EPS
Escola Politècnica Superior
UdG

Projecte/Treball Fi de Carrera

Estudi: Eng. Tècn. Informàtica de Sistemes. Pla 2001

Títol: Implementació d'un sistema xat multiprotocol fent servir el llenguatge de programació ERLANG

Document: Còpia Completa P/TFC

Alumne: Walter Juan Pairó

Director/Tutor: Esteve del Acebo
Departament: Informàtica i Matemàtica Aplicada
Àrea: LSI

Convocatòria (mes/any): 09/2009

UNIVERSITAT DE GIRONA – ESCOLA POLITÈCNICA SUPERIOR

Programació amb Erlang

Xat Multiprotocol

Walter Juan Pairó
Enginyeria Técnica en Informàtica de Sistemes
Setembre 2009
Tutor: Esteve del Acebo

Índex

CAPÍTOL 1

INTRODUCCIÓ

Capítol 1: Introducció

El projecte realitzat ha consistit en l' implementació d'una aplicació de missatgeria instantània multiprotocol, concurrent i distribuït basat en el llenguatge de programació Erlang.

La part principal del projecta ha estat en la construcció del sistema de missatgeria en xarxa local. L'aplicació utilitza el protocol TCP/IP per comunicar-se i consta de dos parts, el servidor i el client.

El servidor es l'encarregat de gestionar els usuaris i les converses entre ells, per poder organitzar correctament totes aquestes dades s'ha utilitzat una petita base de dades anomenada MNESIA la qual la trobem en el mateix paquet Erlang.

Pel que fa el client consta d'una part gràfica per poder ser més amigable amb l'usuari, per poder-la realitzar s'ha utilitzat la llibreria GS, també en el mateix paquet Erlang. El client permetrà dos tipus de converses, unes amb grup i unes altres directament amb un altre client (un a un).

El client també s'ha dotat perquè es puguin realitzar converses amb un altre tipus de sistema que no sigui el local, el IRC.

Aquest document consisteix en diverses parts, la primera d'elles és una petita explicació sobre que es l'Erlang, una mica sobra la seva història, les peculiaritats i diferències que te amb els altres llenguatges i tot això acompanyat d'algun exemple.

El següent apartat consisteix en una explicació sobre la realització del projecta, com s'ha gestionat, que s'ha fet i objectius.

Després es centrarà en l'explicació sobre l'aplicació, aquí s'explicarà que és el que fa cada mòdul, com i perquè. Aquesta part consta dos grans apartats, el client i el servidor. Per cada apartat s'explicarà el disseny de l'aplicació, el seu funcionament i cada mòdul pel qual esta format. El següent ens explicarà amb profunditat el client, i l'últim el servidor local. I en l'últim es veuran els resultats obtinguts del projecte al igual que algunes probes realitzades en l'aplicació.

Finalment hi ha els dos últims apartats, les conclusions obtingudes del projecta i la bibliografia en la qual s'ha obtingut la informació.

També apart del projecte hi ha es disposa d'un apèndix el qual és un manual sobre l'aplicació desenvolupada.

CAPÍTOL 2

LLENGUATGE ERLANG

Capítol 2: Llenguatge Erlang

En aquest apartat inicial es donarà a conèixer una mica més aquest llenguatge de programació.

S'explicarà una mica la seva historia i també, s'ha fet una secció on es veuen les característiques que el fan diferent als altres llenguatges de programació. Es veuran algunes de les diferencies amb altres llenguatges més estàndards (com el C o Java) i també algun exemple més gràfic perquè quedi una mica més clar.

2.1.Una mica d'història

L'Erlang és un llenguatge de programació desenvolupat per Ericsson es va posar al marcat l'any 1986 i a partir de llavors es van començar a programar les anomenades Erlant/OTP, bàsicament són totes les seves llibreries.

Es troba un bon resum de la seva historia a una de les enciclopèdies digitals més conegudes, la *wikipedia* la qual ens diu:

> *Erlang és un llenguatge de programació concurrent i un sistema d'execució que inclou una maquina virtual i llibreries.*
> *El subconjunt de programació seqüencial d'Erlang és un llenguatge funcional, amb avaluació estricta, assignació única. Va ser dissenyat per la companyia Ericsson per realitzar aplicacions distribuïdes, tolerants a errors, execució en temps real i de funcionament ininterromput.*
> *Proporciona un canvi de codi en calent de forma que aquest es pot canviar sense para l'execució del mateix.*
> *Originalment Erlang era un llenguatge propietari d'Ericsson, però va ser cedit com a open-source l'any 1998.*
> *Erlang rep el nom de A. K. Erlang, a vagades es pensa que el nom és una abreviació de ERicson LANGuage, degut al seu ús intensiu a Ericsson.*
>
> - *Wikipedia, mes d'agost de l'any 2009*

Actualment l'Erlang és utilitzat per l'empresa Ericsson sobretot pels productes destinats a connexions en xarxa i aplicacions destinades a Internet, un exemple podria ser el *ejabberd* un servidor Javer (protocol de missatgeria) totalment desenvolupat amb Erlang.

També al mercat trobem algunes companyies que l'utilitzen com podrien ser:

- Bluetail/Alteon/Nortel (distribució, sistema mail tolerant a falles, accelerador SSL)

- Cellpoint (Localització basada en serveis mòbils)
- Facebook (Facebook chat backend) .....

2.2.Programar amb Erlang

Erlang és un llenguatge de programació que conta amb moltes funciones més comunament associades amb un sistema operatiu que amb un llenguatge de programació: els processos concurrents, la gestió de memòria, la distribució, les xarxes, etc...

La primera *open-source* (codi lliure) d'Erlang la qual conte la mateixa aplicació i gran part del *middlewere* d'Ericsson per la construcció de sistemes distribuïts d'alta disponibilitat.

En aquest llenguatge de programació compleix les següents característiques:

Concurrència: l'Erlang esta format per processos extremadament lleugers, la memòria requerida del qual pot variar dinàmicament. Els processos no tenen memòria compartida i la comunicació de missatges entre si és asíncrona. Suporta aplicacions amb un llarg nombre de processos concurrents els quals no fa falta que estiguin en el sistema operatiu amfitrió.

Distribució: L'Erlang està dissenyat per ser executat en un entorn distribuït. Una maquina virtual Erlang crida un node Erlang, un sistema distribuït Erlang es una xarxa de nodes Erlang (normalment una per processador). Un node Erlang pot crear processos en paralel que corrin en altres nodes, els quals pot ser que estiguin en sistemes operatius diferents. Els processos que es troben en diferents nodes es comuniquen exactament igual que si estiguessin en el mateix node.

Robustesa: Erlang disposa de vaires primitives (BIF diminutiu de *buit-in fuction*) de detecció d'errors les quals poden ser utilitzades per estructura un sistema tolerant a errors. Per exemple, un procés de seguiment de la situació i les activitats d'altres processos, inclús si aquests s'executen en nodes diferents.

Hot Swap (Actualització del codi en calent): Es troben molts sistemes que no poden se aturats pel seu manteniment, l'Erlang permet actualitzar el codi de l'aplicació sense que el sistema es pari i segueixi funcionant. El codi antic es eliminat i substituït pel nou, durant aquesta transició tan el codi antic com el nou poden coexistir. Per tant, és possible instal·lar les actualitzacions i correccions d'errors en un sistema en funcionament sense alterà el seu estat.

Carrega de codi incremental: Els usuaris poden controlar en detall com s'ha carregat el codi. En sistemes incorporats, el codi sol ser carregat en el moment d'arrencada. En sistemes de desenvolupament, el codi es carrega quan és necessari, inclús quan el sistema està funcionant. Per tant si es descobreix algun error només es necessita canviar la part de codi amb errors.

Interfícies externes: Si per motius d'eficiència fos necessari l'Erlang permet està directament vinculat amb aplicacions en C o Java.

2.3.Components

Amb l'Erlang *open-source* hi ha varis components que poden ser utilitzats com elements bàsics en el desenvolupament d'aplicacions. Aquest components estan desenvolupats per comprendre el pas de missatge amb Erlang, la càrrega, descarrega, inici, parada, reinici i canvi de codi.

Inets: Servidor i client HTTP i FTP

Mnesia: Base de dades distribuïda i en temps real per l'Erlang. Suporta la carrega de dades en memòria RAM així com l'emmagatzematge en disc, permet canvis dinàmics de l'esquema també permet guardar estructures de dades complexes. Mnesia es molt ràpida ja que s'executa en el mateix espai de direccions que les aplicacions que l'utilitzen (això es possible gracies a que Mnesia i les aplicacions estan escrites amb Erlang).

Orber: CORBA v2.0 Object Request Brocker (ORB).

SNMP: Agent SNMP extensible v1/v2 i compilador MIB.

2.4.Eines i Llibreries que trobarem

Algunes de les eines útils que trobarem que ens porta són:

Appmon: Monotorització (seguiment) gràfic de grups de processos (locals i en nodes remots).

ASN.1: Compilador en temps d'execució el qual també suporta la notació bàsica ASN.1 i les regles de descodificació BER, DER, i PER (alineat).

Compiler: Compilador Erlang.

Debugger: Corrector d'errors, depurador (debugger) gràfic.

GS: Llibreria per escriure interfícies gràfiques d'usuari (GUI – graphical user interface).

IC: Compilador provingut de OMG Interface Definition Language (IDL) per l'Erlang amb el C i Java.

Kernel: Codi en C necessari per executar el sistema Erlang: Erlang built-in functions (BIFs). Codi d'arrencada i servidor de noms. Suport a la distribució i a la xarxa. Carregadors, enllaçadors i *loggers*. Sistema operatiu i sistema de fitxers.

Mnemosyne: Llenguatge opcional per les consultes a la base de dades Mnesia

Mnesia Session: Idiomes d'interfície a Mnesia es defineix en el IDL, proporciona accés a traves de IIOP i protocols erl_interface.

OS monitor (OS_MON): Seguiment de la CPU, disc i memòria utilitzades.

Parse tools: Analitzador LALR-1

PMan: Eina per veure l'estat dels processos en l'Erlang (locals i remots)

SASL: Informes sobre progressos, errors, accidents (*crash*).

Stdlib: Llibreries per l'entrada/sortida.

Table visualizer: Eina per veure les taules ETS i Mnesia

Toolbar: Barra d'eines escrita en Erlang

Tools: Analitzador de cobertura, de perfil, fitxer generador de TAGS en Emacs, utilitat make, utilitat gràfica de trucada,....

2.5.Explicacions i algun exemple

Tot seguit es mostren alguns exemples i algunes explicacions gràfiques sobre les característiques que te programar amb Erlang. Aquest apartat està destinat bàsicament en veure d'una manera més senzilla les diferències que te aquest llenguatge enfront als altres.

2.5.1.Tipus de fitxers amb Erlang

Amb Erlang, al igual que amb altres llenguatges de programació, es te més d'un tipus de fitxer.

En aquest cas podem dir que bàsicament en tenim dos, en quan a executables i codi font:

- Els Mòduls: Aquets fitxers son els que contenen el codi font del programa, son aquells on escrivim totes les funcions que té l'aplicació. Aquest fitxers tenen com a extensió ***erl***

- Els Executables: Els fitxers executables són aquells els quals des de la màquina virtual es poden executar per obtenir un resultat. Són els que es generen un cop es compilen els mòduls, aquest tenen l'extensió ***beam***

2.5.2.Processos i PID's

D'ara en endavant es parlarà tota l'estona de processos i paral·lelisme però què és un procés?

Un procés no és res més que un programa informàtic en execució, i el paral·lelisme significa que es te més d'un procés que col·laboren junts per executar una tasca en concret.

Una altra cosa que serà d'utilitat posteriorment és saber que amb Erlang, tal com s'ha dit, es poden crear processos els quals executen unes ordres. Quan es creen aquests processos, es retorna el que es diu un PID això significa *Proces IDentifiquer,* Identificador del Procés en el nostra cas.

Aquest PID no és res més que un número únic el qual representa un procés, quan es vol dirigir a ell només es te que utilitzar aquest número que sol ser de la forma <X.X.X> i on X és un número i el seu tipus s'anomena *pid*.

```
4> P = spawn(fun() -> hot_swap:loop() end).
<0.42.0>
5> erlang:is_pid(P).
true
6> erlang:is_integer(P).
false
7> erlang:is_list(P).
false
```

- Creació d'un procés

Amb Erlang però, apart de tenir aquest número hi ha la possibilitat de assigna'ls-hi un nom, amb això el que s'aconsegueix és que pels processos generals i més importants se saben quins són sense la necessitat del PID.

I com es poden comunicar entre si aquest processos?

La comunicació amb Erlang és molt fàcil, quan es tenen dos processos i es vol establir una comunicació entre ells és fa via missatges, un envia un missatge a l'altre i l'altre el respon. La comanda que envia un missatge simplement és:

PID ! {Contingut_del_missatge}

On ***PID*** és el PID del procés el qual es vol enviar el missatge i el ***Contingut_del_missatge*** és el missatge que es vol enviar en el procés, per exemple si tenen dos processos que s'han registrat amb els noms exemple_1 i exemple_2:

L'**exemple_1 envia** a l'exemple_2: ***exemple_2 ! {"Hola, soc l'exemple número 1!"}***

L'**exemple_2 rep** de l'exemple_1: ***{"Hola, soc l'exemple número 1!"}***

Continuant amb l'exemple anterior:

```
9> erlang:register(exemple,P).
true
10> P ! {self(), "proba 1"}.
Versio:1.0
 Den <0.47.0> he rebut:"proba 1"
{<0.47.0>,"proba 1"}
11> exemple ! {self(), "proba 2"}.
Versio:1.0
 Den <0.47.0> he rebut:"proba 2"
{<0.47.0>,"proba 2"}
```

- Enviament de missatges

2.5.3.Creació de Processos amb Erlang

Ja s'ha vist que és un procés i com es comuniquen, però ara faltaria saber com es creen, aquest tema és el que es tracta en aquet apartat.

Amb Erlang per crear un procés és molt fàcil, simplement amb la comanda ***spawn(X)***, el que fa, és crear un procés que executa una funció X, *l'spawn* retornarà el PID (numero únic per comunicar-se amb el procés), i per registrar-lo la comanda és ***register(Y,Pid)***, on *Y* és el nom i *Pid* el PID del procés.

Amb Erlang però, també es pot crear un procés amb la comanda ***spawn_link(X)*** aquesta fa el mateix que l'anterior, però amb la diferencia que crear un enllaç bidireccional (link) entre els dos processos, el que s'ha creat i el que l'ha creat. L'ha diferencia bàsica entre el *spawn* i el *spawn_link* és la creació d'aquest enllaç, aquest enllaç el que fa és crear una relació/dependència entre ells, és a dir, si un es mort l'altre també, i tots els que estan enllaçats entre si, tots es moren. Per habitar això tenim el que sen diu ***trap_exit***, això és un flag, que si l'activem (amb la comanda ***process_flag(trap_exit, true)***) i un procés que està enllaçat amb un altre es mort/destrueix en comptes de morir , rep un missatge *d'EXIT*, aquest pot ser de la forma:

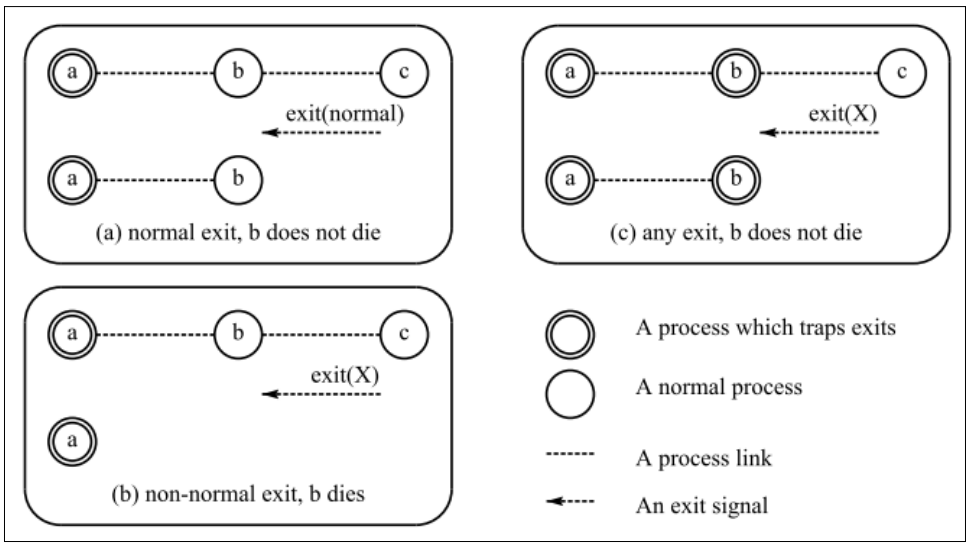
{'EXIT', normal} → destrucció correcta

{'EXIT',{error,Why} → destrucció incorrecta amb error *Why*

Això s'utilitza perquè si un procés és destruït incorrectament es pugui tractar el seu error i si es necessari tornar-lo a crear.

En el cas que es vulgui destruir un procés es pot utilitzar la comanda ***exit(Why)*** la qual destrueix el procés actual. Si es fa un *exit(normal)* el senyal de sortida serà del tipus "destrucció correcta".

Tot seguit es mostra una imatge perquè estigui tot més clar:



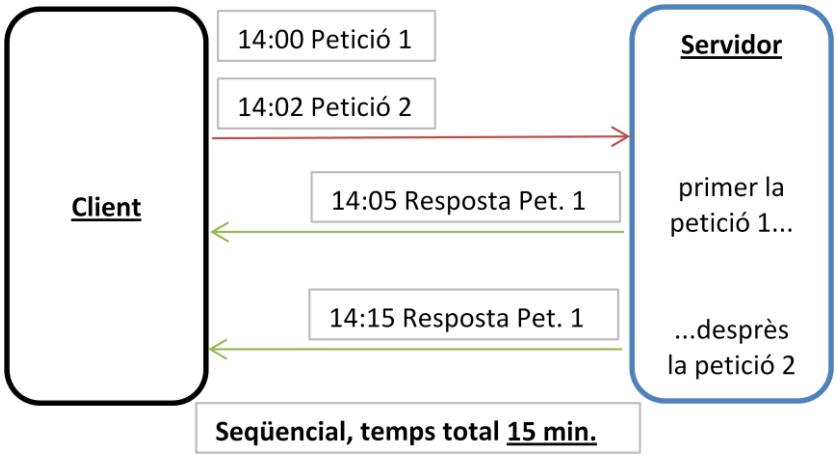
- Llibre Programming Erlang

2.5.4.Diferencies entre Seqüencial i Paral·lel

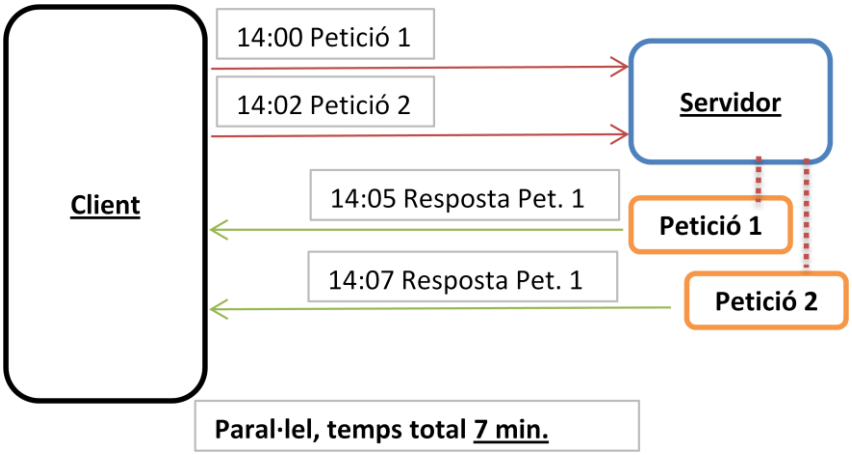
La gran i evident (pel nom) diferencia entre seqüencial i paral·lel és que amb seqüencial s'executen les ordres una després de l'altre en canvi amb paral·lel es poden executar varies ordres a la vegada.

Per exemple si es te un client que fa dos peticions A en un servidor amb una diferència de 2 min. entre elles. El servidor tarda en processar la petició A 5 min.

En les següents imatges es veuen les diferències de temps: (sense tenir en compte retards)

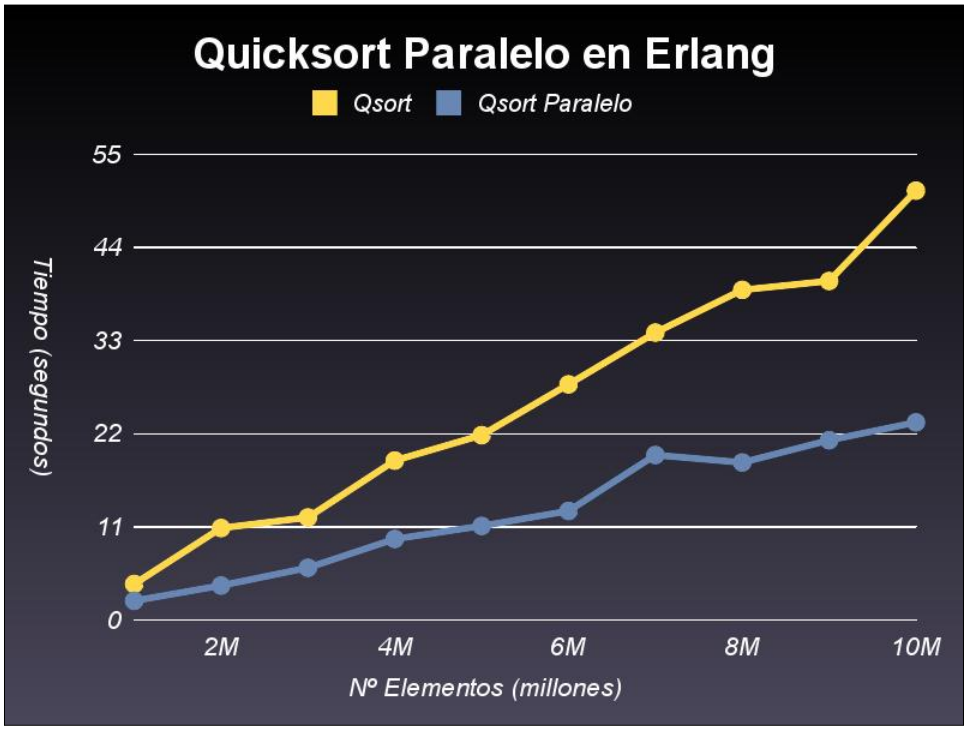


- Exemple aplicació treballant en seqüencial



- Exemple aplicació treballant en paral·lel

Un exemple real és el sistema d'ordenació Quicksort, aquest exemple el sistema ha estat desenvolupat en seqüencial i tot seguit en paral·lel amb el mateix llenguatge, l'Erlang. Finalment s'han comparat els temps de resposta en segons i els milions d'elements a ordenar.



