

UNIVERSITAT DE LLEIDA  
Escola Politècnica Superior  
Grau en Enginyeria Informàtica  
Xarxes

# Pràctica 1

Ian Palacín Aliana

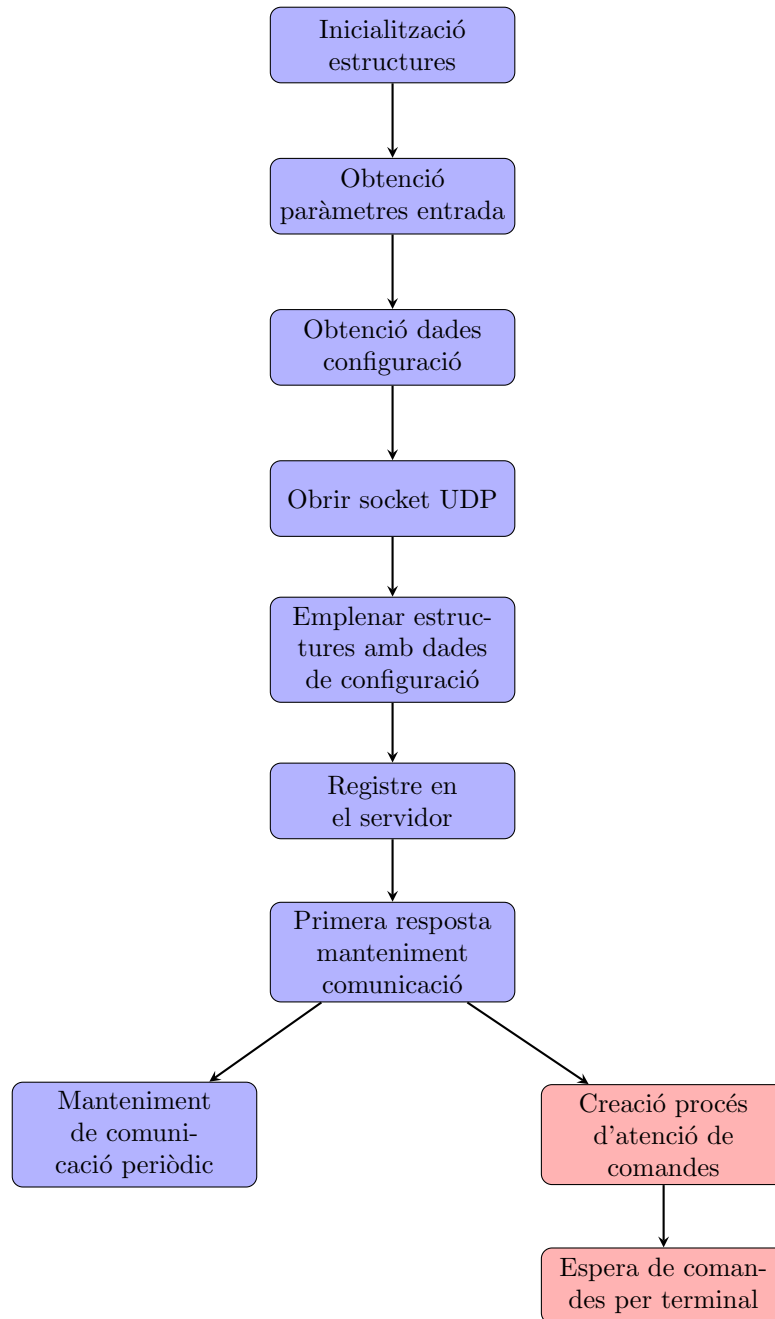
Professorat : ENRIQUE GUITART BARAUT, CARLOS MATEU PIÑOL  
Data : 23 d'Abril de 2019

