UNIVERSITAT DE LLEIDA
Escola Politècnica Superior
Grau en Enginyeria Informàtica
Xarxes

Pràctica 1

Ian Palacín Aliana

Professorat : ENRIQUE GUITART BARAUT, CARLOS MATEU PIÑOL
Data : 23 d'Abril de 2019

Contents

1 Introducció

Documentació de la pràctica 1 de xarxes. L'objectiu d'aquest projecte és el de programar una aplicació de xarxes utilitzant el model client-servidor seguint les pautes d'un protocol de comunicacions. El document està partit en 5 parts, sent el primer la introducció, seguit d'un diagrama de l'estructura del client i el servidor, l'explicació del manteniment de la comunicació, el diagrama d'estats UDP i, per últim, consideracions a tenir en compte.

2 Diagrames d'estructura

2.1 Estructura del client

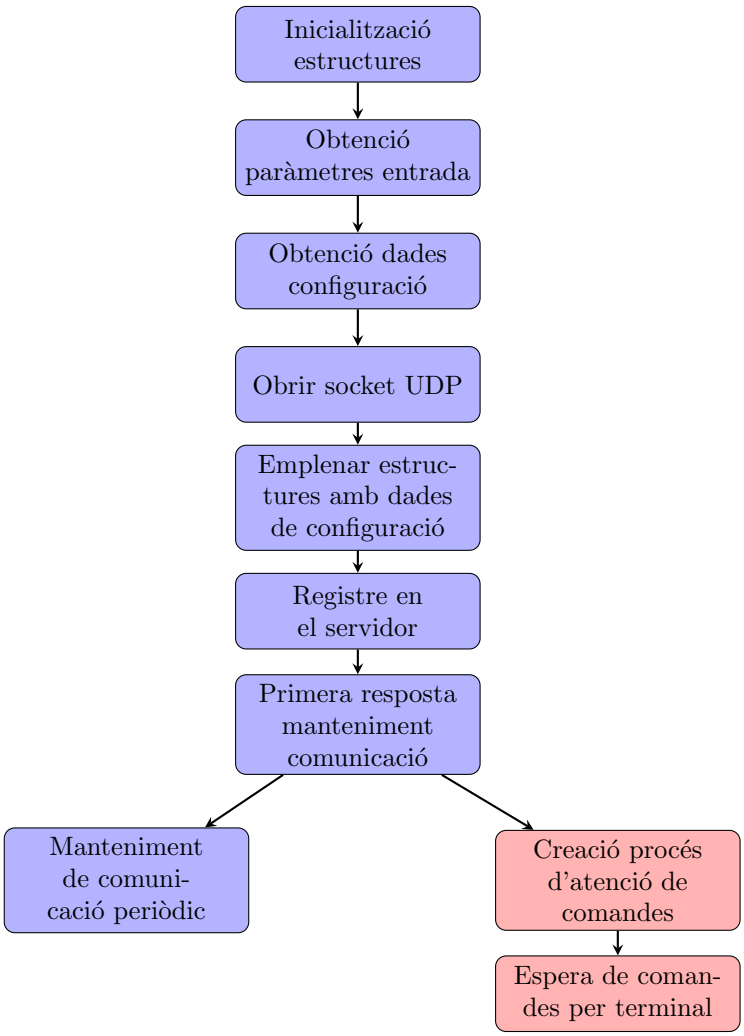

Figura 1: Diagrama de blocs de l'estructura del client

2.2 Estructura del servidor

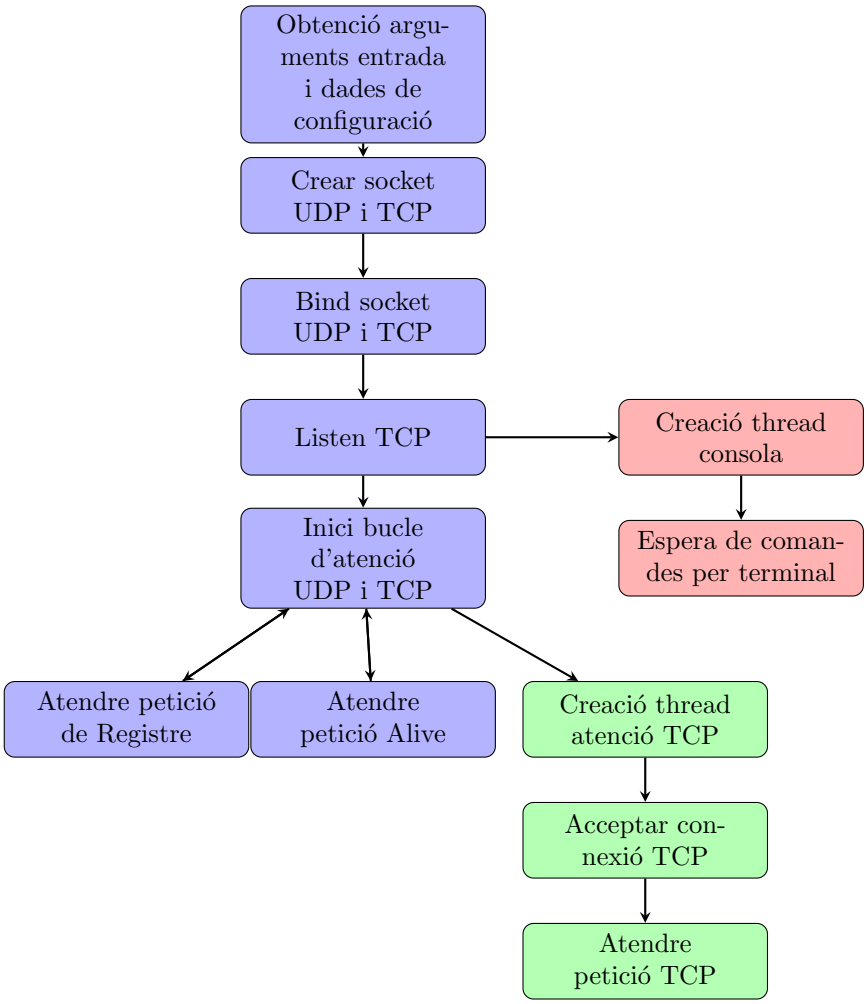

Figura 2: Diagrama de blocs de l'estructura del servidor

3 Manteniment de comunicació

3.1 Manteniment de comunicació del client

Una vegada el client s'ha registrat, passa a la fase de manteniment de comunicació amb el servidor. Per entendre l'estratègia emprada primer s'han d'introduir unes variables importants. La primera és **intent**, és el comptador d'intents d'alive. Variable de tipus enter que s'inicialitzarà a 0 i anirà canviant de valor en funció dels paquets que li arribin (o li deixin d'arribar). **Intent** ajudarà a sortir en el moment adient del bucle principal d'enviament d'ALIVE_INF com, per exemple, quan es rep un ALIVE_REJ (es veuran tots els altres casos més endavant).

