Xarxes

Pràctica 1: Programació d'aplicacions de xarxa

Jordi Rafael Lazo Florensa

20 d'abril de 2020

Grau en Enginyeria Informàtica

Universitat de Lleida
Escola Politècnica Superior

Índex

1- Introducció

Aquest informe exposa el procediment emprat per a la resolució del problema plantejat en la assignatura de Xarxes de la Universitat de Lleida.

El principal objectiu es posar en pràctica es coneixements teòrics adquirits relacionats en la comunicació d'aplicacions en xarxes a través dels sockets així com dissenyar i programar un protocol de comunicacions entre un client-servidor.

La situació proposta consisteix en que un servidor rep i emmagatzema la informació de diferents clients (dispositius encarregats de realitzar les mesures de diferents punts de fabricació).

Per a la implementació d'aquesta solució s'ha utilitzat els tipus de comunicació ensenyats durant les classes de teoria. En la fase del registre i manteniment de la comunicació s'ha utilitzat el protocol UDP i pel que fa al enviament de configuració el protocol TCP.

La implementació d'aquesta pràctica presentada no contempla la fase d'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.

Finalment, es descriuran els motius que han portat a la implementació final, juntament amb els problemes que s'han presentat durant el desenvolupament dels programes en cadascuna de les seves parts; tant els motius causants com les accions empreses per a la seva solució.

2- Estructura client-servidor

2.1- Client

Per a la realització del client el primer pas consisteix en la lectura de fitxer de l'entrada. Posteriorment s'ha procedit a emmagatzemar les dades en un diccionari clau, valor. Es crea el socket UDP, pel qual s'enviarà els paquets de registre i es començarà el procés de registre. Una vegada establert el registre, es redirigeix al port establert per el servidor per on s'envien els paquets REG_INFO i es rebran les respostes a aquests. En cas de no rebre el paquet correcte es procedirà a tornar a començar el procés de registre. Aquesta situació es repetirà en cas de no rebre correctament els paquets INFO_ACK. Una vegada rebut aquest paquet el client passarà a estat registrat i començarà el procés d'enviament dels paquets ALIVE. L'estratègia emprada per el manteniment de la comunicació del client al servidor consisteix en l'ús de dos *threads* utilitzats mitjançant la llibreria *threading* un dels quals s'encarrega de controlar el *timing* ja que cada 2 segons s'encarrega d'enviar un paquet ALIVE mentre que l'altre va revisant si s'ha d'enviar o no. En cas que es perdi la comunicació amb el servidor serà el segon *thread* el que s'encarregarà de notificar-ho a l'usuari, modificar l'estat i informar la resta de processos sobre aquesta situació i començar un nou procés de registre. Paral·lelament el procés pare es mantindrà a l'espera de la introducció de comandes, en aquesta ocasió sol s'han pogut implementar les següents comandes locals: *stat*, *quit*, *set*. Tot i així, els canvis realitzats per aquest última comanda no es veu reflectida a l'hora de executar un últim *stat*.
Per a solucionar els problemes de sincronització de processos s'ha fet ús d'una variable la qual s'estableix un valor *bool*ean quan els altres processos s'hagin de bloquejar.

2.2- Servidor

Per a la realització del servidor el primer pas consisteix en llegir els paràmetres d'entrada i establir correctament el fitxers de configuració. A continuació s'ha procedit a la lectura del fitxers de configuració i en emmagatzemar en variables globals aquestes dades com el id, port i dades.
El pròxim pas ha estat la creació de un *thread* per gestionar la recepció dels paquets de registre i un altre *thread* per gestionar la recepció dels paquets ALIVE (seguint el esquema proposat en el client).
Mentrestant, el procés pare es manté a la espera de rebre alguna comanda introduïda.

S'ha definit una estructura per emmagatzemar totes les dades dels clients del servidor i la seva modificació s'ha aconseguit mitjançant a la utilització de punters. En aquesta estructura s'emmagatzemarà el estat de cada client, el seu ,id, dispositius assignats etc.

