Informe Programació Aplicacions de Xarxa

Marc Alsina Martínez

Doble grau en Enginyeria Informàtica i Administració i Direcció d'Empreses

4/2022

Abstract

Aquest treball consta de la realització d'una arquitectura client-servidor, programats en llenguatge de programació C i en python, respectivament, per la simulació de la connexió entre uns sensors o actuadors amb un equip encarregat de l'emmagatzemament de dades rebudes. Tot això, s'aconsegueix mitjançant l'ús de diversos protocols, UDP i TCP, descriptors d'arxiu i un sistema d'estats per garantir la bona comunicació entre ells.

Índex

1 Introducció

1.1 Motivació i objectius del treball

Amb la realització d'aquest treball de l'assignatura de Xarxes s'ha volgut aconseguir una simulació del funcionament real entre un client i un servidor a nivell empresarial, sobretot en les anomenades indústries 4.0 que han proliferat aquests últims anys.

1.2 Estructura del document

Aquest document consta d'un desenvolupament, separat en tres parts principals, on en cadascuna d'elles es respon a les diverses preguntes plantejades a l'enunciat d'aquesta pràctica. Finalment, s'afegeixen unes conclusions per interpretar els resultats i valorar el treball i els objectius plantejats.

2 Desenvolupament

2.1 Estructura en forma de diagrama de blocs del client i del servidor

Fotografia adjunta al fitxer Diagramablocsclient-servidor.png, degut a que la mida d'aquesta no s'adequa al format presentat, i per tant no es podia visualitzar correctament.

2.2 Estratègia emprada pel manteniment de la comunicació, tant del client com del servidor

Per realitzar el manteniment de la comunicació periòdica al client, he decidit primerament, partint de que aquest es troba en l'estat REGISTERED, crear un procés mitjançant la funció *pthread* del llenguatge C, que ens permet iniciar un nou procés, i que s'encarrega d'enviar paquets ALIVE cada v segons, en el nostre cas $v = 2$, mentre el client es trobi en l'estat REGISTERED o SEND_ALIVE. A continuació, i mentre aquest procés segueix executant-se, fent ús de l'estructura *timeval* amb un retard establert de $r * v$ segons, i la funció *select* amb el descriptor d'arxiu de tipus UDP aconseguim saber si el primer ALIVE s'ha rebut dins del temps definit. Si és així, utilitzem la funció *recvfrom* per llegir de la comunicació UDP el contingut, en forma de *bytes*, fem les comprovacions (tipus de paquet, id. transmissor correcte, id. comunicació correcte i camp dades amb el id del client) i si tot es cumpleix passem el client a l'estat SEND_ALIVE, reiniciem a 0 el nombre de registres i inicialitzem una variable anomenada *time_alive* amb la funció *time* per saber en quin moment s'ha rebut el paquet ALIVE. Si no és així, el client passarà a l'estat NOT_REGISTERED.

Seguidament, si tot ha estat correcte, es crearà un nou *socket* de tipus TCP i amb l'ús de la instrucció *while* mentre el client es trobi en l'estat SEND_ALIVE i la funció *select* amb els tres descriptors d'arxiu: UDP, ja utilitzat anteriorment, TCP i *STDIN*, encarregat de saber si l'usuari ha introduït comandes per terminal, esperarem a rebre els paquets ALIVE correcte per la comunicació UDP reiniciant cada cop la variable *time_alive* amb la funció *time*. D'aquesta forma, a cada iteració del bucle afegirem una comprovació per saber si s'han deixat de rebre tres paquets ALIVE consecutius. Aquesta comprovació consta de la comparació entre el temps actual i la variable *time_alive* que ens indica quan s'ha rebut l'últim paquet ALIVE correcte per saber si el resultat de la diferència d'aquests dos és major a $v * 3$, ja que l'enviament de dos paquets ALIVE és de v segons i el nombre 3 perquè és el número màxim de paquets d'aquest tipus que es poden deixar de rebre per desconnectar el client.

