Pràctica 1:

Programació d'aplicacions de xarxa

Abril 2022

Marc Gaspà Joval

Curs 2021-22

17 abril 2022

Índex

Introducció

Aquesta practica esta orientada al desenvolupament d'un protocol de comunicació per al control i supervisió del maquinari de fabricació tot emprant *sockets* i el model client-servidor. Aquest ha de ser capaç de comunicar varis clients amb un servidor i emmagatzemar dades dels sensors o enviar dades als actuadors, tot de de forma simulada a traves de comandes en la terminal.

La practica esta dividida en dues parts, una es desenvolupament del client i l'altra la del servidor. Totes dues parts han de ser capaces de dur a terme un procés de registre, mantenir una comunicació periòdica i permetre l'enviament de dades a traves del protocol TCP (*Transmision Control Protocol)*. Ambdues estan explicades en detall i per separat a continuació.

Client

El client esta dividit en diferents mòduls, on cada mòdul es un '.c' i un '.h', d'aquesta manera es mes fàcil modificar i organitzar les diferents parts de la practica. A continuació explicaré els mòduls mes importants per ordre.

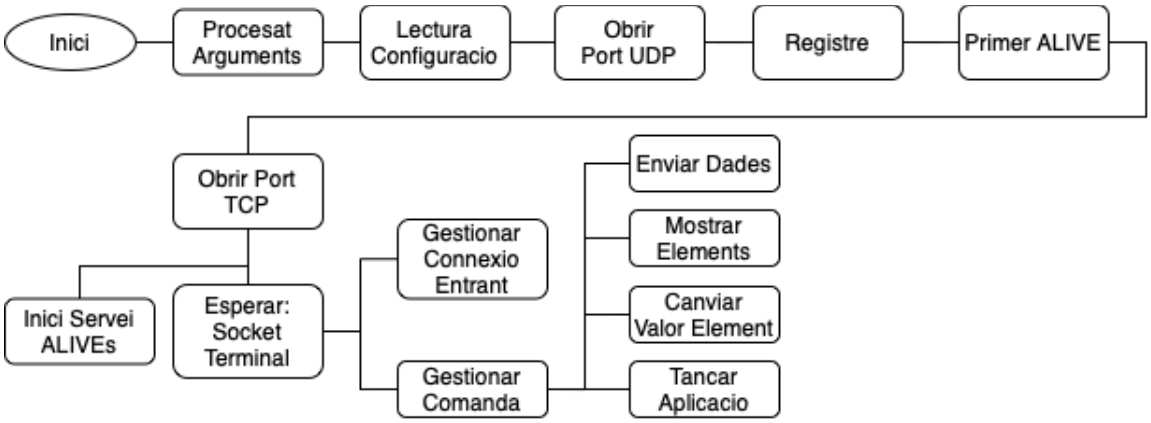

Figura 1 Diagrama de Blocs del Client

Client (*main*)

Aquest es l'arxiu principal del client i conte el mètode *main* i un mètode per a processar els arguments. En el main es crida aquest mètode per a processar els arguments, seguit es crida al mètode encarregat de carregar la configuració i s'obre el *socket* UDP (*User Datagram Protocol*) per a tal de començar amb el registre.

A continuació entrem en un bucle que mira constantment el estat, en el cas de estar NOT_REGISTERED aquest inicia el procés de registre i un cop completat satisfactòriament envia el primer ALIVE, engega el procés que manté la comunicació periòdica i obre el *socket* TCP per l'enviament de dades.

Un cop el estat es SEND_ALIVE el procés es queda bloquejat a la espera de o be una entrada del usuari per terminal o be una connexió entrant per el *socket* TCP. En cada cas gestiona la entrada, i que serà explicat en el seu mòdul corresponent.

Configuració

Aquest es el primer mòdul que entra en acció dins del client, i que conte tots els mètodes relacionats amb la configuració del servidor, com un mètode per carregar-la, per imprimir-la per terminal entre altres mètodes auxiliars.

Per carregar la configuració es van considerar dues maneres. La primera seria 'tokenitzant' cada línia del arxiu i tractar cada *token* per separat. La segona es utilitzant el *scanf* i fent que es salti els caràcters no desitjats posant-los dins del format.

