PROJET INGENIEUR INFORMATIQUE PREMIERE ANNEE

Implémentation

PROJET FORUM

COULON Anthony
ROLLAND Cyrille

26/05/2009

INDEX

Introduction

Dans le cadre des études en école d'ingénieur informatique première année à Sup Galilée, nous devons effectuer un projet de programmation système. Notre projet consiste à développer un forum en langage C. Ce forum doit pouvoir contenir plusieurs articles traitant de multiples thèmes dans lesquels les utilisateurs peuvent poster des messages. Ce projet doit se composer de deux programmes, « le Serveur » qui sera chargé de stocker et de gérer les articles du forum, et « le client » qui devra gérer l'interface homme machine et permettre à l'utilisateur d'utiliser le forum le plus simplement possible. Nous verrons dans ce document comment implémenter un tel projet en listant et en expliquant le fonctionnement de l'ensemble des fonctions à utiliser.

I/ Les fonctionnalités

Le programme serveur est un programme autonome qui ne nécessite aucune intervention humaine. Le serveur est capable de gérer la liste des articles postés par les utilisateurs. S'il y a trop d'articles ou que certains ne sont plus utilisés depuis une semaine alors un archivage sera mis en route afin de ne pas avoir une liste d'articles trop importante qui rendrait l'utilisation du forum désagréable. De plus le serveur limite son nombre d'archives et la taille accordée au répertoire « Users » qui contient les utilisateurs de façon à ne jamais dépasser l'espace disque de 2 Go sur la machine sur laquelle il tourne. En réalité les 2Go ne devraient jamais être atteint.

Cependant le serveur propose à l'ingénieur système une petite interface graphique lui permettant de supprimer des articles, des archives ou des comptes utilisateurs et de consulter les plaintes déposées par les utilisateurs. Cette fonctionnalité que représentent les plaintes peut s'avérer très utile dans les débuts de l'application car il est possible que certains bugs ne soient pas détectés lors des tests. Par conséquent les plaintes des utilisateurs permettront d'organiser les éventuelles mises à jour du programme.

Le programme client est principalement une interface graphique permettant de simplifier l'utilisation du forum pour l'utilisateur. Nous avons fait le choix d'une interface graphique assez simple et plutôt conviviale pour donner envie à l'utilisateur, même le moins expérimenté en informatique, d'utiliser ce forum. En effet le choix d'une interface graphique qui donne une impression trop « logiciel » peut avoir tendance à repousser les utilisateurs.

Cependant malgré la convivialité de l'interface graphique lorsque ces deux programmes tournent ils offrent à l'utilisateur un nombre non négligeable de possibilités que voici :

- L'utilisateur possède un compte unique protégé par un mot de passe
- Il peut s'enregistrer lui-même avec une facilité à toutes épreuves
- Il peut créer un article et y poster des messages contenant des images (une image par message)
- Il peut trouver facilement quels articles il a déjà lus et ceux qu'il n'a pas lus grâce a une petite lumière verte qui s'allume dés qu'un article contient un message non lu par l'utilisateur.
- Les articles possédant les messages les plus récents seront situés en tête de liste. Les articles sont donc classés dans l'ordre chronologique de celui qui contient le dernier message le plus récent à celui dont le dernier message est le plus ancien ce qui permet à l'utilisateur de toujours avoir en tête de liste les sujets d'actualité.

- Il peut répondre aux articles des autres utilisateurs toujours avec la possibilité de joindre une image à ses messages
- Il peut mettre en forme le texte de ses messages grâce à un système de balises très simple à utiliser d'autant plus que le programme client offre une interface graphique qui gère ces balises.
- Il peut de plus rendre plus attractif ses messages grâce à de nombreux smileys
- Il peut éditer ses messages (mais pas ceux des autres utilisateurs).
- II peut signaler les erreurs qu'il a rencontrées durant l'utilisation du forum auprès de l'administrateur.
- Il peut consulter les statistiques du forum.
- Des jeux sont mis à la disposition de l'utilisateur pour patienter en attendant une réponse à un article.
- Il peut aussi discuter avec tous les utilisateurs via une messagerie instantanée.

On peut donc dire que notre objectif de rendre ce forum accueillant et accessible à tous tout en offrant un nombre de fonctionnalités raisonnables est une réussite.

Il en est de même pour le serveur qui est comme dit précédemment très simple d'utilisation et ne nécessite aucune connaissance approfondie dans le domaine de l'informatique.

II/ Le programme client :

1/ Présentation :

Le programme client est composé de fonctions qui gèrent l'interface graphique en GTK et de fonctions qui gèrent la communication avec le serveur. Ces fonctions sont implémentées dans quatre fichiers sources :

- *Le fichier main.c :* Ce fichier contient la fonction main du programme.
- *Le fichier interface.c :* Ce fichier contient l'ensemble des fonctions gérant l'interface graphique qui ne nécessite aucune communication avec le serveur.
- *Le fichier communication.c :* Ce fichier contient les fonctions qui gèrent toutes la communication avec le serveur. On y trouve aussi les fonctions de gestion de l'interface graphique qui ont besoin de communiquer avec le serveur afin de déterminer leur contenu.
- *Le fichier communicationDialogue.c :* Ce fichier contient l'ensemble des fonctions qui permettent la communication par le T'Chat (bouton «Dialogue» dans l'interface graphique). Certaines de ces fonctions font également évoluer l'interface graphique du T'Chat (par exemple : celle qui affiche les nouveaux messages postés dans le T'Chat).

L'affichage de la liste des articles dans la fenêtre principale se fait dans l'ordre de l'article qui contient le message le plus récent a celui qui contient le message le plus ancien.

Lors de la lecture d'un article composé d'images, celles-ci seront récupérées dans le répertoire « tpmfile » pour ensuite être affichées. Le contenu de ce répertoire sera effacé à la fin de chaque session du forum.

Les jeux sont stockés dans le répertoire « bin » du forum, il ne faut donc pas les supprimer.

Les images du forum sont dans le répertoire « img » qui ne doit donc pas être supprimé si on veut un rendu correct.

Lors du lancement du T'chat, un serveur est lancé sur la machine du client afin de pouvoir recevoir l'ensemble des messages de la messagerie instantannée.

ATTENTION : Pour que le forum fonctionne correctement s'il est lancé a partir d'un terminal, il faut qu'il soit lancé a partir du répertoire client. Si cela n'est pas respecté alors les images et les jeux ne fonctionneront pas.

2/ Listes des variables globales du client :

- *char SERVNAME[100]* : Adresse IP du serveur entrée par l'utilisateur sauvegardée dés la première page.
- *int dialogue :* Variable qui permet de savoir si le t'chat est lancé ou non vaut 1 si le T'chat est déjà lance 0 sinon
- *int acce :* Utilisée pour la connexion, cette variable vaut 1 si l'utilisateur a le droit de se connecter 0 sinon
- *int enreg* : utilisée pour l'enregistrement, cette variable vaut 0 si l'utilisateur n'a pas tenté de s'enregistrer, vaut 1 si le Pseudo est disponible, vaut -1 si le pseudo demandé n'est pas disponible
- *char lastArticle[SIZETITRE+1]* : chaine contenant le dernier article lu par l'utilisateur
- *fenetrePrincipale *GlobalFen :* Pointeur vers la fenêtre principale lorsque la connexion est acceptée.
- *fenetrePrincipale *DialogueFen :* Pointeur vers la fenêtre du T'chat
- *int PidSudoku :* Pid du processus qui permet de jouer au sudoku vaut -1 si le jeu n'est pas lance
- *int PidZork :* pid du processus qui permet de jouer a Zork vaut -1 si le jeu n'est pas lance

3/ Listes des fonctions du programmes client:

a. Fonctions du fichier main.c :

- *int main (int argc, char *argv[])* : Main du programme qui lance l'interface graphique et redirige la sortie d'erreur standard vers un fichier erreur.txt qui pourra être lu par le client afin que celui-ci puisse signaler une éventuelle erreur.

b. Fonctions du fichier interface.c :

Rappel : Les fonctions ci-dessous ne font jamais appel au serveur. Elles appellent des fonctions contenues dans communication.c et communicationDialogue.c qui se chargent de faire la communication avec le serveur. Voila pourquoi on retrouve dans interface.c des fonction qui on besoin de communiquer avec le serveur mais ces fonctions appellent juste les fonctions de communication.c et commnunicationDialogue.c en leur donnant les bons arguments.

- *void pageConnexion(char *message) :* fonction qui génère la fenêtre de connexion.
- *void pagePrinc(gpointer pwindow2) :* fonction qui génère la page principale .
- *void verifMdp(GtkWidget* pBouton, gpointer data)* : fonction qui gère l'interface graphique en fonction de la validité du pseudo et du mot de passe de l'utilisateur.
- *void actualiser(GtkWidget* pBouton, gpointer pwindow2)* : Fonction qui permet d'actualiser la liste des sujets dans la fenetre principale.
- *void enregistrement(GtkWidget* pBouton, gpointer data)* : fonction qui gère l'appel à la fenêtre enregistrement.
- *void aideEnregistrement(GtkWidget* pBouton, gpointer data)* : fonction qui génère une fenêtre d'aide pour l'enregistrement.
- *void retourConnexion(GtkWidget* pBouton, gpointer data)* : fonction qui permet de retourner à la page de connexion.
- *void verifEnregistrement (GtkWidget* pBouton, gpointer data)* : fonction qui gère l'interface graphique en fonction du succès ou de l'échec de l'enregistrement de l'utilisateur
- *void fenetreEnregistrement(conect *arg)* : fonction qui génère la fenêtre d'enregistrement

- *void traitementText(char *s)* : cette fonction est utilisée pour empêcher les utilisateurs de s'enregistrer avec des pseudo comportant les caractères *,.,\,#, » … tous les caractères spéciaux seront remplacés par le caractère « _ ».
- *void aideForum(GtkWidget* pBouton, gpointer data)* : cette fonction génère la fenêtre d'aide pour l'utilisation du forum.
- *void pageJeu(GtkWidget* pBouton, gpointer FenPrecedente)* : fonction qui génère la page jeu.
- *void sudoku(GtkWidget* pBouton, gpointer FenPrecedente)* : fonction qui lance le jeu SUDOKU
- *void zork(GtkWidget* pBouton, gpointer FenPrecedente)* : fonction qui lance le jeu ZORK
- *void statistisques(GtkWidget* pBouton, gpointer Fen)* : fonction qui lance la fenêtre de statistiques.
- *void lectureArticle (GtkWidget* pBouton, gpointer data)* : fonction qui appelle la fenêtre lecture article et prépare les variables globales en conséquent. En effet le titre de l'article lu est stocké en variable globale et la fonction lecture article le récupère sur le label du bouton correspondant à l'article sur lequel a cliqué l'utilisateur.
- *void aideLecture(GtkWidget* pBouton, gpointer data)* : fonction d'aide qui expose les choix qui s'offrent a l'utilisateur lorsqu'il se trouve sur une fenêtre lecture article
- *void repondre(GtkWidget* pBouton, gpointer nom*) : fonction qui génère la fenêtre permettant de répondre à un article.
- *void aideRepondre(GtkWidget* pBouton, gpointer data)* : fonction qui génère la fenêtre d'aide pour aider l'utilisateur lorsqu'il se trouve sur la fenêtre répondre.
- *void nouvelArticle(GtkWidget* pBouton, gpointer FenPrecedente)* : fonction qui génère la fenêtre permettant de créer un nouvel article
- *void envoyerMessage(GtkWidget* pBouton, gpointer data)* : fonction qui gère l'interface graphique lors de l'envoi d'une réponse à un article en fonction du résultat de l'envoi (C'est-à-dire si le message a été correctement envoyé ou non). Cette fonction connait le nom de l'article auquel l'utilisateur répond car il a été stocké en variable globale lorsque celui-ci a lu l'article auquel il répond.
- *void envoyerArticle(GtkWidget* pBouton, gpointer data))* : fonction qui gère l'interface graphique lors de la création d'un article en fonction du résultat de l'envoi (C'est-à-dire si le message a été correctement envoyé ou non).
- *void fentreLectureArticle(GtkWidget* pBouton, gpointer nom)* : fonction qui génère la fenêtre avec tous les messages d'un article
- *void fenetreErreur(char* msg,fenetrePrincipale *arg)* : cette fonction génère une fenêtre d'erreur avec le message msg sur une fenêtre arg. On appelle cette fonction dans toutes les fonctions qui génèrent des fenêtres lorsque celles-ci peuvent rencontrer des erreurs (La plupart étant des erreurs de connexion avec le serveur).
- *GtkWidget* listeSmiley()* : fonction qui génère la liste des smileys elle est utilisée par les fonctions repondre et nouvelArticle
- *void quitterForum()* : fonction qui permet de fermer le forum proprement (c'est-à-dire en prévenant le serveur que le client se déconnecte et en quittant l'interface graphique)
- *void returnPagePrinc(GtkWidget* pBouton, gpointer pwindow2)* : fonction qui permet le retour vers la page principale.
- *void chercherImage(GtkWidget* pBouton, gpointer Fen)* : fonction qui génère la fenêtre de recherche d'une image permettant à l'utilisateur de choisir l'image qui veut joindre à son message.