- Fonts quicksort, llibre Programming Erlang i pàgina web www.bicosyes.com

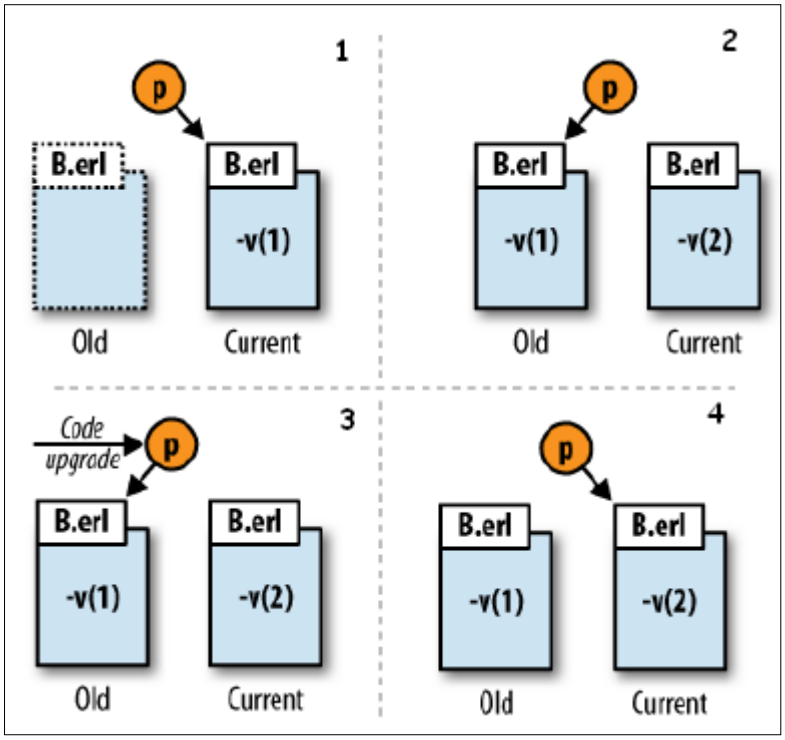
Tal com s'observa el sistema Quicksort mostra uns temps molt similars si es tracta d'ordenar petits elements, però a la que s'augmenten els elements es veu com l'aplicació en paral·lel es bastant millor.

2.5.5.Hot Swap

Amb Erlang quan es parla de *Hot Swap* el que es vol dir és actualitzar el codi sense tenir que "parar el sistema".

Això vol dir per tant, que es pot tenir un procés X que s'està executant sempre, per exemple un procés general d'un servidor. Si pel que sigui es troba un error en aquest procés l'únic que es te que fer és arreglar el codi i compilar-lo de nou, llavors el codi s'actualitzarà "automàticament".

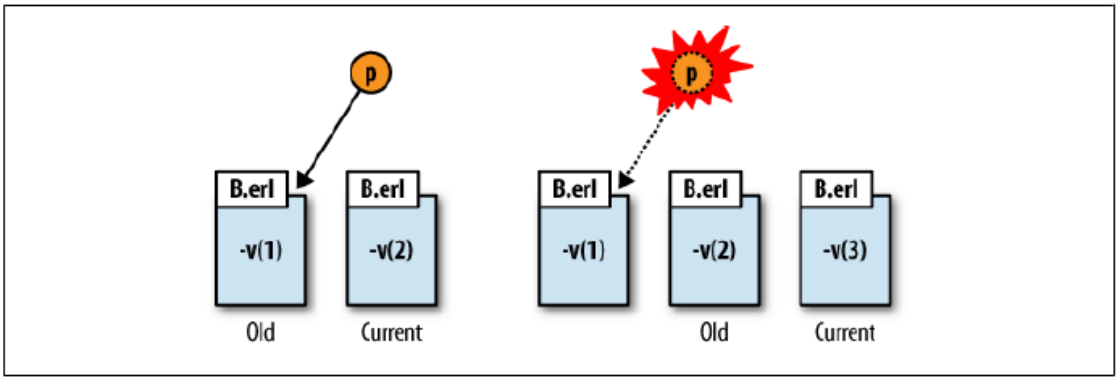
Si es parla de processos el codi d'aquest sol ser un bucle infinit, per tant el codi serà actualitzat quan acabi aquesta volta. Per aquest motiu es habitual posar un missatge de *update*.



- Exemple Hot Swap

En la primera imatge la numero 1 s'observa un procés p que executa un mòdul anomenat B.erl, llavors en la mateixa carpeta es compila un altre cop el mòdul però amb alguna modificació i amb un nom de versió diferent (imatge 2). Llavors a la que s'executa la funció *update* (rep un missatge *d'update*, imatge 3) i tot seguit veu que hi ha una altre versió i aquesta es carregada a memòria, imatge 4.

Però una de les coses que es tenen que tenir en compte és que Erlang no suporta fer el canvi amb més de dos versions tal com es veu en la següent imatge



- Exemple error Hot Swap

Tot seguit es mostra un exemple, posem pel cas que tenim el mòdul *hot_swap.erl*

```
1  -module(hot_swap).
   -export([main/0,loop/0]).
   -vsn(1.0).
   -define(VERSION, 1.0).

2  main() ->
       spawn(?MODULE,loop,[]).

3  loop() ->
       receive
           {_,update} ->
                io:format("Actualitzacio de la versio:~p~n",[?VERSION]);
           {Sender, N} ->
                io:format("Versio:~p~n Den ~p he rebut:~p~n",[?VERSION,Sender,N])
       end,
    ?MODULE:loop().
```

- Codi del mòdul hot_swap.erl

A la primera part es defineix el mòdul i la versió *-vsn(1.0)*. La segona part es la funció principal que crea un procés i ens retorna el seu PID. La tercera es el bucle que executa el procés es tracta nomes de que per cada missatge rebut executa un tipus d'acció, observeu el **missatge *d'update*!**

Seguidament per comprovar-ho s'obra una consola d'Erlang, es compila el mòdul (línea 1), a la segona línea es veu com a la variable P es posa el resultat de l'execució de la funció *main*, el qual és el PID del procés creat, P servirà per comunicar-nos amb ell. Llavors observem la línea 3 com li enviem un missatge.

```
Erlang R13B01 (erts-5.7.2) [smp:4:4] [rq:4] [async-threads:0]

Eshell V5.7.2  (abort with ^G)
1> c(hot_swap).
{ok,hot_swap}
2> P = hot_swap:main().
<0.39.0>
3> P ! {self(), "proba 1"}.
Versio:1.0
 Den <0.32.0> he rebut:"proba 1"
{<0.32.0>,"proba 1"}
```

- Exemple Hot Swap 1

Llavors per veure el *Hot Swap* simplement realitzem uns petits canvis en el mòdul, es canvia la versió, la definició de versió i el text:

```
-module(hot_swap).
-export([main/0,loop/0]).
-vsn(1.2).
-define(VERSION, 1.2).

main() ->
    spawn(?MODULE,loop,[]).

loop() ->
    receive
        {_,update} ->
            io:format("Actualitzacio de la versio:~p~n",[?VERSION]);
        {Sender, N} ->
            io:format("Versio:~p~n De tu, ~p he rebut:~p~n",[?VERSION,Sender,N])
    end,
 ?MODULE:loop().
```

- Modificació del codi hot_swap.erl

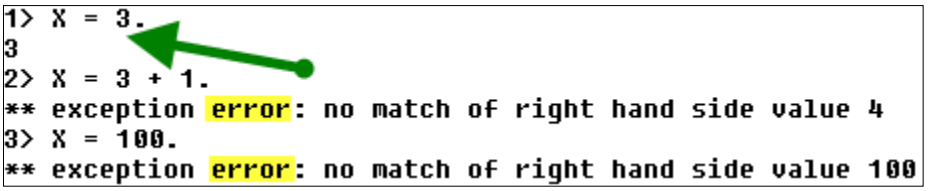
Finalment per acabar-ho de comprovar es compila un altre cop el mòdul, li enviem un *update* al procés, i fem una proba per veure com realment s'ha canviat el codi del mòdul carregat a memòria.

```
4> c(hot_swap).
{ok,hot_swap}
5> P ! {self(), update}
Actualitzacio de la versio:1.0
{<0.32.0>,update}
6> P ! {self(), "proba 2"}.
VERSIO:1.2
 De tu, <0.32.0> he rebut:"proba 2"
{<0.32.0>,"proba 2"}
```

- Exemple Hot Swap2

2.5.6.Variables amb Erlang?

Una de les grans diferencies en que ens troba al principi quan es comença a programar amb Erlang és l'ús de les variables. Amb Erlang les variables són variables algebraiques, això vol dir que per exemple, si s'assigna un valor a la variable X el valor 3 sempre valdrà 3 (durant l'execució de la funció).



- Exemple Variables algebraiques

Quan anava a escola, el meu professor de matemàtiques va dir: "Si hi ha una X en diferents parts de la mateixa equació, totes les X signifiquen el mateix. Així podem resoldre les equacions! Si sabem que X+Y=10 i X-Y=2 llavors la X serà 6 i la Y 4 en ambdues equacions!" Però quan vaig aprendre el primer llenguatge de programació fèiem coses de l'estil X=X+1?! Tots protestàvem i dèiem que això no podia ser, no era el que havíem après a matemàtiques! Però el professor deia que estàvem equivocats i teníem que despendre'ns de tot el que havíem après a classe de mates.
Amb Erlang això no passa, les variables son com a mates, nosaltres associem un valor a la variable.

- *Programming Erlang de Joe Armstrong*

CAPÍTOL 3

REALITZACIÓ DEL PROJECTE

Capítol 3: Realització del Projecte

Tot seguit s'explicarà com s'ha gestionat la el projecte, que és el que s'ha fet des de que es va començar fins al final, la organització del temps i els objectius.

3.1.Objectius

Els objectius d'aquest projecta eren en un principi dos, el més gran **era l'aprenentatge del llenguatge de programació Erlang** i el segon la construcció d'una aplicació realitzada amb aquest.

El primer es tracta d'aprendre, llegir i practicar i que millor que practica sobre un llenguatge que realitzar una pràctica amb ell?

Això és el que s'ha fet amb aquest projecte s'ha realitzat una pràctica, en concret un sistema de missatgeria instantània (xat) multi protocol.

L'objectiu era realitzar una aplicació on es veies bé el funcionament de l'Erlang, i com que l'Erlang és un llenguatge pensat per les comunicacions es va decidir realitzar un xat ja que és una de les aplicacions més agradables i divertides de realitzar quan tens que programar amb xarxes sobretot perquè un cop realitzada la pots provar amb els teus amics i companys!

Inicialment l'aplicació només era construir un **xat (client i servidor) en xarxa local**, que fos capaç de comunicar-se entre els diferents usuaris, després també se li van afegir altres opcions com poder transferir fitxer entre usuaris i també poder canviar d'estat.

L'objectiu del servidor i client en el xat local era de veure com actuaven els processos en paral·lel i de pel servidor a també de desenvolupar una petita base de dades amb Mnesia.

Més tard quan encara es pensava com desenvolupar la pràctica es va decidir també incloure, en el client, la capacitat de connectar-se en algun altre tipus de sistema de missatgeria instantània, en un principi es va pensar amb un dels més utilitzats el Messenger, però es va veure que era molt difícil ja que el seu protocol de comunicació no es lliure. Per tant es va fer que el client fos capaç de connectar-se en el servidor local i a més a més **connectar-se amb un canal del IRC**.

3.2.Gestió Projecte

Aquest projecte consistia de dos grans parts com molts dels projectes tecnològics:

La Primera Recerca

La recerca en aquest cas consistia en aprendre un nou llenguatge de programació bastant diferents als vistos fins el moment. En aquest cas Erlang s'assemblava en un vist anteriorment a una assignatura, el Prolog. Tret d'aquesta referència/semblança no se'n sabia absolutament res més.

Primer de tot es va començar per buscar informació en la pagina web oficial. Donat que la oficial estava amb angles es va utilitzar un cercador per veure si es trobava alguna pàgina útil amb Català o Castellà, la cerca va ser profitosa, però no va durar gaire, ja que la majoria de pagines només eren una introducció al llenguatge Erlang, no aprofundien del tot.

Després es va estudiar el llenguatge a partir d'un dels seus llibres més reconeguts *Programming Erlang* de Joe Armstrong. Aquest llibre és de gran utilitat gràcies a la gran quantitat d'exemples que té.

Més tard cap al mes de Juny/Juliol va sortir a la venda un nou llibre el *Erlang Programming* de Francesco Cesarini i Simon Thompson, aquest però al sortir a unes dates tan ajustades nomes s'ha utilitzat per fer consultes i no s'hi ha aprofundit tan com en el primer.

S'ha de tenir en compte que també a la recerca d'informació, apart dels llibres de text, es combinava amb la cerca d'informació sobre funciones, mòduls, etc..., la qual gran part va ser proveïda per la pàgina web oficial d'Erlang (www.erlang.org).

La Segona Recerca:

Aquesta segona recerca no va d'una durada tan elevada com va ser la primera, ja que en aquesta nomes es tractava de cerca informació sobre el protocol de missatgeria instantània IRC per poder realitzar un client capaç de comunicar-se amb ell.

Aquesta, es va portar a terme un cop la primera part de la pràctica ja estava en part mig realitzada (el servidor i part del client) ja que no era tant difícil de portar a terme.

Ja que només es tractava de realitzar el procés encarregat de descodificar el protocol del IRC i modificar una mica el de comunicació amb el servidor.

Pràctica

La part pràctica és una de les més grans d'un projecta d'aquest típus ja que és allà on es veu tot el que s'ha aprés. La part pràctica es portava a termer m'entres s'aprenia a programar amb Erlang, sobretot la primera part, el xat local.

També avanç de començar a realitzar la pràctica gran s'en va realitzar una de probes petita, nomes consistia en llistar els fitxer d'un directori que estava en un altre ordinador de la mateixa xarxa local.

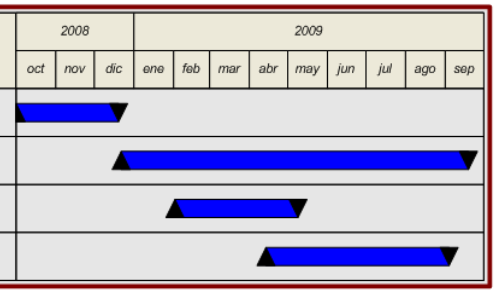
Això va portar un petit problema ja que la realització d'aquesta pràctica més petita va alentir la gran perquè es veia que sempre es podien fer modificacions per petites que focin. I el problema va ser que la va fer més dura de portar a terme ja que el temps empleat per realitzar la pràctica de probes s'hagués pogut utilitzar per iniciar-se de mica en mica amb el xat i anar fent petites pràctiques (de durada de dies) per anar provant les diferents parts

Gestió del temps:

Tot seguit es mostra un petit gràfic el qual es pot apreciar una mica millor el temps utilitzat.

S'ha de tenir en compte que les dates són nomes orientatives i no exactes.

Nombre de tarea	Comienzo	Fin
Recerca d'un Projecte Final de Carrera	01/10/2008	19/12/2008
Aprenantatge llenguatge Erlang	22/12/2008	18/09/2009
Realitzacio Aplicació de Proba	02/02/2009	08/05/2009
Realització Xat Multiprotocol	14/04/2009	03/09/2009



- Gràfic amb la gestió del temps

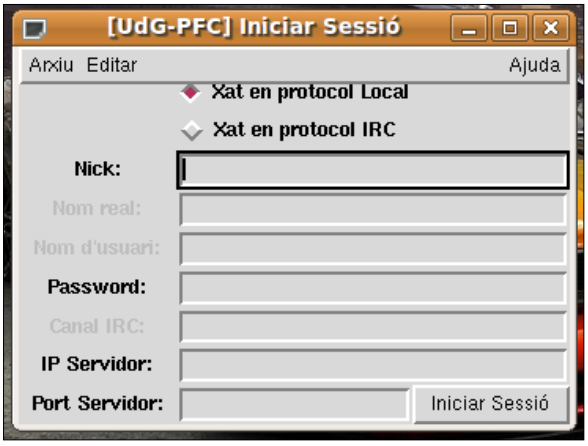
CAPÍTOL 4

INTRODUCCIÓ A L'APLICACIÓ

Capítol 4: Introducció a l'aplicació

Aquest apartat està dedicat bàsicament a ser una introducció del que vindrà després, s'explicarà el contingut pel qual esta format l'aplicació, fitxers, carpetes i altres.

En moltes ocasions durant el capítol 4 i el 5 es parlarà de *widgets*, en aquest document quan es parla de widget es referència a una aplicació gràfica, com per exemple una finestra d'una aplicació. En la següent imatge es mostra una finestra d'inici de sessió, o un widget:



- Exemple widget

4.1.Servidor

A dins de la carpeta del servidor hi consten els següents fitxers (sense tenir en compte els fitxers compilats *.beam*):

Aquest mòdul conté tot el necessari per engegar tot el sistema del qual esta format el servidor. Inicia el procés encarregat de gestionar i controlar els altres, el ***servidor_adm***, inicia el procés de comunicació ***servidor_do_accept***, el qual en el cas del servidor és el que accepta peticions, també al inici arrenca la base de dades i és ell qui hi realitza totes les modificacions. Llavors quan algun client sol·licita crear una conversa ell és l'encarregat de crear els processos corresponents i si la conversa necessita crear és la general, la registra amb el nom ***servidor_grup***.

En el *grup.erl* hi ha tot el necessari per poder crear el procés encarregat de gestionar les converses en grup, en el cas del grup general és el ***servidor_grup***.

Aquest altre mòdul te tot el necessari per comunicacions, és el que s'encarrega de accepta les peticions per part del client amb el procés ***servidor_do_accept*** i un cop acceptada crea un procés específic per comunicar-se amb el client pel socket.

Aquí s'hi troba el codi que s'utilitza per arrencar la base de dades i realitzar les consultes necessàries a ella.

Aquest mòdul el està en dos llocs, en el client i en el servidor, bàsicament del que s'encarrega és de fer apareixia un missatge d'error amb una imatge i el text d'error.

Imatge que es veu quan sens apareix una pantalla d'error

Carpeta que conté els fitxers de configuració i les dades per la base de dades amb Mnseia

4.2.Client

A la carpeta del client una de les diferencies amb que ens trobarem és que a dins i n'hi ha dos carpetes més.

La ***Fitxers Rebuts***, és on hi s'hi troben els fitxers que es reben pel xat en protocol local. L'altre, ***Moduls***, és allà on hi ha els mòduls que s'explicaran tot seguit.

Amb el client s'hi troba un mòdul i una imatge que també estan en la carpeta on es troba el servidor, és el *miss_err.erl* i el *error_icon.gif*. També es troben aquí perquè sinó sense aquests dos fitxers no es podríem mostrar els missatge d'error. També hi han dos imatges més, necessàries per poder mostrar la finestra d'informació sobre l'aplicació correctament.

Els altres que i trobarem son:

Aquest mòdul és el principal de tot el client és l'encarregat de crear els processos principal i de gestionar els errors d'aquests. També conté la funció que crea el procés el qual entrem totes les dades per connectar-nos en un servidor o en un altre, el ***client_wis***. També és l'encarregat de crear el procés corresponent al protocol escollit en el *client_wis*, el ***client_plocal*** o be el ***client_pirc***.

Aquest és l'encarregat de gestionar part dels missatges rebuts pel servidor local, també és l'encarregat de crear alguns processos com son els *widgets* (finestres) per les converses. En el cas de la conversa general, que és en la que tots els usuaris parlen amb tots els altres rep el nom de *client_general*.

Aquest és com l'anterior però en comptes de gestionar els missatges rebuts pel local gestiona els rebuts per un servidor que utilitza el protocol IRC. I al igual que amb *plocal* la sala principal es registra amb el nom de ***client_general.***

Aquest conté el codi per comunicar-se amb un servidor IRC i amb un servidor local. Pel que fa al servidor local el *pcomc* també és l'encarregat de gestionar l'enviament de fitxers.

Mòdul que té el codi necesari per crear el widget d'inici de sessió.

Aquest altre crea el widget que s'utilitza per mantenir les converses.

Widget on demana que entrem la ruta per enviar un fitxer a un client.

Aquet, també crea un widget, en aquest cas crear una finestra amb una llista d'usuaris.

Finalment el *winf.erl* també s'utilitza per crear una finestra, en aquest cas una finestra que mostra informació sobre l'aplicació.

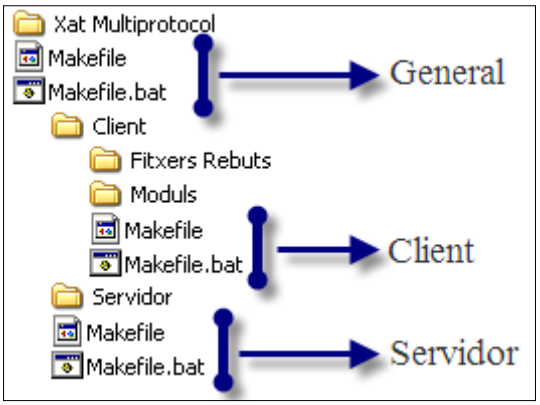
4.3.Makefiles

Pel que fa els *makefiles* ni han 6 en diferents llocs i de diferents tipus, un del tipus és el típic que es troba en sistemes operatius Linux el *Makefile* i l'altre és un executable de Windows *Makefile.bat.*

Se'n troben tants perquè n'hi han dos de generals (un per Linux i l'altre per Windows) que es troben a la carpeta general, la Xat Multi protocol. Llavors en trobem dos més a dins la carpeta Servidor, aquests són específics del servidor, al igual que la carpeta Mòduls del client que n'hi ha dos més d'específics.

Per tant podríem dir que segons el sistema operatiu en trobem un de general i dos d'específics. El seu funcionament és senzill, els específics són els que fan tota la feina i el general el que fa és executar cada un dels específics.

Els que trobem són els següents:



- Distribució dels Makefiles

Amb aquets *makefiles* el que es pot fer és compilar els mòduls, eliminar tots els fitxers *.beam* i *.dump* (mòduls executables i fitxers d'error respectivament) o bé executar.

També s'ha de pensar que cada *makefile* disposa d'una mica d'informació, per executar-ho amb Linux és la comanda "make info" i amb Windows no s'ha posat perquè ja es disposa d'un petit menú, observar la següent imatge:

- Exemple Makefile.bat en Windows

CAPÍTOL 5

L'APLICACIÓ, XAT MULTI PROTOCOL

Capítol 5: L' Aplicació, Xat Multiprotocol

Tal com s'ha explicat anteriorment la part principal del projecte ha estat el desenvolupament del sistema de missatgeria multi protocol, un xat.

