Universitat de Lleida

Escola Politècnica Superior
Enginyeria Informàtica

Xarxes: Pràctica 1, Programació d'aplicacions de xarxa

Autor
David Muñoz Sánchez

Índex de continguts

Índex de figures

Abstract

L'objectiu d'aquest informe és, principalment, l'explicació de les idees prèvies i d'implementació d'aquesta pràctica, així com l'esquema que en sorgeix, tant de client com de servidor, per a la realització d'aquesta pràctica.

També s'explicarà les causes darrere de les decisions de disseny i implementació, junt amb els problemes trobats i les conseqüents solucions en el desenvolupament.

Diagrames de blocs

Partint del fet que els clients poden estar en estats diferents dels atribuïts pel servidor, cal diferenciar i establir un diagrama de blocs pel funcionament en cadascun, tot i que, al cap i a la fi, els seus comportaments s'assemblen.

Aquests dos diagrames representen les fases al llarg del procés de subscripció i la posterior comunicació entre ells, fent menció a les principals funcions implementades en el servidor i el client en cada respectiva fase.

Estructura del client

En la implementació es va seguir l'estructura especificada en la *figura 1*. En aquesta figura es remarquen les diferents etapes del client i tant les funcions més importants que duen a terme la tasca (en negreta) com les auxiliars.

Per controlar els canvis d'estat que esdevé el client s'utilitza la funció *subscribe_process()*. Aquesta s'encarrega d'establir l'estat del client corresponent en conseqüència al punt del diagrama d'estats on ens trobem. També és la que controla que les dades dels paquets siguin correctes, esdevenint llavors, en la funció nucli del procés de subscripció.

Dins de la funció *subscribe_process()*, s'empren crides a altres funcions com *create_subs_req()* o *create_hello()* per crear els diferents paquets de dades que s'envien durant el procés.
Així i tot, cal recalcar la funció *time_out_subscribe*, que tot i com es pot veure en la *figura 1*, no esdevé la principal funció dins de la tasca, però el seu paper és molt important. És la funció encarregada de temporitzar i contar els paquets del tipus SUBS_REQ que s'envien en cada procés de subscripció. A més a més de tractar la resposta del servidor en conseqüència amb l'ajuda de *subs_ack_treatment()*.

Un cop completada la subscripció, el procés encarregat de mantenir la comunicació romandrà en la funció *subscibe_process()* entrant contínuament, en l'apartat del codi corresponent, a enviar HELLO's fins que no passi una irregularitat.

Finalment, respecte a la part del port de protocol de control de transmissió (TCP), s'utilitza un *select* per determinar si s'ha introduït una comanda per la terminal o si ha arribat un paquet pel part del servidor, és a dir, mitjançant la funció *periodic_comunication()* tractem les dues entrades d'informació alhora.

El mateix procés que controla el select, un cop rep un paquet DATA_REJ, envia el senyal SIGUSR2 perquè el procés pare comenci un nou procés de subscripció i tanca el socket TCP.