La configuració es carrega dins una estructura que a part també contindrà els valors que es guardaran durant el procés de registre, com l'adreça, els ports...

Registre

El mòdul del registre conte dos mètodes, el primer es el que es crida quan es vol registrar al client, aquest serà el encarregat de comptar el nombre de processos de registre i continuar o acabar amb el programa. Tot això ho fa cridant a un segon mètode que seria un procés de registre, aquest porta a terme l'enviament i recepció de paquets a part del control dels estats. En aquest cas, si un procés de registre es satisfactori el mètode retorna el nombre 1, i en cas contrari el -1.

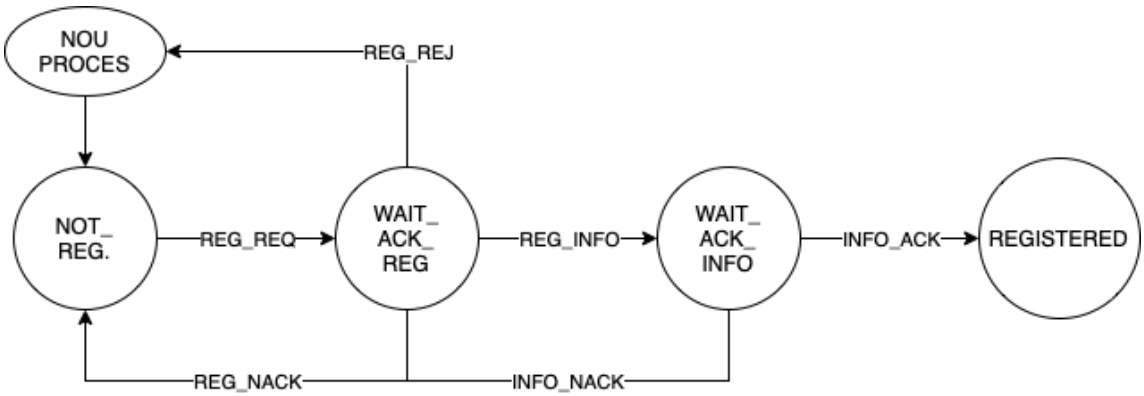

Figura 2 Taula d'Estats del Registre

El diagrama mostrat en la *Fig. 2* es el corresponent al procés de registre, aquest conte tots els estats per els que passa el client fins a arribar al estat de registrat, cada fletxa mostra un paquet rebut o enviat que causa el canvi d'estat i encara que no esta en el diagrama, en el cas de que es rebés un paquet amb dades errònies sempre es torna al estat NOT_REGISTERED.

Manteniment de comunicació periòdica

Un cop el client esta en REGISTERED encara no pot enviar ni rebre fins que no es trobi en l'estat SEND_ALIVE.

Per a arribar-hi es fa anar un dels mètodes del mòdul, que es encarregat de enviar un ALIVE i tractar la resposta del servidor. Si el ALIVE rebut en resposta es correcte, el mètode retorna 0, en cas de que hagi passat un temps preestablert sense rebre resposta retornarà -1 i en cas de que es rebi amb informació errònia o un ALIVE_REJ es retornarà -2.

En aquest cas si el ALIVE rebut es correcte el client passarà d'estat i obrirà el port TCP i engegarà el servei encarregat de mantenir la comunicació periòdica que es troba en aquest mòdul.