Aquest xat és multiprotocol, és a dir, un client i un servidor es comuniquen entre si, però amb la diferencia que el client també és capaç de comunicar-se amb dos tipus de servidors diferents, el local en un primer cas i el IRC en el segon.

Tal com s'ha descrit s'ha realitzat amb el llenguatge Erlang i s'ha intentat paralitzar de tal manera que fos comprensible. Per poder-lo desenvolupar amb aquest llenguatge no fan falta gaires eines, simplement un ordinador amb un sistema operatiu Linux o Windows 2000 o XP i un editor de text que permeti guardar amb la codificació de caràcters ISO-8859-1, com el Gedit ja que sinó alguns caràcters no surten bé (accents, dièresis,...).

Pel que fa el funcionament de l'aplicació és mol similar a qualsevol altre, el servidor que respon les peticions del client i el client que executa una sèrie d'ordres donades per l'usuari.

Segurament la diferencia és que al utilitzar un protocol local s'han pogut realitzar converses client a client, això vol dir que quan tenim dos client que volen parlar entre si en comptes de enviar sempre les dades pel servidor se les envien només entre ells.

En l'apartat sobre l' introducció a l'aplicació ja s'ha parlat més o menys de que era el que feia cada mòdul, en aquest però s'explicarà més en detall el disseny de l'aplicació, el seu funcionament (que és el que fa) i també es comenten les funcions que conté cada mòdul, les principia'ls (publiques) es donarà la pre-condició i les privades una petita explicació.

Pel que fa al codi de l'aplicació es pot trobar tot en el CD inclòs en el projecte.

5.1.Llegenda

En un primer lloc el que es farà és descriure la llegenda sobre les imatges que s'aniran veien durant les explicacions sobre el funcionament del client i el servidor.

Aquesta imatge fa referència a Internet o bé a una connexió en xarxa LAN.

 Una imatge d'aquesta tipus significa que es tracta d'un procés

Les fletxes que es veuran també tenen un significat especial, es trobaran les següents:

- Una comunicació amb els clients "especial" únicament s'escolten peticions:

- Comunicació amb els client a través de la xarxa local o Internet:

- Comunicació entre els processos de l'aplicació (pot ser bidireccional o unidireccional):

- Un procés nou ha estat creat:

5.2.Servidor

El servidor s'ha intentat desenvolupar d'una manera senzilla i entenedora. El servidor a diferencia del client no utilitza una interfície gràfica, ja que per les consultes a realitzar és simplement entrar una comanda senzilla (mirar manual servidor).

Les llibreries utilitzades en el servidor han estat:

- **erlang**: És la llibreria principal, s'utilitza per la majoria de coses, crear processos, destruir, comprovar, etc...
- **io**: Aquesta és l'encarregada de mostrar el text per la sortida corresponent, en el nostre cas la pantalla (és com el *printf* en C).
- **lists**: Per la gestió de llistes.
- **string**: Aquesta és per la gestió dels elements tipus string.
- **mnesia**: Per engegar la base de dades, parar-la, crear, destruir i modificar els elements.
- **qlc**: S'utilitza per les cerques en els elements de la base de dades.

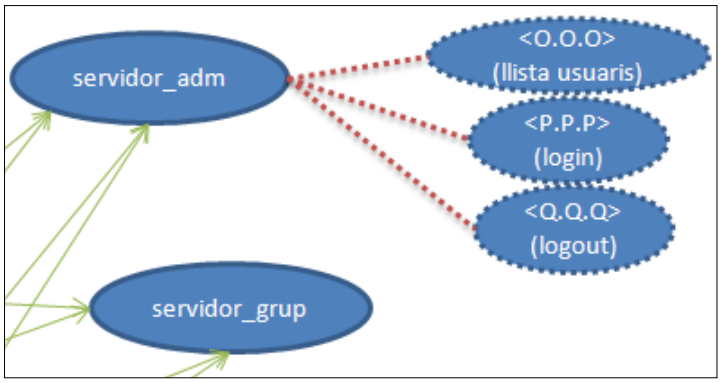
5.2.1.Disseny

El disseny del servidor es basa bàsicament en 4 grans grups de processos, el primer, per ordre d'importància, és el *servidor_adm*, el que algun cop se l'anomenarà **servidor** "a seques", en la imatge és el numero 3. Aquet és l'encarregat de gestionar tots els processos, els crear, els pot destruir i si s'han destriuit mira si és normal o bé ha estat per algun error. També s'ha de tenir en compte però que el servidor quan se li sol·licita segons quina petició aquest crea un petit

procés el qual és l'encarregat de respondre aquesta petició, així d'aquesta manera obtenim la paralització, o sigui es poden realitzar més d'una cosa a la vegada. Això si, sempre depèn del que es realitzi a la base de dades, ja que si es realitza una operació d'escriptura aquest queda bloquejada i no en deixa realitza cap altra, en canvi si se'n realitzen dos de consulta aquestes poden ser a al mateix temps. Les peticions en les quals passa això són:

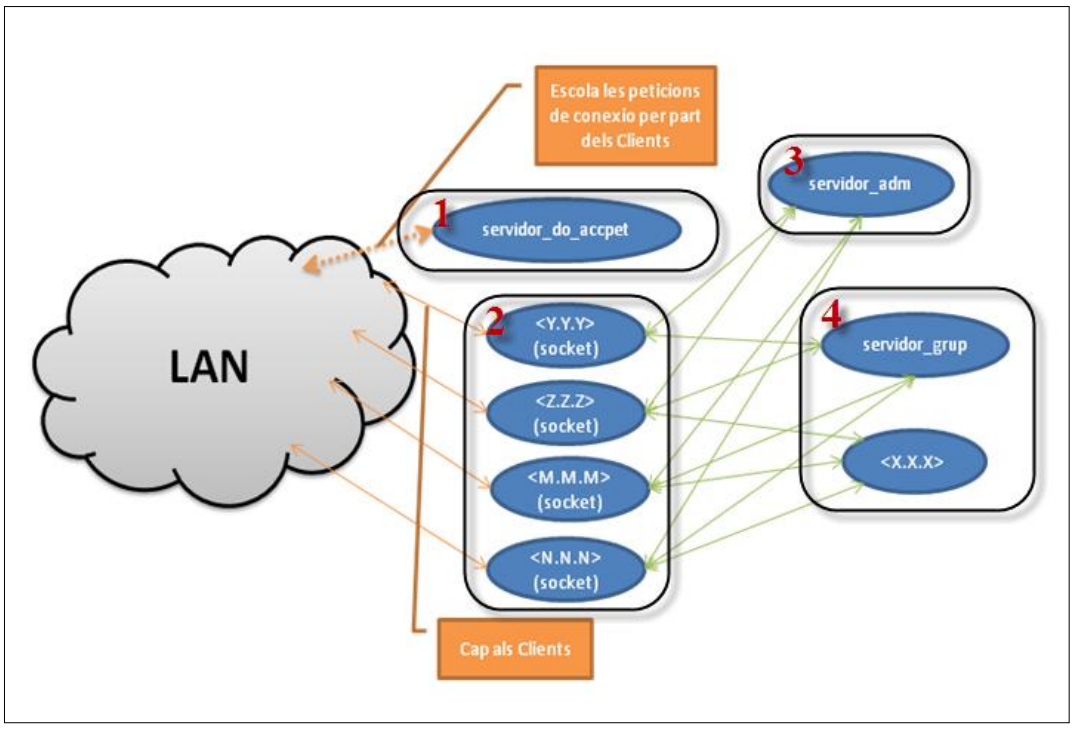
- Quan es realitza el login d'un usuari
- Quan es sol·licita un canvi d'estat
- Quan es sol·licita la llista d'usuaris
- Quan es sol·liciten les dades per la creació de la conversa client a client
- Quan es realitza el logout del servidor

O sigui un petit esquema seria el següent:



- Exemple de la realització de més d'una consulta a la vegada

El grup que te el número 1 en que només hi ha un procés és el *servidor_do_accept* que es pot dir el **do_accept**. El numero 2, el format pels processos encarregats d'escoltar els sockets, en poden dir **pid_sockets** i finalment el grup número 4 és el format pel *servidor_grup* i el <X.X.X>, també anomenat **grups**

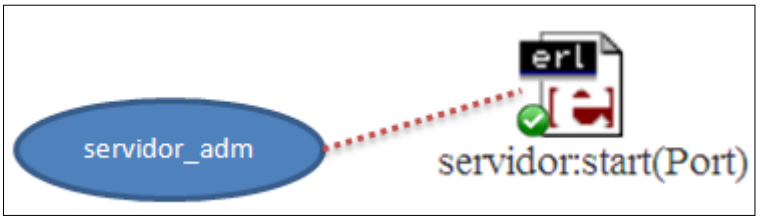


- Imatge dels processos en un moment concret del servidor

També s'ha de tenir en compte que el servidor no s'utilitza una interfície gràfica. Això és degut a que les operacions que es realitzen amb ell són mínimes, simplement engegar, parar, donar d'alta algun usuari i si s'escau expulsar-ne algun.

5.2.2.Funcionament

El funcionament del servidor és relativament senzill, en un principi des de la consola d'Erlang, per engegar el servidor nomes es te que executar la comanda (funció) *start* que es troba en el mòdul *servidor.erl*



- S'executa la funció servidor:start(Port) i es crea el servidor_adm

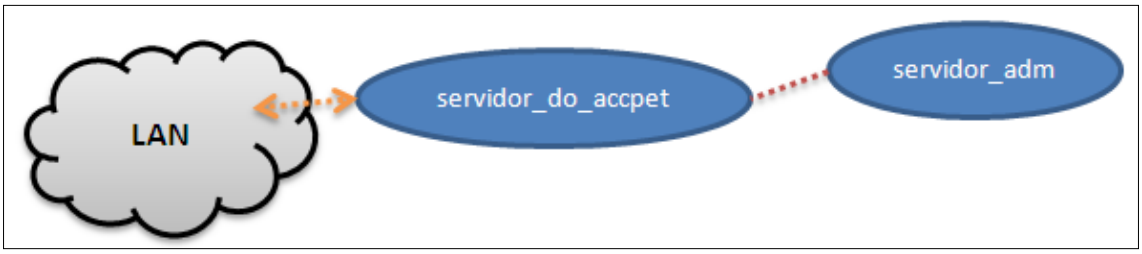
Aquesta funció el que fa és crear el procés que s'encarrega d'administrar-los a tots, el *servidor_adm* (també anomenat en algunes ocasions servidor "a seques").

Per poder fer això tots els processos que creen el servidor i els altres estan enllaçats, o sigui creats amb la comanda *spawn_link*, això es fa perquè com s'ha dit el *servidor_adm* fa com

"d'administrador" per tant és l'encarregat d'executar unes instruccions o unes altres tot depenent de com hagi estat destruït un procés o un altre.

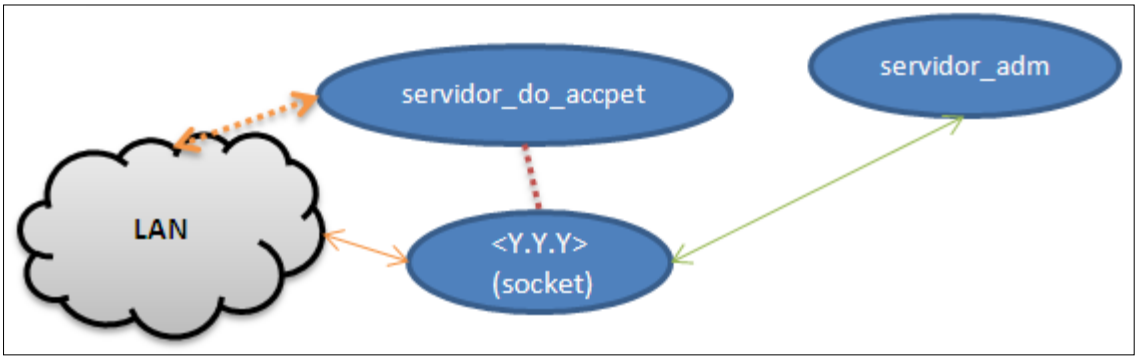
Llavors el servidor és l'encarregat de, primer carregar la base de dades Mnesia a partir de les funcions definides en el mòdul *servidor_bdd*. Després crea el procés encarregat d'escolta les peticions dels clients pel port entrar en la comanda *start*, el *servidor_do_accept*.

En un principi només tenim aquest dos processos, i fins que no es connecta un client no comença tot, quan es connecta el *do_accept* accepta la connexió i crea un procés que esta dedicat únicament a escoltar el socket d'aquest client, a partir d'ara en direm també el *pid_socket*.



- Es crea el procés que accepta les connexions

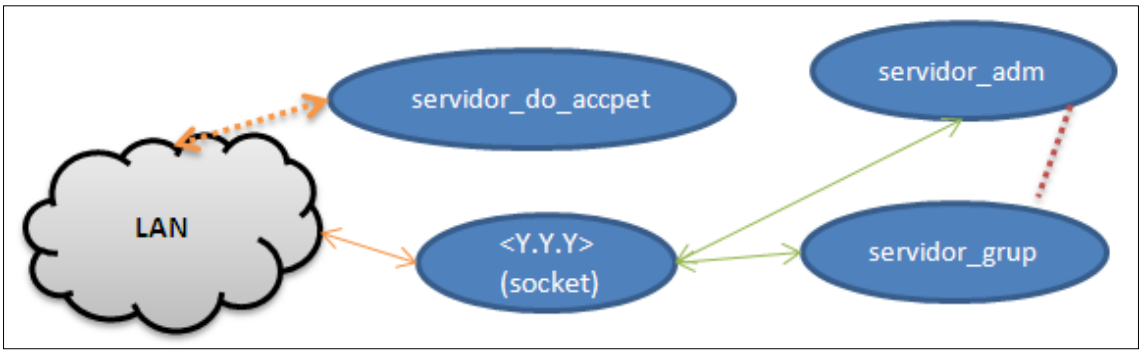
Quan un d'aquests *pid_sockets* és creat el que el client te que fer en primer lloc és enviar la petició de *login*, entre que es connecta i que envia la petició te un temps màxim de 3,5 segons sinó el procés que escolta el seu sockets és destruït, per tant es tenca el socket automàticament.



- S'ha creat un proces que escolta el socket d'un usuari

En cas que *el pid_socket* enviï la petició de *login* dins el *timout* (temps d'espera de 3,5seg.), el *servidor_adm* és l'encarregat de comprovar les dades, per fer-ho crea un procés en que ell és l'encarregat de comprovar les dades, així obtenim més paralització. Si aquestes són incorrectes envia un missatge de *login* incorrecta al client a través del *pid_socket*, en cas contrari li envia un missatge conforme el *login* ha estat correcta.

En el missatge de *login*, un dels arguments que te és el Pid del grup general perquè el client pugui mantenir converses amb tots els usuaris. Si el grup general no existeix significa que no hi ha cap usuari connectat, per tant el que fa el *servidor_adm* és crear-lo.

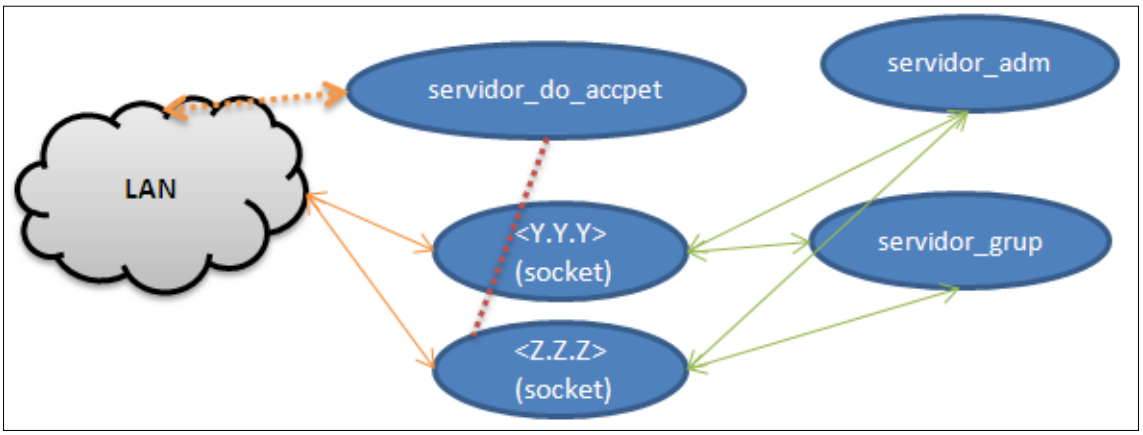


- Es crea el grup general en el qual estaran tots els usuaris

Un cop arribats aquest punt ja es poden començar a enviar missatges en el grup general, o sigui al procés servidor_grup. Quan el client envia un missatge al servidor sempre el primer argument és a qui va dirigit aquest missatge (mirar protocol), llavors l'encarregat de enviar el missatge a un o altre procés és el *pid_socket*.

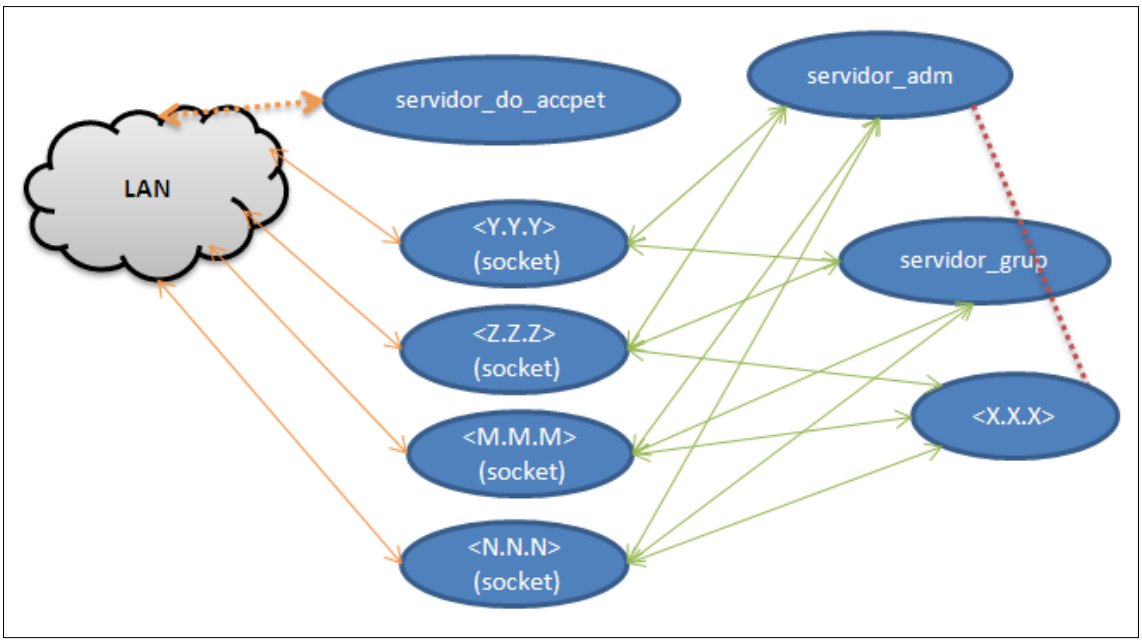
Un cop arribats aquest moment només es poden crear més processos si sorgeix alguna de les següents situacions:

- Es connecta un altre usuari, en aquest cas es crear un *pid_socket*, el qual escolta totes les peticions que l'usuari envia pel socket.



- Es connecta un altre usuari

- L'altre cas és que s'iniciï una conversa amb més de dos usuaris, llavors es crearà un procés com el servidor_grup, però en aquest cas no li donem cap nom (no el registrem) ja que no és un procés principal.



- S'inicia un altre grup de conversa

Si ens hi fixem s'ha dit que nomes es crearia el procés que gestiona els grups en el cas que es vulgui iniciar una conversa amb un mínim de 3 usuaris, això és degut que el client és capaç de realitzar converses client a client independents. En el cas que el client sol·liciti en el servidor crear una conversa amb nomes 2 usuaris (ell i un altre) el servidor envia a cada un una resposta diferent, ja que un tindrà que escolta per un port i l'altre tindrà que connectar-s'hi.

Un dels altres processos que es crea però que no s'ha dit és un procés el qual ens mostra una finestra amb un missatge d'error. En el cas del servidor és molt difícil que se'ns mostri perquè esta tot molt mercat, si es mostres seria més perquè s'ha programat malament alguna funció. El codi del procés es troba en el mòdul *miss_err.erl*.

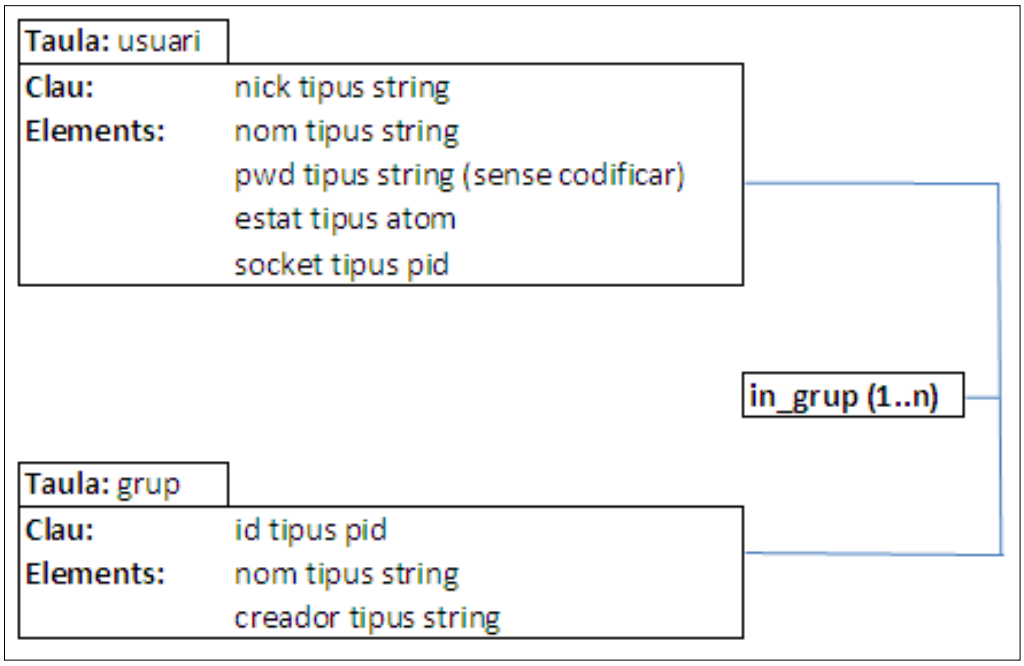
S'ha de tenir en compte també que en tot moment es disposa d'una base de dades per enregistrar en tot moment els canvis realitzats, en el següent apartat es veurà el seu funcionament i com s'ha realitzat.