- *int imageValide(char *chm)* : Fonction qui s'assure que l'image choisie par l'utilisateur est bien au format jpg. *Cette fonction est utilisée dans la fonction chercher image.*
- *void editer(GtkWidget* pBouton, gpointer data)* : Fonction qui génère la page éditer qui permet d'éditer un message.
- *fenetrePrincipale * loadWindow(fenetrePrincipale * Fenprec)* : fonction qui génère la page de chargement lors de chaque changement de page.
- *void editerMessage(GtkWidget* pBouton, gpointer data)* : fonction qui gère l'interface graphique en fonction de la réussite de l'édition d'un message.
- *void dialoguer(GtkWidget* pBouton, gpointer nom)* : fonction qui lance le thread qui va gérer le T'CHAT
- *void finDialogue(GtkWidget* pBouton,gpointer Fen)* : fonction qui permet de quitter le T'Chat proprement.
- *void finDialogueErr(GtkWidget* pBouton,gpointer data)* : fonction qui permet de quitter le T'Chat proprement en cas d'erreur.
- *void dialoguerThread(void *nom)* : fonction qui génère la fenêtre du t'chat qui lance le thread qui va gérer le serveur du t'chat.
- *void connectDialogue(gpointer *Fen)* : fonction qui gère l'interface graphique en fonction du résultat de la demande la connexion d'un utilisateur au t'chat
- *void envoyerMessageDialogue(GtkWidget* pBouton,gpointer *Fen)* : fonction qui permet d'envoyer un message à partir du t'chat
- *void *lancerServeurDialogue(gpointer fen)* : fonction qui permet de lancer le serveur du t'chat à partir d'un thread
- *void aideDialogue(GtkWidget* pBouton, gpointer data)* : fonction qui génère la fenêtre d'aide au dialogue.
- *void signalerErreur(GtkWidget* pBouton, gpointer data)* : fonction qui génère la fenêtre signaler erreur permettant d'envoyer un message à l'administrateur du forum concernant une éventuelle erreur rencontrée.
- *void balisesGras(GtkWidget* pBouton, gpointer text)* : fonction ajoutant les balises permettant de mettre le texte en gras lorsque le client clique sur le bouton correspondant lors de la rédaction d'un message
- *void balisesItalique(GtkWidget* pBouton, gpointer text)* : fonction ajoutant les balises permettant de mettre le texte en italique lorsque le client clique sur le bouton correspondant lors de la rédaction d'un message.
- *void balisesSouligne(GtkWidget* pBouton, gpointer text)* : fonction ajoutant les balises permettant de mettre le texte en souligné lorsque le client clique sur le bouton correspondant lors de la rédaction d'un message.
- *void balisesBig(GtkWidget* pBouton, gpointer text)* : fonction ajoutant les balises permettant de mettre le texte en gros lorsque le client clique sur le bouton correspondant lors de la rédaction d'un message.
- *void balisesBarre(GtkWidget* pBouton, gpointer text) :* fonction ajoutant les balises permettant de mettre le texte en gras lorsque le client clique sur le bouton correspondant lors de la rédaction d'un message.
- *void balisesInd(GtkWidget* pBouton, gpointer text)* : fonction ajoutant les balises permettant de mettre le texte en indice lorsque le client clique sur le bouton correspondant lors de la rédaction d'un message
- *void balisesExp(GtkWidget* pBouton, gpointer text)* : fonction ajoutant les balises permettant de mettre le texte en exposant lorsque le client clique sur le bouton correspondant lors de la rédaction d'un message

- *void balisesSmall(GtkWidget* pBouton, gpointer text)* : fonction ajoutant les balises permettant de mettre le texte en petit lorsque le client clique sur le bouton correspondant lors de la rédaction d'un message.
- *void balisesTt(GtkWidget* pBouton, gpointer text)* : fonction ajoutant les balises permettant de mettre le texte en télétype lorsque le client clique sur le bouton correspondant lors de la rédaction d'un message
- *GtkWidget* boxMiseEnPage(GtkWidget *textView)* : génère les boutons de mise en page du texte. Cette fonction est appelée lors de la création des fenêtres répondre et nouvel article.

c. Fonctions du fichier commnunication.c :

- *GtkWidget* listeMessage(char *titre, char *Pseudo)* : Fonction qui crée l'objet graphique contenant les messages de l'article « titre » qu'elle récupère suite à une communication avec le serveur, elle envoie aussi au serveur le Pseudo de l'utilisateur qui lit l'article afin que le serveur puisse faire le nécessaire pour que l'article soit affiché comme non lu pour l'utilisateur en question. Elle gère aussi les simley et les éventuelles mises en page du texte afin d'obtenir le rendu visuel souhaité par l'utilisateur. Elle crée aussi pour chaque message un bouton permettant de l'éditer mais uniquement si l'utilisateur à le droit (ce droit est une information donnée par le serveur). Cette fonction est appelée lors de la lecture d'un article.
- *GtkWidget* listeArticle(char *Pseudo)* : Fonction qui génère la liste des articles suite à une commnunication avec le serveur et qui détermine l'ordre des article (du dernier modifié a celui qui n'as plus été modifié depuis longtemps) et si l'utilisateur a déjà lu ou non l'article grâce aux informations fournies par le serveur (c'est-à-dire date de la dernière modification de l'article et date de la dernière lecture de l'article par l'utilisateur en question). Le serveur trouvera les informations de l'utilisateur car le client envoie le pseudo de l'utilisateur au serveur lors de cette manipulation. Cette fonction est appelée lors des appels à la création de la fenêtre principale.
- *int controleIdentifiant(char *Pseudo, char *Mdp)* : fonction qui est appelée lors de la vérification de la validité du mot de passe et du pseudo, renvoie 1 si c'est bon , 0 si il y a une erreur et -2 si il y a une erreur de connexion. Cette valeur est déterminée suite à une communication avec le serveur.
- *int controleEnregistrement(const char *Pseudo,const char *Mdp)* : Cette fonction est appelée lors de la demande d'enregistrement d'un client, elle reçoit le pseudo à créer et le mot de passe à lier à ce pseudo. Elle renvoie 1 si tout est bon, -1 si le pseudo est déjà pris, -2 si il y a une erreur de connexion et -3 si les enregistrements sont fermés.
- *int nouvelArticleSocket(char *art,char *mess,char *pseudo, char* chmImg)* : fonction qui est appelée lors de la demande de création d'un nouvel article. Elle envoie au serveur le titre de l'article à créer, le message qu'il contient, le pseudo de l'utilisateur, le chemin vers une éventuelle image jointe. Elle renvoie 1 si tout s'est bien passé, 0 si le titre de l'article existe déjà et -2 si on a une erreur de connexion.

- *int repondreSocket(char *art,char *mess,char *pseudo,char *chmImg)* : Cette fonction est appelée lors de la demande d'envoi d'une réponse à un article. Elle reçoit le titre de l'article auquel l'utilisateur veut répondre, le pseudo de l'utilisateur, le message et le chemin vers une éventuelle image à joindre. Elle envoie au serveur toutes ces informations et retourne à la fonction appelante 1 si tout s'est bien passé, 0 si l'article n'a pas été trouvé (ce qui peut se produire suite a une opération de l'administrateur pendant que l'utilisateur rédige son message), -2 si il y a eu un échec de connexion et -4 si l'article est complet (c'est-à-dire que le serveur refuse de mettre plus de messages dans l'article car il estime qu'il y en a trop).
- *int connection()* : Cette fonction est appelée au début de chaque fonction qui nécessite une communication avec le serveur afin de connecter le client au serveur. Renvoie -2 s'il y a eu un échec de connexion, renvoie le numéro de la socket de communication sinon.
- *void deconnection(int desc)* : Cette fonction est appelée à la fin de chaque fonction qui nécessite une communication avec le serveur, elle déconnecte l'utilisateur du serveur.
- *void deconnectionUtilisateur(char *pseudo)* : Cette fonction est appelée lorsque l'utilisateur quitte le forum afin de prévenir le serveur que le client s'est déconnecté. C'est pour cela que cette fonction prend en argument le pseudo de l'utilisateur.
- *GtkWidget* statistiqueSocket(char *profil)* : Cette fonction génère les éléments contenus dans la fenêtre statistique qu'elle obtient grâce au serveur.
- *void chargerImage(int sock,char *chemin)* : Cette fonction est appelée par la fonction listeMessage si il y a une image à réceptionner. Elle a donc pour rôle de charger une image et de la mettre au chemin « chemin ».
- *GtkWidget* constructionImageDimentionne(char *chm,int width,int height)* : fonction qui construit une image située au chemin « chm » dans les dimensions width et height données. Cette fonction est appelée juste après chargerImage dans la fonction listeMessage.
- *void envoieImage(int sock,char *chmImg)* : fonction qui est appelée lors de l'envoi d'un nouvel article ou lors de l'envoi d'un message pour déterminer si une image est jointe, et si c'est le cas elle envoie l'image au serveur, sinon elle prévient le serveur qu'il n'y aura pas d'image jointe
- *void OnZoomIn(GtkWidget *pWidget, gpointer data)* : appelée par le bouton Zoom avant créé lors de la création d'une image, cette fonction permet de zoomer sur l'image si l'utilisateur le souhaite.
- *void OnZoomOut(GtkWidget *pWidget, gpointer data)* : appelée par le bouton Zoom arrière créé lors de la création d'une image, cette fonction permet de dézoomer l'image si l'utilisateur le souhaite.
- *char* get_text_edit(char* article,int numMess,char* buffer)* : Appelée lors d'une demande d'édition d'un message cette fonction permet de récupérer le texte à éditer. Pour cela elle devra envoyer au serveur le nom de l'article concerné et le numéro du message à éditer.
- *int editerSocket(char *art,char *mess,char *pseudo,char *chmImg,int numMess)* : Cette fonction envoie le nouveau texte d'un message lorsque celui-ci est édité. Pour cela elle a besoin du pseudo, du nom de l'article, du nouveau contenu, du nouveau chemin vers une éventuelle image et du numéro du message. Puis elle envoie tout au serveur qui va gérer lui-même les changements.

- Les deux prochaines fonctions sont appelées dans la fonction listeMessage. Elles permettent de gérer les balises de mise en page du texte afin qu'entre chaque smiley et entre le début et la fin d'un message on ait le même nombre de balises ouvrantes que de balises fermantes. On rappelle qu'un message dans l'interface graphique est composée de plusieurs labels qui sont séparés par des images (les smileys), d'où l'intérêt d'avoir ces deux fonctions qui permettent de conserver les propriétés du texte lorsque celui-ci se retrouve coupé en deux par un smiley.
  *void traiterFinlabel(char *sTexte,int it,int b,int big,int s,int sub,int sup,int small,int tt,int u);*
  *int traiterDebutlabel(char *sTexte,int it,int b,int big,int s,int sub,int sup,int small,int tt,int u);*

d. Fonctions du fichier communicationDialogue.c :

Rappel : Les fonctions contenues dans communicationDialogue.c concernent uniquement le T'CHAT. Le t'chat permet à plusieurs utilisateurs inscrits au forum de communiquer entre eux.

- *int traiterCommande(int socketService,fenetrePrincipale* fen) :* Cette fonction permet de traiter les commandes envoyées au serveur du client (serveur qui est lancé lors du lancement du T'Chat). Cette fonction est appelée par le serveur pour chaque client.
- *void lancerServeur(gpointer fen) :* Fonction qui lance le serveur du client pour le T'chat. En cas d'erreur l'interface graphique sera modifiée par cette fonction afin d'avertir l'utilisateur.
- *void finDialogueServeur()* : Fonction qui permet au forum de se connecter à son propre serveur de T'CHAT pour lui envoyer la commande lui demandant de se couper. Cela permet de couper proprement le serveur de T'CHAT sans briser le relai pendant l'envoi d'un message du serveur T'CHAT du programme serveur.
- *int connectionDialogue() :* Cette fonction permet de se connecter au serveur T'Chat du programme client. Elle est appelée par la fonction finDialogueServeur.
- *void deconnectionDialogue(int desc) :* Cette fonction permet de se déconnecter du serveur T'Chat du programme client. Elle est appelée par la fonction finDialogueServeur.
- *int connectionUserDialogue(char *nom) :* Cette fonction permet d'informer le serveur T'CHAT du programme serveur que l'utilisateur s'est connecté au T'CHAT.
- *void deconnectionUserDialogue(char *nom)* Cette fonction permet d'informer le serveur T'CHAT du programme serveur que l'utilisateur s'est déconnecté au T'CHAT.
- *void envoyerMessageDialogueSocket(char *user,char *txt) :* Cette fonction est appelée lorsque l'utilisateur envoie un message via le t'chat. Elle permet d'envoyer le message tapé par l'utilisateur au serveur. On envoie donc le pseudo de l'utilisateur et le message au serveur T'chat du programme serveur qui se chargera de lier les deux et d'envoyer ce message a tous les membres connectés.
- *void lireMessageDialogue(int socketService,fenetrePrincipale *fen, int taille) :* fonction appelée par la fonction traiterCommande qui s'occupe de réceptionner et d'afficher dans la fenêtre du t'chat les messages reçus (Ces messages sont envoyés par le serveur t'chat du programme serveur)

III/ Le programme serveur :

1/ Présentation générale :

Le programme serveur lance deux serveurs :

- *Un serveur multi-clients pour le forum :* serveur qui lance un thread par client ce qui permet de gérer plusieurs clients à la fois.
- *Un serveur simple pour le T'Chat de forum :* Serveur qui traite les messages envoyé un par un et renvoie les messages reçus à l'ensemble des utilisateurs connectés au t'chat.