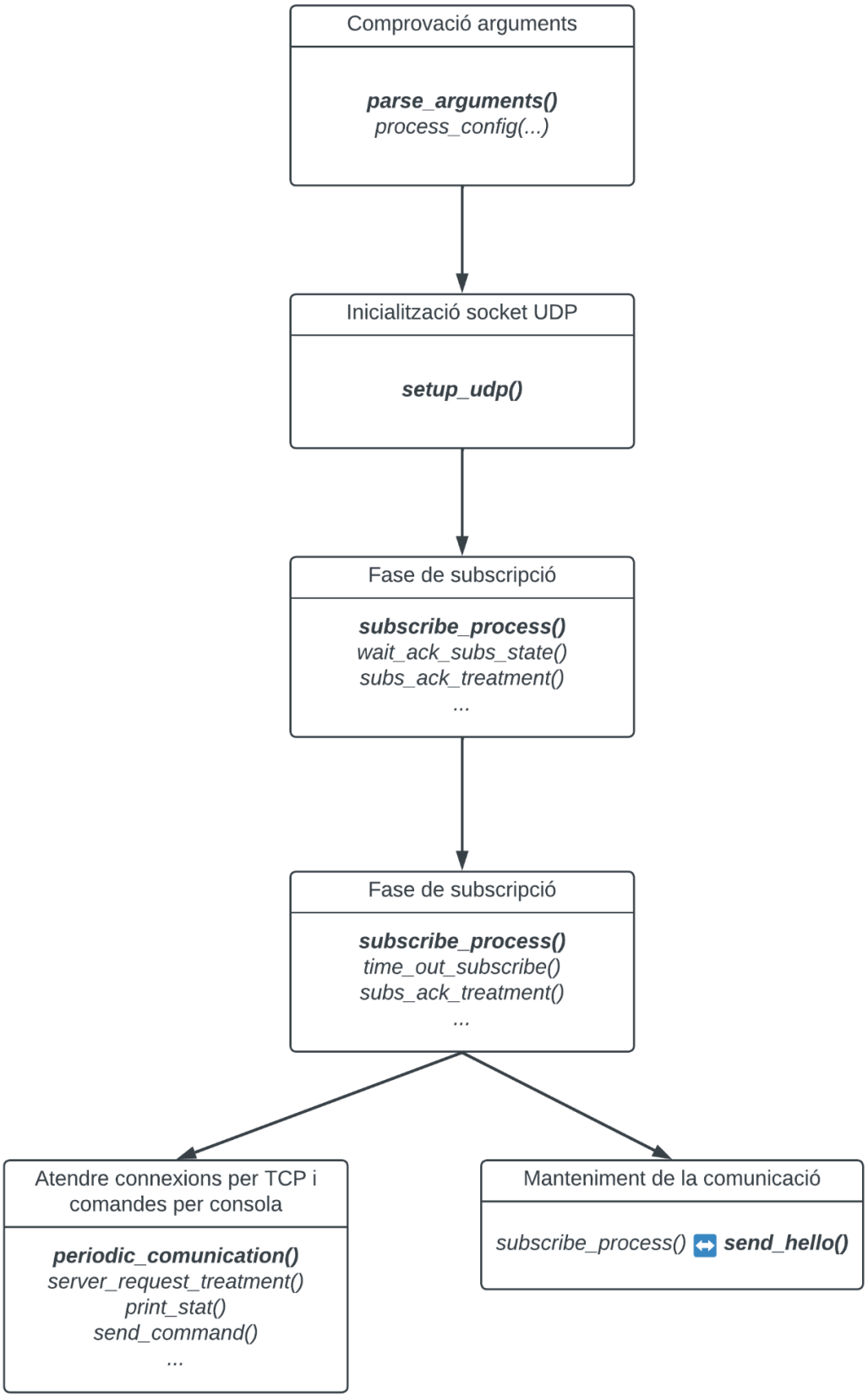

Figura 1: Diagrama de blocs del client

Estructura del servidor

En la implementació es va seguir l'estructura de la *figura 2*.
Es passa de tenir una funció que esdevé el nucli del procés de subscripció a dividir-lo entre 3 funcions i a utilitzar *threads* en lloc de fills. Com a resultat, un *thread* s'encarrega de tractar els paquets entrants pel port UDP, un altre els entrats pel port TCP i el procés principal de llegir la terminal.

En primer lloc, quan es rep una connexió UDP, com especifica la *figura 2*, només podem anar a parar a *send_hello()* o *recieve_info()*.

Per una banda, *recieve_info()* tracta els paquets que són del tipus SUBS_REQ. Seguidament, la funció *send_subs_ack()* és l'encarregada tant d'enviar el paquet del tipus SUBS_ACK com d'esperar una resposta pel part del servidor i tractar aquesta.
Finalment, si ens trobem en el cas d'una subscripció amb èxit, el programa entra en la funció *send_info_ack()*, on s'envia el paquet del tipus INFO_ACK i establint el temporitzador pel primer paquet del tipus HELLO.
Per l'altra banda, la funció *send_hello()*, com el seu nom diu, s'encarrega d'enviar els paquets HELLO i establir el comptador pel *time out*.

Pel que fa a la lectura del port TCP, a cada paquet entrant es crea un nou *thread* per tractar-lo. D'aquesta manera es pot continuar llegint el *buffer* del port.

Finalment, pel que fa a la lectura de comandes, no es creen *threads* per resoldre cada comanda que es passa per la terminal, simplement s'utilitza el mateix procés que llegia la terminal.