Tal com s'ha vist el seu funcionament en general tampoc és mol complicat, només es tracta en la creació del procés corresponent i prou.

5.2.3.Base de dades

La base de dades tal com ja s'ha comentat ha estat desenvolupada amb la Mnesia, la qual és una base de dades que no ha estat desenvolupada per emmagatzemar grans quantitats de dades (tamanys superiors al giga), sinó que és per gestionar petites dades que demanen molts canvis i ràpids.

Per la realització de l'aplicació s'ha intentat utilitzar gran part de les opcions que proporciona la base de dades, en un primer lloc observem el diagrama de la gestió de dades:



- Gràfic de la base de dades del Servidor

Tal com s'observa es tenen dos taules, la *usuari* i la *grup* i una relació la *in_grup*. La relació in grup és una relació 1 a molts, això vol dir que es pot tenir un usuari en més d'un grup, però no un mateix usuaris repetit en un grup.

Per realitzar la creació de taules i la relació amb Mnesia és molt senzill, de fet tot són taules, si, inclús la relació es tracta d'una taula, en que la seva clau és el *pid* de la taula *grup* i l'element el *nick* de la taula *usuari* (evidentment necessitem la clau primària de cada taula), amb la única diferencia de les opcions. Seguidament s'expliquen algunes de les opcions:

- **set**: Una única clau per element de la taula.
- **bag**: Per cada element de la taula la clau es pot repetir, però no la tupla (la resta d'elements tenen que ser únics).
- **disc_copies**: Quan es crea una nova entrada en la taula es guarda a memòria RAM i a DISC.
- **ram_copies**: Les entrades de la taula es guarden nomes a memòria RAM.

I les opcions per cada taula són:

- **Taula usuaris**: Tipus *set* i *disc_copies*
    - Tipus *set* perquè evidentment no es podrà tenir més d'un usuari amb la mateixa clau. La clau és el nick perquè es considera que el nick te que ser únic. El tipus *dis_copies* be donat perquè es tracta d'una taula la qual les seves dades tenen que esta sempre disponibles, encara que el servidor es tanqui per

algun error o encara que el computador on esta es pari de cop, per aquest motiu es guarden les dades a memòria de disc

- **Taula grup**: Tipus *set* i *ram_copies*
  - Aquesta és tipus *set* perquè no podrem tenir processos repetits amb el mateix Pid. En aquest cas aquesta taula se'n fa copia nomes a memòria Ram perquè al tractar-se d'una taula la qual s'aniran creant elements, eliminant i consultat molt sovint es necessita un accés ràpid. I es considera que si el servidor "peta" per algun error al reiniciar-lo nomes importarà la taula *usuari.*
- **Taula in_grup**: Tipus *bag* i *ram_copies*
  - Aqueta per altra banda és del tipus *bag* perquè es poden tenir molts usuaris en un mateix grup, però un usuari no pot estar repetit en un mateix grup, per aquest motiu la clau principal és el pid del grup i la secundaria el nick de l'usuari. La copia es fa nomes a Ram es fa pel mateix motiu que la taula grup, per la gran quantitat d'accessos que es faran i perquè es considera que si el servidor es "mort" només ens importa la taula *usuari*.

5.2.4.Mòdul servidor.erl

Aquest mòdul te les com a funcions públiques, start/1, stop/1, stop/0, expulsar_usuari/1 (el /X indica el nombre d'arguments d'entrada de la funció).

I està dividit en les següents parts:

- **Iniciar i parar el servidor:**
  - start(Port)
    - Pre-condició: El Port te que ser un número que faci referència a un port de comunicació de la xarxa.
    - Post-condició: Retorna el Pid del procés servidor_adm creat.
  - stop()
    - Pre-condició: -
    - Post-condició: Si el procés amb el nom *servidor_adm* existeix el l'atura
  - stop(Pid)
    - Pre-condició: (en principi és pel *servidor_adm*)
    - Post-condició: Envia un missatge de sortida en el procés que te el Pid
- **Recepció de dades per part del servidor:**
  - init(Port)
    - Inicia la base de dades i activa el flag per els missatges d'eliminació de processos.
  - loop()
    - Funció que executa el servidor, bucle infinit de recepció de dades.
- **Creació de processos per la gestió del servidor:**
  - Totes aquestes funcions són el cos de processos independents, per això comencen per *proc*.
  - proc_login(Origen,Nick,Pwd)

        - Comprova les dades de l'usuari i si són correcte modifica els paràmetres de la base de dades i li envia la resposta al client.
    - proc_canvi_estat(Origen,Estat)
        - Realitza un canvi d'estat del client, modifica la base de dades.
    - proc_list_usr(Origen)
        - Consulta la base de dades per enviar una llista de tots els usuaris.
    - proc_create_grup(Origen,Usuari,Ip, Port)
        - Genera les dades necessàries per crear una conversa client a client.
    - proc_logout_serv(Origen)
        - Modifica la base de dades i si s'escau elimina processos de comunicació i gestió de grups.
    - proc_logout_grup(Origen,Grup)
        - Modifica la base de dades i si s'escau elimina processos de gestió de grups.
- **Enviament de dades en processos concrets**
    - enviar_logout(Pid,Nick)
        - Envia un missatge de logout en els grups corresponents on esta l'usuari
    - enviar_estat(Pid, Arg)
        - Envia el nou estat en els clients.
    - enviar_create_grup_ok(Pid, Nom, Taula)
        - Envia en els clients corresponents el missatge de la creació correcta del grup
- **Comprovacións**
    - comprovar_grup(Pid)
        - Comprova si queda algun usuari en el grup Pid
- **Operacions amb usuaris connectats**
    - expulsar_usuari(Nick)
        - Pre-condició: -
        - Post-condició: Si l'usuari amb nick Nick esta connectat elimina els seu procés de comunicació provocant així la seva expulsió

5.2.5.Mòdul grup.erl

La seva funció publica és gestio_grup/0.

I està dividit en les següents parts:

- **Gestió del grup:**
    - gestio_grup()
        - Pre-condició: -
        - Post-condició: Entra en un bucle infinit el qual gestiona les dades necessàries pel correcta funcionament d'un grup en el servidor. Les funcions bàsiques són el login, logout d'usuaris i enviament de dades a tots ells.

- **Envia dades als diferents membres:**
  - send_data(Comanda, Arg)
    - Consulta a la base de dades els usuaris i executa la funció send_data(Taula,Comanda,Arg)
  - send_data(Taula, Comanda, Arg)
    - Envia a tots els usuaris de la Taula la comanda Comanda amb arguments Arg
  - envia_llista_usuaris()
    - Envia una llista d'usuaris actuals a tots els membres del grup

5.2.6.Mòdul pcoms.erl

La seva funció publica és la listen/1.

I te les parts:

- **Acceptació de peticions de connexió:**
  - listen(Port)
    - Pre-condició: El Port te que ser un número que faci referència a un port de comunicació de la xarxa.
    - Post-condició: Activa un socket per escolta peticions, un cop activat executa la funció *do_accept(LSocket).*
  - do_accept(LSocket)
    - Bucle infinit el qual va acceptant connexions, quan una connexió és accepta, crea un procés que executa la funció *init_loop_recevie(Socket).*
- **Recepció de dades pel socket:**
  - init_loop_receive(Socket)
  - Inicia la recepció de dades a través del socket, després executa *loop_receive(Socket).*
  - loop_receive(Socket)
    - Bucle per rebre dades pel socket o pel servidor.
- **Altres**
  - comprovar_desti(Desti)
    - Comprova si el procés Desti existeix. Desti pot ser una dada tipus pid o àtom (si és un procés registrat), si existeix retorna *{ok, Pid}*, sinó *error*

5.2.7.Mòdul sevidor_bdd.erl

Totes les seves funcions són publiques, ja que només en tenim per iniciar la base de dades, parar-la, crear valors, eliminar-los, modificar-los, consultar-los i crear-ne de probes.

- **Inici i parament de la base de dades**
  - start()
    - Pre-condició: -

        - Post-condició: Inicia la base de dades. Si no existeix en crea una de nova. Si ja existeix carrega les taules.
- **Creació de taules:**
    - init(LlistaNodes)
        - Funció que utilitza per crear una base de dades.
- **Cerca d'elements (tots):**
    - pregunta(usuari)
        - Retorna tots els elements de la taula usuaris
    - pregunta(usuari_nick)
        - Retorna tots els nicks o be []
    - pregunta(usuari_tots_nick_nom)
        - Retorna una tuple amb el nick i nom de tots els usuaris
    - pregunta(usuari_nick_nom)
        - Retorna una tuple amb els nicks i noms o be []
    - pregunta(grup)
        - Retorna tots els elements de la taula grup o be []
    - pregunta(grup_id)
        - Retorna tots els id's de la taula grup o be []
    - pregunta(in_grup)
        - Retorna tots els elements de la taula (relació) in_grup o be []
- **Cerca d'elements (determinats):**
    - pregunta(usuari, Nick)
        - Retorna tot l'element usuari que coincideix amb el Nick, sinó []
    - pregunta(usuari_socket, Nick)
        - Retorna el socket d'un usuari que coincideix amb el Nick, sinó []
    - pregunta(usuari_nick, Socket)
        - Retorna el nick d'un usuari que coincideix amb el Socket, sinó *fail*
    - pregunta(grup, Id)
        - Retorna tot un element grup que coincideix amb Id
    - pregunta(in_grup_nick, Id)
        - retorna els nick's dels usuaris d'un determinat grup o []
    - pregunta(in_grup_nick_nom_estat, Id)
        - Retorna una tuple amb el nick i nom de tots els usuaris d'un determinat grup o []
    - pregunta(in_grup_socket, Id)
        - Retorna els sockets d'un determinat grup o []
    - pregunta(in_grup_grup, Nick)
        - Retorna els grups on esta un determinat usuari o []
- **Comprovació d'elements:**
    - pregunta(login, Nick, Pwd)
        - Comprova, si existeix, si tot ok retorna *ok*, sino *fail_sock, fail_pwd, fail_usr.*
- **Creació d'elements:**
    - crear_usuari(Nick, Nom, Pwd)

        - Crea un element de la taula usuari.
    - crear_grup(Id, Nom, Creador)
        - Crear un element de la taula grup.
    - crear_grup_usuaris(Id, Nom, Creador, Llista_usr)
        - Crear un grup i afegeix els usuaris de la Llista_usr com a membres.
    - afegir_usuaris_grup(Id, Taula)
        - Afegeix al grup amb id Id els usuaris de la Taula
- **Esborrat d'elements:**
    - esborrar(usuari,Nick)
        - Esborra un usuari amb clau Nick.
    - esborrar(grup, Id)
        - Esborra un grup amb clau Id.
    - esborrar(in_grup_usuari, Nick)
        - Esborra totes les relacions de in_grup que contenen l'usuari.
    - esborrar(taules, Llista)
        - S'esborren totes les taules de la llista.
    - esborrar(in_grup_usuari, Id, Nick)
        - Elimina un determinat usuari d'un grup de la relació in_grup.
- **Modificació d'elements:**
    - modifica_sockets(Socket, Taula)
        - Modifica els socket dels usuaris de la Taula
    - modifica_estats(Estat, Taula)
        - Modifica l'estat dels usuaris de la Taula
- **Reset de prova:**
    - reset_tables()
        - Esborra les taules i posa els les que es troben a *example_tables().*
    - example_tables()
        - Conte unes taules d'exemple.

5.2.8.Mòdul miss_err.erl

Funcions públiques init/1 i trans_dades/2.

- **Conte les funcions:**
    - init(Llista)
        - Pre-condició: La llista te que ser una llista d'elements simples. No pot contenir una altre llista amb altres elements a dins. Poden ser números, tuples, strings, bits, pid's, ports, sockets,....
        - Post-condició: Genera un wdiget que mostra un missatge d'error amb una imatge. El missatge és el contingut de la llista.
    - trans_dades(Llista)
        - Transforma totes les dades de la llista Llista en un string
    - trans_dada(Dada)
        - Transforma una dada en un string.

5.3.Client

El client a diferencia del servidor és una mica més "complicat", pel fet que el client esta format per més mòduls que el servidor.

S'ha intentat fer que cada procés tingui un mòdul apart amb les seves funcions que ell utilitza i amb el que ell fa, així d'aquesta manera és una mica més entenedor (tal com s'ha vist al capítol passat)

Les llibreries utilitzades en el servidor han estat:

- **erlang**: És la llibreria principal, s'utilitza per la majoria de coses, crear processos, destruir, comprovar, etc...
- **io**: Aquesta és l'encarregada de mostrar el text per la sortida corresponent, en el nostre cas la pantalla (és com el *printf* en C).
- **lists**: Per la gestió de llistes.
- **string**: Aquesta és per la gestió dels elements tipus string.
- **inet**: Aquesta llibreria s'utilitza nomes per obtenir la ip del client.
- **gs**: Per la gestió de la interfície gràfica.

5.3.1.Disseny

Pel que fa el disseny del client es una mica diferent que el del servidor però molt similar.

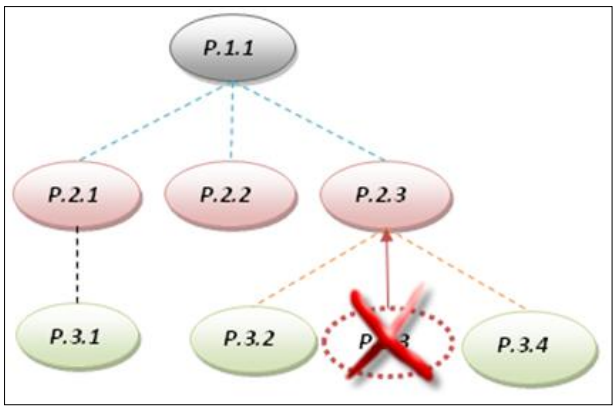
Una de les diferenciés és que els processos que es crearan variaran segons el protocol, en tindrem uns si utilitzem el protocol local i uns altres si utilitzem el IRC, però en essència els dos son els mateixos.

El client també es pot dividir en grups, en concret 4.

Tenim un grup (el número 2 en les imatges de sota) grup format pel procés *client_adm*, el qual és l'encarregat de gestionar els una part dels processos principals, algun cop s'anomenarà **client**.

L'altre encarregat de gestionar alguns altres processos són el *client_pirc* ó el *client_plocal* (alguns cops rebran el nom de **pirc** o **plocal** corresponentment, número 3). Aquests són els processos encarregat de descodificar el que reben del servidor (local o IRC), però també gestionen els seus processos, els processos els quals ells necessiten per funcionar.

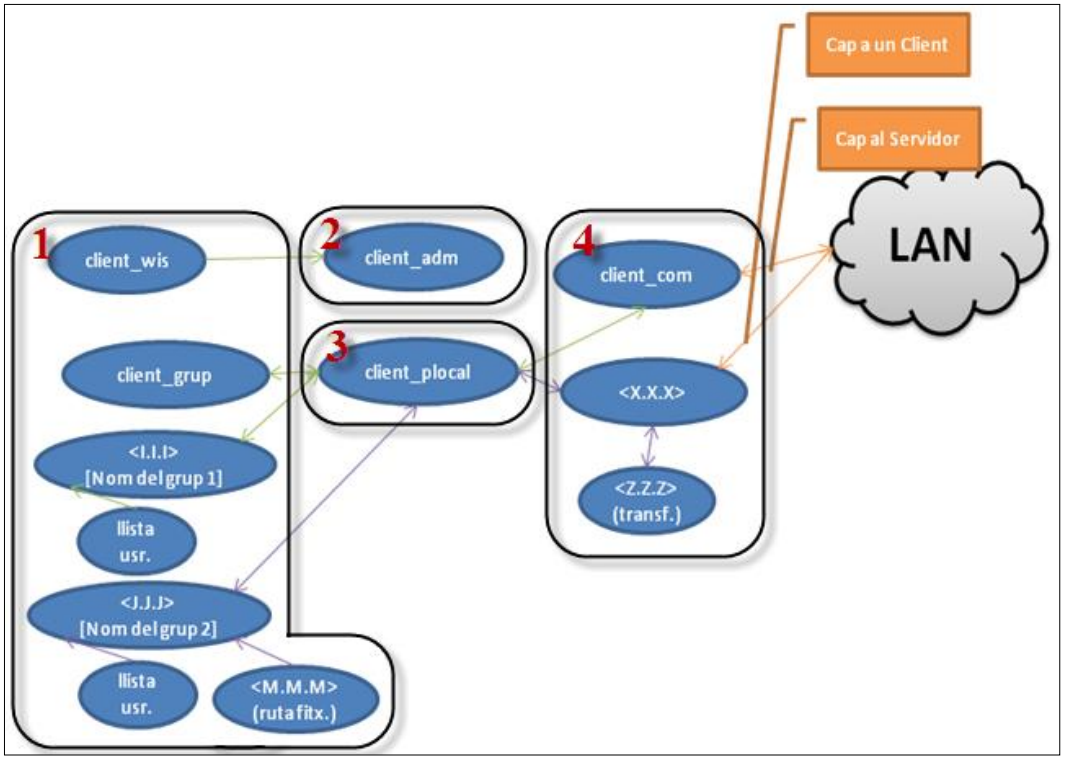
Podríem dir per tant que el tema de la gestió i tractament d'error no només depèn d'un sinó de dos. Ho podríem veure com un arbre, el pare gestiona els errors dels seus fills i aquets els que ell crea. Això també passa amb el servidor, però no es veu tan bé.



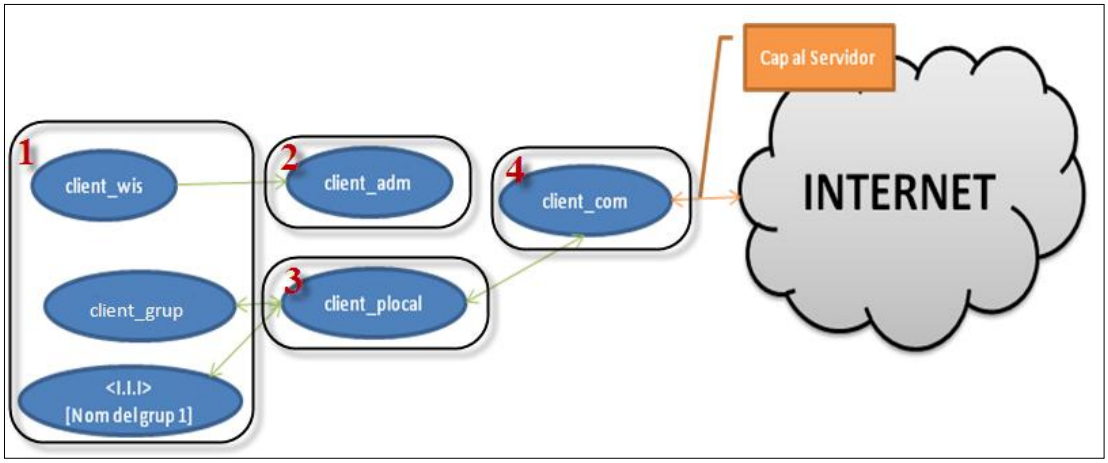
- Exemple sobre el tractament d'errors en processos

Un dels altres dos grups que queden és el que es podria anomenar GUI (*Graphics User Interface*) el qual és el que te totes les finestres on l'usuari interactua amb l'aplicació, algunes vegades cada un d'aquest processos s'anomenarà **widget**,en les imatges del final te el número 1. En aquet grup també es trobarà un widget el qual està registrat amb un nom, aquest és el *client_grup*.

L'últim grup és el de la comunicació, és el procés encarregat de transmetre les dades per la xarxa, aquest cas és similar el del servidor, hi ha un procés general que rep el nom de *client_com* i si es tracta del protocol local hi poden haver també altres processos de comunicació, al igual que el servidor alguns cops s'anomenaran amb el nom de **pid_sockets**, el 4 ens les imatges.



- Imatge dels processos en un moment concret del client amb protocol local

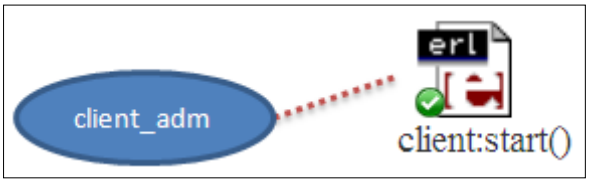


- Imatge dels processos en un moment concret del client amb el protocol IRC

5.3.2.Funcionament – Elecció Protocol

El funcionament del client potser és una mica més complicat que el del servidor, ja que amb aquest es tenen dos protocols diferents hi depenent de quin s'escolli es farà una cosa o una altre, però el principi, fins que no s'escull el protocol tenen el mateix funcionament.

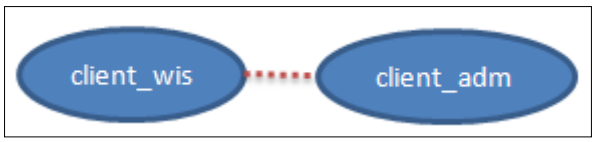
Tot s'inicia amb la maquina virtual Erlang, el mòdul *client.erl* executa la funció *start().* Aquesta funció el que fa és crear el procés *client_adm*, el qual és l'encarregat de gestionar els errors dels seus processos.



- Es crea el procés client_adm

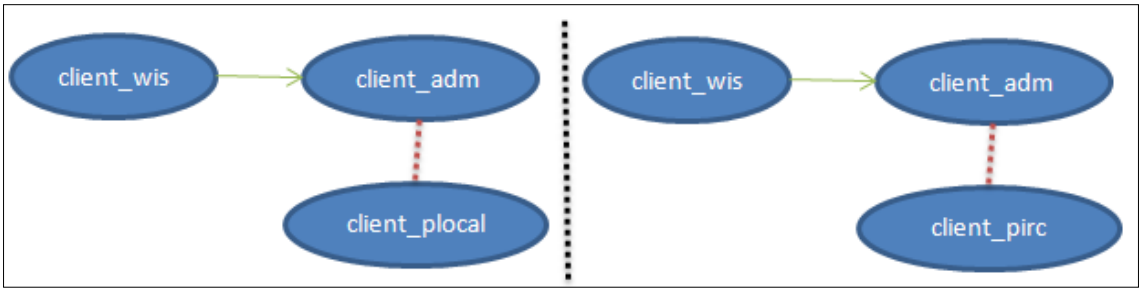
Un cop es te el *client*, aquet és l'encarregat de crear el procés que s'encarrega de l'entrada de dades segons el protocol, aquest procés és un dels que s'anomenen diuen widgets.

Aquí podem escollir en quin dels protocols ens volem connectar (observar imatge), depengen de quin escollim s'hauran d'emplenar una dades o unes altres.