Le serveur enregistrera les utilisateurs dans le répertoire « Users », les articles dans le répertoire « Articles », et les archives dans le répertoire « archives ».

Il est autonome, en effet Il est prévu qu'un nombre maximum de 100 articles soit gérés par le serveur (si ce nombre est atteint on archive les articles les plus vieux), les articles ne pourront pas être trop anciens car au bout d'une semaine ils seront archivés, de plus chaque articles pourra contenir un maximum de 100 messages. Le serveur ne permettra d'archiver que 50 articles, une fois ce nombre atteint les archives les plus anciennes seront supprimées pour laisser la place aux plus récentes. A cela s'ajoute le fait que le serveur autorise son répertoire « Users » qui contient les répertoires des utilisateurs à atteindre une taille maximal de 1Go (Ce qui permet une quantité importantes d'utilisateurs enregistrés car on estime que la taille maximale d'un répertoire utilisateur est aux environs de 10.5Ko). Tout cela permet d'assurer le fait que le serveur ne dépassera jamais une certaine taille sur le disque de la machine sur laquelle il tourne. Après un petit calcul on trouve que la taille maximale que le serveur peut atteindre sur le disque est aux environs de 1.165Go ce qui est convenable sachant que nous avions prévu une taille maximale à ne jamais dépasser qui est de 2Go. Par conséquent pour le bon fonctionnement du serveur nous demanderons à l'ingénieur qui gère le serveur de s'assurer que celui-ci a toujours 2Go d'espace mémoire qu'il peut occuper.

Une fois par jour le serveur va lancer une opération d'archivage qui va archiver les articles les plus vieux.

Malgré cette autonomie le serveur lance une petite interface graphique permettant à l'ingénieur système de supprimer les utilisateurs, les articles ou les archives qu'il souhaite. Il peut de plus consulter les plaintes déposées par les utilisateurs sur le fonctionnement du logiciel. Ces plaintes pourront êtres utilisées pour faire d'éventuelles mises à jour sur des bugs non constatés lors de la période de tests du programme. Elles seront stockées dans le répertoire « 0 » du répertoire « Articles » car elles fonctionnent comme des messages postés dans l'article « signaler erreur ! » sauf que cet article n'est pas concerné par les opérations d'archivages.

L'ingénieur pourra avoir un aperçu des mouvements des utilisateurs sur le forum s'il lance le serveur depuis un terminal. En effet le serveur utilise la sortie standard pour afficher les opérations demandées par les utilisateurs.

Pour implémenter cela le programme serveur est composé de cinq fichiers sources :

- *Le fichier serveurMain.c :* Fichier qui contient le main du programme, les fonctions qui lancent les deux serveurs.
- *Le fichier serveur.c :* Qui contient l'ensemble des fonctions qui gère la communication avec le client et les enregistrements que cela implique.
- *Le fichier serveurDialogue.c : Qui contient l'ensemble des fonctions qui permettent la communication entre tous les clients du T'chat.*
- *Le fichier gestionServeur.c :* Qui contient toutes les fonctions permettant l'archivage et la suppression des d'éléments gérés par le serveur.
- *Le fichier interfaceServeur.c :* Qui contient toutes les fonctions permettant de gérer l'interface graphique du serveur.

ATTENTION :

- Si le serveur est lancé depuis un terminal il faut le lancer depuis le répertoire « serveur » sinon celui-ci ne fonctionnera pas !
- Assurez vous que le répertoire serveur contient bien les répertoires : « Users », « Articles » et « archives ».
- Le répertoire Articles doit contenir le répertoire 0 avec un fichier 0 qui contient la chaine de caractères : « signaler erreur ! ».

2/ Présentation détaillée du fonctionnement du serveur :

Voici quelques explications concernant le fonctionnement du serveur :

Comme on l'a vu précédemment : « *Le serveur enregistrera les utilisateurs dans le répertoire « Users », les articles dans le répertoire « Articles », et les archives dans le répertoire « archives ».* »

Les utilisateurs les articles et les archives sont tous stockés dans des répertoires numérotés allant de 0 à un nombre maximal.

- *Pour les utilisateurs :* Les répertoires des utilisateurs contiennent un fichier « user » contenant le pseudo de l'utilisateur, un fichier « mdp » contenant son mot de passe et des fichiers allant de 0 à un nombre maximal (<=150 car 100 articles + 50 archives) qui contiennent les titres des articles lus par l'utilisateur. A chaque lecture, réponse ou création d'un article, ces fichiers sont crées (on crée un fichier nommé par le numéro du dernier fichier du répertoire de l'utilisateur + 1 et on y écrit le titre de l'article lu) ou si l'utilisateur a déjà lu l'article ils sont modifiés (on efface le contenu du fichier et on réécrit le titre). Le but d'une telle opération est que pour chaque utilisateur on puisse, à partir des dates de dernières modifications, savoir si l'utilisateur a déjà lu ou non un article et toutes ses réponses.

- *Pour les articles et les archives :* Les archives étant d'anciens articles on se contentera de l'explication sur les articles. Les articles des fichiers numérotés de 0 à 100 (car 100 messages par articles au maximum). Le fichier 0 contient le titre de l'article, les autres fichiers sont donc les messages de réponses à cet article. Chaque fichier de 1 à 100 a un fichier « u » portant le même numéro que lui (c'est-à-dire le fichier 1 a un fichier associé qui est u1 le fichier 2 à un fichier u2 etc.) contenant le pseudo de l'utilisateur qui l'a écrit. Cela nous permet de savoir pour chaque message quel utilisateur à le droit de l'éditer. Le principe est le même si une image est jointe à un message alors elle porte le même numéro que le message auquel elle est associée avec pour suffixe .jpg (si une image est associée au message 1 alors elle aura pour nom 1.jpg).

Pour finir, la liste des utilisateurs connectés au T'chat et au forum est tout simplement gérée dans une liste chainé qui contient le pseudo de l'utilisateur pour la liste du forum et on ajoute l'adresse IP pour celle du t'chat.

3/ Listes des principales variables globales du serveur :

- Pour le serveur du forum :
  - *int finForum :* vaut 0 quand le serveur tourne vaut 1 quand on veut quitter le serveur
  - *users *listeUsers :* liste des utilisateurs connectés au serveur (une structure user contient un champ nom et un pointeur vers l'utilisateur suivant).
  - *pthread_mutex_t mutexListeUsers :* Mutex permettant de gérer les accès au niveau de la liste des utilisateur et au niveau des accès au répertoire « Users ».
  - *sem_t mutex[NBMAXARTICLES+1][NBMAXMESSAGES] :* Tableau de sémaphore pour les accés aux messages des articles
  - *sem_t mutexArchive :* sémaphore permttant l'accés à la liste des archives.

- Pour le serveur du T'chat :
  - *userDialogue *liste :* liste des utilisateurs connectés au serveur du T'Chat

4/ Listes des fonctions Utilisées :

a. Dans le fichier serveurMain.c :

- *void* lancerServeurForum() :* Fonction qui lance le serveur du forum
- *void * lancerServeurDialogue(void * arg) :* fonction qui lance le serveur du T'chat
- *int main(int argc, char *argv[]) :* fonction main du serveur qui lance trois thread un pour le serveur du forum, un pour le serveur du t'chat et un pour la gestion de l'archivage. Le processus père s'occupe ensuite de l'interface graphique.

b. Dans le fichier serveur.c :

L'ensemble des fonctions de ce fichier permettent la gestion du serveur. Voila pourquoi les sémaphores sont gérés dans ces fonctions de façon à ce qu'on ne puisse pas lire et écrire ou lire et supprimer ou encore écrire et supprimer en même temps un même message. Voila pourquoi lors de l'implémentation de ces fonctions il faut penser à gérer le sémaphore. Pour cela on a un sémaphore par message et un sémaphore qui autorise l'accès au répertoire « Users » et a la liste des utilisateurs connectés. Les fonctions suivantes doivent donc pour la plupart tenir compte de ces contraintes.

- *void* traiterCommande(void* socket) :* cette fonction permet de traiter les demande du client. A chaque connexion, le client envoie une commande est c'est à partir de cette commande que le serveur détermine l'action qu'il doit exécuter. Les commandes possibles sont :
  - */e :* appelle la fonction qui gère l'enregistrement d'un utilisateur
  - */c :* appelle la fonction qui gère la connexion d'un utilisateur
  - */r :* appelle la fonction qui gère la réponse a un article
  - */n :* appelle la fonction qui gère la création d'un nouvel article
  - */l :* appelle la fonction qui gère la demande de lecture d'un article
  - */s :* appelle la fonction qui gère la demande de la liste des articles
  - */i :* appelle la fonction qui gère la demande des statistiques du serveur
  - */f :* appelle la fonction qui gère la demande de déconnexion d'un utilisateur
  - */re :* appelle la fonction qui gère la demande de récupération d'un message précis (utilisée pour l'édition d'un message)
  - */ed :* appelle la fonction qui gère la demande d'un remplacement d'un message précis (utilisée pour l'édition d'un message)

- *void connection(int socketService) :* Cette fonction gère la connexion d'un client. Elle informe le programme client si le mot de passe ou le pseudo sont incorrects et connecte l'utilisateur dans le cas contraire
- *void enregistrer(int socketService)* : Fonction qui gère l'enregistrement d'un client. Elle enregistre le client si le pseudo envoyé par le programme client n'est pas déjà utilisé et informe le client que l'enregistrement est impossible dans le cas contraire.
- *void newArticle(int socketService) :* fonction qui gère la création d'un nouvel article. Elle récupère via une communication le pseudo de l'utilisateur, le titre de l'article, le contenu gère le fait que si le client a écrit cet articles alors c'est comme s'il l'avait déjà lu et donc sera affiché comme lu sur le programme client. Si le titre de l'article est déjà utilisé alors le client sera informé.
- *void repondre(int socketService) :* fonction qui gère la réponse d'un utilisateur à un article. Elle gère aussi le fait que si le client a répondu à un article alors si son message est le dernier, l'article est considéré comme lu par ce client. Si l'article n'a pas été trouvé (ce qui peut se produire après des opérations de maintenance) ou que l'article est complet (nombre maximal de messages atteint) alors le client est informé.
- *void listerArticles(int socketService) :* Cette fonction envoie la liste des titres des articles au client afin qu'il puisse savoir quels articles sont disponibles. Elle reçoit le pseudo de l'utilisateur ce qui lui permet d'envoyer au programme client la date de la dernière lecture d'un article (grâce aux fichiers contenus dans le répertoire de l'utilisateur) et la date de la dernière modification de l'article. Le programme client pourra ainsi déterminer s'il doit afficher l'article comme lu ou comme non lu. Si jamais le client n'a pas encore lu l'article alors le fichier comportant le titre de l'article n'apparait pas dans la liste des fichiers qui contiennent les titres des articles lus par l'utilisateur. Le serveur envoie donc la date 0 et ainsi l'article concerné sera considéré comme non lu par le programme client.
- *void listerMessages(int socketService) :* Cette fonction récupère via une communication le pseudo de l'utilisateur et le titre de l'article qu'il veut lire. Elle s'occupe ensuite de lister les messages de l'article puis de faire le nécessaire dans le répertoire de l'utilisateur pour que le l'article apparaisse comme lu. Si le fichier n'a pas été trouvé alors cette fonction informe le programme client qui s'occupera de gérer l'interface afin d'informer l'utilisateur que le message qu'il veut lire n'existe plus (ce qui peut se produire après des opérations de maintenance).
- *void supprimeAllUsersConect()* : cette fonction permet de déconnecter tous les utilisateurs connectés. Elle est appelée lors de la fermeture du serveur.
- *void supprimerUserConect(char *pseudo) :* cette fonction retire l'utilisateur Pseudo de la liste des utilisateur connectés. Elle est appelée lors de la demande de déconnexion d'un utilisateur
- *int connectUser(char *pseudo) :* Cette fonction ajoute un utilisateur à la liste des utilisateurs connectés. Elle est appelée par la fonction qui gère la connexion d'un utilisateur.
- *void deconnecterUser(int socketService) :* fonction qui gère la déconnexion d'un utilisateur.
- *void information(int socketService) :* Fonction qui envoie les statistiques du serveur au client qui les demande (liste des utilisateurs, nombre d'utilisateurs connecté, nom de ces utilisateurs, nombre de messages, nombre d'article etc…).
- *int nombreDeMessages() :* renvoie le nombre total de messages
- *int nombreDeArticles() :* renvoie le nombre total d'articles
- *int nombreUtilisateurs() :* renvoie le nombre total d'utilisateurs enregistrés
- *int nombreUtilisateursConnect() :* renvoie le nombre total d'utilisateurs connectés
- *void envoieImg(int socketService, char *chm) :* fonction qui est appelée lors de la lecture d'un article par un utilisateur à chaque fois qu'une image est liée à un message de l'article concerné.