En quant al servidor, el procediment d'ALIVE comença si es rep un paquet del client i aquest es troba en estat REGISTERED o SEND_ALIVE. En aquest punt, es mira si el paquet rebut és correcte i si s'ha rebut el paquet dins del temps w des de que el client ha passat a l'estat REGISTERED, si és així, vol dir que s'ha rebut el primer paquet ALIVE i s'envia un paquet ALIVE de resposta. Addicionalment, si el client es troba en l'estat REGISTERED, és a dir, és el primer paquet ALIVE que rebem canviem l'estat d'aquest a SEND_ALIVE i posem el comptador ja utilitzat a zero perquè ja no cal revisar-lo pels paquets ALIVE següents. Si aquest paquet ALIVE no és correcte, s'informa al client i es passa el client a l'estat DISCONNECTED.

A més a més, quan es rep un paquet ALIVE es posa en marxa un altre comptador mitjançant la funció *time.monotonic* per saber en quin punt s'ha rebut, de la mateixa forma que he

fet al client. D'aquesta forma, com tenim un bucle iterant sempre per saber si s'han rebut paquets per alguna de les comunicacions, afegim un procés encarregat de mirar si s'ha rebut el primer ALIVE dins del temps w comparant el temps actual amb el comptador iniciat quan el client passa l'estat REGISTERED i, també, si el client es troba en l'estat SEND_ALIVE i la diferència entre el temps actual i el comptador és major de $v * 3$, voldrà dir que s'han deixat de rebre 3 paquets ALIVE consecutius, per la mateixa raó explicada en l'estratègia del client. Finalment es passa el client a l'estat DISCONNECTED.

2.3 Diagrama d'estats del protocol UDP durant el registre i el manteniment de la comunicació

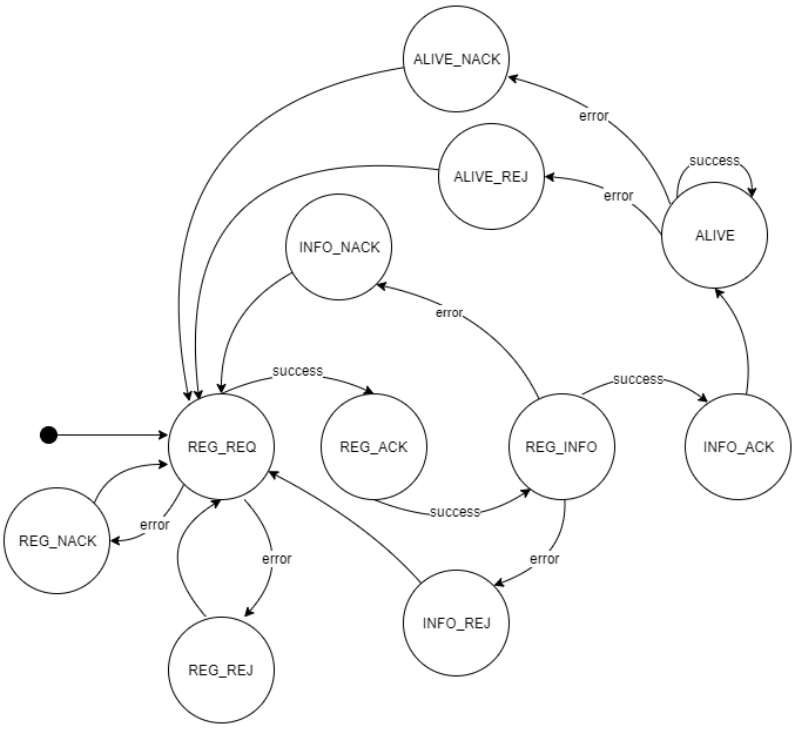

Figura 1: Diagrama d'estats del protocol UDP

3 Conclusions

Per finalitzar i com a conclusió, remarcar que la realització d'aquest treball crec que ha simulat correctament la situació client-servidor que es pot donar en la realitat, i a més a més, m'ha servit personalment per millorar el meu nivell de programació tant en C com en Python i conèixer els protocols i les comunicacions que s'utilitzen en la realitat.