La segona variable a mencionar és **first**, variable de tipus enter però que funciona com una booleana. Aquesta variable serveix per saber si ja s'ha rebut el primer ALIVE_REQ. S'inicialitza a 1 i en recepció de la primera confirmació d'alive passa a 0, nombre en el qual es quedarà durant tota la resta de procés de manteniment de comunicació.

El procés de manteniment de comunicació consta d'un bucle principal, que s'executarà mentres **intent** sigui menor a **u**, el nombre màxim de temps d'alive permesos sense rebre confirmació. A cada iteració del bucle s'enviarà una petició d'alive i s'esperarà un temps d'alive (r). Si després de l'espera no s'ha rebut resposta del servidor, s'incrementarà **intent** i es tornarà a dalt del bucle. Si després de l'espera s'ha rebut resposta, aquesta es tractarà de la següent manera:

- Si la resposta és un ALIVE_ACK del servidor corresponent, es ficarà el comptador d'intents a 0 de nou, d'aquesta manera es podran tornar a permetre **r** ALIVE_INF sense resposta. Si **first** val 1 voldrà dir que és la primera confirmació d'alive i, per tant , el client canviarà d'estat registrat a alive. En aquest moment és també quan es crea el procés de consola.
- La recepció d'un ALIVE_REJ del servidor corresponent es considerarà com suplantació d'identitat i es ficarà el comptador d'intents a **u** (nombre màxim d'intents sense confirmació permesos), d'aquesta manera acabarà el bucle i la fase d'alive, tornant a la fase de registre
- En qualsevol altre cas, que és la recepció d'un paquet no autoritzat o d'un paquet de tipus ALIVE_NACK es considerarà com no haver rebut resposta del servidor, i simplement s'incrementarà el comptador d'intents

Utilitzant aquest sistema s'aconsegueix el manteniment de connexió corresponent amb el servidor per part del client.

