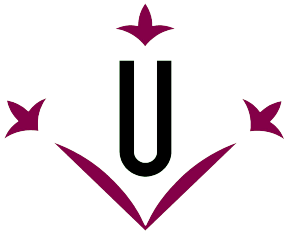

Universitat de Lleida
Escola Politècnica Superior

XARXES

Practica 1: Programació d'aplicacions de xarxa

Aaron Arenas Tomas

17 de Abril 2022

Grau en Enginyeria Informàtica

Índex

Índex de figures

1 Ejecución código

Per a executar el servidor, es pot executar el fitxer amb els respectius paràmetres python servidor.py {-d} {-c software_config_file } { -u allowed_devices_file } Per a compilar el client, es pot utilitzar la comanda make all. Aquesta comanda compila el cliente. Per a executar el Cliente entrant algun paràmetre es pot fer executant el fitxer cl, resultant de la compilació feta amb make. Una possible manera seria ./cl {-d} {-c cfg_file }

2 Cliente

2.1 Estructura Cliente

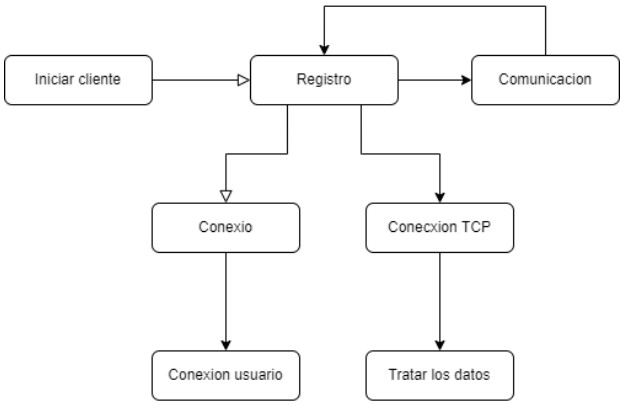

Figura 1: Esquema estructura cliente

2.2 Estados Cliente

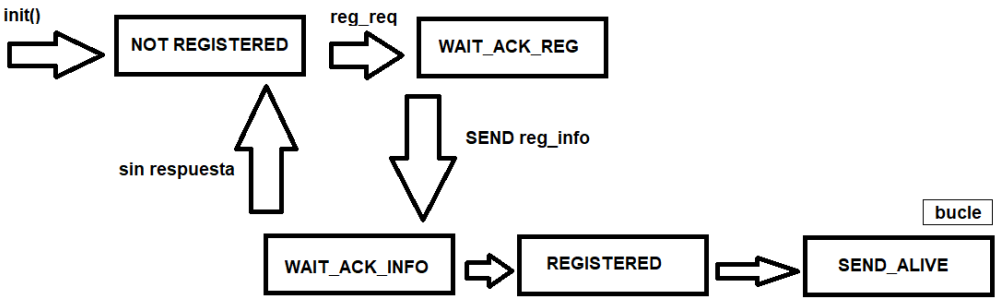

Figura 2: Esquema Estados cliente

Cliente hecho en (C). El cliente lo he realizado con threads para la gestión de cominica-

ciones simultaneas. El client primero lo he utilizao para registralo al servidor. Una vez registrado el usuario mantendra una comunicación periodica con el servidor a partir de la función realizada para la gestion de paquetes ALIVE. Aparte el usuario puede escribir comandes aun estando en comunicacion. estan implementades les 4 per defecte, en caso de poner una incorrecta se le mostrara una ayuda al usuario con las distintas comandas. (si ponemos "help"nos saldra una ayuda)

2.3 Registro al servidor

Para implementarlo, he utilizado 1 thread para cuando empezemos los envios al servidor podamos interactuar con el terminal. Tenemos una funcion principal que es service_loop() en la cual primero arrancaremos con la funcion registered()
que es la que se encarga de el proceso de registro, tiempo que ha de pasar el cliente entre envios de REG_REQ si el servidor no contesta. Esperaremos la respuesta del servidor con una funcion creada para realizar dichas esperas y no colapsar de peticiones periodicas, y tambien contaremos el numero de paquetes enviados y intentos de registro. (Los intentos de registro los veremos en el modo de debug).Quando hay una resposta del servidor se realiza el proceso de registro entero si hi hay exito. Si no, Realizaremos el envio de paquets REG_REQ. Una vez finalizado inicia el proceso de comunicación periodica.

2.4 Comunicación periodica con el servidor

Para esta comunicacion gestionaremos el envío y recibimiento de paquetes, mientras realizamos dicha comunicacion disponemos un metodo con una variable realizarlos cada 2 segundos, para indicar en que debe enviar un paquete.Si se pierde comunicación con el servidor, acabarán por poder reiniciarse después, y reanudar el envío y recibimiento de paquetes ALIVE. En caso de no recivir 3 alive consecurivos volveremos a empezar con la fase de registro ya que no tendremos comunicaión con el servidor

2.5 Enviamento de información con el servidor

No lo he podido realizar. pero explicare una posible solucion de como pienso que se podria resolver. .Para este envío, el cliente simplemente crea un socket TCP y se conecta al servidor. Una vez conectado envía el paquete, espera una respuesta y cierra la conexión. Lo ejecutaremos justamente despues de escribir la comanda deseada y esperaremos el envio del paquete se exitoso o fallido

2.6 Atender peticiones de informacion del servidor

Per a atendre peticions d'informació del servidor utilizaria un thread diferente al principal. Este realizara un bind del socket TCP i se quedara en espera de la conecxion

Quando recibe una peticion, creara otro thread que sera el encargado de recibir otro paquete y contestar con el paquete correpondiente y realizar las variables si el paquete es SEND_DATA.