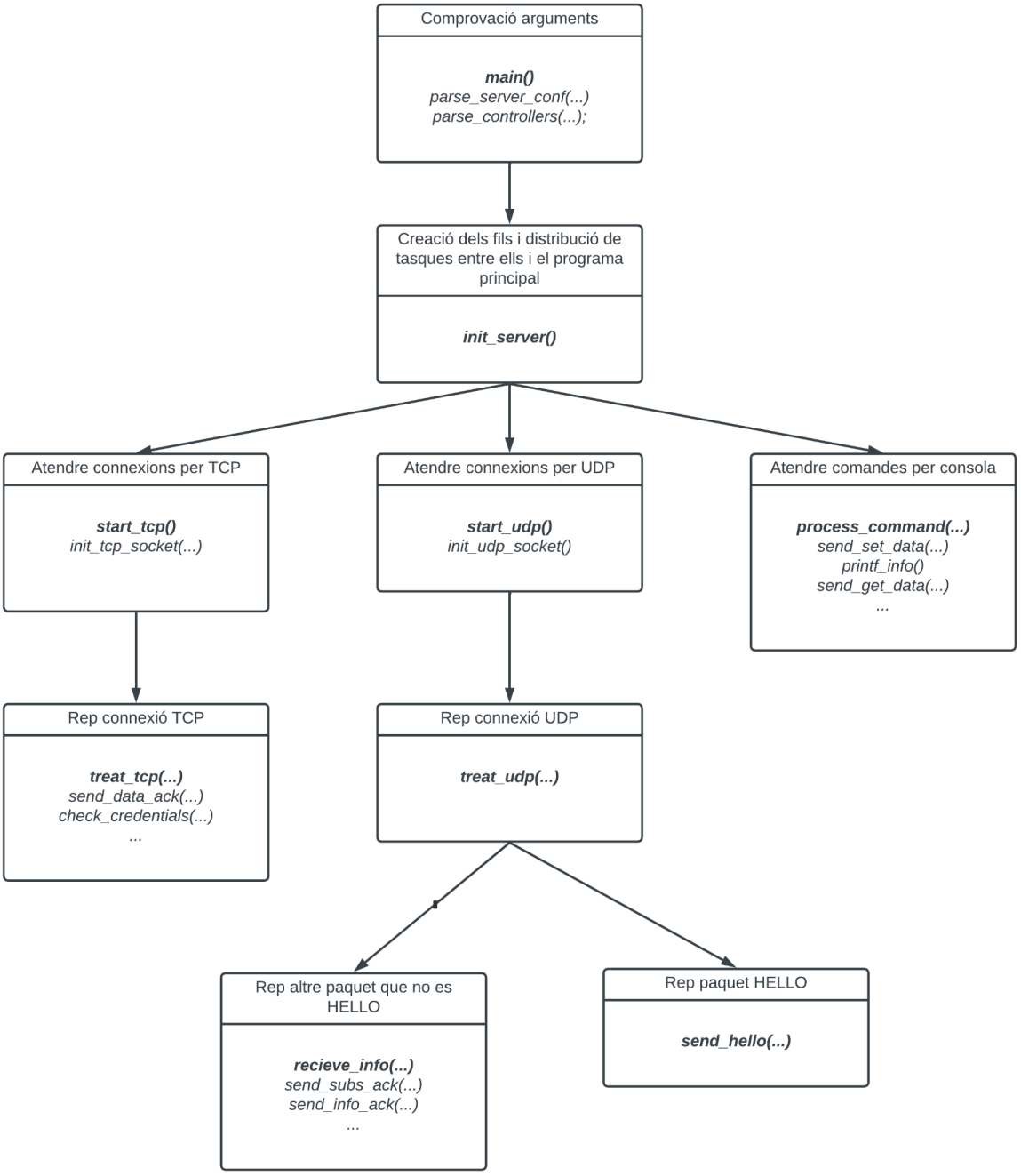

Figura 2: Diagrama de blocs del servidor

Fase de manteniment de comunicació

Client

En aquesta fase el client es troba ja en l'estat SEND_HELLO, per tant, té l'opció de llegir comandes per la terminal activada, llegeix constantment pel port TCP i envia paquets HELLO pel port UDP.

Per resoldre aquesta situació, en rebre el primer HELLO per part del servidor es crea un procés fill que s'encarregarà de llegir tant el port TCP com les comandes de la terminal mitjançant un select. Com a conseqüència, el procés pare s'encarregarà d'enviar constantment els paquets HELLO.

