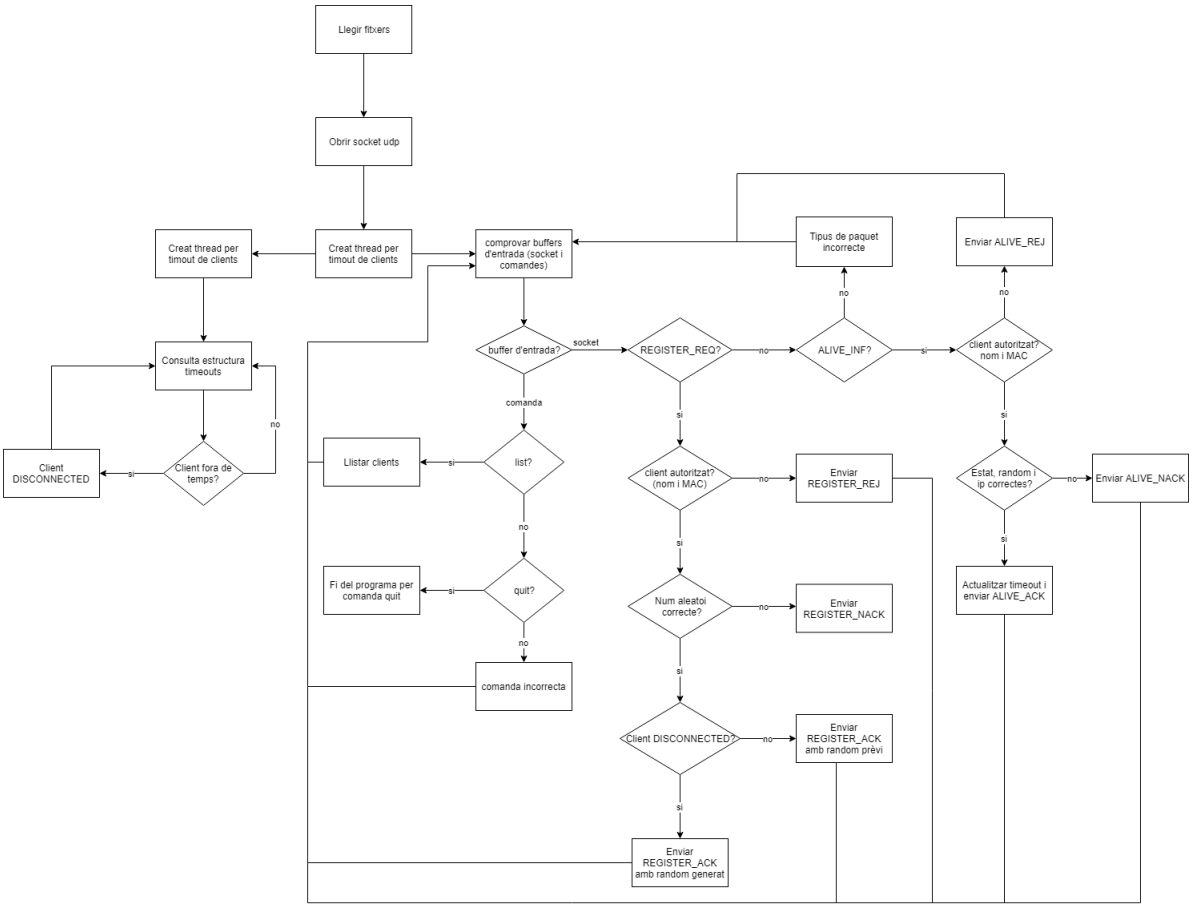

Figura 4.1: Diagrama de flux del servidor.

4.1 Decisions de disseny del servidor

Tot seguit veurem diferents implementacions i decisions de disseny que es van prendre durant el desenvolupament del servidor i perquè es va implementar d'aquesta manera.

La principal dificcultat amb la que em vaig trobar a l'hora d'implementar el servidor va ser com detectar que un client havia deixat d'enviar els paquets de manteniment de la comunicació en el temps esperat, i per tant, desconnectar-lo.

Per a aquesta tasca vaig crear un fil d'execució que comprovés una estructura de dades on es contenia el temps de l'ultim paquet de manteniment de la comunicació rebut i quin tipus de comprovació havia de fer. Aquest ultim valor podia prendre tres possibles estats: 0 si no s'ha de comprovar per que encara no esta a la fase de manteniment de la comunicació, 1 si esta registrat per'q no s'ha rebut el primer paquet de manteniment de comunicació o 2 en el cas d'estar ja en aquesta fase.

D'aquesta manera aquest thread estarà constantment mirant els clients que superin el timeout, en cas de rebre un paquet de manteniment de comunicació s'actualitzarà el temps del corresponent temporitzador. Si es detecta que no s'han rebut els paquets pertinents, es canviarà l'estat del dispositiu a desconnectat, i es reiniciaran el seu número aleatori i la seva ip.

La resta de funcions i diferents decisions de disseny del servidor es troben comentades en el codi o bé no requereixen d'explicació ja que la seva pròpia naturalesa es compren amb el propi seguiment del codi, tot i que de la mateixa forma que en el client, les dificultats es van anar trobant i superant a mesura que s'avançava amb el desenvolupament de la pràctica.

Capítol 5: Conclusions

De cara a l'enfocament de la pràctica trobo que es una pràctica molt útil per tal d'entendre la comunicació d'aplicacions a través de la xarxa mitjançant sockets. Si bé també trobo que la dificultat d'aquesta requereix de coneixements posteriors de la carrera que he adquirit en els cursos superiors, sobretot per tal de mantenir una estructura de codi més encarada a projectes més grans, o de més complexitat, i també per tal de conèixer els mecanismes de threading i sincronització que requereix una implementació curosa d'un protocol com aquest.

També he valorat l'ajuda que poden arribar a donar els diagrames i la planificació prèvia a la pràctica, ja que ajuden a comprendre el funcionament i simplificar el codi, tenint clar des del principi el rumb que prendrà l'execuciño del programa, la qual cosa no serveix de res un cop finalitzada la pràctica, i fets els diagrames i la planificació a posteiori per justificar el que s'ha fet sense planificació.

També he aprés a utilitzar una mica millor, i conjuntament al desenvolupament de la pràctica, un control de versions, en aquest cas Github, el qual m'ha ajudat a desenvolupar la pràctica per fases, i permetent-me tornar a versions anteriors funcionals, realment trobo que es una eina molt útil en quan es vol desenvolupar codi amb un mínim de tamany. El repositori utilitzat per fer la pràctica és el següent:

`https://github.com/marcmova/xarxes_prac_1`

El qual he decidit configurar com a public després d'acabar la pràctica per tal de col·laborar amb totes aquelles persones que puguin necessitar un exemple d'una implementació amb sockets o bé per fer un seguiment de com ha estat implementada la pràctica.