- *void verificationImage(int socketService,char *chemin) :* Fonction qui vérifie si une image est jointe à un message lorsque le client envoie ce message et si c'est le cas elle l'enregistre en prenant soin du fait que l'image doit porter le même numéro que le message auquel elle est liée (nom de l'image = nomMessage.jpg nomMessage étant en fait un numéro).
- *void recupererMessage(int socketService) :* Cette fonction est utilisée pour l'édition d'un message. Elle récupère grâce à une communication avec le programme client un titre d'article et un numéro de message à partir desquels elle retrouve le message et l'envoie au client qui s'occupera de l'intégrer à l'interface graphique pour le modifier.
- *void editer(int socketService) :* Le principe est le même que celui de la fonction précédente sauf que cette fonction récupère le nouveau message et remplace l'ancien message.
- *void initSemaphore() :* Cette fonction est appelée au lancement du programme dans la fonction main afin d'initialiser l'ensemble des sémaphores.
- *void * gestionArchivage(void * arg) :* fonction qui est lancée par un thread au lancement du programme serveur ce thread lance une fois par jours un archivage. Il lance aussi un archivage dés le lancement du serveur. Les sémaphores sont donc tous bloqués au cours d'une telle opération.
- *void lockAllSemaphore() :* fonction qui bloque tous les sémaphores liées aux articles.
- *void unlockAllSemaphore() : ;*fonction qui débloque tous les sémaphores liées aux articles.
- *int removeArticle(char *article) :* fonction qui appelle la fonction de suppression de l'article article tout en gérant les sémaphores nécessaires.
- *int removeArchive(char *article) :* fonction qui appelle la fonction de suppression de l'archive article tout en gérant les sémaphores nécessaires.
- *int removeUser(char *user) :* fonction qui appelle la fonction de suppression d'un utilisateur user tout en gérant les sémaphores nécessaires .
- *GtkWidget* lirePlaintes() :* fonction qui permet à l'ingénieur système de consulter les plaintes des utilisateurs qui se trouvent dans le répertoire « 0 » du répertoire « Articles »
- *void lockEndSemaphore() :* fonction qui bloque la totalité des sémaphores du serveur. Elle est appelée à la fermeture du serveur afin qu'on ne puisse pas avoir de clients en écriture au moment ou le serveur se coupe et ainsi éviter les erreurs lors du redémarrage du serveur.
- *GtkWidget* listeArchive()* : Fonction qui génère la liste des archives sous forme d'un objet graphique pour l'interface du serveur tout en tenant compte des sémaphores.
- *GtkWidget* listeArticles() :* Fonction qui génère la liste des articles sous forme d'un objet graphique pour l'interface du serveur tout en tenant compte des sémaphores.
- *GtkWidget* scrollListeUsers() :* Fonction qui génère la liste des utilisateurs enregistrés sous forme d'un objet graphique pour l'interface du serveur tout en tenant compte des sémaphores.

c. <u>Dans le fichier gestionServeur.c :</u>

- *int nbArchive() :* fonction qui retourne le nombre d'archive
- *void reorganiserListeArticles(int a) :* fonction qui est appelée lors de l'archivage ou de la suppression d'un article afin que la liste des articles soit composée uniquement de répertoires qui ont pour nom des numéros qui se suivent. (c'est-à-dire : si on a dans le répertoire « Articles » les répertoires suivant : « 0 1 2 3 4 5 » si on supprime (ou archive) l'article qui est dans le répertoire 3 on doit obtenir la liste suivante « 0 1 2 3 4 » et non « 0 1 2 4 5 » il faut toujours que les numéros se suivent). L'argument « a » étant l'indice de l'article supprimé
- *void reorganiserListeArchives(int a) :* Exactement comme la fonction précédente mais pour les archives.

- *void supprimerArchive(int a) :* supprime l'archive d'indice a
- *void supprimerArticle(int a) :* supprime l'article d'indice a
- *void archivage();* gère l'archivage des fichiers non modifiés depuis 7 jours. Cette fonction récupère tous les articles non modifiés depuis plus de 7 jours et les archive. Elle appelle donc la fonction reorganiserListeArticles après avoir archivé un article d'indice a et passe au suivant.
- *void RemoveUsersArticles() :* cette fonction supprime tous les fichiers contenant les titres des articles lus dans les répertoires des utilisateurs dépassant une durée de 9 jours sans modification. Cela permet de supprimer les articles qui ont été supprimés ou archivés et ainsi de libérer de la place.
- *void archiverOldArticle() :* fonction qui est appelée lorsque le nombre maximal d'articles est atteind afin d'archiver les articles les plus anciens même s'ils sont moins vieux que 7 jours.
- *void removeOldArchives(int nb) :* fonction qui supprime les « nb » plus anciennes archives (cela permet de toujours avoir 50 archives au maximum) elle est appelée après un archivage.

d. Dans le fichier serveurDialogue.c :

Toutes les fonction ci-dessous sont des fonctions qui permettent la gestion du serveur du t'chat :

- *int traiterCommandeDialogue(int socketService) :* Fonction qui traite les demandes des utilisateurs mais pour le serveur du T'CHAT. Les commandes possibles sont :
  - */c :* appelle la fonction qui gère la connexion d'un utilisateur.
  - */m* : appelle la fonction qui gère l'envoi d'un message sur le T'Chat.
  - */d :* appelle la fonction qui gère la déconnexion d'un utilisateur.
  - */q2009_admin_pass : appelle* la fonction qui déconnecte le serveur du t'chat. Pour assurer le bon fonctionnement du serveur il est préférable que cette dernière commande reste connue uniquement de l'ingénieur système.
- *void connectionDialogue(int socketService) :* Fonction qui connecte un utilisateur aux T'chat en récupérant son pseudo et son Ip en appelant la fonction connectUserDialogue après avoir créé la structure userDialogue nécessaire.
- *void connectUserDialogue(userDialogue *user) :* Fonction qui ajoute un utilisateur à la liste des utilisateurs connectés en y mettant son pseudo et son adresse IP afin de pouvoir se connecter au serveur de celui-ci pour pouvoir y envoyer les messages postés par les autres utilisateurs.
- *void supprimerUserConectDialogue(char *pseudo) :* Fonction qui supprime l'utilisateur Pseudo de la liste des utilisateurs connectés
- *void supprimeAllUsersConectDialogue()* : Fonction qui supprime tous les utilisateurs de la liste des utilisateurs connectés au T'chat. Cette fonction est appelée lors de la déconnexion du t'chat.
- *void gestionMessage(int socketService) :* Fonction qui récupère un message envoyé par un utilisateur et le renvoie à l'ensemble des utilisateurs connectés grâce à leur adresse IP sauvegardée lors de leur connexion.
- *int connectServUser(char *SERVNAME,int SERVPORT) :* Fonction qui permet de se connecter au serveur du programme client lors d'une session de t'chat.
- *void deconnectionDialogue(int socketService) :* Fonction qui gère la déconnexion d'un utilisateur au niveau du serveur du T'chat.

e. Dans le fichier interfaceServeur.c :

Les fonctions ci-dessous gèrent l'interface graphique du serveur et permettent de lier les boutons aux actions qu'ils doivent produire.

- *void pagePrinc() :* Cette fonction génère la page principale du serveur
- *void fenetreSupressionArticle(GtkWidget* pBouton, gpointer data) :* Cette fonction génère la page permettant de supprimer un article.
- *void fenetreSupressionArchive(GtkWidget* pBouton, gpointer data) :* Cette fonction génère la page permettant de supprimer une archive.
- *void fenetreSupressionUser(GtkWidget* pBouton, gpointer data) :* Cette fonction génère la page permettant de supprimer un utilisateur.
- *void deleteArticle(GtkWidget* pBouton, gpointer data) :* Cette fonction lie le bouton permettant de supprimer un article aux fonctions qui exécutent cette requête tout en récupérant le nom de l'article que l'administrateur veut supprimer.
- *void deleteArchive(GtkWidget* pBouton, gpointer data) :* Cette fonction lie le bouton permettant de supprimer une archive aux fonctions qui exécutent cette requête tout en récupérant le nom de l'archive que l'administrateur veut supprimer.
- *void deleteUser(GtkWidget* pBouton, gpointer data) :* Cette fonction lie le bouton permettant de supprimer un utilisateur aux fonctions qui exécutent cette requête tout en récupérant le nom de l'utilisateur que l'administrateur veut supprimer.
- *void consulterPlainte(GtkWidget* pBouton, gpointer Fen) :* Cette fonction génère la fenêtre dans laquelle s'affiche les plaintes des utilisateurs.
- *void deletePlaintes(GtkWidget* pBouton, gpointer data) :* Cette fonction fait en sorte que quand l'administrateur clique sur supprimer les plaintes alors la liste des plaintes est supprimée.
- *void consulterListeArchives(GtkWidget* pBouton, gpointer Fen) :* Cette fonction génère fenêtre dans laquelle s'affiche la liste des archives.
- *void consulterListeArticles(GtkWidget* pBouton, gpointer Fen) :* Cette fonction génère fenêtre dans laquelle s'affiche la liste des articles.
- *void consulterListeUsers(GtkWidget* pBouton, gpointer Fen) :* Cette fonction génère fenêtre dans laquelle s'affiche la liste des utilisateurs.
- *void finServeur(GtkWidget* pBouton, gpointer data) :* Cette fonction permet de quitter le serveur proprement. Elle appelle toutes les fonctions nécessaires pour cela.

IV/ Les tests :

Afin de s'assurer que la totalité de ces fonctions fonctionnent correctement nous avons préparé toute une série de tests à effectuer. Il faut tout d'abord remarquer que pour pratiquement la totalité des fonctions il faut en implémenter d'autres pour les tester. Voila pourquoi nous testerons les fonctions par petits groupes.

Pour cela nous allons en fait implémenter toutes les fonctions qui correspondent à une page de l'interface graphique du programme client et celles qui sont nécessaires pour que cette page fonctionne du coté du programme serveur.

Cependant il faut que le serveur tourne pour pouvoir tester le client donc nous commenceront par implémenter les fonctions qui permettent de lancer le serveur.

1. Lancement du serveur :

Pour pouvoir tester les fonctions du client il faut déjà que le serveur fonctionne correctement

- *int main(int argc, char *argv[])*
  - *void initSemaphore()*
  - *void *gestionArchivage()*
  - void* lancerServeurForum()
  - *void * lancerServeurDialogue(void * arg)*
  - *void pagePrinc()*

La fonction main du serveur lance trois threads qui vont exécuter les fonctions qui lancent le serveur du forum, le serveur du t'chat et la fonction de maintenance puis ensuite le processus père va lancer la page principale de l'interface du serveur.

Grâce à la console on peut voir si les serveurs sont bien lancés et pour la fonction de l'interface graphique voici le résultat :

Il faut donc implémenter la fonction *void* traiterCommande(void* socket)* pour lancer les fonctions du serveur en fonction des requêtes du client

2. La page de connexion :

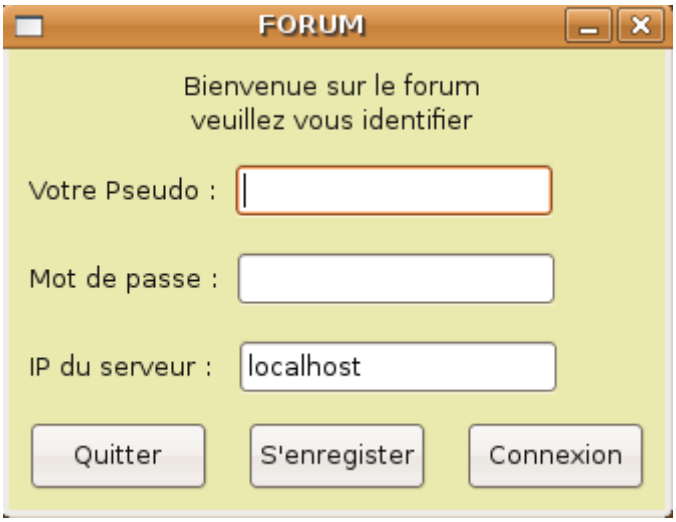
Le seul moyen d'arriver sur cette page est de lancer le programme client. On exécute les fonctions suivantes :

- int main (int argc, char *argv[])
  - *void pageConnexion(char *message)*

Ici : La fonction main appelle la fonction pageConnexion (nous garderons cette notation tout au long des tests)

L'argument message est en fait le message qui va être affiché par la fenêtre de la page de connexion.

Voici le résultat :



Pour tester le bouton quitter il suffit de cliquer dessus et cela doit fermer le programme.

3. La page d'enregistrement et fin connexion :

Comme la page de connexion, atteindre la page d'enregistrement ne nécessite aucune communication avec le serveur, il faut cliquer sur le bouton S'enregistrer

- *void enregistrement(GtkWidget* pBouton, gpointer data)*
  - *void fenetreEnregistrement(conect *arg)*

Si ces fonctions sont correctement codées voici le résultat :

Pour tester la fonction :

- *void aideEnregistrement(GtkWidget* pBouton, gpointer data)* :

Il suffit de cliquer sur le bouton Aide et une fenêtre d'aide doit s'ouvrir.