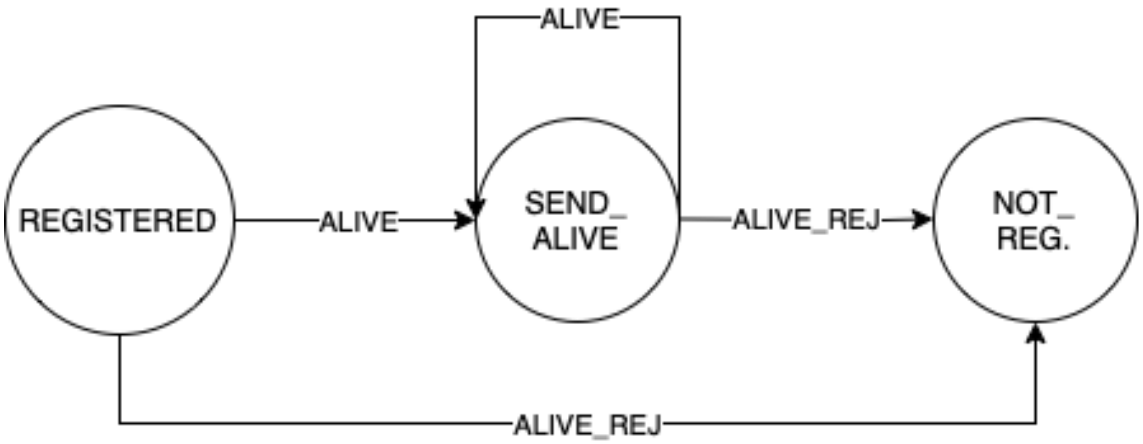

Figura 3 Diagrama d'Estats Manteniment Comunicació Periòdica

En la Fig. 2 podem observar com afecten els diferents paquets en l'estat. En aquest cas, hi ha altres condicions que el poden afectar, com per exemple rebre un ALIVE amb dades errònies o simplement no rebre un nombre predeterminat de ALIVEs, i que l'enviarien a l'estat NOT_REGISTERED.

Enviar i rebre dades del servidor

En aquest cas ens trobem dins del mòdul encarregat de gestionar l'enviament i la recepció de dades.

Enviar dades

En el cas de l'enviament de dades ens trobem un mètode amb el propòsit d'enviar un element, ja que es l'únic que es pot enviar des de el client. Aquest obre un nou *socket* TCP amb un port decidit pel sistema operatiu, i es connecta amb el servidor per el port guardat en el procés de registre.

Un cop ha establert aquesta connexió envia un paquet SEND_DATA amb la informació del element i el temps de la mostra i espera a rebre resposta. En al cas de rebre un paquet amb informació errònia o be un DATA_REJ el client pesarà a l'estat DISCONECTED. Si aquest obté un DATA_NACK ho comunicarà per pantalla, i en cas de rebre un DATA_ACK es considerarà que l'enviament ha estat un èxit.

Rebre dades

En aquest cas, un cop rebem una connexió entrant per el *socket* TCP obert anteriorment l'acceptem i llegim per el nou *socket*.

El servidor ens pot enviar dos tipus de paquets, un es el *SET_DATA* que hauria de canviar el valor d'un element de tipus actuador, i l'altre seria el *GET_DATA*, en el qual hauríem de enviar el valor del element demanat.

En tot cas, si la operació s'ha efectuat correctament enviarem un DATA_ACK, i si hi ha algun error en el tipus d'element per exemple, un DATA_NACK.
Si hi hagués algun error en els camps de identificació del paquet enviaríem un DATA_REJ i passaríem el client a l'estat NOT_REGISTERED.

Connexions

El mòdul connexions conte mètodes auxiliars per a validar paquets i per a comparar adreces. El mètode mes important que conte es el que servirà per substituir el SIGUSR1, que es farà anar per a canviar l'estat del client a desconnectat des de qualsevol dels processos fills.

Terminal

Aquest mòdul es important ja que conte el mètode encarregat de processar les comandes entrades per el terminal, en aquest cas a diferencia de quan es carreguen les configuracions es 'tokenitza' l'entrada.

A mes a mes, també hi ha els mètodes encarregats de imprimir per pantalla. El primer d'ells es una superposició al *printf* que simplement afegeix la hora minut i segon. Els altres mètodes simplement afegeixen un MESSAGE, DEBUG, ERROR abans del text en si. Per a mantenir la similitud amb el *printf* original i els seus múltiples arguments s'ha fet anar les *va_list* que s'utilitza en l'original.

Socket
Per últim tenim el mòdul de *socket* que conte tots els mètodes necessaris per a crear *sockets* tant UDP o TCP amb nomes un port.

Cal destacar que tots dos tipus de *socket* es creen canviant la opció SO_REUSEADDR que permet reutilitzar-los en cas de que es quedin oberts, ja que a vegades, si es tanca el programa sense la comanda *quit*, i es torna a obrir al moment, no dona temps a que el *timeout* dels sockets expiri.

Per últim hi ha un mètode que genera sockaddr_in a partir de un port i una adreça en forma de cadena de caràcters.

