Pràctica 1 – Xarxes

Joel Aumedes Serrano – 48051307Y

GEI UdL – Curs 2019/2020

Índex

Com executar

Per a executar un client, es pot executar el fitxer *cl.py* amb els respectius paràmetres d'entrada, com per exemple `./cl.py -d -c client2.cfg`

Per a compilar el servidor, es pot utilitzar la comanda *make*. Aquesta comanda compila el servidor i l'executa sense cap paràmetre d'entrada. Per a executar el servidor entrant algun paràmetre es pot fer executant el fitxer *sr*, resultant de la compilació feta amb *make*. Una possible manera seria `./sr -d`

Client

Per l'implementació del client, he utilitzat *Python 3.7*, i he fet servir *threads* per a gestionar accions simultànies.

El client primer es registra al servidor. Un cop registrat, manté una comunicació periòdica amb el servidor mentres l'usuari pot escriure comandes per a executar. A part de les 4 comandes estàndard, l'usuari pot executar *"?"* per a mostrar una ajuda o *"debug"* per a activar o desactivar el mode debug.

El client segueix l'estructura del Diagrama 1.

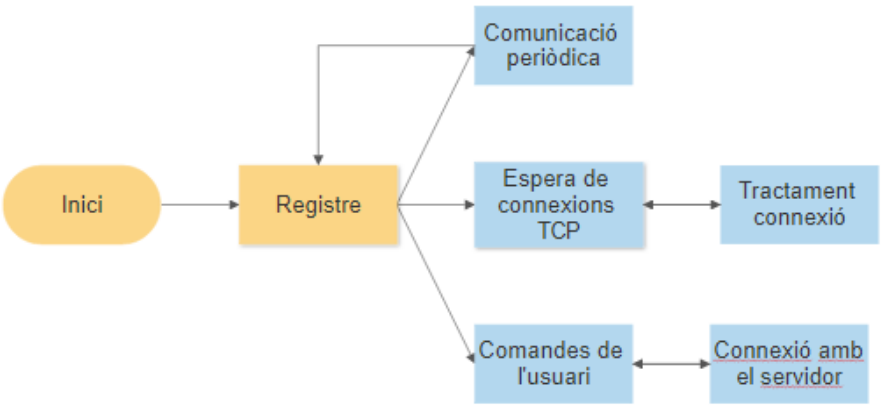
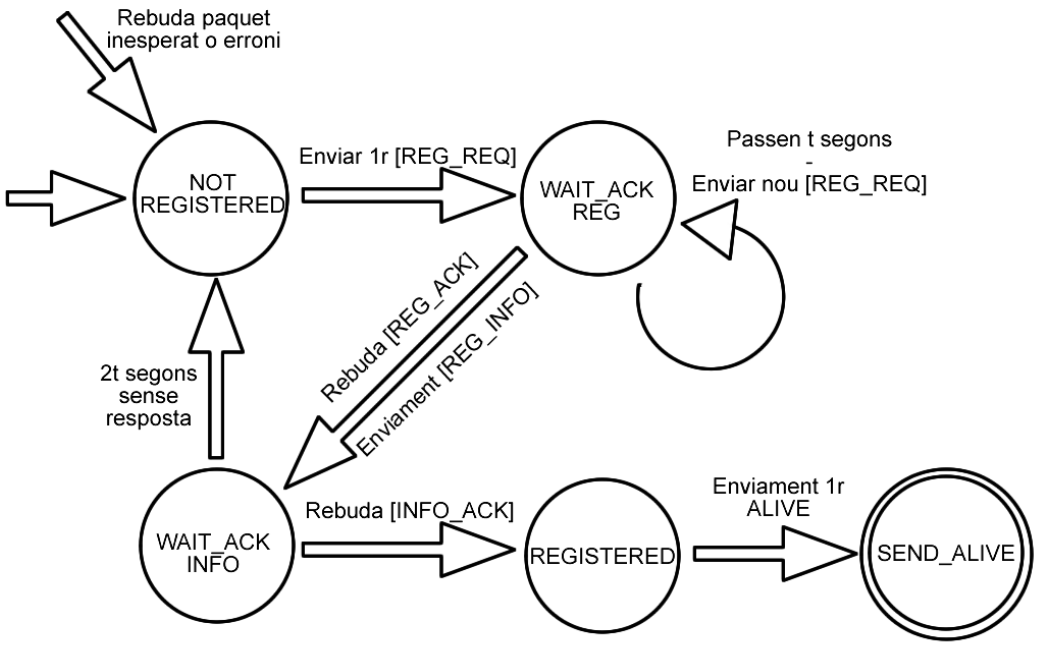


Diagrama 1 - Estructura del client

Registre al servidor

Durant el registre al servidor, el client realitza els canvis d'estat indicats per la Imatge 1.