La funció *send_hello()* s'encarrega d'aquest manteniment constant.
A més a més, mitjançant la variable *no_response* també porta el compte de quants paquets HELLO no han tingut resposta per part del servidor i estableix un comptador de v = 2 segons per rebre cada paquet HELLO.

En el cas on el client deixa de rebre s = 3 HELLO's, per tornar a passar a l'estat NOT_SUBSCRIBED, s'haurà de finalitzar el fill que tractava les comandes de la terminal i el port TCP.
Per aconseguir-ho s'envia el senyal SIGUSR1 des del procés pare al fill. En la funció de tractament del senyal el fill es tanca el port TCP i es finalitza el procés amb èxit. D'aquesta forma es torna al punt de partida, és a dir, a l'estat NOT_SUBSCRIBED.

Servidor

Respecte al manteniment de la comunicació, en el servidor la implementació és més senzilla. Abans de res, en lloc d'utilitzar fills s'empren *threads*.
Com a resultat obtenim el procés principal i dos *threads*.

El procés principal és l'encarregat de llegir comandes per la terminal i és qui crea els dos *threads*, un per tractar els paquets que arriben pel port UDP i l'altre per tractar els que arriben pel port TCP.

En tractar-se d'un servidor concurrent, tractar tota la part del port UDP amb un *thread* no és viable. Per això, cada vegada que el *thread* encarregat de llegir pel port UDP rep un paquet, es crearà un altre *thread* per tractar-lo.
D'aquesta manera es tracten els paquets mentre es poden rebre de nous alhora.

Emfatitzant els paquets HELLO que rep el servidor, és de crucial importància que aquest comprovi si no s'han rebut x = 3 HELLO's consecutius.
És per això que, beneficiant-se del fet que els *threads* comparteixen memòria, establim un comptador de HELLO's dins de cada estructura de client.

Aquesta variable s'anomena *check_pack* i s'incrementa d'un en un cada cop que un HELLO és enviat. Posteriorment, es fa un *sleep()* de v * x segons (són els v = 2 segons que tarda un paquet a enviar-se i x = 3 HELLO's consecutius no rebuts que no podem sobrepassar). En finalitzar el *sleep()* es comprova si el valor de *check_pack* és el mateix que el valor que tenia aquest abans de fer el *sleep()*.

La diferència entre el *check_pack* anterior i posterior si la comunicació va bé és de 3 (doncs són el màxim de paquets HELLO no rebuts consecutius permesos).
En el moment que deixem de rebre HELLO's consecutius, la diferència disminueix, fins a arribar a 0, on es passarà el client a l'estat DISCONNECTED.