Pel que fa a la fase de registre, primer s'ha definit un *socket* UDP per el port del servidor i aquest es manté esperant a la rebuda d'algun paquet. Quan aquest ha rebut un paquet s'executarà un *thread* que tractarà el paquet. Així doncs, la funció de tractar paquet s'encarregarà de la gestió.

Primer es revisa quin dispositiu ha enviat el paquet, si el dispositiu esta desconnectat i el paquet rebut es un REG_REQ el estat passarà a ser WAIT_ACK_REG i establirà un nombre aleatori. Seguidament s'estableix un port UDP per a la fase d'intercanvi de informació. Al usuari se li enviarà el paquet REG_ACK amb la informació (número aleatori i el port UDP). A continuació el servidor esperarà un nou paquet per el *socket*. Si rep el paquet REG_INFO correctament aquest enviarà INFO_ACK i canviarà el estat del dispositiu a REGISTERED. Qualsevol problema que el servidor detecti durant aquest procés farà que enviï un paquet REG_NACK.

Finalment si el paquet original enviat era de tipus ALIVE actualitzarà el temps de recepció del últim ALIVE. En canvi, si el paque ALIVE es enviat a un dispositiu no conectat o desconegut es tornarà un paquet de tipus ALIVE_REJ.

3- Diagrames d'estats

3.1- Registrar-se en el servidor

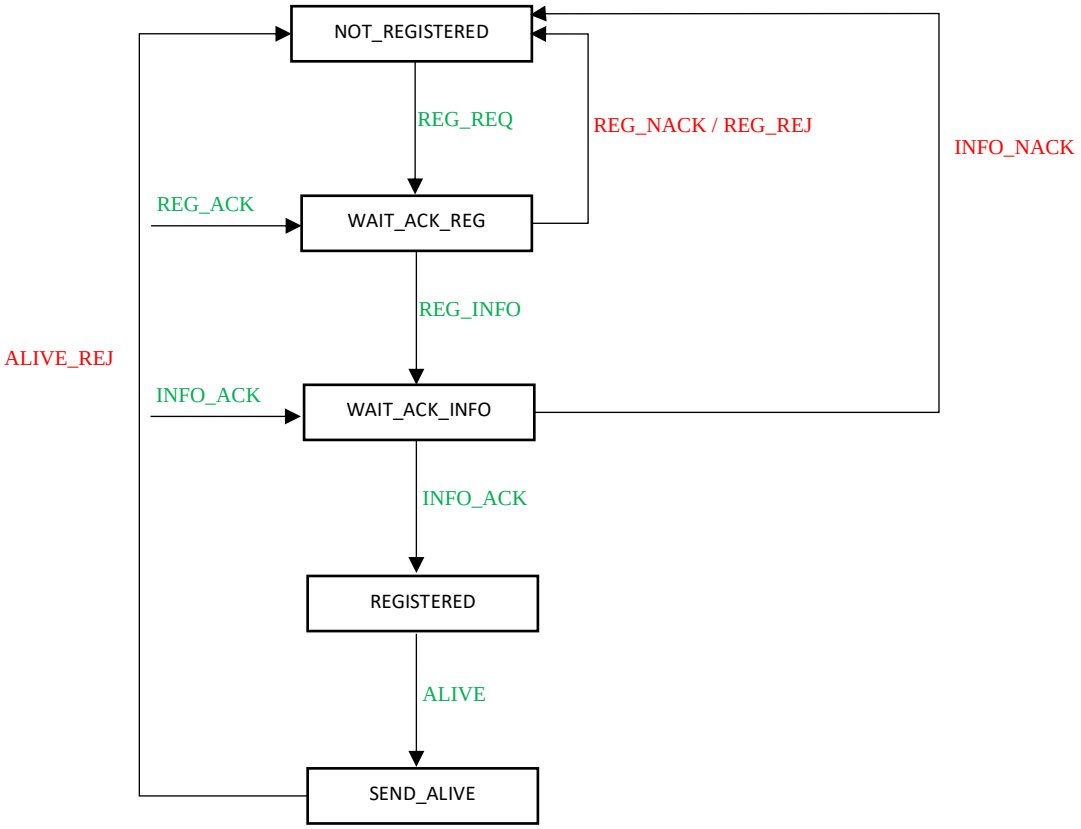

Figura 1

3.2- Mantenir comunicació periòdica amb el servidor

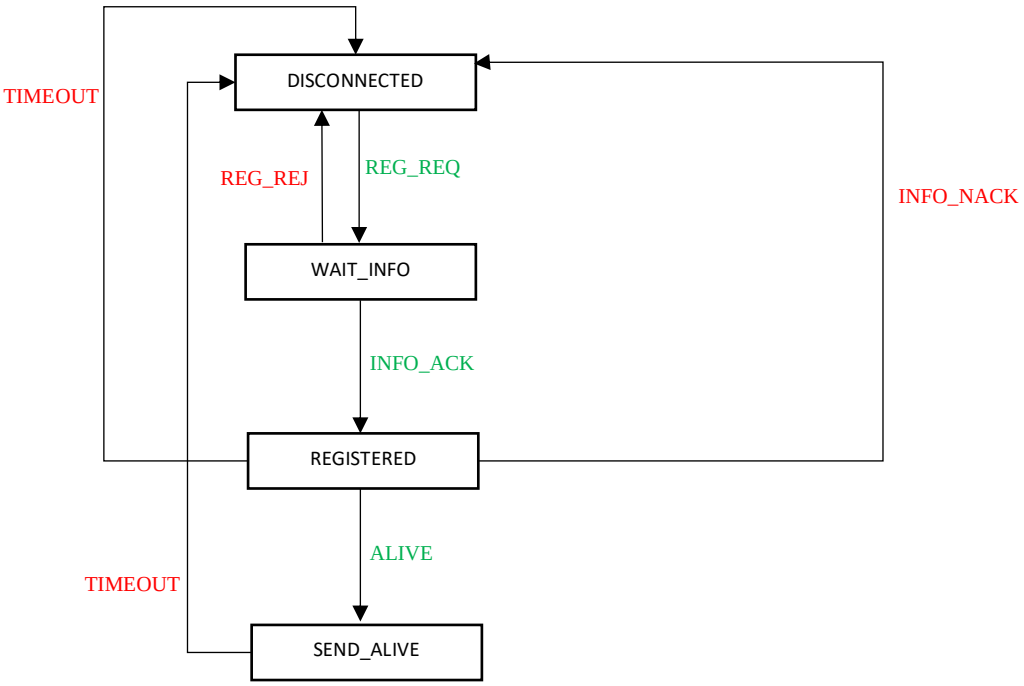

Figura 2

4- Conclusió