3.2 Manteniment de comunicació del servidor

En la part del servidor, cada client que ha superat la fase de registre, té un camp anomenat TTL que fa referència al temps (en temps d'alives) que li queda fins a ser desconnectat. Quan aquest camp arriba a 0, es considera que s'ha perdut connexió amb el client i passa a l'estat DISCONNECTED.

Perquè a cada temps d'alive es puguin actualitzar els TTL dels clients i a l'hora atendre totes les altres peticions del servidor, s'utilitzen threads concurrents. El programa principal és el que atendrà les peticions TCP i UDP. Cada vegada que es rebi un ALIVE_INF correcte es crearà un thread que sumarà **k** (3) al camp TTL del client que ha enviat la petició. Acte seguit a cada temps d'alive li anirà restant 1, si s'arriba a 0 vol dir que no s'ha rebut cap ALIVE_INF en el transcurs d'aquests 3 temps d'alive. Això és degut al fet que si s'hagués rebut una altra petició d'alive del mateix client, s'hauria creat un altre thread, que li hauria sumat k (3) TTLs més.

Aquest sistema és possible gràcies al fet que les variables globals estan compartides entre threads d'un mateix procés; i la informació dels clients (on s'actualitzen els TTLS) estan en una llista de diccionaris de forma global. Una vegada entés el sistema de TTL, es pot explicar de forma més ordenada el procés sencer de manteniment de comunicació.

Una vegada el servidor ha rebut la petició de registre correcta i l'ha respost de forma adequada, el servidor ha de rebre un alive en menys de 2 temps d'alive. És per això que el servidor només acabar la fase de registre del client crea un thread dels comentats anteriorment, però en comptes de sumar 3 TTLs, li suma 2.
Mentre el thread secundari va restant els temps d'alive, el thread principal va atenent les peticions que li arriben pels canals UDP i TCP. Atén els ALIVE_INF de la següent forma:

- Quan rep un ALIVE_INF d'un client, mira si aquest està autoritzat, en l'estat correcte, i si el nombre aleatori i IP concorden amb el client registrat. Si tot és correcte se li enviarà un ALIVE_ACK i es crearà un thread de descompte de TTL de 3 temps d'alive. Si l'estat del client era REGISTERED, aquest passarà a ALIVE.

- Si la petició prové d'un client no autoritzat o que no està en els estats REGISTERED o ALIVE, se li enviarà un ALIVE_REJ informant-lo del motiu.

- Si la IP o el nombre aleatori del client no concorden se li enviarà un ALIVE_NACK informant-lo del motiu.

Amb aquest sistema de TTLs s'aconsegueix mantenir la comunicació corresponent amb els clients necessaris de forma concurrent per part del servidor.