Servidor

El servidor ha estat programat amb *python 3.9* i igual que el client aquest ha estat dividit en diferent mòduls, un '.py' per a cadascun d'aquests i per a tal de portar una millor organització del projecte.

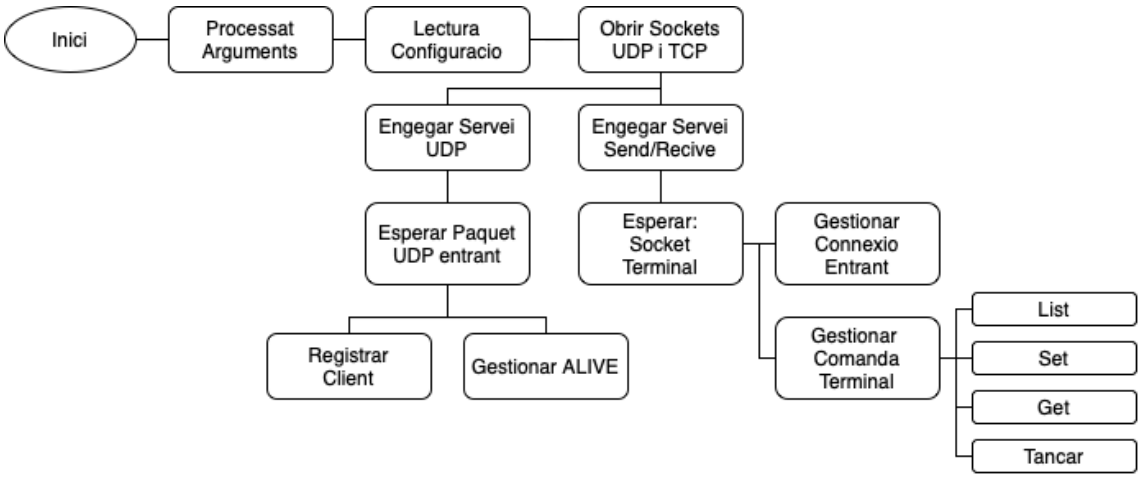

Figura 4 Diagrama de Blocs del Servidor

Servidor (*main*)

Igual que en el client es l'arxiu principal del servidor, i de mateixa manera conte un mètode encarregat de processar els arguments. Seguidament, es carrega la configuració del servidor i la llista de dispositius autoritzats, s'obren dos *sockets* un UDP i un TCP.

Per acabar, com que a diferencia del client s'han d'atendre peticions de múltiples clients s'inicien dos serveis (processos), un encarregat de gestionar les connexions per UDP, i un altre per atendre les connexions TCP i les entrades per terminal.

Connexions

Aquest mòdul conte el servei encarregat de gestionar les connexions UDP entrants. D'un principi inicia el servei dels ALIVE del que en parlarem mes endavant, i es queda bloquejat a la espera de rebre un paquet.

Un cop rebut, primer mira si ve d'un dispositiu autoritzat, en cas que si, si es tracta d'un paquet de registre, comprova les dades d'aquest i es guarda la ip i el port. Seguit inicia un procés de registre en un nou procés (per tal de poder atendre múltiples clients) i del que parlarem en el seu apartat corresponent.

En canvi si es tracta de un ALIVE, mira que el dispositiu que l'ha enviat estigui en els estats corresponents i comprova que contingui les dades correctes i crida a una funció del mòdul corresponent.

Registre

El procés de registre en el servidor es du a terme en processos individuals per a cada procés per a tal de permetre múltiples registres alhora.

En aquest cas un cop es crea aquest fill el primer que fa es obrir un nou *socket* UDP per on continuarà el registre i qual port s'enviarà al client en el REG_ACK.
A continuació, s'espera rebre un REG_INFO, i si aquest es correcte s'enviarà un INFO_ACK, i així seguint el diagrama d'estats de la *Fig. 2* fins que el client estigui registrat.

Igual que en el client si les dades d'identificació son incorrectes es cancel·larà el procés de registre, i el canviaria d'estat però en el de DISCONECTED.

Manteniment de comunicació periòdica

El manteniment de la comunicació periòdica es complica una mica en el servidor ja que s'ha de mantenir entre tots els possibles clients.