- Es veu la creació del widget client_wis i dos imatges d'ell

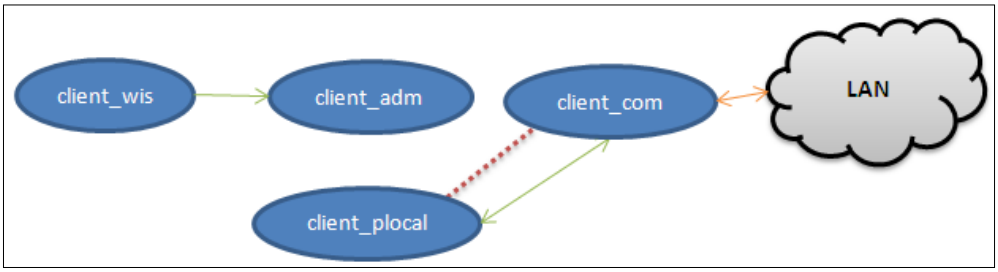
Un cop emplenades les dades el client crea el procés que gestinara les dades rebudes segons el protocol, client_plocal o el client_pirc.
Per veure el funcionament a partir els apartats 5.3.3 i 5.3.4 descriuen el funcionament en el cas del protocol local i el IRC respectivament.



- Es crea el procés client_plocal o el client_pirc

5.3.3.Funcionament - Protocol Local

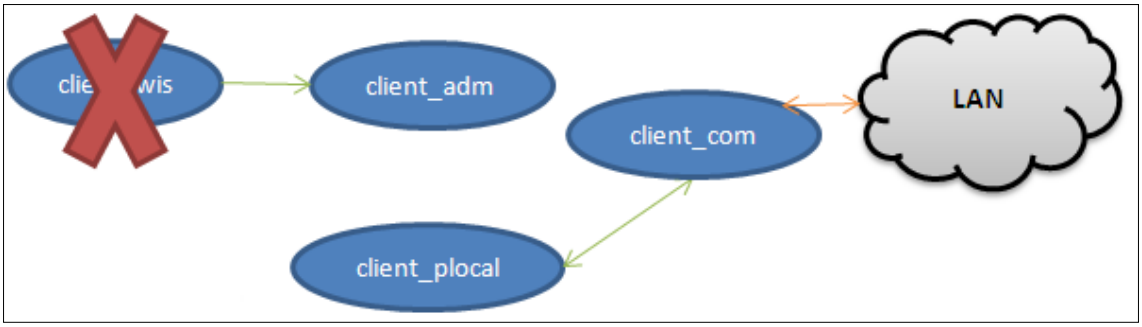
Un cop creat el procés encarregat de gestionar-ho tot perquè el protocol local funcioni correctament, el primer que fa és crear el procés encarregat de la comunicació amb el servidor, el client_com.



- Es crea el client_com

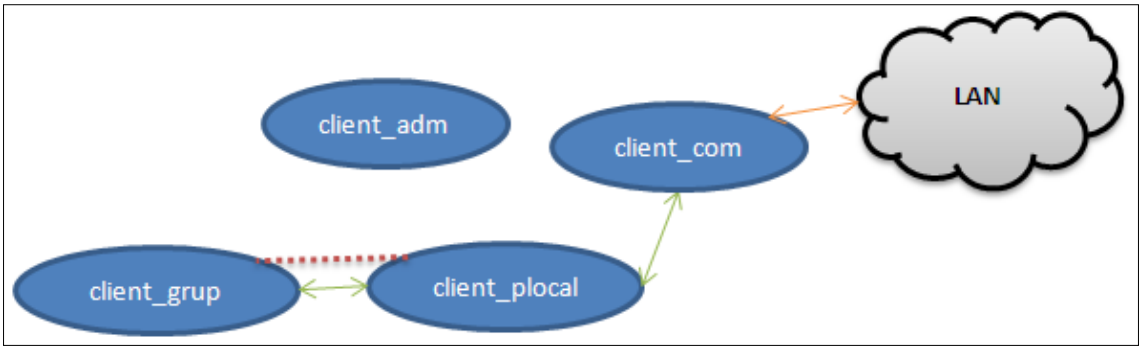
Aquest procés el primer que fa és connectar-se amb el servidor, si per algun motiu el port entrat o la ip son incorrectes envia un missatge al *plocal* i s'autodestrueix, llavors el *client_plocal* al rebre aquest missatge de connexió errònia ell s'autodestrueix i com a motiu dona l'error rebut. Finalment el *client_adm* rep el senyal de destrucció d'un procés l'interpreta i si cal mostra el missatge d'error corresponent.

Per altra banda si la connexió és correcta s'envien les dades de registra (login) al servidor. En el cas que siguin incorrectes el *plocal* mostra un missatge d'error i es destrueixen el *client_com* i ell. En canvi si el login ha estat correcta el *client_plocal* envia un missatge al *client_adm* conforme ja pot destruir el *client_wis*.



- Si les dades són correctes es destrueix el client_wis

Un cop el login ha estat correcta el que fa el servidor és enviar les dades per crear el widget que contindrà la conversa general, registrat amb el nom de *client_grup*. S'ha de tenir en compte, tal com s'explica en el manual que si pel que sigui es tenca aquest widget al ser el general (com una llista d'usuaris, però amb conversa tots a tots) es destrueix tot i es desconnecta del servidor.



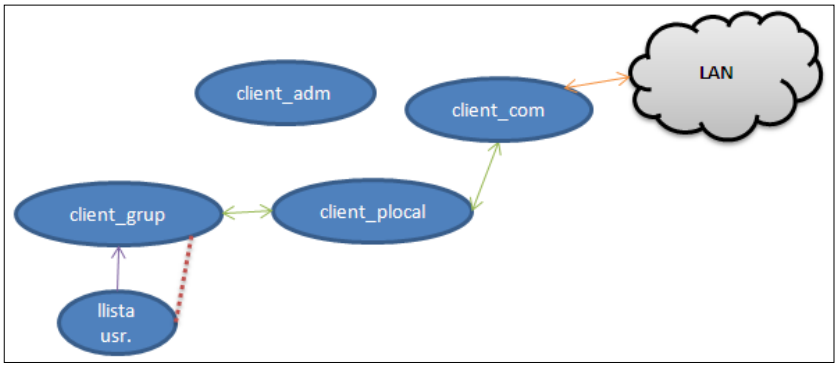
- Creació del client_grup

- Widget de la conversa general

Un cop arribats aquest punt es poden realitzar diverses accions, una d'elles per exemple és canviar l'estat el qual ens trobem, l'estat pot estar en disponible, absent o ocupat. Per realitzar un canvi simplement s'envia el missatge al servidor conforme volem realitzar un canvi d'estat.

L'ajuda simplement crea un altre widget que ens informa sobre l'aplicació.

L'altre acció és realitzar una conversa en grup, per la conversa en grup el que es fa és envia un missatge al servidor conforme ens enviï una llista d'usuaris. Un cop rebuda es crea un procés que conte aquesta llista i podem seleccionar els usuaris amb els quals volem mentir la conversa.

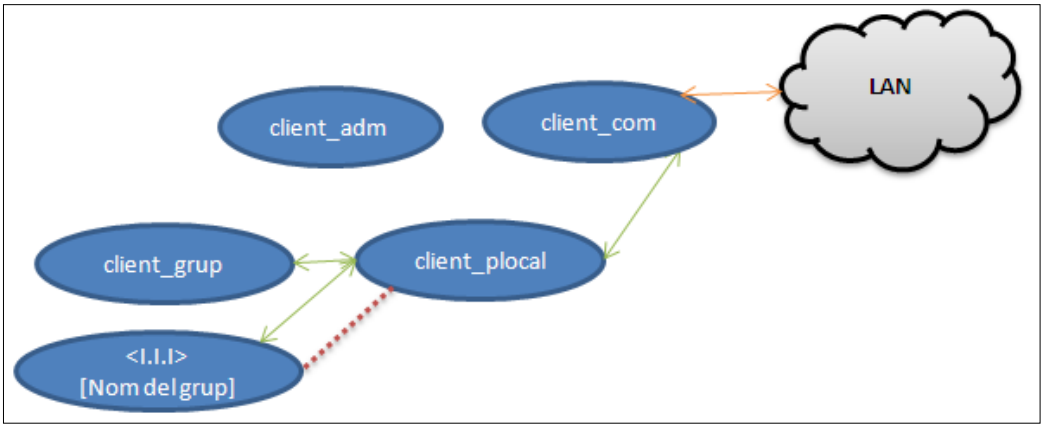


- Es crea el procés el qual ens mostra la llista d'usuaris (esquerra), a la dreta un exemple del widget

En aquest moment hi han dos possibilitats, realitzar una conversa en grup o bé realitzar una conversa un a un. Quan es te aquest widget es seleccionen els usuaris amb els quals es vol mantenir una conversa, llavors quan se li dona al *Ok* s'envia un missatge al *plocal* amb la llista

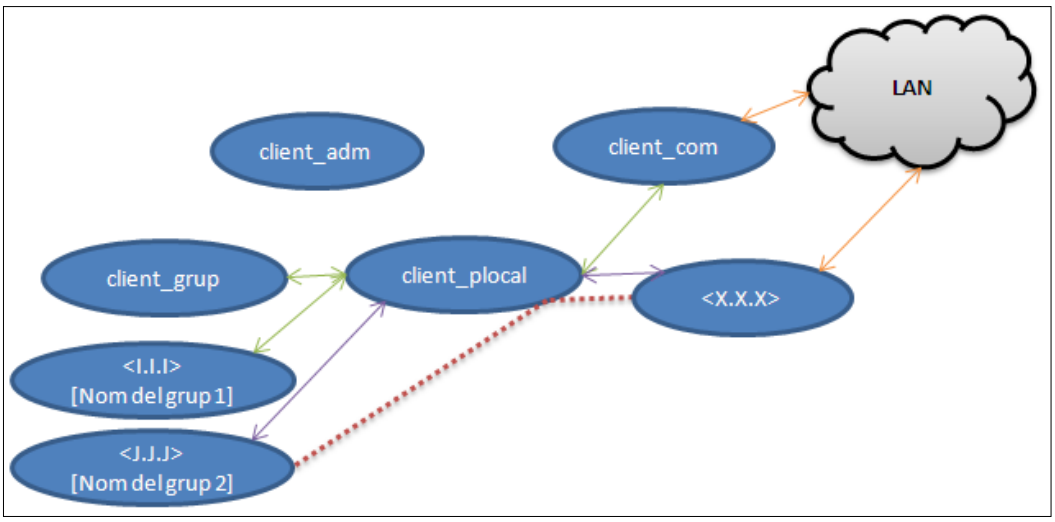
d'usuaris, si veu que és superior a dos envia una petició al servidor per realitzar una conversa client a client sinó envia una petició per realitzar una conversa en grup.

- **Conversa en grup:** Si la petició és acceptada rebrem la contestació, amb aquest el que es fa és crear un altre procés el qual és un altre widget per mantenir conversacions similar en el de la conversa general amb la única diferencia que en aquest cas nosaltres escollim el nom i els usuaris.



- Creació del procés <I.I.I>

- **Conversa un a un:** En el cas que escollim nomes un usuari el que es fa és envia una petició al servidor, aquest ens contesta amb un nom de grup a qui dirigí les converses. Llavors es crearà el procés per mantenir la conversa amb l'altre usuari. Aquest procés sera similar al *client_com*, amb la principal diferencia que el servidor ens dirà si es te que escoltar peticions de connexió o bé es te que realitzar la connexió a l'altre client. S'ha de tenir en compte que el que escolta les connexions té un timout i el que es connecta també te un timout i límit de intents.
  Si la connexió surt bé es crea un nou widget per mantenir la conversa amb l'altre usuari, aquests és una mica diferent que el de les converses amb més de dos usuaris ja que no hi ha la llista dels usuaris amb qui parlem, sinó que a la part superior, en el nom de la conversa hi apareixen els dos nicks de les persones que estan parlant.



- Creació d'una conversa client-client (primer es crea el <X.X.X> i després el <J.J.J>

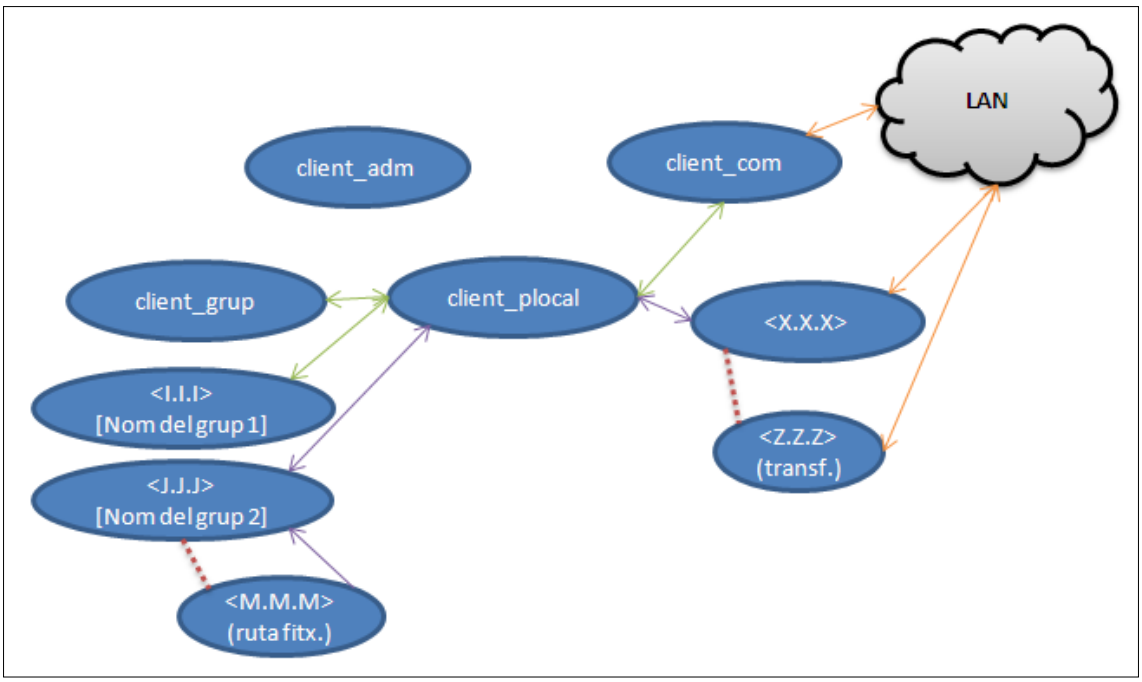
El principal avantatge de mantenir una conversa client a client és que es dona la possibilitat de transferir fitxers.

Quan s'escull la opció de transferir un fitxer el primer que es fa és crear un procés que és un widget. La principal funció del widget és entrar la ruta del fitxer i quan és dona al *Ok* aquest procés envia un missatge al *plocal* i s'autodestrueix.

El *plocal* envia un missatge en el corresponent procés de comunicació i aquest inicia els passos per transferir el fitxer. El primer que fa és comprovar que el fitxer existeixi en l'ordinador, si no existeix es mostra un missatge d'error. En cas contrari s'envia una petició de transferència a l'altre client.

L'altre client comprova que el fitxer no existeixi en la carpeta de *fitxers rebuts* si es així denega la transferència i els dos usuaris mostren el missatge d'error corresponent. En el cas que aquesta carpeta no existeixi es crea de nou.

En canvi si tot surt bé es crearà un procés que va lleguin el fitxer a transferir i va enviant les dades directament pel socket. L'altre el que fa és també crear un procés però en comptes de llegir un fitxes l'escriu.

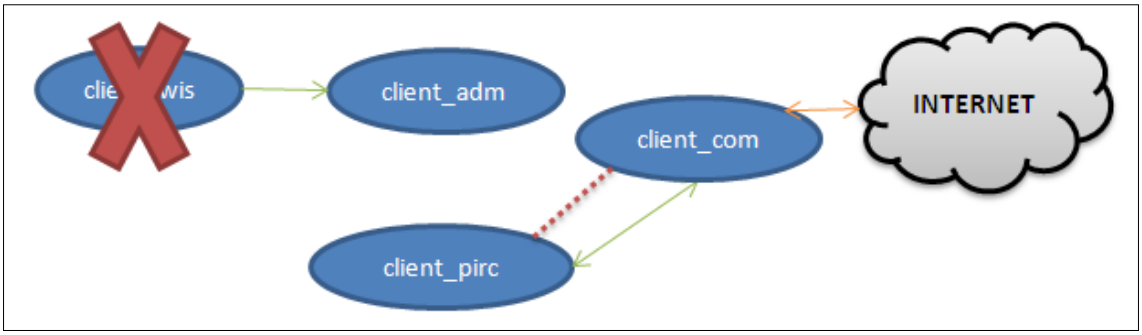


- Transferència de fitxers, primer es crea el <M.M.M> i després si tot surt bé el <Z.Z.Z>

5.3.4.Funcionament - Protocol IRC

El funcionament del xat amb el protocol IRC és bastant més senzill que en el local, ja que aquest no inclou ni converses client a client ni transferència de fitxers.

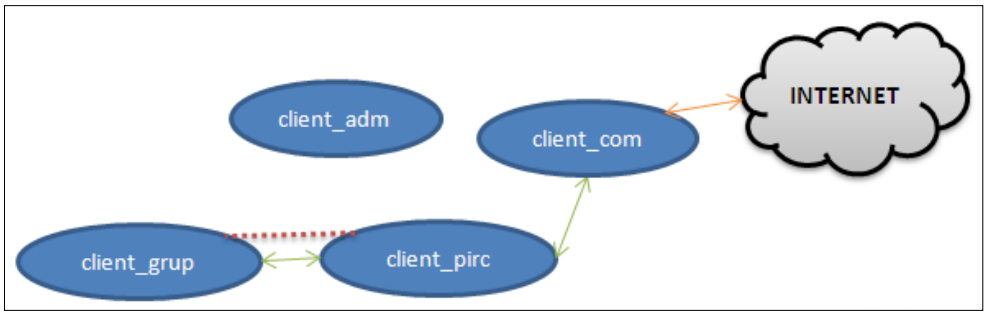
En un primer lloc al igual que en el local, un cop creat el *client_pirc* es procedeix a crear el *client_com*, aquest és l'encarregat de en un inici, connectar-se amb el servidor, i un cop connectats entrar en el canal.



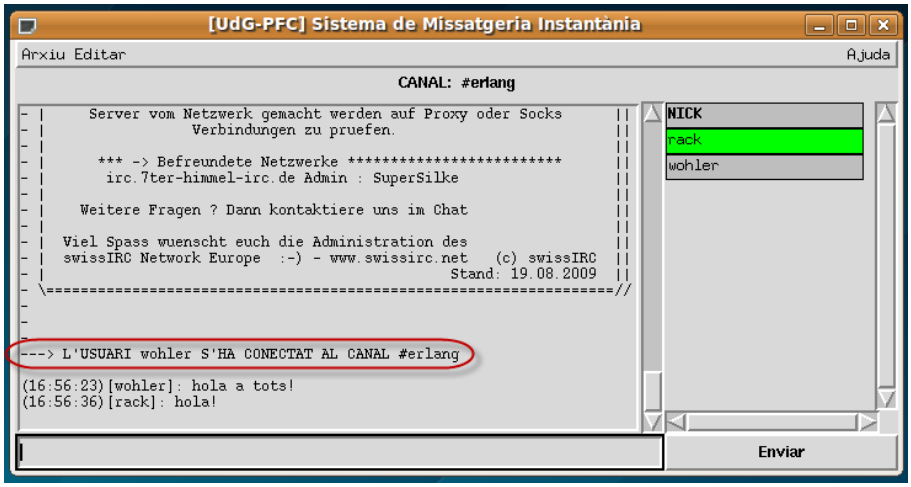
- Es crea el procés de comunicació i es destrueix el d'inici de sessió

Un cop es connecta amb el servidor es crea el widget principal. Aquest és el que conté la conversa tots a tots del IRC, el *client_grup*. Si per algun motiu falles la connexió amb el servidor (alguna comanda mal entrada) apareixeria la finestra amb la conversa general i mostraria el missatge d'error.

Això esta fet perquè s'ha preparat l'aplicació perquè l'usuari tingui la possibilitat de entrar ell les comandes a enviar al servidor.

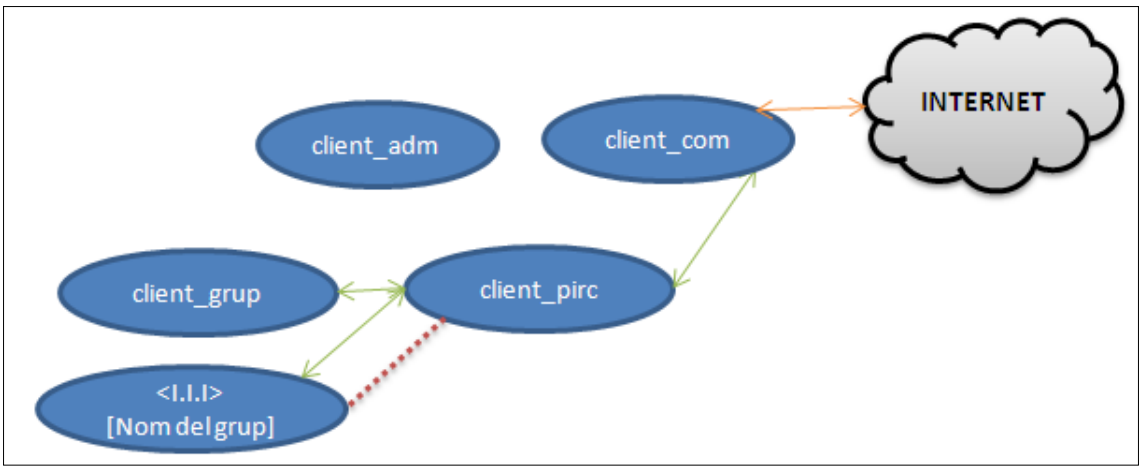


-Creació del client_grup



- Exemple conversa general, i connexió correcta

Si la connexió resulta satisfactòria i entrem en el canal es podran mantenir converses amb usuaris individuals, com un client a client del local però en aquest cas els missatges passen pel servidor. Per tant per fer això l'únic que es fa és crear un altre widget però només serveix per mantenir una convers amb un altre usuari.



- Es crea un altre widget per poder mantenir una conversa privada amb un altre usuari

El funcionament del client en el cas del protocol IRC s'acaba aquí. Tal com es veu és molt més senzill que el local, però també s'ha de tenir en compte que es poden realitzar menys coses.

5.3.5.Mòdul client.erl

Les seves funcions públiques és start/0

I esta dividit pels apartats:

- **Inici del procés encarregat d'administrar el client:**
  - start()
    - Pre-conidició: -
    - Post-condició: Inicia el procés *client_adm* i el registre.
- **Inici i bucle per la recepció de dades del client:**
  - init()
    - Activa els flag per la recepció de la destrucció d'un procés, crea el procés *client_wis*, i executa la funció *loop_receive*.
  - loop_receive()
    - Recepció i gestió de dades *del client_adm*.