## Contents

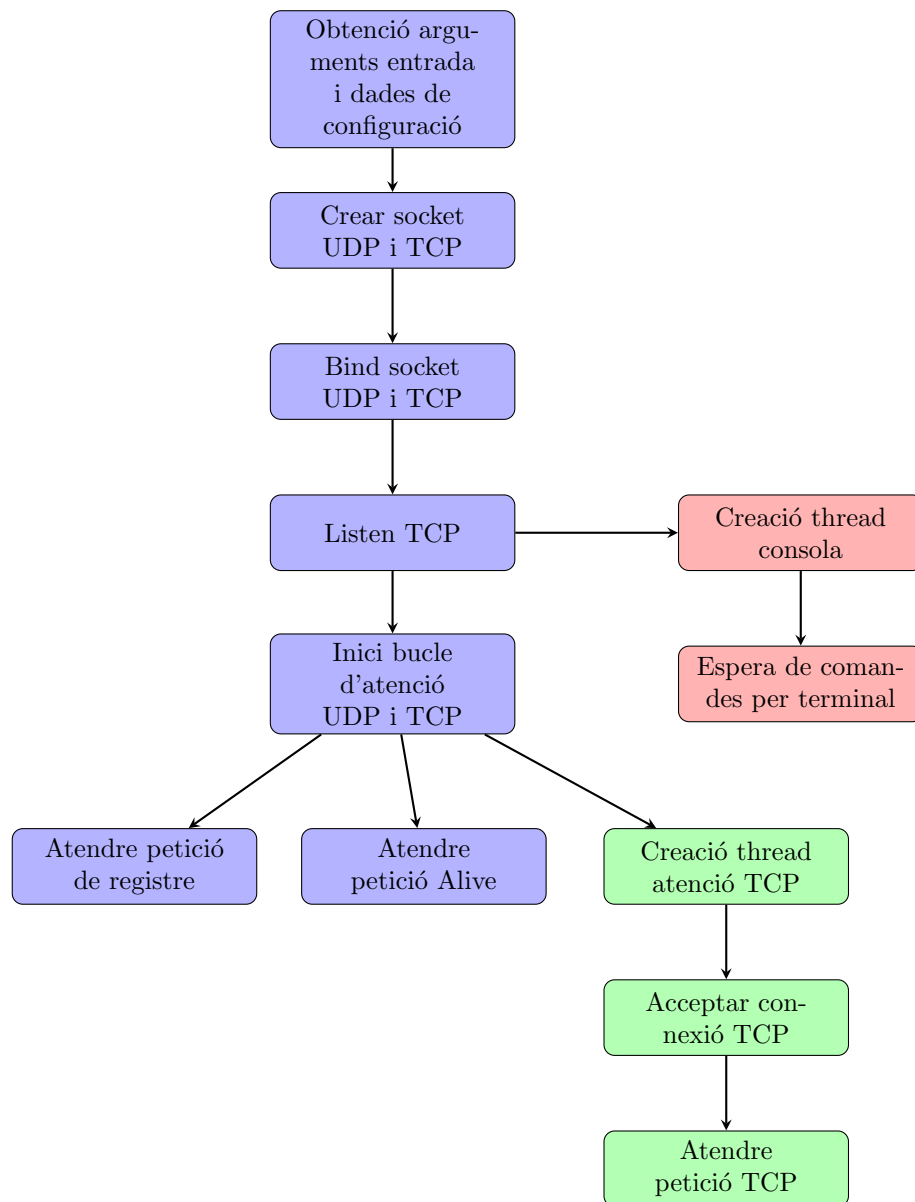
<b>1</b>	<b>Diagrames d'estructura</b>	<b>1</b>
1.1	Estructura del client . . . . .	1
1.2	Estructura del servidor . . . . .	2
<b>2</b>	<b>Manteniment de comunicació</b>	<b>3</b>
2.1	Manteniment de comunicació del client . . . . .	3
2.2	Manteniment de comunicació del servidor . . . . .	3
<b>3</b>	<b>Diagrama d'estats UDP</b>	<b>5</b>
<b>4</b>	<b>Consideracions</b>	<b>5</b>

# 1 Diagrames d'estructura

## 1.1 Estructura del client



## 1.2 Estructura del servidor



## 2 Manteniment de comunicació

### 2.1 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 contador 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 que 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 mentre **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 contador d'intents a 0 de nou, d'aquesta manera es podran tornar a permetre **r** ALIVE\_INF sense resposta. Si **first** té el valor 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 contador 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 contador d'intents

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

### 2.2 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

ser desconnectat. Quan aquest camp arriba a 0, es considera que s'ha perdut connexió amb el client i passa a l'estat DISCONNECTED.

Per a que a cada temps d'alive es puguin actualitzar els TTL dels clients i a l'hora atendre totes les demés peticions del servidor, s'utilitzen threads concurrents. El programa principal és el que atendra les peticions TCP i UDP. Cada vegada que es rebí 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 a que si s'hagués rebut un 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 a 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 ves de sumar 3 TTLS, li suma 2.

Mentres 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 número 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 número 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.

### **3    Diagrama d'estats UDP**

### **4    Consideracions**