4 Diagrama d'estats UDP

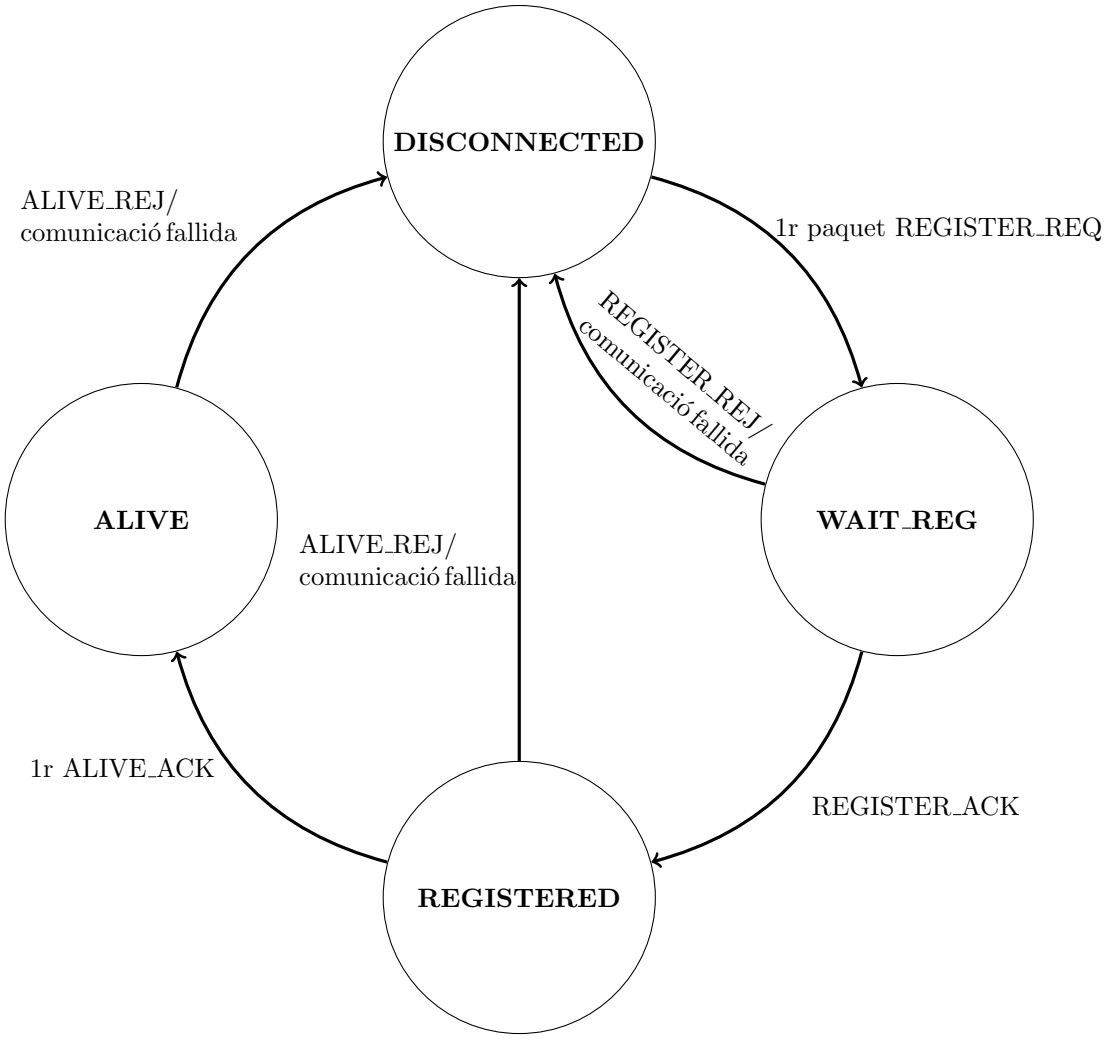

Figura 3: Diagrama de la màquina d'estats del protocol implementat

5 Consideracions

5.1 Buffer consola

La consola del client s'activa una vegada s'ha arribat a la fase d'alive. Si s'escriu alguna comanda quan això encara no ha passat, quan la consola s'activi llegirà de l'entrada estàndard tot el que s'havia escrit quan aquesta no estava activada. He pensat que deixar-ho així, o buidar el buffer de l'entrada estàndard primer és una qüestió subjectiva més que un error ja que, per exemple, si el client fa un quit per marxar, i la consola encara no està activada, el primer que farà quan s'activi és fer el quit. És per això que he decidit deixar-ho així.

5.2 Llargada cua de clients

A l'enunciat de la pràctica s'especifica que en les proves hi hauria 2 clients de forma concurrent, així que s'ha triat de forma arbitrària que la cua de clients que el servidor és capaç d'escoltar és de 3 clients.

5.3 SO_REUSEADDR

Fent una sèrie de tests al client, me'n vaig adonar que si s'utilitzava get-conf, es tancava el client i es tornava a executar amb inmediatesa saltava el següent error durant un petit termini de temps:

```
[Errno 98] Address already in use
```

Pensant-me que era un error al meu codi a causa de un mal tancament dels sockets, vaig cercar l'origen. Resulta ser que encara que el socket es tanqui de forma correcta, el sistema operatiu té un temps TIME_WAIT abans que la stack de TCP consideri que la connexió ha estat tancada completament. Aquest fet es pot evitar fàcilment amb l'opció REUSEADDR:

```
tcp_sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

No obstant això, en ser una decisió del sistema operatiu, he considerat no ficar-ho, i deixar que el meu programa sigui afectat per aquest temps d'espera.

5.4 Get-conf servidor de prova

Una petita observació és que el servidor de proves penjat al campus virtual envia dues vegades l'última línia de configuració quan se li demana amb get-conf. Simplement a tenir en compte si s'utilitza per fer algun tipus de test.