5.3.6.Mòdul plocal.erl

La seva funció pública és init/4. Consta de un *define* el qual indica el port per la transmissió de fitxers.

Esta compost per les parts:

- **Inici del protocol local:**
  - init(Ip,Port,Nick,Pwd)

  - Pre-condició: La ip Ip i el port Port tenen que ser números o la ip un string. El Nick i el Pwd strings.
  - Post-condició: Activa el flag, crea el procés de comunicació amb el servidor i si la connexió i login son correctes entre el *loop(Llista)* i crea la conversa general.

- **Bucle de recepcio de dades del protocol local:**
  - loop(Llista)
    - Bucle de recepció de dades.
- **Connect i listen per les converses client-client:**
  - grup_connect(Grup,Ip,Port,Pid,Nick,Intents,Usuari)
    - Fa uns intents de connexió a la ip IP amb port Port.
  - grup_listen(Grup,Port,Pid,Nick,Usuari)
    - Escolta un socket pel port Port.
- **Enviament de missatges als processos:**
  - envia_al_widget(Llista, Grup, Comanda, Arg)
    - Envia una comanda al widget.
  - envia_al_servidor(Llista, Wid, Comanda, Arg)
    - Prepara una comanda pel *client_com* perquè l'enviï al servidor.
  - envia_al_grup(Llista, Wid, Comanda, Arg)
    - Prepara una comanda pel *client_com* perquè l'enviï al grup.
  - bloquejar_widget(Pid,Llista)
    - Bloqueja un widget.
  - bloqueix_total_widgets(Taula)
    - Bloqueja una llista de widgets.
- **Operacions amb la llista del client_plocal:**
  - extreure_grup(Wid, Llista)
    - Extreu un grup de la llista.
  - extreure_grups_cc(Llista)
    - Retorna tots els grups que són del tipus string.
  - extreure_widget(Grup, Llista)
    - Extreu un widget.
  - extreure_widget_2(Pid_com, Llista)
    - Extreu un widget.
  - eliminar_widget(Wid, Llista)
    - Elimina un element de la llista.
  - busca_conversa(Usuari,Llista)
    - Busca en la llista el grup que conte part de l'string Usuari
  - busca_conversa_grup(Grup,Llista)
    - Retorna l'element que conte Grup de la llista
  - canviar_grup_llista(GrupAct,Grup,Pid_com,Llista)
    - Canvia un grup de la llista
- **Varis:**
  - obtenir_ip()
    - Retorna una ip diferent a la local 127.0.0.1

5.3.7.Mòdul pirc.erl

La seva funció publica és la init/6

Esta dividit en les següents parts:

- **Inici i bucle del procés encarregat de descodificar el protocol IRC**
  - init(Ip,Port,Canal,Nick,Nom,NomUsr)
    - Pre-condició: La ip Ip i el port Port tenen que ser números o la ip un string. El Nick, Nom i NomUsr strings.
    - Post-condició: Activa el flag, crea el procés de comunicació amb el servidor i entre el *loop(Ant,Llista)* i crea la conversa general.
  - loop(Ant,Llista)
    - Entre en el bucle de recepcio de dades del protocol local.
- **Execució d'una o més comandes**
  - executar_comanda(Data,Llista)
    - Executa una o més comandes que estan a Data.
  - executar_comanda(Pre,Com, Arg, Seg, Llista)
    - Executa la comanda Com.
- **Enviament de missatges als diferents processos**
  - enviar_al_widget(Grup,Nick,Comanda,Dades,Llista)
    - Envia la comanda en el widget.
  - enviar_al_general(Comanda,Dades)
    - Envia la comanda en el *client_general*.
  - enviar_al_serv(Dades)
    - Envia Dades al *client_com* perquè les enviï al servidor.
  - enviar_al_pcom(Comanda,Dades)
    - Envia una comanda Comanda amb argument Dades al *client_com*.
  - destruir_el_pcom()
    - Destrueix el *client_com.*
  - eliminar_widget(Wid, Llista)
    - Destrueix un widget.
  - bloquejar_widgets(Llista)
    - Bloqueja els widgets de la llista Llista.
- **Descodificació de comandes**
  - split_once(String, Char)
    - Separa la prefix de la resta de la comanda.
  - parse_args(String)
    - Busca el primer " :" de String i executa *parse_args(String,Idx).*
  - parse_args(String, Idx)
    - Separa els argument de la comanda.
  - next_command(Data)
    - Si existeix més d'una comanda la separa en dos.
  - scan_string(Data)

        - Separa una Data en el prefix, comanda, arguments i següent comanda.
    - is_command(Data)
        - Mira si Data és una comanda completa del protocol IRC
    - buscar_final_comanda(Com,Data)
        - Busca el final de la comanada Com en Data.
    - buscar_final_comanda(Com,Data,Ret)
        - Busca el final de la comanada Com en Data.
    - ajuntar_comanda(Pre,Com,Arg,Seg)
        - Ajunta una comanda.
    - ajuntar_comanda_namerply(N,Temp,Arg)
        - Ajunta una comanda.

5.3.8.Mòdul pcomc.erl

Té com a funcions públiques connect/4,listen/2. També ho són totes les de comprovació i transferència de fitxers ja que aquestes tindran que ser executades pel procés de transferència de fitxes i el *client_com.*

- **Inici del protocol de comunicació**
    - connect(Prot,Pid,Ip, Port)
        - Pre-condició: Prot te que ser o bé "local" o "irc". Pid és el pid el qual s'enviaran els missatges. Ip i Port són la ip i port a connectar-se.
        - Post-condició: Si la connexió amb el servidor és correcta executa el *loop_receive(Protocol,Socket,Pid)* sinó es fa un *exit(Why).*
    - listen(Pid, Port)
        - Pre-condició: Pid és el pid el qual s'enviaran les dades i Port el port per on s'escoltaran les peticions de connexió, aquest te que ser un número.
        - Post-condició: Si una connexió és acceptada executa *loop_local(Socket,Pid).*
    - loop_receive(Protocol,Socket,Pid)
        - Executa la funció *loop_irc(Socket,Pid) o loop_local(Socket, Pid)* depenent del protocol Protocol.
- **Comunicació amb el servidor IRC**
    - loop_irc(Socket,Pid)
        - Recepció de dades del protocol IRC.
- **Comunicació amb el servidor local**
    - loop_local(Socket, Pid)
        - Recepció de dades del protocol local.
- **Comprovació de fitxers i directoris**
    - info_tam(File)
        - Retorna el tamany del fitxer o be "error"
    - info_tip(File)
        - Retorna el tipus del fitxer o be "error"
    - nom_fitxer(Ruta)

        - Retorna el nom del fitxer d'una ruta completa.
    - comprobar_fitxer(Ruta)
        - Comprova el tipus, grandària i existència del fitxer.
    - comprobar_dir_descarga(Nom)
        - Comprova que un fitxer no existeixi en el directori de descarrega.
- **Transferència de fitxers**
    - transferir(Pid,Pare,Sock, Nom, Ruta, Tamany)
        - Calcula les parts que es tindran que enviar i executa la funció *transferir_env(Pid, Pare, Sock, Nom, Fitx, TotalParts, PartAct).*
    - transferir_env(Pid, Pare, Sock, Nom, Fitx, TotalParts, PartAct)
        - Transfereix un fitxer
    - percentatge_trans(A,T)
        - Calcula el percentatge.

5.3.9.Mòdul wis.erl

Només té com a funció publica init/1. I esta dividit per les parts:

- **Configuració del Widget**
    - init(Pid)
        - Pre-condició: Pid te que ser el pid d'un procés existent.
        - Post-condició:Configura el widget, Pid és el pid del procés el qual s'enviaran les dades
- **Recepció de dades**
    - loop(Pid,Pwd)
        - Inici de la recepció de dades
- **Configuració widgets i obtenció de dades**
    - enviar_parametres(Pid,Pwd)
        - Envia els paràmetres en el pid Pid
    - comprobacio_port(N,String)
        - Comprova el port

5.3.10.Mòdul wgr.erl

Té com a funció publica init/4. I esta dividit per les parts:

- **Configuració del Widget**
    - init(Nom, TipusConv, Pid_prot, Nick)
        - Pre-condició: Pid_prot te que ser el pid d'un procés existent, Nom i Nick tenen que ser tipus string i TipusConv només pot ser "clientclient", " grup", " irc_grup" o " irc_clientclient".
        - Configura el widget, amb un nom Nom , segons el tipus de conversa TipusCon, i enviarà les dades en el Pid_prot
- **Recepció de dades**
    - loop(Pid_prot,Nick)

        - Inici de la recepció de dades
- **Configuració widgets i obtenció de dades**
    - emplenar_llista(Row, Llista)
        - Emplena la llista d'usuaris segons el protocol local.
    - emplenar_llista_irc(Row,Llista)
        - Emplena la llista d'usuaris segons el protocol IRC.
    - obtenir_gridlines(Grid, Llista, Row)
        - Retorna l'objecte gridline.
    - eliminar_gridlines(Grid, Row)
        - Elimina les gridlines.
    - canviar_estat(Nick, Estat, Grid, Row)
        - Busca la gridline per executar la funció canviar_estat(GridLine, Estat)
    - canviar_estat(GridLine, Estat)
        - Realitza un canvi d'estat, canvi de color.
    - obtenir_hora()
        - Retorna l'hora
    - moure_scroll(Pid)
        - Mou el scrollbar del l'objecte amb pid Pid.

5.3.11.Mòdul llista_usr.erl

Com la resta de mòduls que contenen widget la serva única funció publica init/3. I esta dividit per les parts:

- **Configuració del Widget**
    - init(Pid,General,Llista)
        - Pre-condició: Pid te que ser el pid d'un procés que existeixi. General pot ser "true" o "false" i Llista pot contenir els usuaris connectats.
        - Post-condició: Configura el widget.
- **Recepció de dades**
    - loop(Pid_prot,Nick)
        - Inici de la recepció de dades.
- **Varis**
    - construir_llista(Llista)
        - Construeix la llista que mostrarà el widget.

5.3.12.Mòdul wftp.erl

La seva funció pública és init_wftp/1:

- **Configuració del Widget**
    - init_wftp(Pid)
        - Pre-condició: Pid te que ser el pid d'un procés que existeixi.
        - Post-condició: Configura el widget.
- **Recepció de dades**

- loop_wftp(Pid)
  - Inici de la recepció de dades.

5.3.13.Mòdul winf.erl

La seva funció pública és init /1:

- **Configuració del Widget**
  - init ()
    - Pre-condició:-
    - Post-condició: Configura el widget.
- **Recepció de dades**
    - loop()
      - Inici de la recepció de dades.

5.3.14.Mòdul miss_err.erl

Mirar apartat 5.2.8 ja que es tracta del mateix mòdul.

5.4.Protocol de Comunicació Local

Aquest apartat està dedicat a explicar els missatges que es poden enviar al servidor o bé que podem rebre per part del servidor. S'ha de tenir en compte que també hi ha missatge específics que es transmeten via client a client, com ara la transmissió de dades, o missatges.

Totes les dades que s'enviïn el grup a qui va dirigit (variable Grup) te que ser del tipus àtom.

En el cas del protocol local s'han de tenir unes coses en compta avanç d'enviar les dades, ja que aquestes tenen que complir una sèrie de requisits. Les dades tenen que està codificades en binary i els paquets tenen que tenir una capçalera de 4 bytes i en ordre big-endian.

5.4.1.Dades que es poden enviar al Servidor

login

{servidor_adm, login, [Nick, Pwd]}

Comanda utilitzada per iniciar una sessió amb el servidor, on *Nick* te que ser un nick registrat en el servidor i *Pwd* la seva contrasenya.

Possibles valors de retorn:

{Grup, login_ok, [Grup, Nick]}}
{Grup, login_fail, [Str]}

login_grup

{servidor_adm, login_grup, [Pid_grup]}

Al igual que l'anterior, però en aquest cas es sol·licita al servidor fer un login en un grup *Pid_grup* en concret

Valors de retorn:

{Grup, error, [Motiu]}
{ Grup, login_ok, [Grup,Nick]}

sms

{Grup, sms, [Str]}

Quan es vol enviar un missatge *Str*, que serà un string, a un *Grup* en concret

Possibles valors de retorn:

(si surt bé no retorna res)
{Grup, error, [Motiu]}

estat_actual

{servidor_adm, estat_actual, [Estat]}

Per fer un canvi d'estat. Els possibles valors de *Estat* són: 'disponible', 'absent', 'ocupat'.

Possibles valors de retorn:

{Grup, estat_actual, [Nick,Estat]}
{Grup, error, [Motiu]}

list_usr

{servidor_adm, list_usr, []}

Es sol·licita una llista de tots els usuaris actuals connectats al servidor

Valors de retorn:

{Grup, list_usr, [Usuaris]}
{Grup, error, [Motiu]}

create_grup – dos usuaris

{servidor_adm, create_grup, [Usuari,Ip,Port]}

Aquesta sol·licita al servidor permís per crear una conversa client a client a través del port *Port* amb IP *Ip* i amb l'usuari *Usuari* (*Usuari* serà el nick de l'usuari)

Possibles valors de retorn:

{servidor_adm, start_listen, [Port,Nick,Usuari,Grup]}
{servidor_adm, start_connect, [Ip,Port,Usuari,Nick,Grup]}
{Grup, error, [Motiu]}

create_grup – més d'un usuari

{servidor_adm, create_grup, [Nom, Usuaris]}

Es sol·licita al servidor crear una conversa en grup amb el nom *Nom* i la llista d'usuaris *Usuaris*, el nostra nick no formarà part d'aquesta llista.

Nom serà un string i Usuaris serà la llista dels nicks d'usuaris actualment connectats i serà de la següent manera.

Aquesta comanda també és aplicable en el cas de dos usuaris, ja que sinó no es poden transmetre fitxer i no es satura tant el servidor.

Per realitzar una conversa en grup però no n'hi ha prou, un cop s'ha creat el grup s'ha de fer un *login_grup* a aquest així s'entrarà al grup.

Possibles valors de retorn:

{servidor_adm, create_grup_ok, [Pid_Grup,Nom,Nick]}
{Grup, error, [Motiu]}

logout

- ***{servidor_adm, logout, []}***

  Comanda utilitzada per la desconecció d'un servidor. En aquest cas no rep cap resposta per part del servidor ja que el servidor el que fa és desconnectar el socket, per tant si es rep alguna cosa serà un missatge de que el socket ha estat tancat.

- ***{servidor_adm, logout, [Grup]}***

  Aquesta comanda s'utilitza per desconnectar-se d'un grup en concret. Grup te que ser el Pid d'un grup del servidor existent.

  Valors de retorn:

  (si surt bé no retorna res)
  {Grup, error, [Motiu]}

5.4.2.Dades que es poden enviar/rebre del Client

sms

{Grup, sms, [Str]}

Quan es vol enviar un missatge *Str*, que serà un string, a un *Grup* en concret.

Valors que de retorn:

(si surt bé no retorna res)
{Grup, error, [Motiu]}

Transferència

Aquestes comandes estan ordenades per números, estan ordenades de tal manera que és com si es vol enviar un fitxer en un client, però s'ha de tenir en compte que al tractar-se de una connexió client a client aquestes comandes tan poden ser enviades com rebudes.

1. ***{fitx,Nom,Ruta,Tamany}***

   Aquesta s'utilitza quan es vol enviar un fitxer amb nom Nom, ruta Ruta (la ruta on es troba en el nostre PC) i grandària Tamany

   Retorn:

   {fitx_err,Nom,Tamany}
   {fitx_ok,Nom,Ruta,Tamany}

2. ***{trans,{T,A},N,Data}***
   Aquesta s'utilitza per enviar les dades Data del fitxer amb nom N. T és la grandària del fitxer a enviar total i A és l'actual. T i A s'utilitza per calcular el tan per cent del fitxer.
3. ***{trans,{T,A},N,eof}***
   Aquesta significa que s'ha arribat al final de la transmissió del fitxer N

En el cas de la transferència de fitxers trobem dos missatges de que ens poden retornar:

- ***{fitx_err,Nom,Tamany}***

  Hi ha un error a l'enviament de dades del fitxer amb nom Nom.

- ***{fitx_ok,Nom,Ruta,Tamany}***

  Es pot iniciar la transmissió del fitxer amb nom Nom

5.4.3.Dades que es poden rebre del Servidor

{Grup, error, [Motiu]}

Hi ha agut algun error amb les dades enviades al servidor, el motiu és Motiu.

{Grup, login_ok, [Grup, Nick]}}

Si el *login* era correcta ens retornarà la comanda *login_ok* on Grup és la referència (PID en el servidor) que s'haurà d'utilitzar per enviar missatges al grup general. *Grup* és el nom que tothom te en el grup general i *Nick* es el nostra nick, el que utilitzem.

{Grup, login_fail, [Str]}

Si per altre banda el *login* no surt bé rebrem un missatge amb la comanda *login_fail*. *Grup* en aquest cas és el PID del servidor i *Str* és un string que conte el missatge de perquè ha fallat, aquest pot ser: "Usuari ja connectat", "Nick Incorrecta" o "Password Incorrecta".

{Grup,sms, [Str]}

Rebem un missatge Str del grup Grup.

{Grup, estat_actual, [Nick,Estat]}

Es rep un nou estat actual d'un usuari amb nick Nick i l'estat és Estat.

{Grup, list_usr, [Usuaris]}

Es rep una llista d'usuaris Usuaris, la llista esta formada per tuples del tipus: {Nick,Nom,Estat}. On Nick és el nick de l'usuari amb nom Nom i estat actual Estat.

{servidor_adm, start_listen, [Port,Nick,Usuari,Grup]}

Aquesta comanda significa que el client te que iniciar un *listen* del port Port per així poder acceptar la petició del client, ja que s'ha sol·licitat una conversa client a client on el grup que es tindrà que utilitzar és Grup el nostre nick és Nick i el nick de l'altre usuari és Usuari.

{servidor_adm, start_connect, [Ip,Port,Usuari,Nick,Grup]}

Aquesta comanda significa que el client te fer un *connect* a la ip Ip i port Port per així poder connectar-se amb l'altre client, ja que s'ha sol·licitat una conversa client a client on el grup que es tindrà que utilitzar és Grup el nostre nick és Nick i el nick de l'altre usuari és Usuari.

{servidor_adm, create_grup_ok, [Pid_Grup,Nom,Nick]}

La creació d'un grup ha sortir bé i rebem el pid del grup (Pid_Grup), el nom del grup Nom i el nostra nick Nick.

CAPÍTOL 6

PROVES I RESULTATS

Capítol 6: Proves i Resultats

En aquest capítol s'exposaran els resultats obtingut de l'aplicació. S'han realitzat proves tan pel client en protocol local i IRC com pel servidor. S'ha provat el client IRC en un canal públic i s'han mantingut converses simultànies amb uns 10 usuaris. Per la part del local s'ha provat amb uns 6 usuaris mantinguem-te amb tots aquests converses un a un i diverses en grup, també amb alguns s'han transmès fitxers, fins a una grandària màxima de 100mb.

Però aquestes proves realitzades van més enllà, se sap que el seu funcionament és correcta i no te errors, per tant al tractar-se d'una aplicació distribuïda i amb processos treballant en paral·lel s'ha volgut analitzar el seu rendiment.

Es començarà fent una comparativa amb el client en protocol IRC i un altre client IRC, concretament el *XChat* versió 2.8.6, un client per sistemes Linux. Pel que fa el client no s'han realitzat proves en protocol local ja que les provés que es realitzarien serien molt similars a les del protocol IRC. S'ha de tenir en compte però que el *XChat* es tracta d'un client IRC complet i amb suport per *plugins* per tant les dades són orientatives, tot hi que en el *XChat* provat no s'ha utilitzat cap *plugin*.

Per altra banda també s'han realitzat proves amb el servidor, però a diferencia del client aquestes no poden estar comparades amb cap altre ja que es tracta d'un servidor desenvolupat per aquest projecte.

Totes aquestes probes s'han realitzat en un equip portàtil amb un Core 2 Duo a 2,0Ghz de processador, 1 Gb de memòria Ram i sistema operatiu Ubuntu 9.04.

6.1.El Client

Per poder realitzar les proves en el client i comparar-les amb un altre s'ha desenvolupat una espècia de servidor IRC. No es tracta d'un servidor ja que aquest l'únic que fa és esperar que algú es connecti, llavors crea un procés el qual rep dades del client i tot després crea un nombre de processos (definit en el servidor) els quals van enviant missatges privats al client.

Les proves que s'han realitzat han estat, en primer lloc s'ha simulat que 25 usuaris enviaven missatges privats cada segon, i la segona prova ha estat igual però amb 50 usuaris. Si es te en compte que habitualment el nombre màxim d'usuaris en que es parla sol estar entre 5 i 15 les proves que s'han realitzat són amb més del doble d'usuaris.

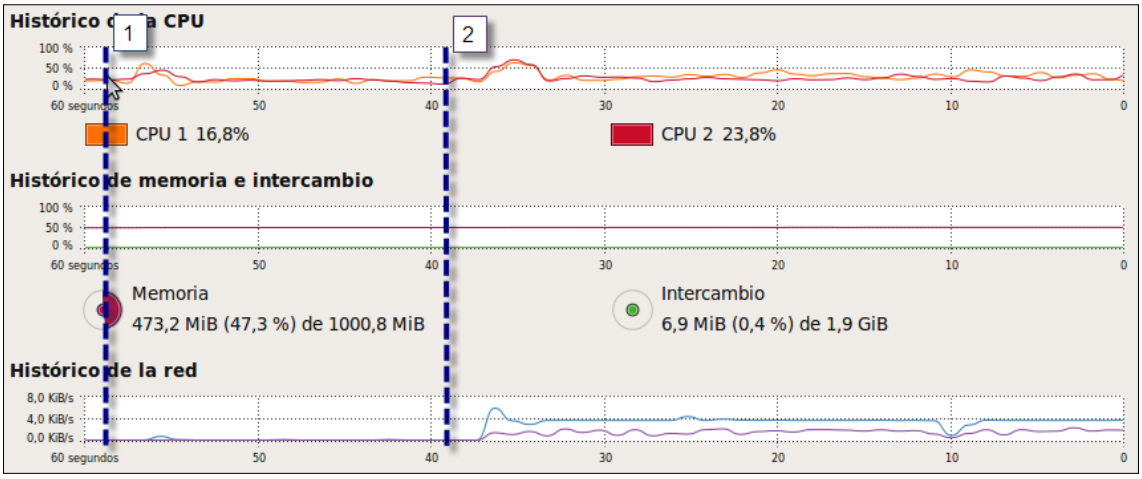
6.1.1.Resultats Client Local

Els resultat que es mostren a continuació són uns gràfics de rendiment de la CPU i de dades rebudes, en el cas del Local es veurà com aquest gràfics estan dividits en dos parts.