Per fer-ho possible s'ha fet amb un servei que contínuament, en bucle, esta mirant quant de temps ha passat des de l'últim ALIVE per als clients que estan en el estat SEND_ALIVE o REGISTERED.

En cas de que es hagi passat el suficient temps i falti un ALIVE es mira si el client esta en REGISTERED, ja que llavors es passaria el client a DISCONECTED a causa que faltaria aquest primer ALIVE. Si en comptes es troba en l'estat SEND_ALIVE s'incrementaria el comptador de ALIVES no rebuts i si supera el límit es consideraria que s'ha perdut la comunicació i es passaria el client al estat DISCONECTED

Per altra banda, quan es rebi un ALIVE, com hem comentat abans, el servei UDP el processarà i en el cas de que sigui correcte farà anar un mètode d'aquest mòdul per a processar-lo. En tot cas el comptador de paquets no rebuts tornarà a 0 i el temps del últim passarà a ser el actual, i a mes en cas de que el dispositiu estes en REGISTERED, l'acabaria de posar en SEND_ALIVE

Enviar i rebre dades dels clients

En aquest cas tenim un servei/procés encarregat de esperar o be connexions entrants per el *socket* TCP o be una entrada de l'usuari per terminal, en cas de que sigui una connexió entrant, es processa utilitzant el mètode trobat dins del mateix mòdul, per altra banda si es tracta d'una entrada per terminal per part del usuari es processarà aquesta comanda i en cas de que sigui un '*get*' o '*set*' es cridaria a la funció encarregada de fer el procés d'enviament.

Sigui que un client ha enviat dades o, des de el servidor haguem fet anar una de les dues comandes, si s'ha completat la operació amb èxit es guardarà en un arxiu '.data' únic al dispositiu, l'element relacionat, el seu valor, el paquet que ha causat l'event i la data i hora. El arxiu sobra sempre en mode *append* per tal de que no s'esborri el contingut del arxiu donat un reinici del servidor.

S'ha de comentar que no es crearà cap subproces ja que el servei anirà iterant sobre les diferents peticions de un amb un per a tal de evitar conflictes.

Sockets

El mòdul de sockets conte les classes corresponents a cada tipus de paquet/trama. Una instancia d'aquestes pot ser creada a partir de un *buffer* de *bytes* obtingut en rebre d'un *socket* o be a partir de cada un de les variables que el formen en el constructor.

Cal destacar que per a realitzar la conversió s'ha separat cadascun dels valors amb un mòdul de *python* anomenat *struct* i que permet donat un format transformar un *buffer* en primitives, i viceversa. En tots dos casos s'ha de codificar o descodificar aquestes en el format corresponent.

Per altra banda també hi han els mètodes que permeten llegir o enviar d'un *socket*, però que en comtes de treballar amb un *buffer* de *bytes* treballen directament en les classes esmentades anteriorment. I que també imprimeixen la trama un cop rebuda o enviada.

Per acabar també podem trobar les funcions encarregades de crear i configurar els *sockets*. Igual que en el client, als *sockets* se'ls activa la opció per a ser reutilitzats per a tal de no haver de esperar al *timeout* en cas de no tancar el servidor amb el *quit*.

Conclusions

La pràctica ha estat molt interesant, es la primera vegada que es treballa en un projecte tan ampli, i aquest ha ajudat a entendre com pot ser una aplicació de xarxa i es clar, a entendre com funcionen els *sockets* en els dos llenguatges, els protocols de comunicació i el model client-servidor.

Per altra banda la practica també ha estat molt útil per a entendre i agafar agilitat en els dos llenguatges utilitzats ja que per exemple en c he après a fer anar els headers, millorat amb els punters, com fer funcions que permetin múltiples paràmetres (*prints* mòdul terminal) entre altres, i en python el crear nous processos, passar de *structs* a primitius de python, fer anar alguns dels mètodes '__algo__' de les classes, fer anar *locks* (per una problema amb els prints), etc.
I no nomes aprendré coses noves sinó també ha estat una oportunitat per a practicar alguns dels coneixements prèviament apresos en altres assignatures com la programació orientada a objectes, estructures de dades, processos, ...

Apèndix

Taula d'acrònims

TCP. *Transfer Control Protocol*

UDP. *User Datagram Protocol*

Taula de figures