3 Servidor

3.1 Estructura Servidor

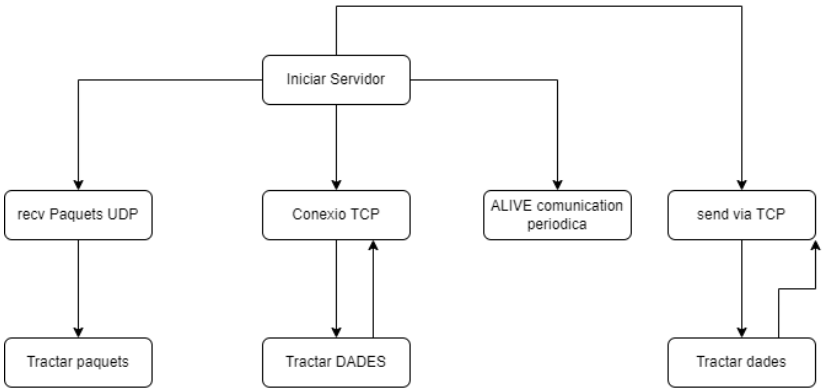

Figura 3: Esquema estructura cliente

3.2 UDP/TCP diagram

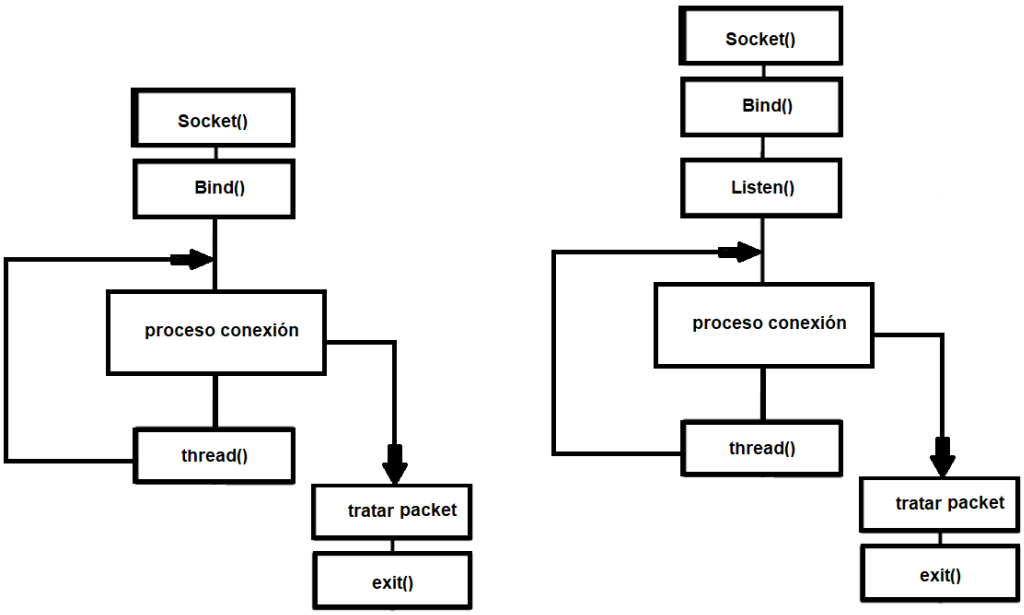

Figura 4: Esquema Paquetes recividos UDP/TCP

La implementación del servidor la he hecho en Python3, y, igual que conl client, he gestionado las acciones simultaneas con threads. El servidor tiene 2 threads sin contar

con el principal para poder gestionar los clients. Un thread atiende los registres, otro atiende las peticiones ALIVE. El thread principal ejecutara las instruccions que escribira el cliente.

3.3 Atender peticiones de registro

Para atender peticiones de registro, utilizo 1 threads. Un thread está siempre atento a los paquetes por el puerto UDP, por lo que cada vez que recibe uno crea un otro thread que tratará el paquete recibido, de forma que el servidor puede atender un nuevo cliente sin acabar de atender a otro para poder realizar dicho servidor concurrentemente El segon thread tracta el paquet rebut. Empezaremos con el proceso de comunicacion peridica y arrancaremos su thread y iniciaremos el tiempo de espera del primer paquete ALIVE

3.4 Gestión de comunicación periodica con los clientes

tenemos un thread que se encarga de si los paquetes alive no llegan a nuestro servidor desconectar nuestro cliente o si el cliente no empieza a perder paquetes y no recibe 3 ALIVE consecutivos se encargara de finalizar dicho registro, esto lo acemos para no tener una comunicación periodica con un cliente desconectado y no saturar dicho servidor

3.5 Gestiónar informacion recibida de los clientes

No realizado Posible solucion: crear un tercer thread se encarga de recibir conexiones TCP de los clientes de la misma modo que se atienden paquetes UDP. Este thread espera conexiones TCP y, cuando recibe una, crea otro thread que se encargará del intercambio de paquetes, siguiendo de nuevo la arquitectura de servidor concurrente

3.6 Enviar información i peticiones de información a los clientes

Podriamos realizarlo, Las peticiones o envíos de información a los clientes se ejecutarlas desde el thread principal, ya que se ejecutaria desde los pedidos del usuario del servidor.

4 Conclusión

La realización de dicha practica me a gustado, ya que he aprendido bastante de como funciona el desarollo de aplicaciónes con sockets y su funcionamiento