La part 1 és quan es compila i s'inicia el protocol local, és a dir la pantalla per iniciar sessió i entrada de dades. Per aquest motiu es veu que al principi hi ha molta activitat, ja que és quan es compilen els mòduls (si és necessari) i s'inicia, després però hi ha una estona sense casi funcionament de CPU és perquè s'entren les dades per connectar-se amb el servidor.

La segona part en canvi és quan es reben tots els missatges privats i es creen totes les finestres per tant és quan es troba un rendiment de CPU més elevat.

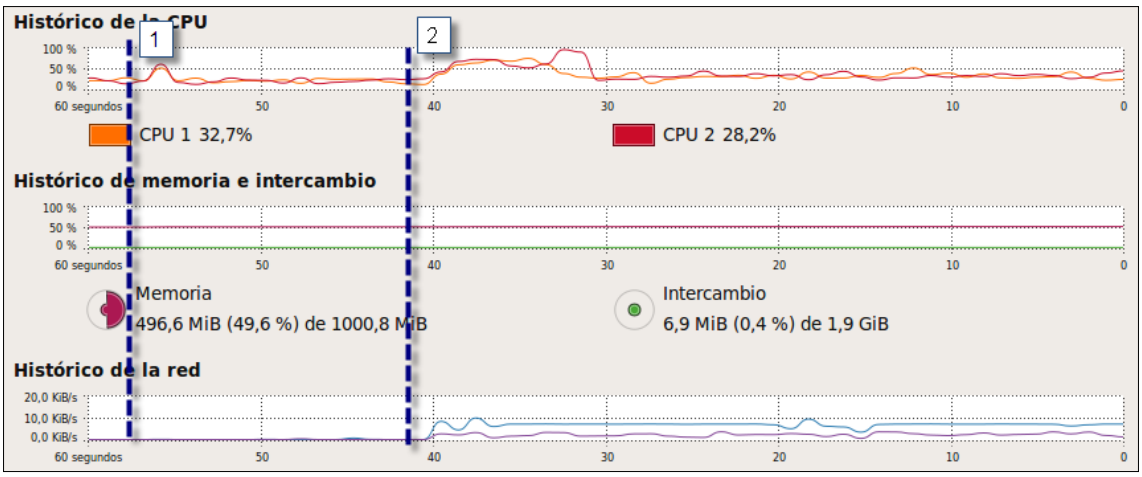
Els resultats obtinguts amb 25 usuaris i rebent de cada un d'ells un missatge cada segon han estat els següents:



- Gràfic protocol local i 25 usuaris

En el moment de la creació de totes les converses hi ha agut uns pics màxim que estaven entre 60% i 65% de consum de la CPU i després el consum mitja rondava entre el 10% i 20%.

Amb 50 usuaris els pics han estat de casi el 90% de la CPU en la creació de les finestres i després es manté entre un 20% i 30% de consum.



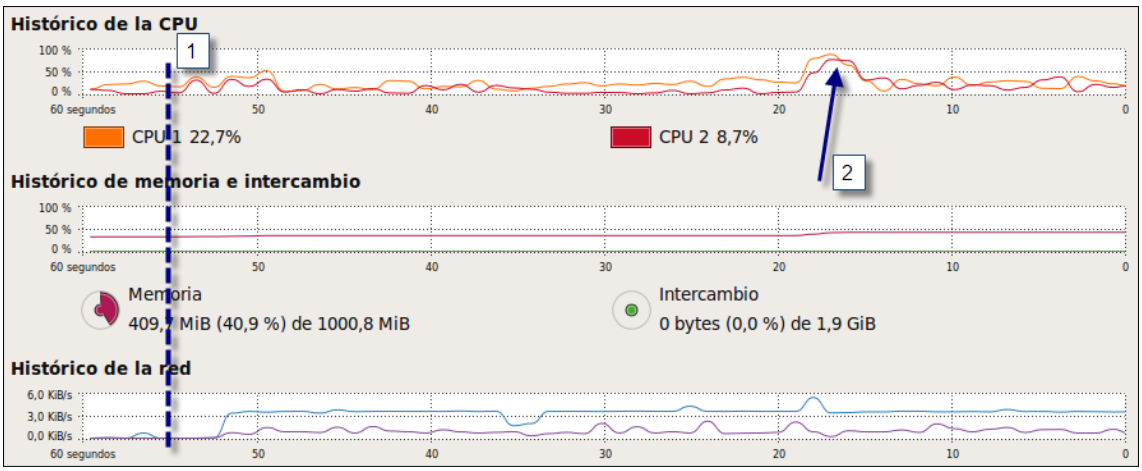
- Gràfic protocol local i 50 usuaris

6.1.2.Resultats *XChat*

Les proves realitzada amb el *XChat* són exactament les mateixes realitzades amb el protocol local, la diferència però està en que en el *XChat* s'ha tingut que canviar les opcions de configuració perquè crees cada conversa en una nova finestra, per així poder estar en un mateix nivell.

En les imatges que es mostren a continuació amb els resultats es veuen dos punts diferents. El primer (número 1) és quan s'inicia l'aplicació i aquesta automàticament és connecta i comença a crear les finestres de conversa. El problema està en que inicialment només crea 4 finestres (5 amb la general) i no és fins el punt número 2 que les crea totes, no se sap si es per algun tipus de bloqueig que porta el programa o bé és que tarda molt a crear-les totes, però no afecta molt.

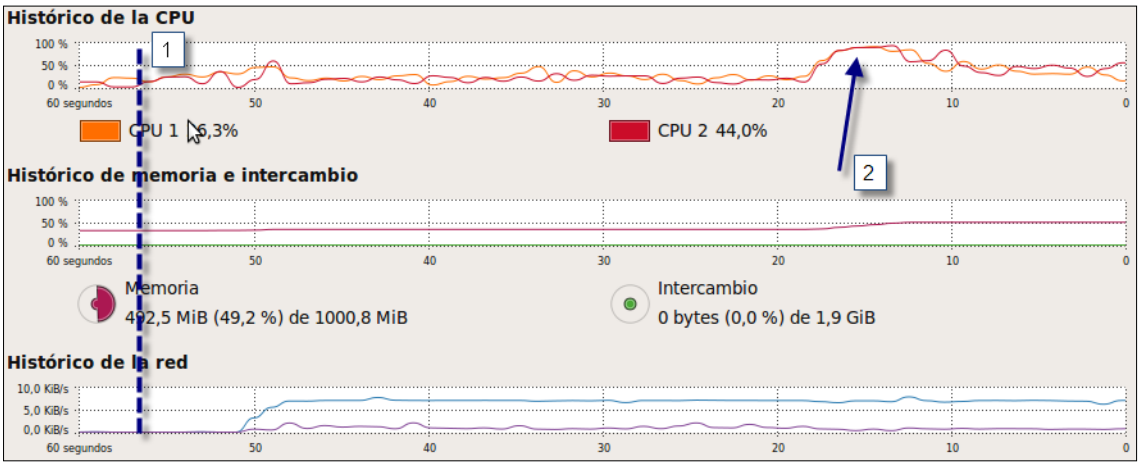
Els resultat amb 25 usuaris són els següents:



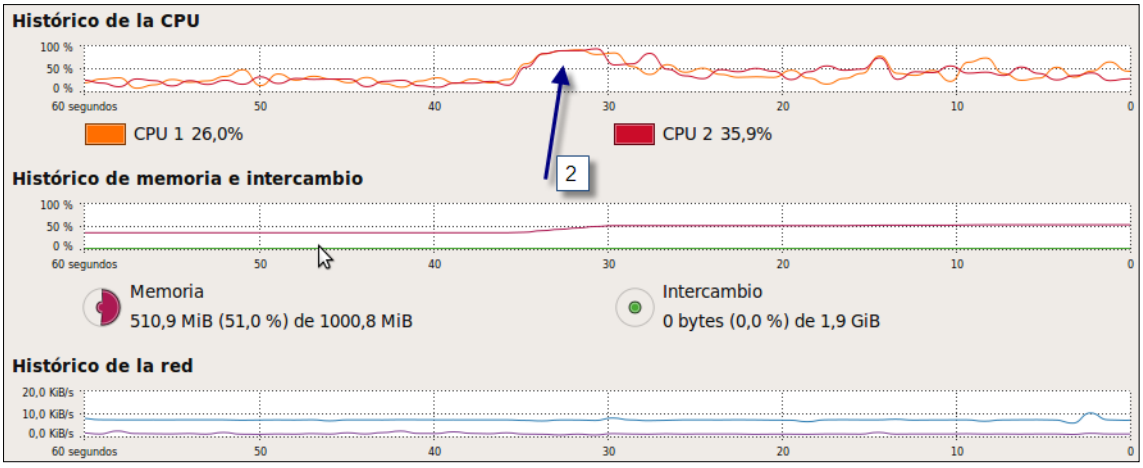
- Gràfic Xchat i 25 usuaris

En el moment d'inici el consum no supera el 55% però nomes amb 4 finestres, llavors un cop es creen les restants s'arriba a algun pic entre el 80% i 95%, en canvi després, un cop creades els finestres està entre el 20% i 30%.

En canvi amb 50 usuaris ha una mica més elevats s'ha arribat a superar el 90% en el moment de la creació de totes les finestres i després els consum no és gaire constant, sinó que va fent molts de pics que estan entre un mínim de 25% i màxim de 75%



- Gràfic Xchat i 50 usuaris



- Gràfic Xchat i 50 usuaris (continuació de la imatge anterior)

6.1.3.Conclusions

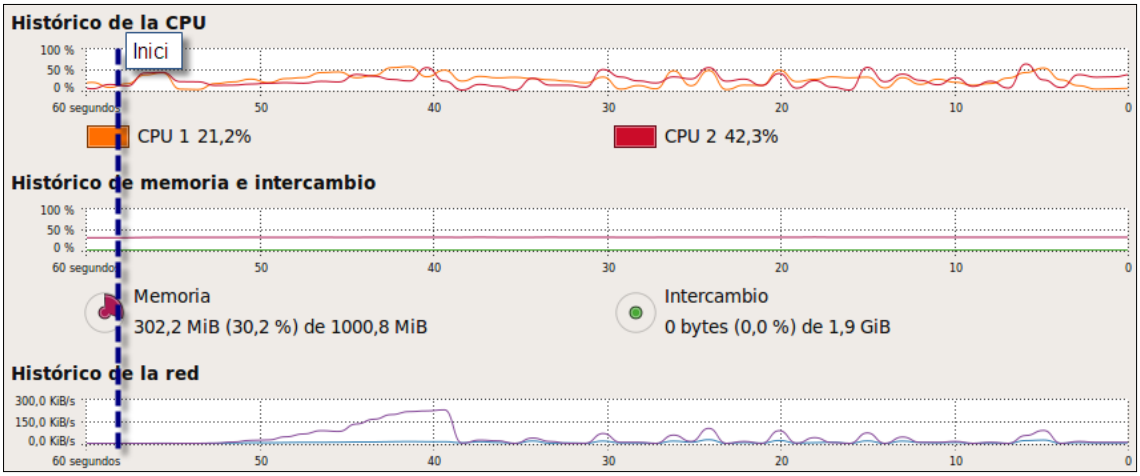
Les conclusions que se'n poden extreure són que tal com es veu amb l'aplicació realitzada i pocs usuaris casi no es troba diferencia alguna entre paral·lel i seqüencial, en canvi ala que s'augmenten els usuaris es veu com el rendiment millora quan es treballa en paral·lel, ja que el pic de consum és una mica inferior però dura menys, a més també quan estan totes les finestres creades l'aplicació que treballa en paral·lel es manté mes constant i més no consumeix tant.

6.2.El Servidor

Pel que fa el servidor, al igual que al client s'ha realitzat un petit *boot*, el qual el que fa és crear tants processos com els que s'han definit, aquests processos el que fan és conecectar-se amb el servidor i al cada "X" segons definits s'envia un missatge en la conversa general, també a si es creen més de 5 procés és a partir del sisè quan cada un crea una conversa en grup amb els usuaris definits en el mòdul.

La primera prova realitzada ha estat de 50 usuaris que es connecten al servidor un després de l'altre cada 250 milisegons i a partir del cinquè es creen converses en grup amb els quatre primers usuaris i el creador de la conversa. Els missatges en aquest cas s'envien cada 2 segons en tots els grups on esta l'usuari. En definitiva es tenen 50 processos de comunicació la conversa en grup general i 44 en més.

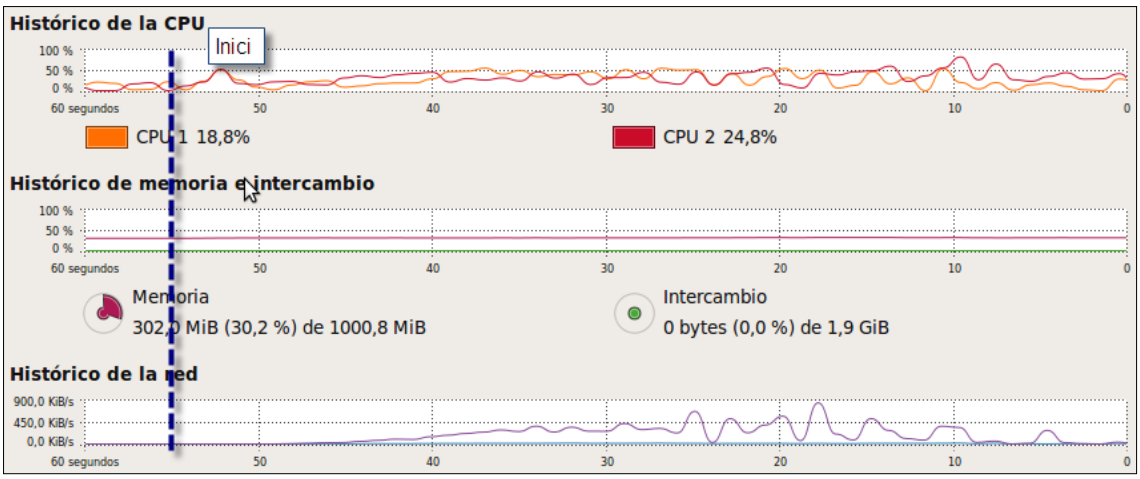
En definitiva, es un cop realitzades aquestes proves es veu que el consum de CPU casi no supera el 50% nomes en alguns moments en concret es tenen alguns pics elevats.



- Gràfic Servidor local amb 50 usuaris i 45 converses en grup

A la segona proba s'ha doblat el nombre d'usuaris, és a dir 100 usuaris i 95 converses en grup.

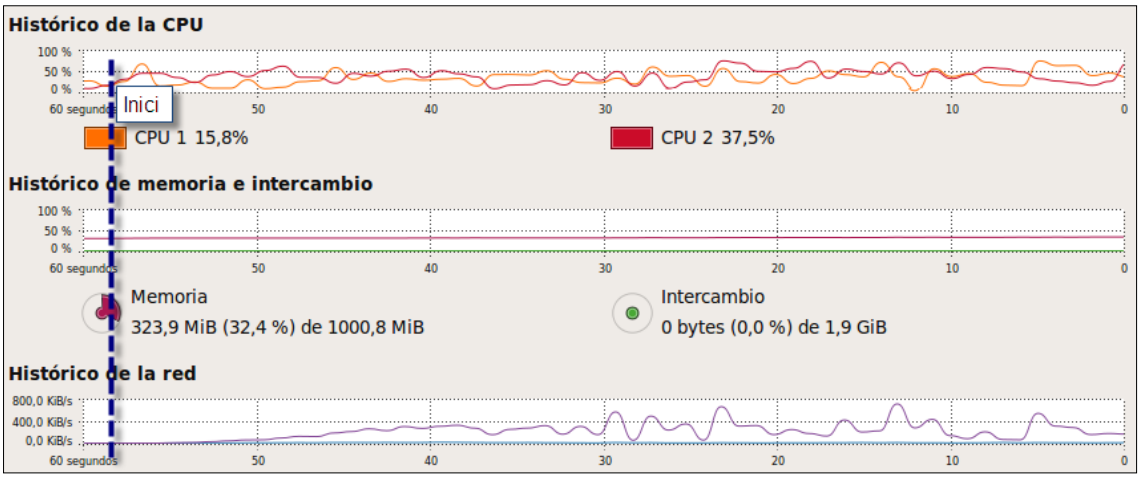
Tal com s'observa en el gràfic que es veu a continuació del text s'aprecia que al doblar el nombre d'usuaris i superar amb més del doble el nombre de converses en grup el consum de CPU s'incrementa en un 10/15% més i es manté més constant en la línea del 50% amb alguns pics que superen aquest 50% algun inclús arribant casi al 80%.



- Gràfic Servidor local amb 100 usuaris i 95 converses en grup

Per acabar s'han realitzat dos proves amb 500 usuaris i 495 converses en grup.

La primera s'ha realitzat amb les mateixes característiques que les anteriors proves, el resultat ha estat el següent:

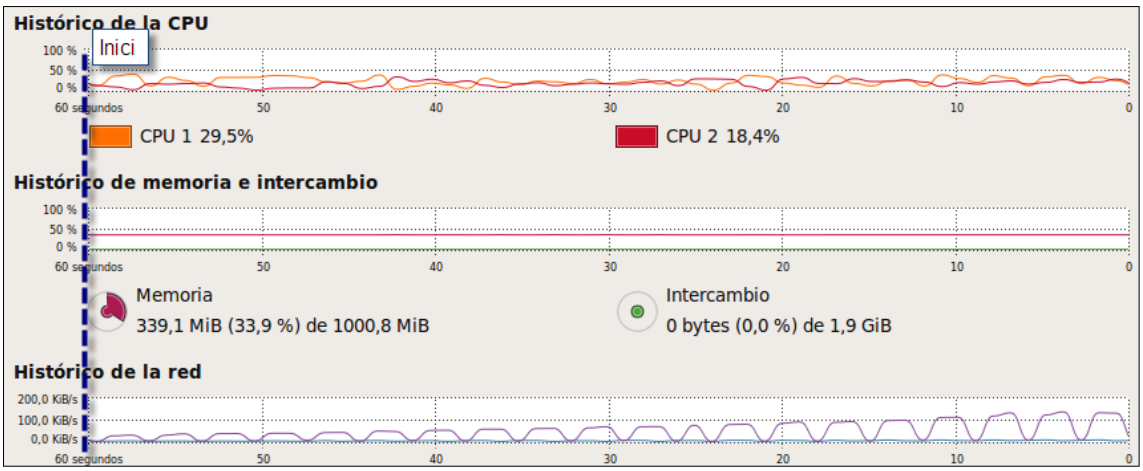


- Gràfic Servidor local amb 500 usuaris i 495 converses en grup (Prova 1)

Tal com s'observa el consum de CPU s'incrementa molt més degut a que es connecten molts usuaris en molt poc temps. Això també comportava alguns problemes d'espera a l'ora de la connexió amb el servidor.

Per aquest motiu s'ha dut a terme una segona prova amb els mateixos usuaris però amb alguns canvis, un d'ells ha estat que en comptes de connectar-se cada 250 milisegons passa a 1,5 segons, és a dir en tenir tots els usuaris connectats es tarden casi 12 minuts. L'altre canvi ha estat l'enviament de missatges que passa a ser cada 4 segons un missatge a tots els grups.

Inicialment el consum de CPU ha estat el següent:



- Gràfic Servidor local amb 500 usuaris i 495 converses en grup (Prova 2)

En aquest últim m'entres els usuaris es van connectat la CPU no consumeix gaire sinó que es troba per sota el 50%, en canvi passats uns 15 minuts, quan tots els 500 usuaris estan connectats i es troben 495 converses en grups el consum de CPU augmenta fins arriba entre el 70% i 80%.

CAPÍTOL 7

CONCLUSIONS I TREBALLS FUTURS

Capítol 7: Conclusions i Treballs Futurs

El punt central d'aquest treball ha estat la programació amb un llenguatge distribuït i concurrent anomenat Erlang i el desenvolupant d'un sistema de missatgeria instantània el qual s'ha realitzat un client i un servidor, tot seguit s'extreuen algunes conclusions i es plantegen possibles millores.

7.1.Conclusions

Tal com s'ha explicat en el principi d'aquest capítol el punt central del projecte ha estat l'aprenentatge d'un nou llenguatge de programació distribuït i concurrent que es desconeixia.

Amb tot l'aprés s'ha vist que es tracta d'un llenguatge agradable de treballar però que alguns cops és una mica complicat quan s'està acostumant a altres com el C o Java.

Un dels punts fort que es troba amb l'Erlang és que es pot arribar a fer exactament el mateix que alguns altres però amb menys línees de codi i amb paral·lel. També una de les coses bones és la facilitat de fer, veure i construir sentenciés recursives.

Però una de les coses que més m'ha agradat ha estat el *hot-swap*, explicat en l'apartat 2.5.5 del capítol 2, el qual permet actualitzar el codi de l'aplicació sense tenir que parar-la.

Per altra banda te algunes coses que no agradant tant, com per exemple un primer que es tracta de la interfície gràfica, no disposa de cap mòdul amb una bona interfície. El mòdul amb que s'ha desenvolupat el projecte (gs) és l'únic de que es disposava propi, llavors se'n troben alguns altres però presenten errors segons el sistema operatiu amb que es treballi. Però per altra banda un dels avantatges que te és la facilitat de "lligar" el codi fer en C o Java amb el codi programat amb Erlang, de fet pel que s'ha llegit el que es fa molt és desenvolupar la interfície en Java.

Pel que fa a l'aplicació s'ha vist com es programava una aplicació concurrent i distribuït i al desenvolupar-la i acabar-la s'ha vist com sí que el codi de l'aplicació feta amb Erlang és més curt, fàcil (un cop es sap programar) i es podria dir inclús que molt més entenedor que algun altre, posant com a exemple el xat desenvolupat en un pràctica de la carrera d'ETIS.

Per acabar una de les coses que més m'han sorprès ha estat que em pensava que aquest llenguatge no s'utilitzava casi en res, nomes en aplicacions concretes basades en la comunicació en xarxa i s'ha vist que l'Erlang s'ha utilitzat per fer una part de del famós *facebook*, en concret el xat.

7.2.Treballs Futurs

En aquest apartat es presenten algun possibles treball amb aquest llenguatge i inclús alguna millora del projecta.

El projecte pot presentar possibles millores com la de donar més opcions i més protocols al client. Algunes de les opcions podrien ser una base de dades per guardar les converses, dades per múltiples usuaris.

Algun altre treball que es podria realitzar seria la implementació d'un servidor, per exemple un servidor IRC, o qualsevol altre aplicació en base a les comunicacions en xarxa, ja que Erlang ofereix moltes opcions per aquests programes.