Diagrama d'estats del protocol implementat sobre UDP

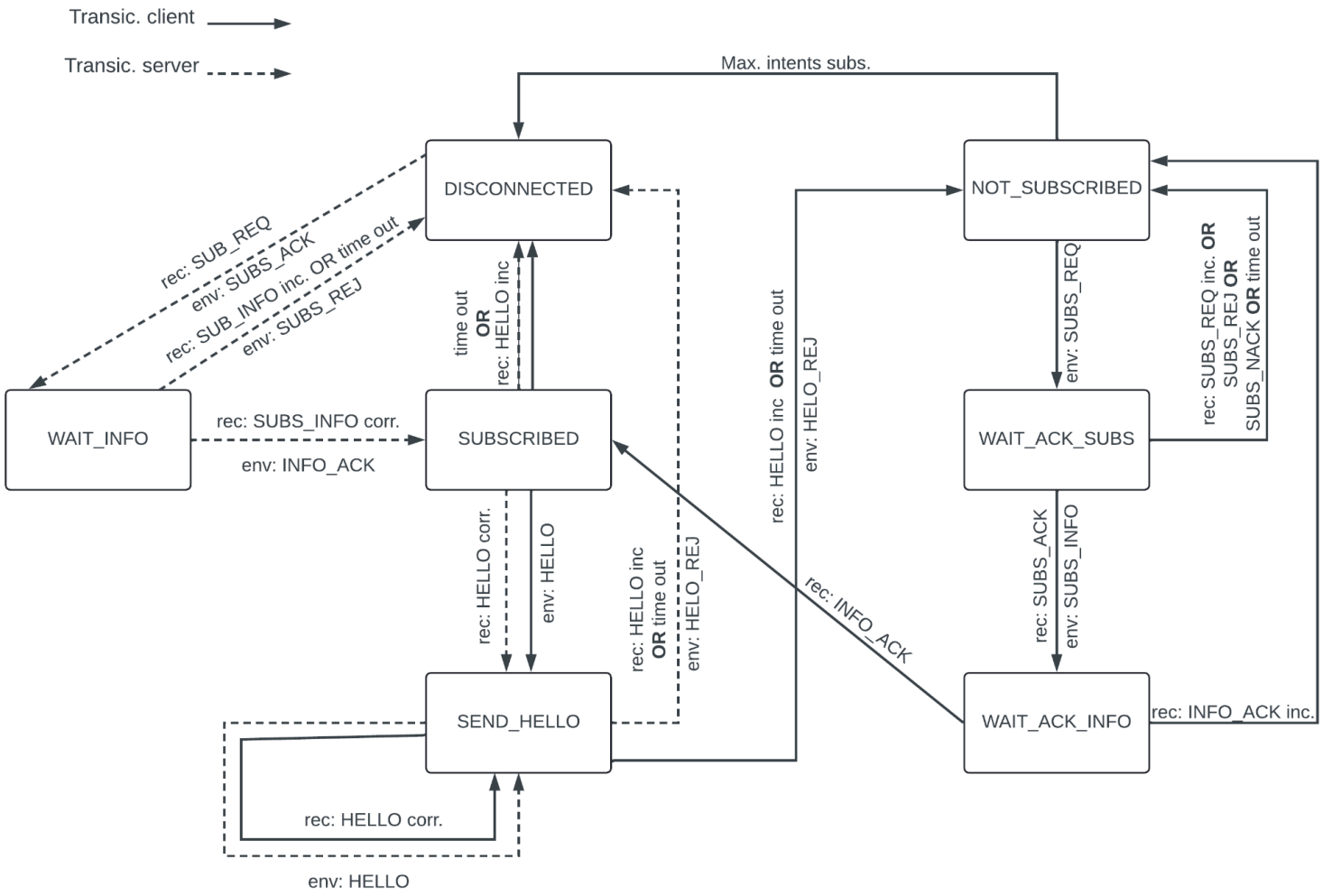

Figura 3: Diagrama d'estats del protocol implementat en UDP

Decisions d'implementació

Client

Respecte al client, el punt més destacat a comentar és el tractament d'un nou procés de subscripció un cop està en l'estat SEND_HELLO, amb el port TCP obert i llegint la terminal.
En primer lloc, ja que no s'utilitzen *threads*, la comunicació entre diferents processos és més difícil. Si el procés que escolta el port TCP rep un paquet DATA_REJ, no pot simplement tancar el socket i finalitzar, li ha de fer saber a l'altre procés que ha d'inicialitzar un nou procés de subscripció.

Per aconseguir aquest comportament es fan servir senyals, en concret s'utilitzen SIGUSR1 i SIGUSR2.

SIGUSR1 és llançada pel procés principal en cas que *s* paquets del tipus HELLO no tinguin resposta. Aquest senyal comportarà el tancament del port TCP de l'altre procés i la seva finalització.

Per una altra part, SIGUSR2 és llençada pel procés fill que llegeix tant la terminal com el port TCP un cop rep un paquet del tipus DATA_REJ o rep un paquet del tipus SET_DATA o GET_DATA que contenen discrepàncies en les dades d'identificació del servidor. Aquest senyal comportarà que el procés principal esperi *u* segons abans de començar un nou procés de subscripció.

Servidor

Respecte al servidor, el fet més important a destacar és el funcionament del manteniment de comunicació amb els clients.

En primer lloc, el problema es presenta quan hem de comprovar quan no es reben x = 3 HELLO's consecutius. Per això, dins de cada estructura de *Client* hi ha una variable anomenada *check_pack* que compta quants paquets HELLO s'han enviat.

D’aquesta manera, aprofitant la memòria compartida, si se surt del *sleep* i el valor de *check_pack* s’ha mantingut, significa que l’últim HELLO enviat era el seu, i que, com llavors no s’han enviat més HELLO’s des de llavors, com a conseqüència no haurem rebut els HELLO’s corresponents.