Imatge 1 - Canvis d'estat del Client

Per a poder implementar-lo, he utilitzat 1 *thread* (a part del *thread* principal). Aquest altre *thread* conta el temps que ha de passar el client entre enviaments de *REG_REQ* si el servidor no contesta. El *thread* principal espera la resposta del servidor amb un *select*, i també conta el número de paquets enviats i intents de registre.

Quan hi ha una resposta del servidor, el *thread* extra es tanca i es realitza el procés de registre sencer si hi ha èxit. Si no n'hi ha, es torna a engegar el *thread* i es reprèn l'enviament de paquets *REG_REQ*.

Un cop finalitzat el registre, el client tanca el *thread* extra i inicia el procés de comunicació periòdica.

Comunicació periòdica amb el servidor

Per a la comunicació periòdica he utilitzat 2 *threads*, sense contar el *thread* principal, ja que aquest estarà ocupat executant comandes locals i connexions TCP amb el servidor.

Per aquesta etapa, un dels *threads* gestionarà l'enviament i rebuda de paquets, mentres l'altre modifica una variable cada 2 segons, per indicar al primer *thread* que ha d'enviar un paquet. Això ho he fet així ja que, si el primer *thread* també espera els 2

segons, durant aquests 2 no pot estar preparat per rebre un paquet que podria enviar el servidor.

Aquests 2 *threads* acabaran quan acabi el programa principal o, si es perd comunicació amb el servidor, acabaran per a poder reiniciar-se després, i reprendre l'enviament i rebuda de paquets *ALIVE*.

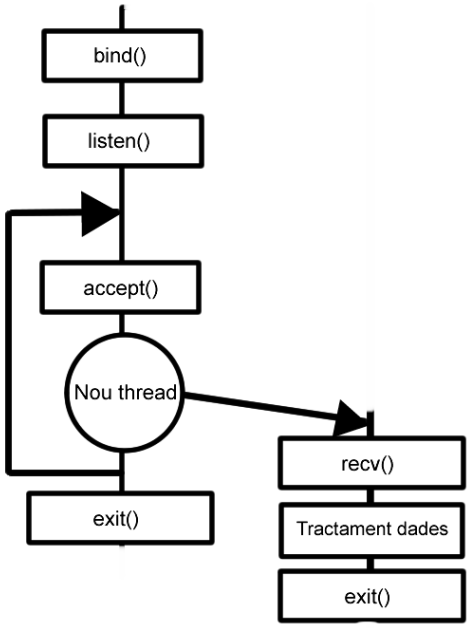
Enviament d'informació al servidor

L'enviament d'informació al servidor l'he executat al *thread* principal, i s'executa just després de que l'usuari entri la comanda, de manera que l'usuari no pot escriure més comandes fins que l'enviament acabi, sigui exitosament o en fallada.

Per a aquest enviament, el client simplement crea un *socket* TCP i es connecta al servidor. Un cop connectat envia el paquet, espera una resposta i tanca la connexió.

Atendre peticions d'informació del servidor

Per a atendre peticions d'informació del servidor sí que utilitzo un altre *thread* diferent al principal. Aquest thread fa un *bind* del *socket* TCP i es queda a l'espera de connexions, tal com es mostra a la Imatge 2.



Imatge 2 - Connexions TCP al Client

Quan en rep una, crea un altre *thread* que serà l'encarregat de rebre el paquet i contestar amb el paquet corresponent, i també de modificar les variables del client si el paquet és un *SEND_DATA*.

Servidor

La implementació del servidor l'he fet en C, i, igual que amb el client, he gestionat les accions simultànies amb *threads*.

El servidor té 3 *threads* a més del principal per a poder gestionar els clients. Un *thread* atén els registres, un altre atén l'enviament d'ALIVE i un altre atén les connexions TCP. El *thread* principal executarà les instruccions que escriu el client.

El servidor funciona amb l'estructura mostrada al Diagrama 2.