La realització d'aquesta pràctica ajuda a entendre la programació d'aplicacions xarxes, començant per la creació de *sockets* fins a rebre múltiples peticions o paquets i saber gestionar aquests amb els *threads*. La creació i gestió dels *threads* tan en C con en Python es essencial i clau per la organització dels paquets enviats i rebuts així com per entendre com funciona un client-servidor concurrent. Amb la implementació dels *sockets* i dels *threads* en la pràctica s'ha pogut observar que, per norma general, no hi ha cap complicació a implementar una comunicació mitjançant *sockets* entre dos llenguatges de programació tan diferents com C i Python.

A més a més, la utilització de la funció *select* ha sigut clau per facilitar les múltiples connexions al mateix temps ja que es mes eficient que escriure un bucle utilitzant els temps de espera del connector.

Cal remarcar que la utilització dels diagrames i la planificació prèvia a la pràctica han ajudat a comprendre el funcionament i simplificar el codi tenint clar des del principi quina seria la execució del programa.

Finalment, des del meu punt de vista durant el desenvolupament de la pràctica me he trobat en bastants problemes derivats de la falta de coneixements previs de programació tan en C com en Python, com l'existència dels *threads*. A nivell general, malgrat no haver aconseguit tots els objectius de la pràctica i tenint errors, hi ha certes parts del codi del client i del servidor que crec que han estat ben implementades o almenys estaven ben plantejades. No obstant això, em quedo només amb la part positiva de haver adquirits uns coneixements de programació totalment nous per a mi i el haver dedicat tantes hores a un projecte que pensava que no seria capaç entregar.