Pour tester la fonction :

- *void retourConnexion(GtkWidget* pBouton, gpointer data)*

Il suffit de cliquer sur le bouton retour et la page de connexion doit réapparaitre

Le bouton enregistrement permet de tester plusieurs fonctions y compris sur le serveur

Fonction testée sur le client :

- *void verifEnregistrement (GtkWidget* pBouton, gpointer data)*
  - *void traitementText(char *s)*
  - *int controleEnregistrement(const char *Pseudo,const char *Mdp)*
    - *int connection()*
    - *void deconnection(int desc)*

Fonctions testées sur le serveur :

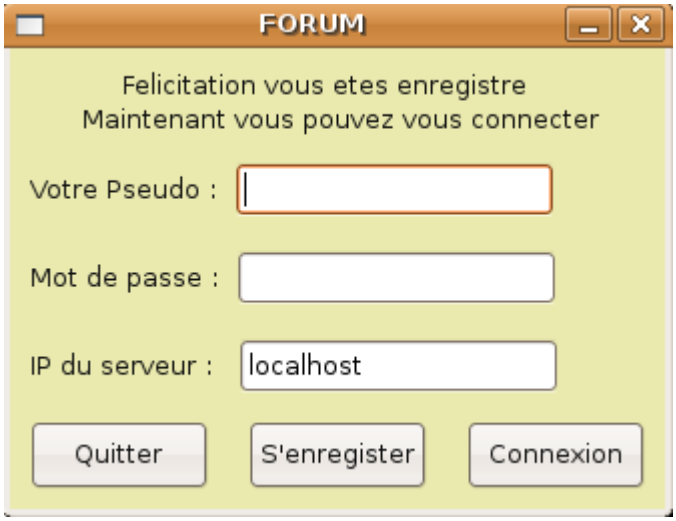
- *void* traiterCommande(void* socket)*
  - *void enregistrer(int socketService)*

Pour vérifier que ces fonctions fonctionnent correctement il faut se mettre dans les quatre situations possible :

a. les inscriptions sont impossibles car le serveur est plein
b. échec de la connexion
c. pseudo déjà utilisé
d. Succès de l'enregistrement

Pour la fonction *void traitementText(char *s)* il suffit de rentrer un pseudo avec des caractères spéciaux et de verifier qu'ils ont bien été remplacés par les « _ ».
Dans les cas a et b on aura une boite de dialogue GTK qui signalera le problème à l'utilisateur, dans le cas c on affichera que le pseudo est déjà utilisé dans le label où il a été écrit : « vous êtes sur le point de vous enregistrer » et dans le dernier cas on affiche la page de connexion avec un message de succès. Si toutes ces situations sont validées alors on considère que les fonctions sont correctes.

Supposons que nous somme dans le cas d'un succès, on se retrouve donc avec la fenêtre suivante :



Cette fenêtre est en fait la page de connexion avec un message d'accueil différent.
Nous allons donc maintenant essayer de nous connecter en cliquant sur le bouton connexion, et cela va tester les fonctions suivantes :

Pour le programme client :

- *void verifMdp(GtkWidget* pBouton, gpointer data)*
  - *void traitementText(char *s)*
  - *int controleIdentifiant(char *Pseudo, char *Mdp)*
    - *int connection()*
    - *void deconnection(int desc)*

Pour le programme serveur :

- *void* traiterCommande(void* socket)*
  - *void connection(int socketService)*
    - *int connectUser(char *pseudo)*

Ici on a quatre possibilités à tester :

a. Le serveur n'est pas connecté
b. Le pseudo ou le mot de passe est incorrect
c. Pseudo et mot de passe correct
d. Utilisateur déjà connecté

Dans les cas a, b et d une boite de dialogue GTK indiquant l'erreur doit s'ouvrir et dans le troisième cas on doit avoir la page principale qui s'ouvre.

4. La page principale :

La page principale est la page la plus importante du forum son fonctionnement permet donc de tester plusieurs fonctions du client et du serveur. Elle peut être appelée dans le programme client par les fonctions

- *void verifMdp(GtkWidget* pBouton, gpointer data)*
- *void returnPagePrinc(GtkWidget* pBouton, gpointer pwindow2)*

Fonction du client pour cette page :

- *void pagePrinc(gpointer pwindow2)*
  - *GtkWidget* listeArticle(char *Pseudo)*
    - *int connection()*
    - *void deconnection(int desc)*

Fonction du serveur :

- *void* traiterCommande(void* socket)*
  - *void listerArticles(int socketService)*

On a plusieurs situations :

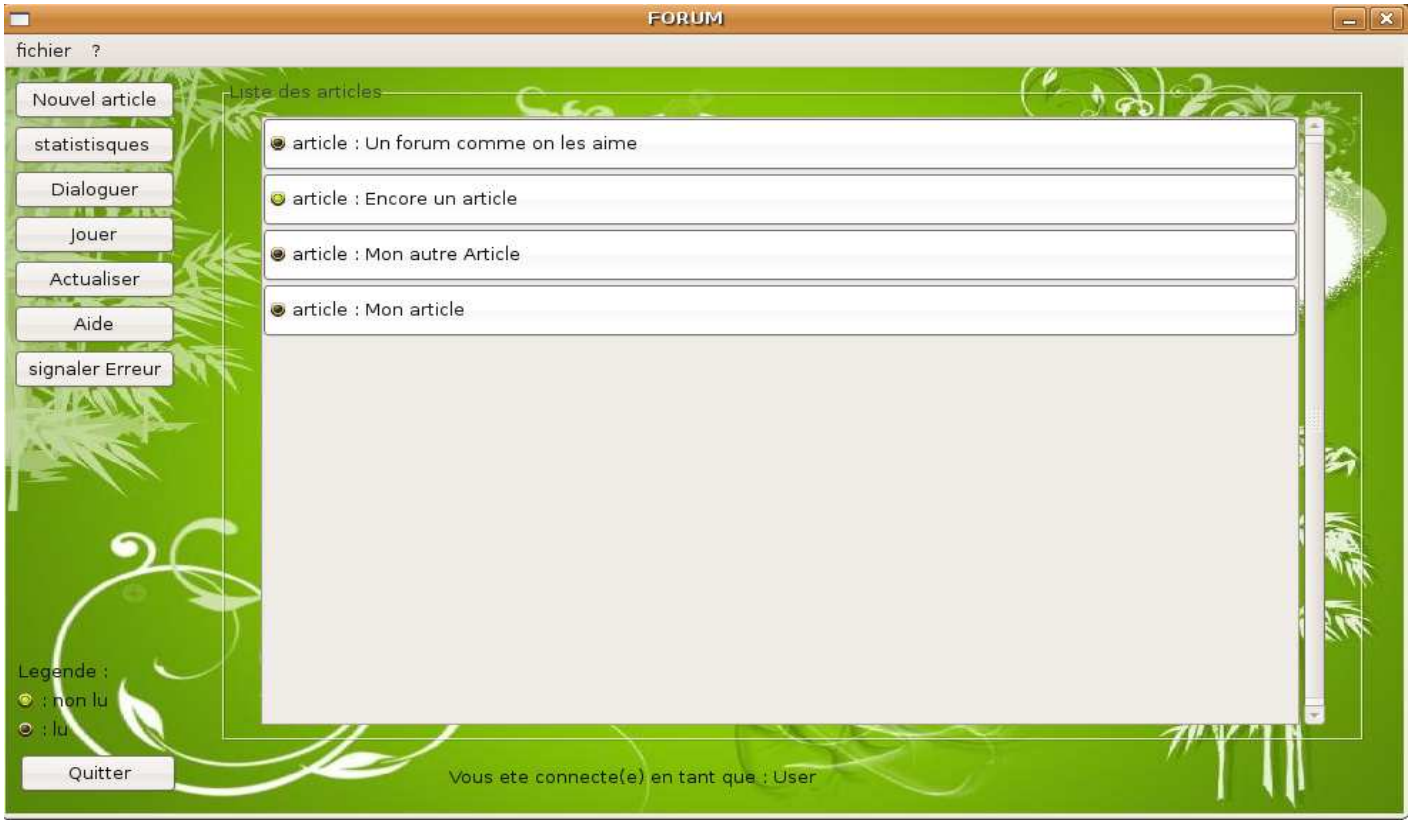
a. Il n'y a aucun article
b. Erreur de connexion
c. On a un ou plusieurs article(s) de disponible

Dans les deux premiers cas la fonction *GtkWidget* listeArticle(char *Pseudo)* remplace la liste des articles par un label où il est précisé soit que la liste est vide dans le cas a. soit qu'il y a eu une erreur de connexion si on est dans le cas b.

Dans le cas c. elle doit générer la liste des articles sous forme de boutons qui contiennent une petite lumière allumée si le dernier message de l'article n'a pas été lu par l'utilisateur, et une lumière éteinte dans le cas contraire pour informer l'utilisateur qu'il a déjà lu l'article. A côté de cette lumière doit être écrit le nom de l'article.

Pour lire un article l'utilisateur aura juste à cliquer sur le bouton qui correspond à cet article.

Voici un exemple d'une page principale qui fonctionne correctement :



Arrivé à cette page on peut tester le fonctionnement de nombreuses fonctions car les possibilités de déplacement dans le forum sont importantes.

Pour tester la fonction *void aideForum(GtkWidget* pBouton, gpointer data)* il suffit de cliquer sur le bouton aide ou dans le menu de faire « ? -> aide » et une fenêtre d'aide doit s'ouvrir qui décrit les possibilités qui s'offrent à l'utilisateur quand il se trouve sur la page principale.

Commençons par la page de lecture d'un article.

Pour tester la fonction *void actualiser(GtkWidget* pBouton, gpointer pwindow2)* il suffit de se connecter avec un autre client, de rajouter un message et de cliquer sur actualiser avec le premier client pour s'assurer que la liste des sujets a bien été actualisée.

5. Page de lecture d'un article :

On accède à ce type de page soit en cliquant sur un article dans la page principale soit en cliquant sur retour depuis une page de réponse à un article. Les fonctions appelées pour générer une telle page sont :

Pour le client :

- *void lectureArticle (GtkWidget* pBouton, gpointer data)*
  - *GtkWidget* listeMessage(char *titre, char *Pseudo)*
    - *int connection()*
    - *void traiterFinlabel(char *sTexte,int it,int b,int big,int s,int sub,int sup,int small,int tt,int u);*
    - *int traiterDebutlabel(char *sTexte,int it,int b,int big,int s,int sub,int sup,int small,int tt,int u);*
    - *void chargerImage(int sock,char *chemin)*
    - *GtkWidget* constructionImageDimentionne(char *chm,int width,int height)*
      - *void OnZoomIn(GtkWidget *pWidget, gpointer data)*
      - *void OnZoomOut(GtkWidget *pWidget, gpointer data)*
    - *void deconnection(int desc)*

Pour le serveur :

- *void* traiterCommande(void* socket)*
  - *void listerMessages(int socketService)*
    - *void envoieImg(int socketService, char *chm)*

Si toutes ces fonctions sont bien codées on devrait obtenir une page de lecture qui ressemble à celle-ci :

Si une image est jointe à un message alors elle doit apparaitre comme celle-ci :

Il ne faut pas oublier de vérifier si la mise en page du texte est correcte (gras, italique etc…).

S'il y a eu une erreur de connexion ou que la liste des messages est vite suite à une opération de maintenance alors un message s'affiche à la place des messages de l'article pour prévenir l'utilisateur.

Pour tester la fonction *void aideLecture(GtkWidget* pBouton, gpointer data)* il suffit de cliquer sur le bouton aide et une page d'aide doit s'afficher.

Pour tester la fonction *void returnPagePrinc(GtkWidget* pBouton, gpointer pwindow)* il suffit de cliquer sur le bouton retour et on doit se retrouver sur la page principale.

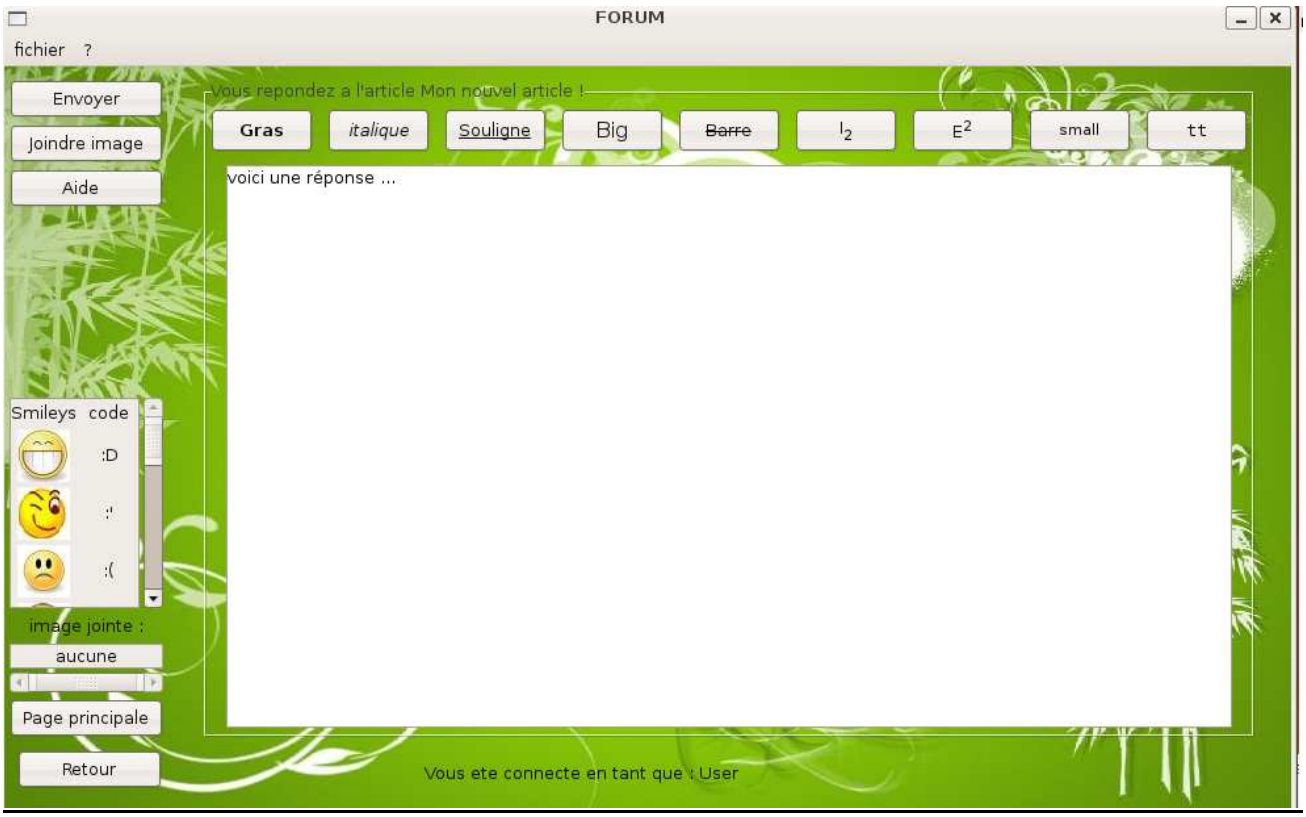
La suite logique après avoir lu un article étant de répondre à cet article nous allons voir les fonctions appelées pour générer la page de réponse à un article pour cela il suffit de cliquer sur répondre.