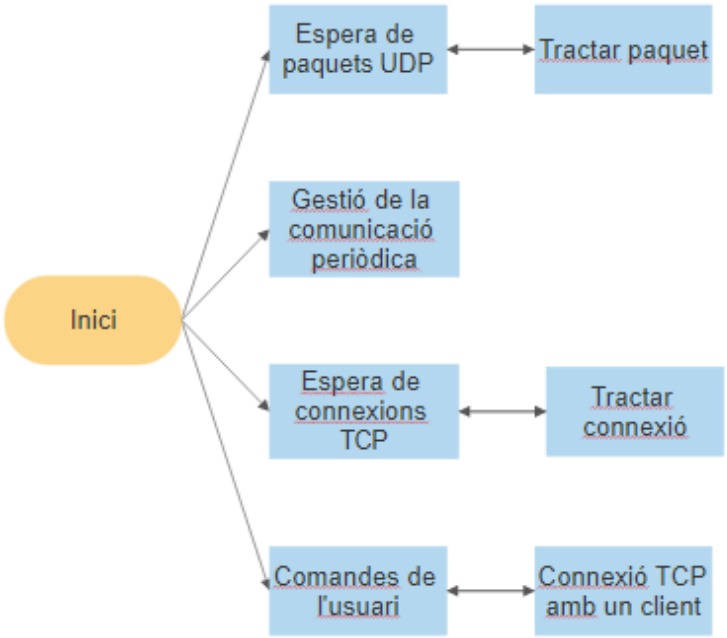


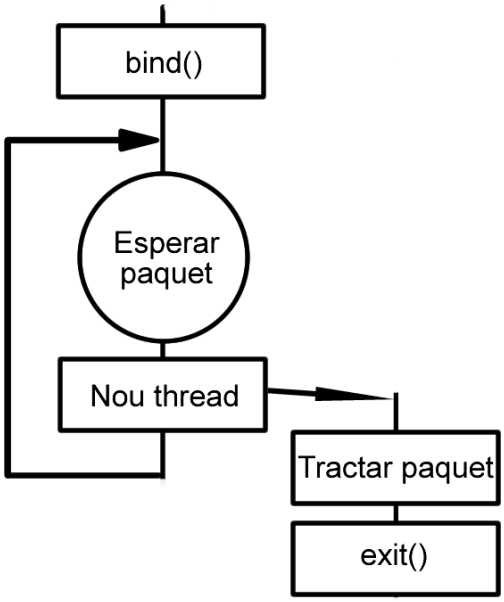
Diagrama 2 - Estructura del Servidor

Tal com amb el client, el servidor accepta les 4 comandes establertes, juntament amb *"?"* per mostrar una ajuda i *"debug"* per activar o desactivar el mode debug.

Atendre peticions de registre

Per a atendre peticions de registre, utilitzo 2 *threads*. Un *thread* està sempre atent a la rebuda de paquets pel port UDP, de manera que cada cop que en rep un crea un altre *thread* que tractarà el paquet rebut, de forma que el servidor pot atendre un nou client sense acabar d'atendre'n un altre (servidor concurrent).

El funcionament del registre al servidor es mostra a la Imatge 3.



Imatge 3 - Rebuda de paquets UDP al Servidor

El segon *thread* tracta el paquet rebut. Si és un *REG_REQ*, aquest mateix *thread* s'encarregarà de tot el registre, ja sigui exitós o hi hagi algun error. Si el paquet és un *ALIVE*, el servidor el comprova i respon amb un *ALIVE* o un *ALIVE_REJ* segons correspongui.

Gestionar la comunicació periòdica amb els clients

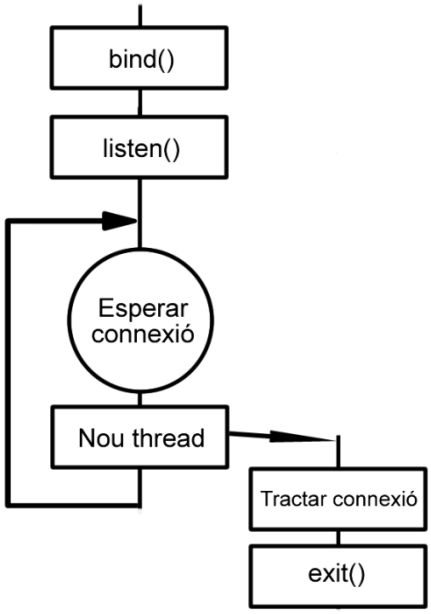
Per a gestionar la comunicació periòdica amb els clients utilitzo un altre *thread* diferent. Aquest *thread* contabilitza els *ALIVE* rebuts dels clients, i, segons si el client n'ha enviat o no, el servidor n'envia.

Això és per no saturar el client de paquets *ALIVE*, ja que quan es rep un *ALIVE* del client ja s'hi contesta des del mateix *thread* que l'ha rebut.

Gestionar informació rebuda dels clients

Un tercer thread s'encarrega de rebre connexions TCP dels clients de la mateixa manera que s'atenen paquets UDP.

Aquest *thread* espera connexions TCP i, quan en rep una, crea un altre *thread* que s'encarregarà de l'intercanvi de paquets, seguint un altre cop l'arquitectura de servidor concurrent que es mostra a la Imatge 4.



Imatge 4 - Rebuda de connexions TCP al Servidor

Enviar informació i peticions d'informació als clients

Les peticions o enviaments d'informació als clients s'executen des del thread principal, ja que s'executen segons les comandes que escriu l'usuari.

Això també ens permet bloquejar l'entrada de comandes de l'usuari fins que aquest sàpiga si aquest enviament o petició ha estat exitós o no.