CAPÍTOL 8

BIBLIOGRAFIA

Capítol 8: Bibliografia

La bibliografia per la realització d'aquest projecta ha estat molt extensa en quant a enllaços web per aquest motiu només es posen els més importants. En quant a llibres el més consultat ha estat el de Joe Armstrong *Programming Erlang* que va ser un dels primers.

- Joe Armstrong. *Programming Erlang*. Dallas, Texas: Pragmatic Bookshelf, 2007
- Francesco and Simon. *Erlang Programing*. Gravenstein Highway North, Sebastopol: O'Reilly, 2009
- Martin, Eric, Richard and Bob. *Concurrent Programming with Erlant/OTP*. US, Greenwich: Manning, 2009
- *Erlang*. Gener 2008 – Setembre 2009. Open Source Erlang. 20 agost 2009 http://www.erlang.org/
- *Wikipedia*. Agost 2009. *Erlang*. Wikipedia Foundation, Inc. 22 juliol 2009 *http://es.wikipedia.org/wiki/Erlang*
- Blaxter. Juliol 2009. *Paralelizando quicksort en Erlang*. Bicosyes: 20 setembre 2008 *http://bicosyes.com/paralelizando-quicksort-en-erlang/*
- C. Kalt. Maig 2009. *Internet Relay Chat: Client Protocol*. Network Working Group: Abril 2000 *http://www.isi.edu/in-notes/rfc2812.txt*
- C. Kalt. Maig 2009. *Internet Relay Chat: Server Protocol*. Network Working Group: Abril 2000 *http://www.isi.edu/in-notes/rfc2813.txt*

També s'han consultat fòrums de discussió següents:

- Erlang Community Site: *http://www.trapexit.org/*
- Nabble – Erlang: *http://www.nabble.com/Erlang-f14095.html*

APÈNDIX A

MANUAL DE L'APLICACIÓ

Apèndix A: Manual de l'aplicació

Aquest apartat és un manual sobre l'aplicació. En primer lloc s'explicaran el requeriments mínims de l'equip, després com compilar i executar i finalment un manual pel client i un altre pel servidor.

S'ha de tenir en compte també que l'aplicació ha estat desenvolupada com un projecte per veure la creació de processos i la recepció de missatges, per tant en tots els casos es veurà també una finestra on és mostren els missatges rebuts i els processos creats.

Requeriments

Per poder executar l'aplicació els requeriments de l'equip no tenen perquè ser molt elevats tot depèn de com s'utilitza l'aplicació. Per exemple si el servidor necessita un nombre elevat de clients evidentment es necessitarà un bon equip.

Els requeriments bàsics són:

- **Sistema Operatiu:**
  - Qualsevol basat en Linux (Ubuntu, RedHat,....)
  - Windows 2000 o superior.
  - Algunes opcions no estan suportades amb Windows Vista, com les converses client a client.
- **Processador i equip:**
  - Processador: S'aconsella un processador de 1Ghz o superior
  - Memòria Ram: 250Mb o més
  - Espai en disc: 5 Mb
- **Paquets instal·lats prèviament:**
  - Erlang R13B o superior

Instal·lació

El programa no necessità instal·lació, simplement copiem la carpeta amb nom "Xat MultiProtocol" allà on es vulgui de l'equip. En el cas de que nomes es vulgui utilitzar el client o el servidor es pot eliminar una de les dos subcarpetes que es troben a dins.

Compilació i Execució

En aquest cas pel que fa la compilació i execució es troben dos fitxers makefiles, un per sistemes operatius Linux i un altre amb nom *Makefile.bat* per sistemes Windows.

A dins de la carpeta "Xat MultiProtocol" hi ha un makefile general, Aquest el que fa és executar el makefile del servidor o del client, per tant si es realitza algun canvi de nom en alguna de les carpetes s'haurà de canviar la ruta dels makefiles generals.

El makefile del Windows disposa d'un menú per escolli la opció que es vulgui.

En el dels sistemes Linux per executar és "make run", per compilar "make compile". En el cas que es tracti del general però per compilar per exemple el client és "make compile_client" i per executar el del client és "make run_client". Per qualsevol dubte sobre les opcions disponibles disposa d'una comanda d'informació "make info".

El Client

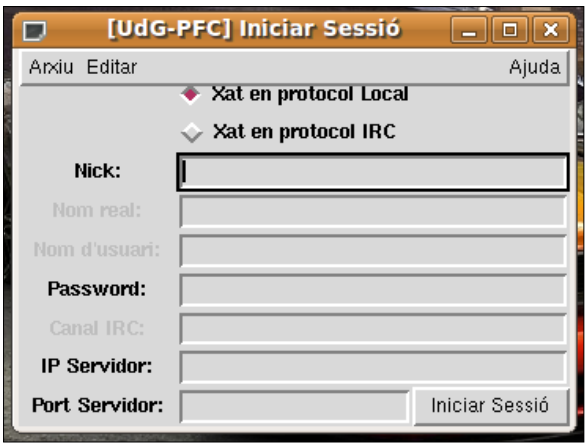
Per executar el client, si estem a la carpta "Xat MultiProtocol":

- Sistemes Linux: make run_client
- Sistemes Windows: Doble clic al fitxer *Makefile.bat* i escollir la opció corresponent

Per altra banda si estem a dins al carpeta on es troben tots els mòduls en sistemes Linux la comanda canvia i s'ha d'executar "make run", per sistemes Windows no canvia.

Iniciar Sessió

El primer que es veurà és aquesta finestra:

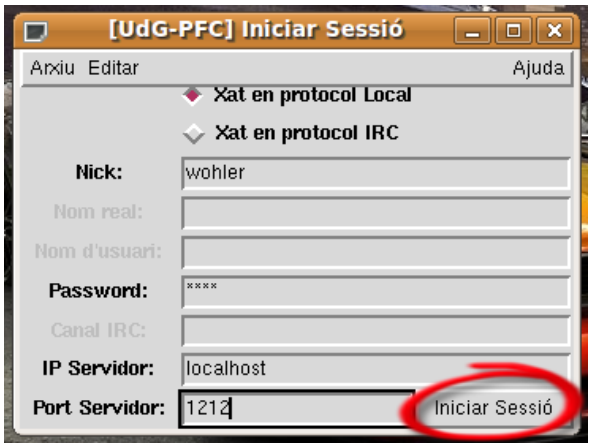


- Finestra inicial

En aquesta s'ha d'escollir el tipus de protocol al qual es vol connectar. En el cas de tractar-se del protocol Local només es necessita un nick i un password, aquests però tenen que esta donats d'alta en el servidor. També farà falta conèixer la direcció ip o el hostname del servidor al igual que el seu port.

En el cas del protocol IRC es te que posar el nick que es vol utilitzar, el nom real d'usuari, un nom d'usuari, el canal IRC on es vol connectar, la ip del servidor a utilitzar i el seu port.

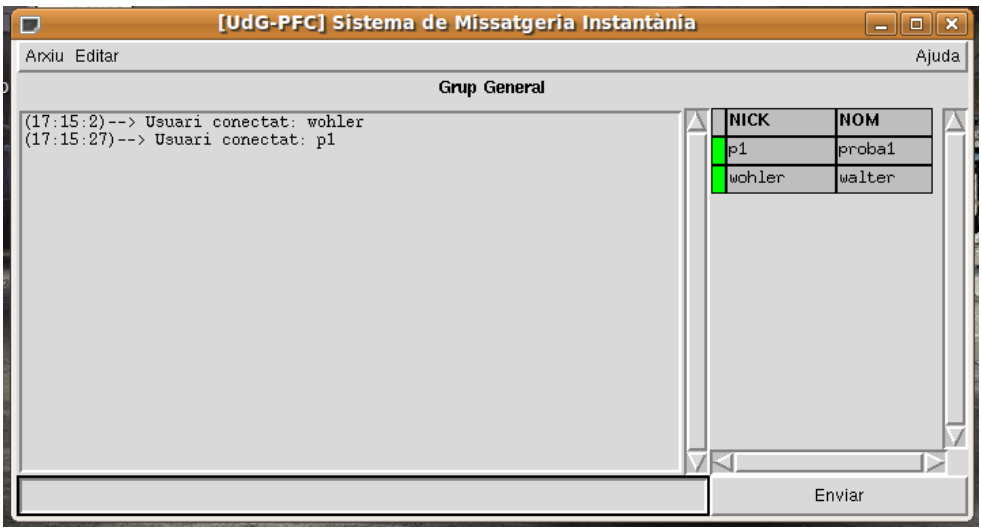
Finalment un cop es tinguin les dades emplenades nomes cal clicar a "Iniciar Sessió"



- Finestra inicial 2

Protocol Local

En el cas d'escollir el protocol local es poden realitzar diverses accions que es trobaran en els següents subapartats, la finestra principal és una com la següent:

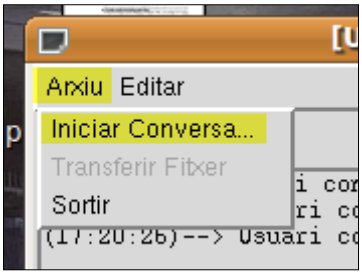


- Finestra de la conversa tots a tots

Aquesta finestra és per les converses generals tots a tots. A mà dreta es troba la llista d'usuaris en la quan es mostra el seu estat, disponible (verd), ocupat (blau) o absent (vermell). Al costat de l'estat d'usuari hi ha el seu nick i el seu nom.

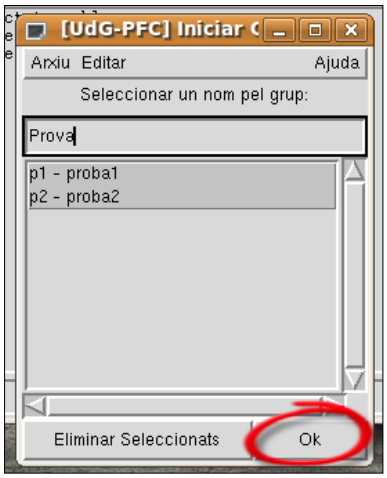
Mantenir un convers en grup

Per iniciar una conversa amb grup s'ha d'accedir en el menú *Archiu/Iniciar Conversa....*



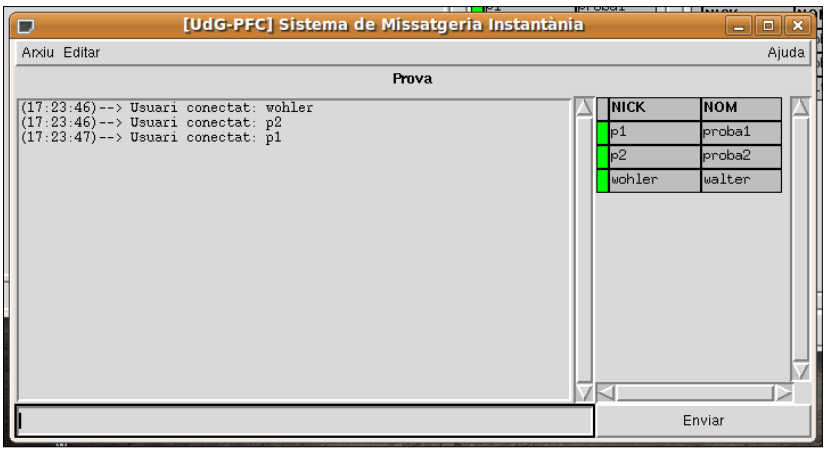
- Menú Iniciar Conversa....

Un cop aquí s'escullen els usuaris i un nom pel grup. En aquest cas el nom és "Prova" i els usuaris són en p1 i el p2. Aquesta llista es mostra al igual que la de la conversa general el nick i el nom de cada usuari. Un cop seleccionats els usuaris només s'ha de clicar en el boto *Ok*



- Finestra de selecció d'usuaris

I seguidament s'obrirà una altre conversa però en aquest cas només amb els dos usuaris que es volen.



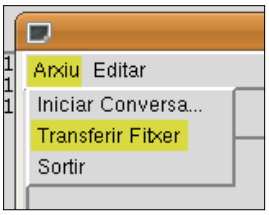
- Exemple conversa en grup

Mantenir una convers client a client

Per mantenir una conversa client a client els passos són exactament que els del punt anterior "Mantenir una conversa en grup". Amb la única diferencia que aquí si escollim un nom pel grup aquest no es mantindrà.

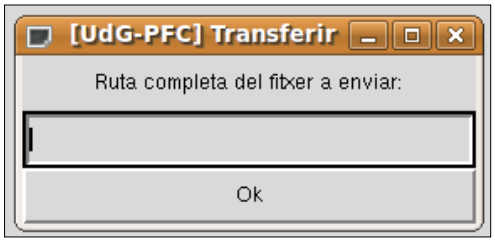
Enviar un fitxer a un altre usuari

Per poder enviar un fitxer a un usuari en primer lloc es te que mantenir una conversa client a client, llavors es veurà que s'ha activat la opció *Transferir fitxer* del menú *Arxiu*.



- Menú Transferir Fitxer

Quan es selecciona aquesta opció es veurà una finestra com la següent:

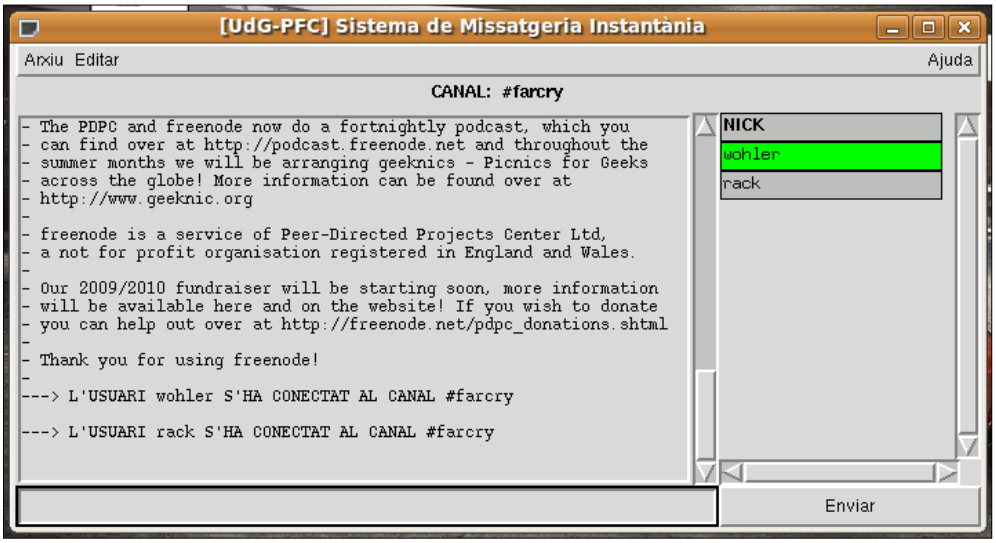


- Finestra per entrar la ruta del fitxer

En aquesta simplement s'ha de entrar la ruta completa del fitxer que es vulgui enviar.

Protocol IRC

En el cas d'haver escollir el protocol IRC la finestra principal la qual es veurà serà una similar a la següent:



- Exemple conversa principal IRC

En aquesta finestra podem mantenir les converses en el grup general, tots a tots en el canal escollit. A la mà dreta es veu una llista amb els usuaris actualment connectats.

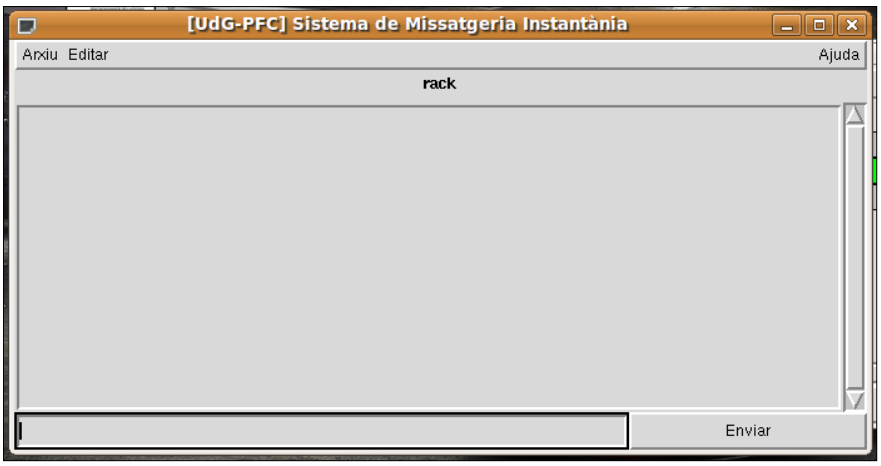
Cada un d'aquest usuaris pot tenir un color diferents segons la categoria dins el canal:

- Gris: Usuari normal.
- Verd: L'usuari es tracta d'un operador del canal.
- Groc: Te permisos com a moderador del canal.

Mantenir una conversa amb un altre usuari

Per mantenir una conversa amb un altre usuari simplement cal fer un doble clic sobre el nom que hi ha a la llista de la conversa general del canal.

En el següent exemple es manté una conversa amb l'usuari amb nick *rack*.



- Exemple conversa usuari a usuari IRC

El Servidor

En aquest apartat s'explica tot el necessari per poder utilitzar el servidor. El servidor a diferencia del client no s'utilitza una interfície de gràfica sinó que es a través de comandes fàcils i intuïtives.

Iniciar el Servidor

Per iniciar el servidor es pot fer de dos maneres, a través del **makefile** que executa la funció ***start()*** o a través d'una consola d'Erlang a la qual li podem entrar la funció *start()* o bé ***start(Port).*** La diferència és que en una se li diu el port a escoltar i l'altre el port esta definit en el mòdul.

En el cas d'escollir executar la funció *start()* (ja sigui per la consola o pel makefile) s'ha de tenir en compte que el port posat per defecte és el 1212, si es vol canviar simplement es te que anar al mòdul ***servidor.erl*** i canviar el ***"-define(PORT, 1212)."*** pel port que es vulgui utilitzar.

En tots dos casos aquestes dos funcions retornen el pid del procés del servidor el qual després servirà, si es vol, per aturar-lo. S'ha de tenir en compte però que nomes es podrà guardar en cas que executem qualsevol de les funcions però de la consola manualment, sense utilitzar el makefile. Per exemple, sense makefile, es va al directori on estan els mòduls i s'executen les següents comandes, les quals executen el servidor i guarden el pid a la variable P.

```
Erlang R13B01 (erts-5.7.2) [smp:4:4] [rq:4] [async-threads:0]

Eshell V5.7.2  (abort with ^G)
1> P = servidor:start().
```

- Inici del servidor

Donar d'alta un Usuari

Per donar d'alta un usuari no fa falta executar el servidor, en cas que el servidor no estigui obert simplement es te que obrir la consola Erlang i executar la funció *servidor_bdd:start().*

Després en qualsevol de les dos situacions, tan amb el servidor obert com parat, simplement s'executa la comanda ***servidor_bdd:crear_usuari(Nick,Nom,Pwd),*** on Nick serà el nick de l'usuari, Nom el seu nom i Pwd la seva contrasenya per iniciar sessió. Si es pot crear bé ens donarà *{atomic, ok}* sinó *{error, Why},* on *Why* és el motiu de l'error.

Exemple:

```
2> servidor_bdd:pregunta(usuari).
[{usuari,"p4","proba4","45",false,false},
 {usuari,"p1","proba1","12",false,false},
 {usuari,"p5","proba5","56",false,false},
 {usuari,"p2","proba2","23",false,false},
 {usuari,"wohler","walter","w123",false,false},
 {usuari,"p3","proba3","34",false,false}]
3>  servidor_bdd:crear_usuari("XxPROVAxX","Albert","tysW34").
{atomic,ok}
4> servidor_bdd:pregunta(usuari).
[{usuari,"p4","proba4","45",false,false},
 {usuari,"p1","proba1","12",false,false},
 {usuari,"p5","proba5","56",false,false},
 {usuari,"XxPROVAxX","Albert","tysW34",disponible,false},
 {usuari,"p2","proba2","23",false,false},
 {usuari,"wohler","walter","w123",false,false},
 {usuari,"p3","proba3","34",false,false}]
```

- Crear un usuari

Expulsar un usuari

Per expulsar un usuari s'ha de tenir el servidor en execució ja que el que fa és tancar el seu procés de comunicació i eliminar-lo dels grups on es troba.

La comanda és ***servidor:expulsar_usuari(Nick),*** on Nick és el nick de l'usuari connectat. En el cas que l'usuari no estigui connectat ens avisa.

Consultes

Aquest apartat està dedicat a mostrar algunes de les consultes que es poden realitzar en la base dades.

- **Usuaris donats d'alta**
  - Comanda: *servidor_bdd:pregunta(usuari_tots_nick_nom)*
  - Exemple:

```
8> servidor_bdd:pregunta(usuari_tots_nick_nom).
[{"p4","proba4"},
 {"p1","proba1"},
 {"p5","proba5"},
 {"XxPROVAxX","Albert"},
 {"p2","proba2"},
 {"wohler","walter"},
 {"p3","proba3"}]
```

- Usuaris donats d'alta

- **Grups actuals**
  - Comanda: *servidor_bdd:pregunta(grup)*
  - Exemple:

```
10> servidor_bdd:pregunta(grup).
[{grup,<0.145.0>,"Grup General","servidor_adm"},
 {grup,<0.155.0>,"Prova de Conversa en grup","p2"}]
```

- Grups actuals

- **Usuaris connectats**
  - Comanda: *servidor_bdd: pregunta(in_grup_nick, Id)* on Id és el número identificatiu del grup general. Per saber-lo simplement cal executar comanda anterior. S'ha de tenir en compte però que el número identificatiu és un pid per tant si s'escriu a mà avanç s'haurà de passar a pid.
  - Exemple:

```
11> Gen = erlang:list_to_pid("<0.145.0>").
<0.145.0>
12> servidor_bdd:pregunta(in_grup_nick, Gen).
["wohler","p1","p2"]
```

- Pasar un tipus string a tipus pid

  - Si no es volen utilitzar variables:

```
13> servidor_bdd:pregunta(in_grup_nick, erlang:list_to_pid("<0.145.0>")).
["wohler","p1","p2"]
```

- Usuaris connectats

- **Usuaris en determinats grups**
  - Comanda: *servidor_bdd: pregunta(in_grup_nick, Id)* on Id és el número identificatiu del grup. Es tracta de utilitzar exactament el mateix mètode que en l'anterior.
  - Exemple:

```
14> G1 = erlang:list_to_pid("<0.155.0>").
<0.155.0>
15> servidor_bdd:pregunta(in_grup_nick, G1).
["p2","p1","wohler"]
16> servidor_bdd:pregunta(in_grup_nick, erlang:list_to_pid("<0.155.0>")).
["p2","p1","wohler"]
```

- Usuaris en determinats grups