6. Page de réponse à un article :

Pour générer cette nouvelle page, le programme client n'a pas besoin de communiquer avec le serveur. Voici les fonctions appelée :

- *void repondre(GtkWidget* pBouton, gpointer nom)*
  - *GtkWidget* listeSmiley()*
  - *GtkWidget* boxMiseEnPage(GtkWidget *textView)*

Si ces fonctions sont correctes on obtient une page du même type que la page suivante :



Pour tester la fonction *void aideRepondre(GtkWidget* pBouton, gpointer data)* il suffit de cliquer sur le bouton aide qui doit générer une page d'aide.

Pour tester les fonctions :

- *void balisesGras(GtkWidget* pBouton, gpointer text)*
- *void balisesItalique(GtkWidget* pBouton, gpointer text)*
- *void balisesSouligne(GtkWidget* pBouton, gpointer text)*
- *void balisesBig(GtkWidget* pBouton, gpointer text)*
- *void balisesBarre(GtkWidget* pBouton, gpointer text)*
- *void balisesInd(GtkWidget* pBouton, gpointer text)*
- *void balisesExp(GtkWidget* pBouton, gpointer text)*
- *void balisesSmall(GtkWidget* pBouton, gpointer text)*

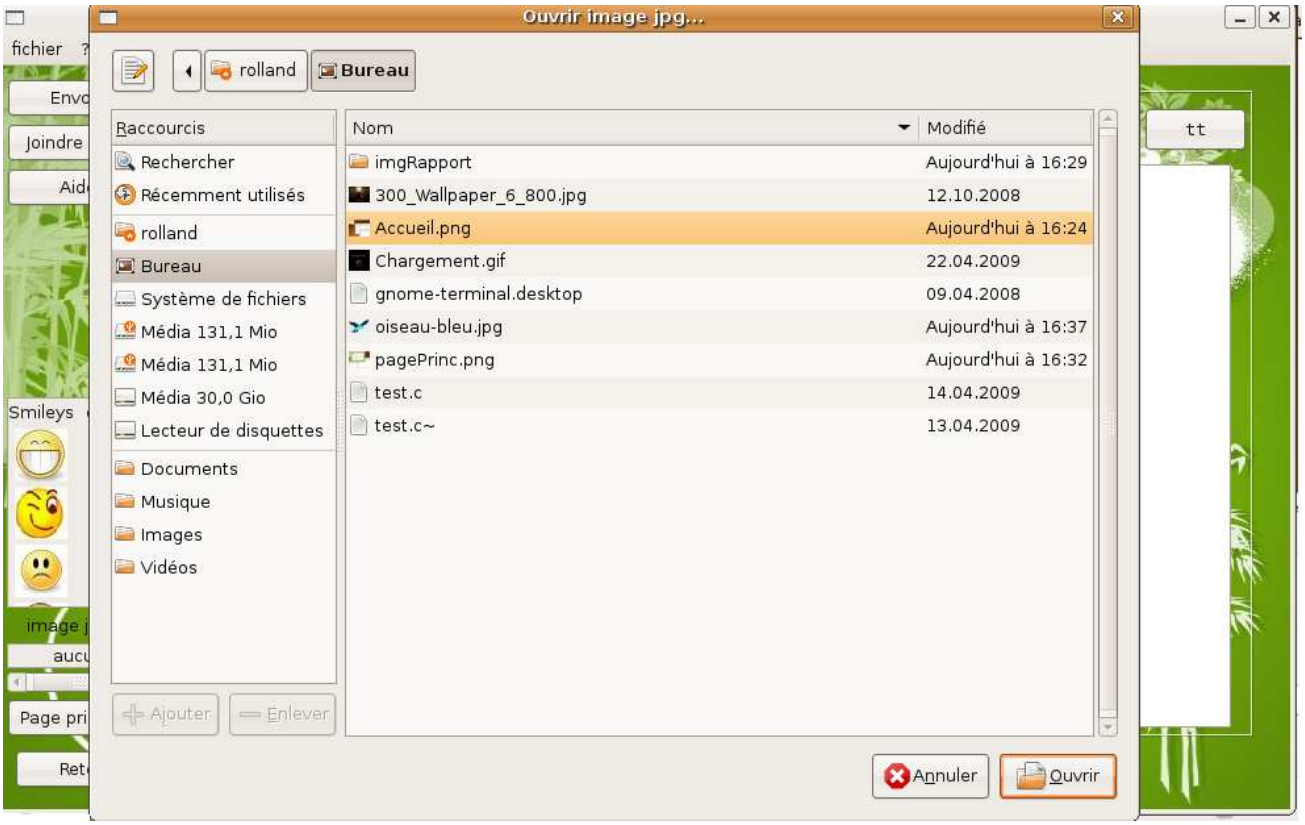
Il suffit de cliquer sur les boutons permettant de faire de la mise en page et de vérifier que les balises s'affichent bien dans la zone de texte.

Pour tester les fonctions :

- *void chercherImage(GtkWidget* pBouton, gpointer Fen)*
  - *int imageValide(char *chm)*

Il faut cliquer sur le bouton joindre image et choisir des types de fichier différents du type « .jpg » pour s'assurer que le forum va les refuser. Si c'est le cas alors il affiche une fenêtre d'erreur pour avertir le client qu'il n'a pas choisi un fichier au bon format. De plus il faut s'assurer que l'on ne peut pas joindre d'images qui ont une taille supérieure à 10Ko.

La fenêtre de recherche dune image est de la forme :



Le bouton « Retour » appelle la fonction *void lectureArticle (GtkWidget* pBouton, gpointer data)* déjà testée précédemment et permet à l'utilisateur de retourner lire l'article.

Le bouton « Page principale » permet de retourner à la page principale en appelant la fonction *void returnPagePrinc(GtkWidget* pBouton, gpointer pwindow2).*

Le bouton envoyé appelle les fonctions suivantes :

Pour le programme client :

- *void envoyerMessage(GtkWidget* pBouton, gpointer data)*
  - *int repondreSocket(char *art,char *mess,char *pseudo,char *chmImg)*
    - *int connection()*
    - *void envoieImage(int sock,char *chmImg)*
    - *void deconnection(int desc)*
  - *void returnPagePrinc(GtkWidget* pBouton, gpointer pwindow2)*

Pour le programme serveur :

- *void* traiterCommande(void* socket)*
  - *void repondre(int socketService)*
    - *void verificationImage(int socketService,char *chemin)*

Si les fonctions sont bien implémentées alors le message sera ajouté a la fin de l'article et le programme client affichera la page principale. Si le serveur n'a pas réussi à trouver l'article auquel veut répondre l'utilisateur, où que l'article est considéré comme plein alors le serveur informe de programme client qui informe l'utilisateur en ouvrant une page d'erreur qui explique le problème rencontré et la page principale n'est pas chargée, le client reste sur la page de réponse afin de ne pas faire perdre tout le texte tapé par le client.

7. Page des jeux :

Nous voila revenu à la page principale, profitons en pour vérifier que la page des jeux fonctionne correctement. Pour cela il suffit de cliquer sur le bouton « Jouer » ce qui va appeler la fonction *void pageJeu(GtkWidget* pBouton, gpointer FenPrecedente)* qui va nous générer la page suivante :

Pour tester les fonctions *void sudoku(GtkWidget* pBouton, gpointer FenPrecedente)* et *void zork(GtkWidget* pBouton, gpointer FenPrecedente)* il faut juste lancer le Sudoku et Zork et vérifier que ces jeux fonctionnent correctement. Après on retourne sur la page principale grâce au bouton retour qui appel la fonction *void returnPagePrinc(GtkWidget* pBouton, gpointer pwindow2).*

8. La fenêtre des statistiques :

On peut lancer cette page à partir de la page principale en cliquant sur « Statistique » ce qui va appeler les fonctions suivantes.

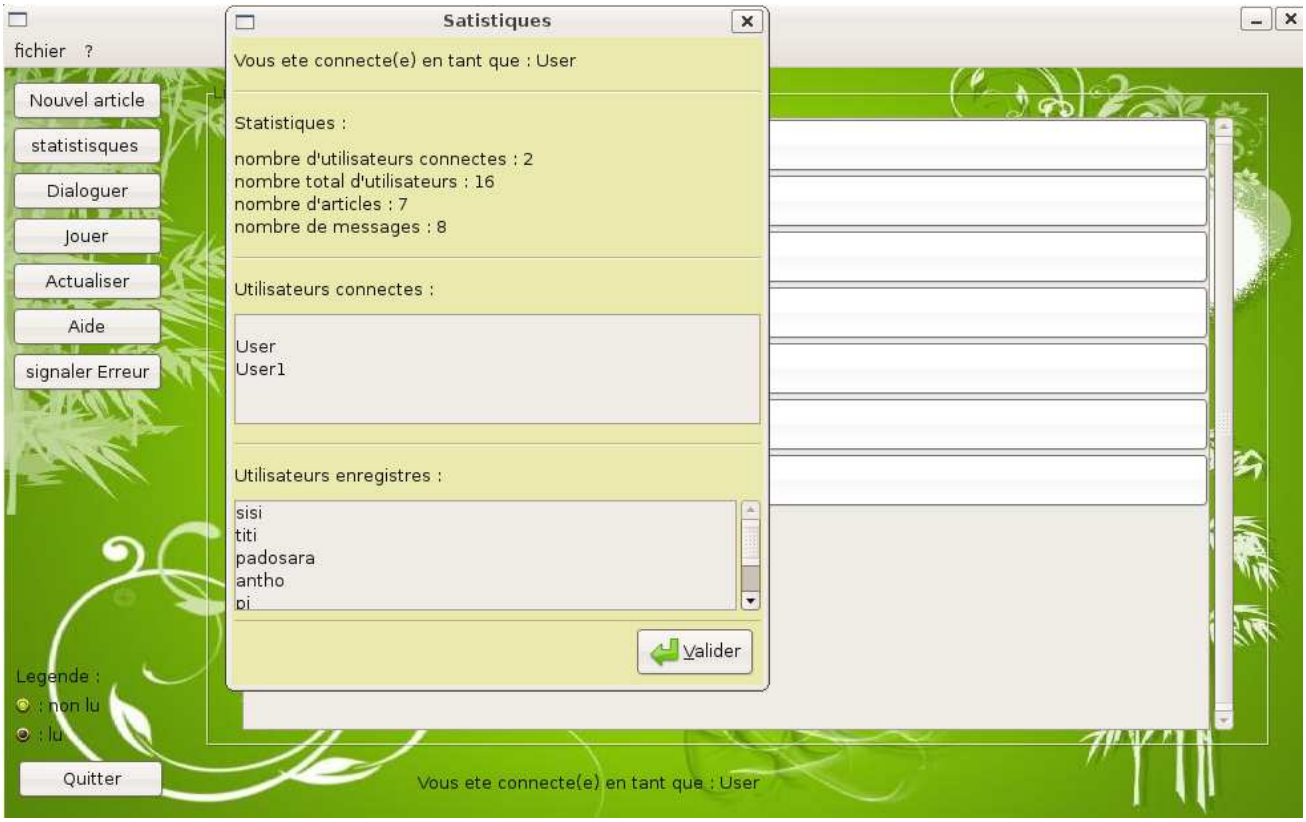
Pour le client :

- *void statistisques(GtkWidget* pBouton, gpointer Fen)*
  - *GtkWidget* statistiqueSocket(char *profil)*
    - *int connection()*
    - *void deconnection(int desc)*

Pour le serveur :

- *void* traiterCommande(void* socket)*
  - *void information(int socketService)*
    - *int nombreDeMessages()*
    - *int nombreDeArticles()*
    - *int nombreUtilisateurs()*
    - *int nombreUtilisateursConnect()*

Si ces fonctions fonctionnent correctement on obtient alors une fenêtre dans le programme serveur qui doit ressembler a cela :



Si une erreur de connexion a lieu alors le client en est informé car un message d'erreur s'affiche a la place des informations qu'il souhaite obtenir.

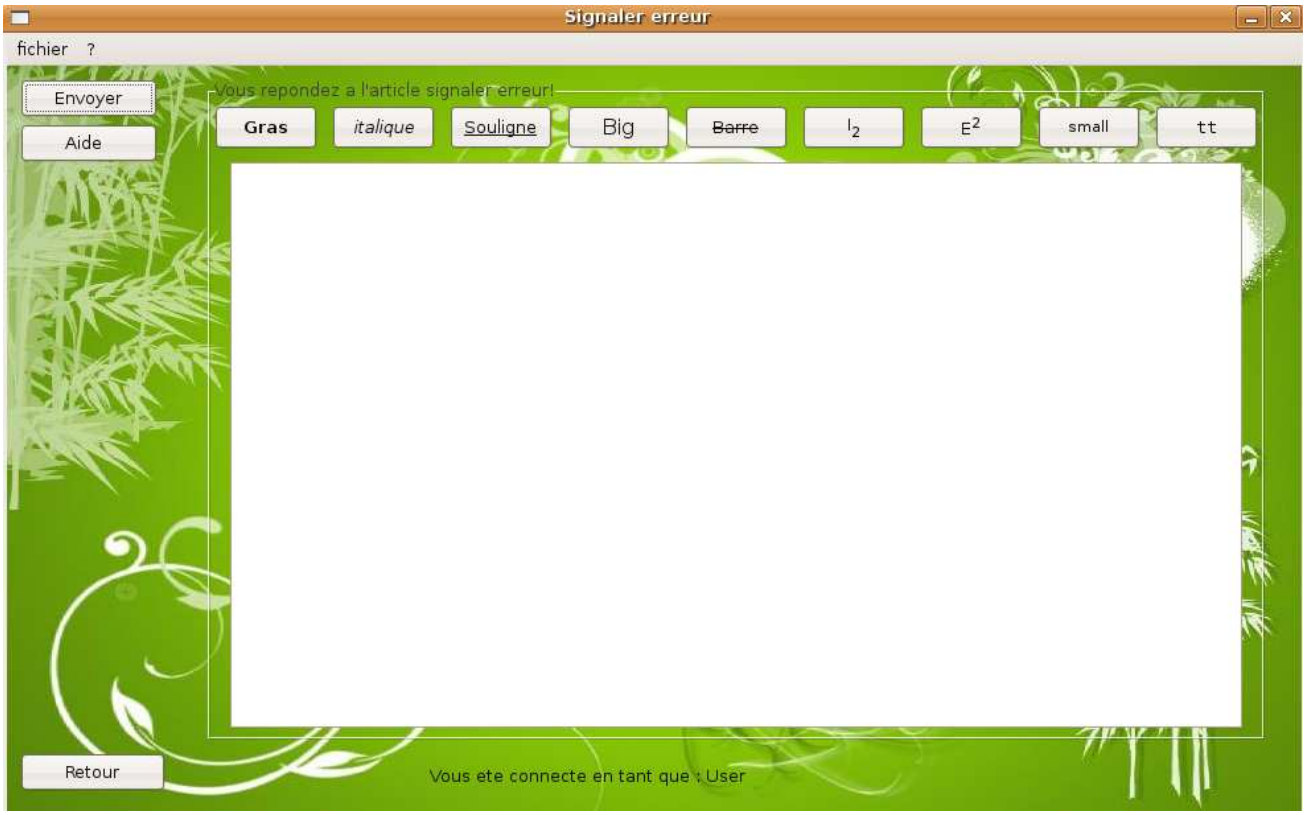
9. La page Signaler erreur :

Cette page fonctionne exactement comme une page de réponse à un article. Voilà pourquoi on va s'attarder uniquement sur la fonction qui permet de la générer et non sur les fonctions appelées lorsque l'on clique sur le bouton envoyer.

La fonction qui génère cette page est donc la fonction :

- *void signalerErreur(GtkWidget* pBouton, gpointer data)*

Si elle est bien codée on obtient une fenêtre comme la fenêtre suivante :



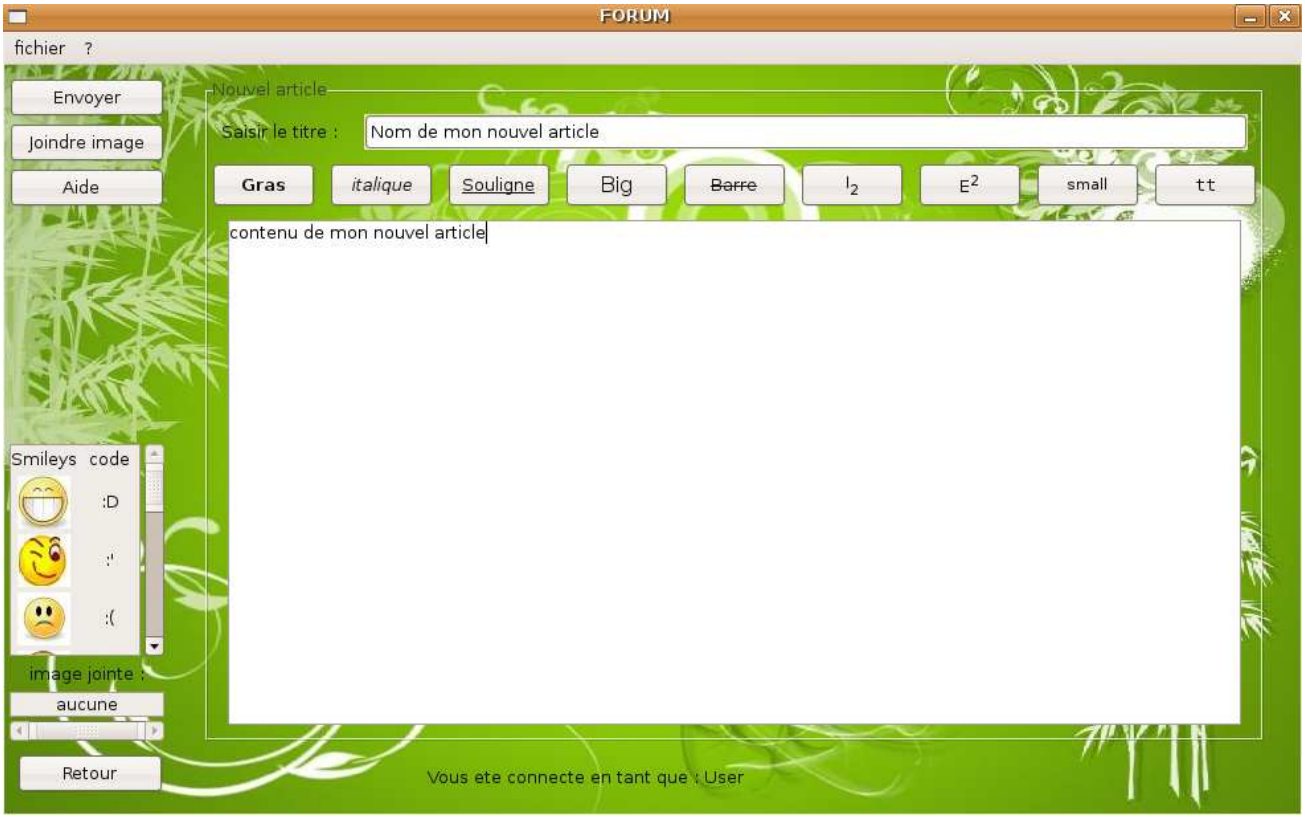
Les seules différences avec une page de réponse sont que l'on ne peut pas envoyer d'image et que l'on n'affiche pas les smileys car de ces options ne sont pas traitées pour l'administrateur qui lira les plaintes.

10. La page nouvel article :

La façon de générer cette page et de traiter le nouvel article diverge légèrement de celle avec laquelle on répond à un article. En effet la seul différence au niveau du client est qu'on envoie le titre d'un nouvel article et au niveau du serveur qu'on crée un nouveau répertoire pour cet article. Cependant le serveur doit aussi gérer le nombre maximal et le nombre d'archive maximale.
Fonction appelée par le client pour générer cette fenêtre :

- *void nouvelArticle(GtkWidget* pBouton, gpointer FenPrecedente)*
  - *GtkWidget* listeSmiley()*
  - *GtkWidget* boxMiseEnPage(GtkWidget *textView)*

Si la fonction fonctionne on doit obtenir la fenêtre suivante :



Les boutons de mise en page ayant déjà été essayés lors du teste des fenêtres réponse, il n'est pas nécessaire de s'attarder sur les fonctions qui génèrent les balises. Il faut juste vérifier que cela fonctionne sur notre nouvelle fenêtre. Il en est de même pour le bouton « joindre image ».

Le bouton aide ouvre la même page d'aide que les pages de réponse donc la fonction n'est plus à tester.

Malgré les faibles différences entre cette page et les pages de réponse les fonctions appelées lorsque l'utilisateur clique sur « Envoyer » divergent.

Liste des fonctions appelées :

Pour le client :

- *void envoyerArticle(GtkWidget* pBouton, gpointer data))*
  - *int nouvelArticleSocket(char *art,char *mess,char *pseudo, char* chmImg)*
    - *int connection()*
    - *void envoieImage(int sock,char *chmImg)*
    - *void deconnection(int desc)*
  - *void returnPagePrinc(GtkWidget* pBouton, gpointer pwindow2)*

Pour le serveur :

- *void* traiterCommande(void* socket)*
  - *void newArticle(int socketService)*
    - *void verificationImage(int socketService,char *chemin)*
    - *void archiverOldArticle()*
      - *void reorganiserListeArticles(int a)*
    - *void removeOldArchives(int nb)*
      - *void supprimerArchive(int a)*
        - *void reorganiserListeArchives(int a)*

Les dernières fonctions qui concernent l'archivage sont exécutées seulement si c'est nécessaire.

Si tout s'est bien passé le programme client recharge la page principal et l'article créé doit apparaitre. Sinon s'il y a eu un problème de connexion, que le titre n'a pas été saisi ou que l'article existe déjà, une fenêtre d'erreur s'ouvre pour avertir l'utilisateur.

11. Les pages d'éditions :

Les pages d'édition fonctionnent exactement comme les pages de réponse sauf qu'à l'ouverture, la zone de texte contient déjà du texte qui est le texte à éditer.

Les fonctions appelées pour ouvrir une telle fenêtre sont :

Pour le client :

- *void editer(GtkWidget* pBouton, gpointer data)*
  - *char* get_text_edit(char* article,int numMess,char* buffer)*
    - *int connection()*
    - *void deconnection(int desc)*

Pour le serveur :

- *void* traiterCommande(void* socket)*
  - *void recupererMessage(int socketService)*

Les fonctions appelées lorsque l'utilisateur clique sur envoyés sont :

Pour le client :

- *void editerMessage(GtkWidget* pBouton, gpointer data)*
  - *int editerSocket(char *art,char *mess,char *pseudo,char *chmImg,int numMess)*
    - *int connection()*
    - *void envoieImage(int sock,char *chmImg)*
    - *void deconnection(int desc)*
  - *void returnPagePrinc(GtkWidget* pBouton, gpointer pwindow2)*

Pour le serveur :

- *void* traiterCommande(void* socket)*
  - *void editer(int socketService)*
    - *void verificationImage(int socketService,char *chemin)*

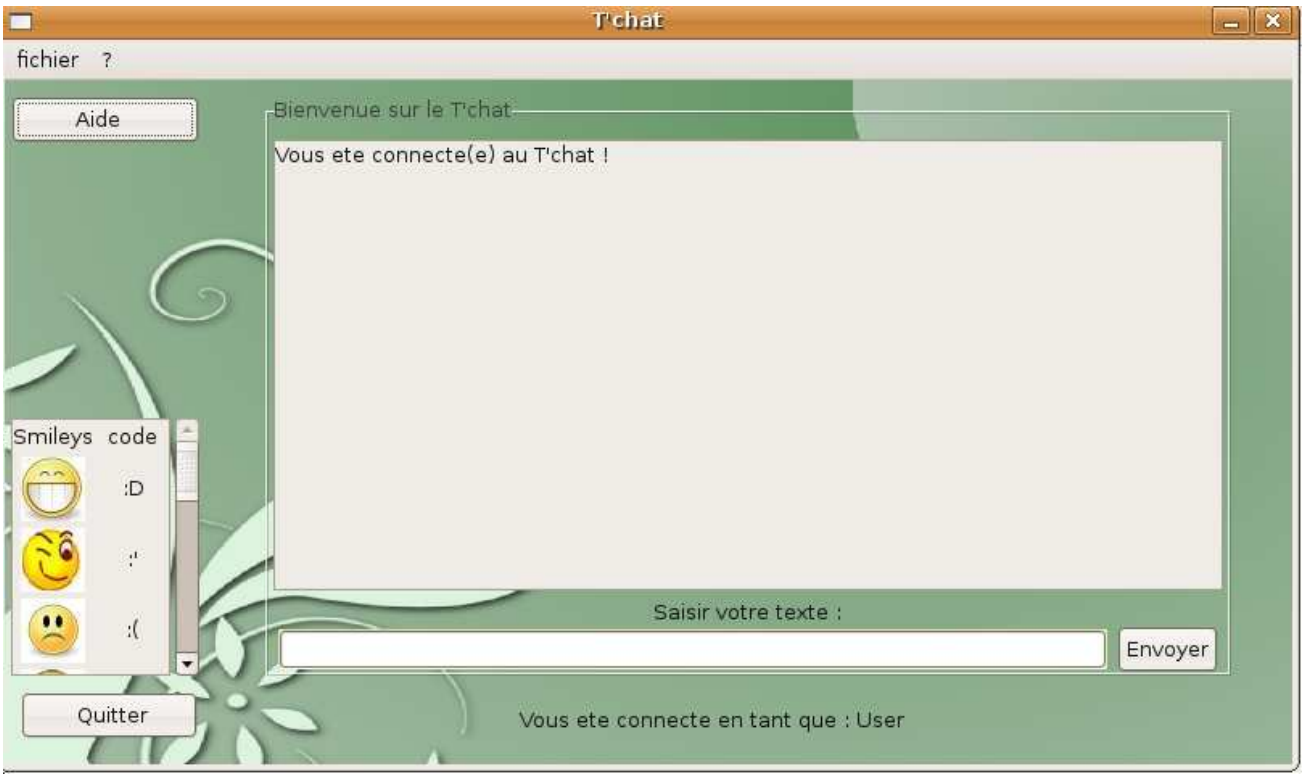
12. Le T'Chat :

Le T'Chat fonctionne sur un serveur à part et est très simple à mettre en place. Voici le principe :

Lorsqu'un utilisateur se connecte au T'chat il est rajouté à la liste des connectés et reçoit tous les messages qui sont envoyés sur le t'chat y compris les siens.

De plus un serveur est lancé sur la machine du client pour qu'il puisse recevoir tous les messages instantanément.

Voici à quoi ressemble la fenêtre du T'chat :



On s'aperçoit vite que les smileys sont gérés sur le T'chat.

Si une erreur se produit lors de l'ouverture du t'chat le client sera averti par le contenu de la fenêtre du t'chat. Si cette erreur est due au fait qu'un T'chat est déjà ouvert par un autre utilisateur sur la même machine alors la fenêtre de t'chat sera fermée par la fonction *void finDialogueErr(GtkWidget* pBouton,gpointer data)* afin de ne pas déconnecter l'autre utilisateur.

Voici les fonctions qui doivent fonctionner sur le client pour que le T'chat fonctionne :

void dialoguer(GtkWidget pBouton, gpointer nom)*
*void dialoguerThread(void *nom)*
*void connectDialogue(gpointer *Fen) ;*
void finDialogue(GtkWidget pBouton,gpointer Fen)*
void envoyerMessageDialogue(GtkWidget pBouton,gpointer *Fen)*
*void *lancerServeurDialogue(gpointer fen)*

Pour verifier le fonctionnement de la fonction *void aideDialogue(GtkWidget* pBouton, gpointer data)* il suffit de cliquer sur Aide et une fenêtre d'aide doit s'ouvrir.

Voici la liste des fonction qui doivent fonctionner sur le serveur pour que le T'chat fonctionne :

int traiterCommandeDialogue(int socketService)
void connectionDialogue(int socketService
*void connectUserDialogue(userDialogue *user) :*
*void supprimerUserConectDialogue(char *pseudo)*
void supprimeAllUsersConectDialogue()
void gestionMessage(int socketService)
*int connectServUser(char *SERVNAME,int SERVPORT)*
void deconnectionDialogue(int socketService) :

Chaque fonction ayant un rôle bien précis, comme vu précédemment dans les partie II et III du rapport, pour tester le t'chat il suffit de le faire fonctionner et de tester les point ci-dessous :

- ✓ Lancer un t'chat
- ✓ Envoyer un message
- ✓ Fermer le T'chat et relancer (si la déconnection n'a pas fonctionné on recevra deux fois les messages)
- ✓ Essayer d'ouvrir un T'chat avec le même utilisateur (doit ouvrir une fenêtre d'erreur)
- ✓ Essayer d'ouvrir Un T'achat avec un autre utilisateur mais sur une même machine (doit afficher un message d'erreur)
- ✓ Lancer un autre (voir plusieurs) utilisateur(s) sur une (des) machine(s) différente(s) et communiquer
- ✓ Quitter le T'chat

Si tous ces points fonctionnent alors on pourra affirmer que toutes les fonctions sont correctes.

On est obligé de tester toutes ces fonctions en même temps car contrairement au forum rien ne nous permet de voir si on est correctement connecté sans envoyer des messages, or les message sont l'intérêt même d'une messagerie instantanée. De plus il serait illogique de tester un serveur sans même avoir prévu de quoi se déconnecter. Les fonctions du T'chat doivent donc être vérifiées ensemble.

13. <u>Les dernières fonctions non testées du programme client :</u>

a. *void fenetreErreur(char* msg,fenetrePrincipale *arg)*

Si les messages d'erreur des étapes précédentes se sont affichés alors cette fonction est vérifiée.

b. *void quitterForum()* :

*void deconnectionUtilisateur(char *pseudo)*

Si l'utilisateur est bien déconnecté du serveur et que le programme client est correctement fermé alors ces fonctions sont vérifiées. Ces fonctions se valident avec les fonctions.

void deconnecterUser(int socketService)

*void supprimerUserConect(char *pseudo)*

du serveur.

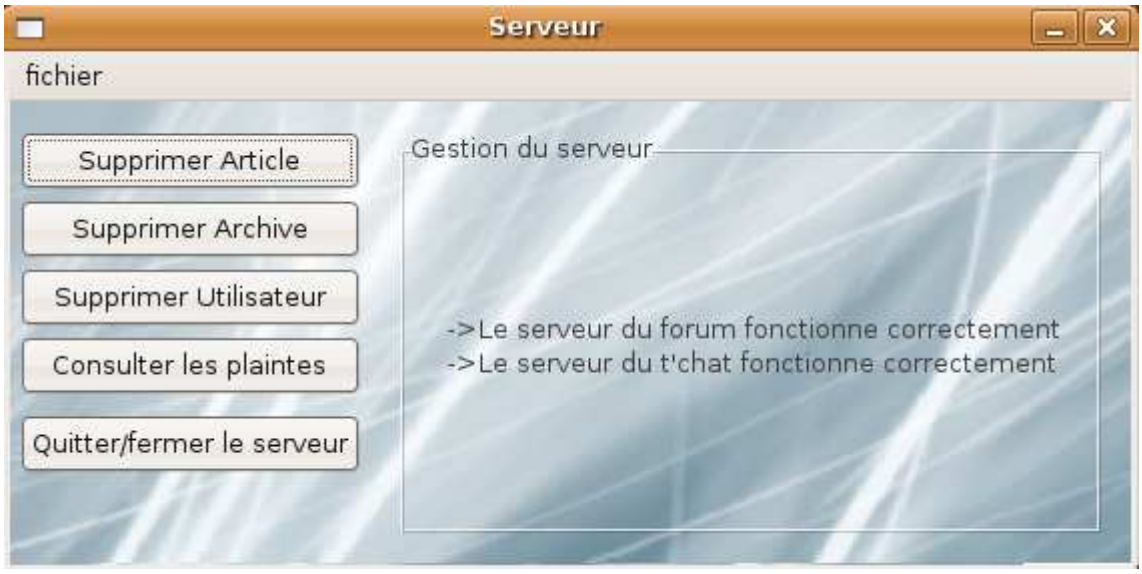
c. *fenetrePrincipale * loadWindow(fenetrePrincipale * Fenprec)*

Si une fenêtre de chargement apparait à chaque fois que le chargement d'une fenêtre est long alors cette fonction est correcte.

14. <u>Les fonctions du serveur qui ne sont pas nécessaires pour le programme client :</u>

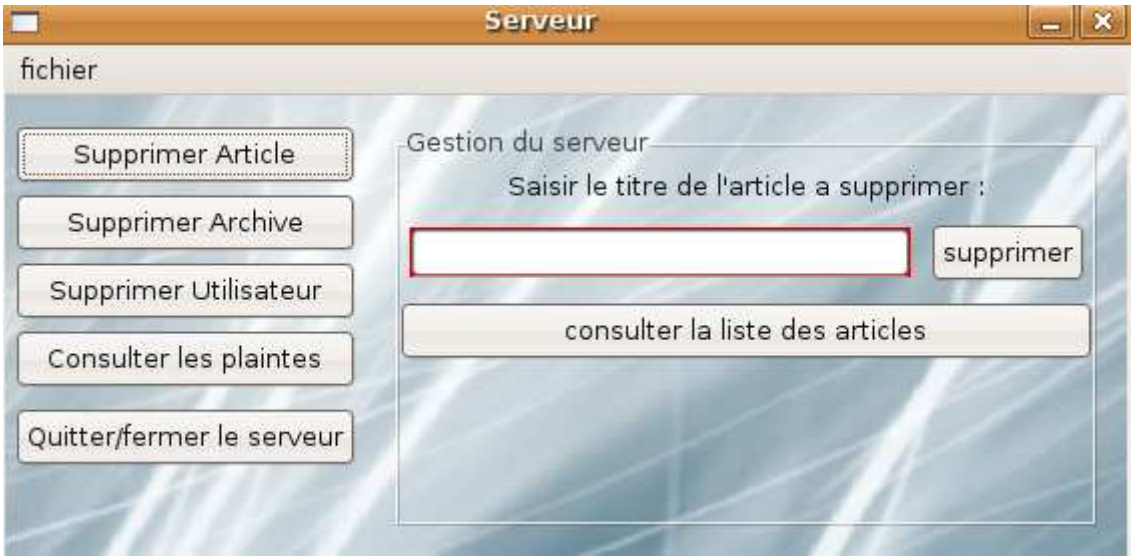
Il reste maintenant à tester les fonctions qui sont liées à l'interface graphique du serveur

<u>Rappel de la page d'accueil du serveur lancée avec la fonction *void pagePrinc()* dans le main du serveur :</u>



A. Fonctions liées à la fenêtre de suppression des articles :

Voici la fenêtre du serveur que l'on doit obtenir pour supprimer un article si la fonction *void fenetreSupressionArticle(GtkWidget* pBouton, gpointer data)* de l'interface serveur est bien implémentée :



Pour tester les fonctions :

- *void deleteArticle(GtkWidget* pBouton, gpointer data)*
  - *int removeArticle(char *article)*
    - *void supprimerArticle(int a)*
      - *void reorganiserListeArticles(int a)*

Il faut cliquer sur le bouton supprimer après avoir saisi le nom de l'article et vérifier que le tout fonctionne correctement. C'est à dire :

a. Si on supprime un Article qui est dans la liste des articles alors un message qui confirme la suppression doit apparaitre et l'Article ne doit plus faire parti de la liste des articles.
b. Si on supprime un Article qui n'existe pas alors un message d'erreur doit s'afficher et la liste des articles ne doit pas être modifiée.

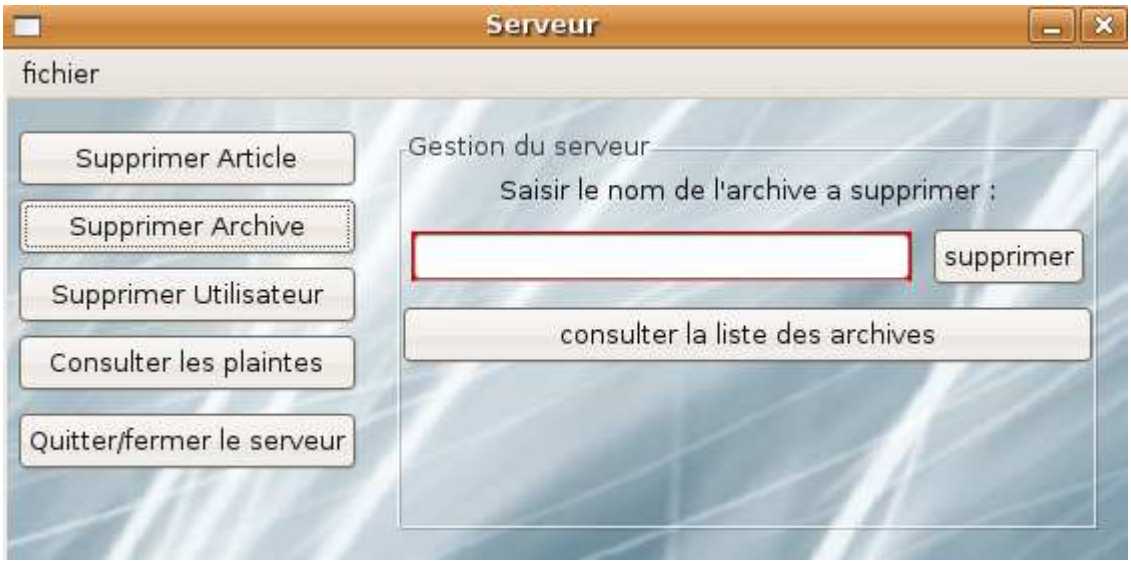
Pour tester les fonctions :

- *void consulterListeArticles(GtkWidget* pBouton, gpointer Fen)*
  - *GtkWidget* listeArticles()*

Il suffit de cliquer sur le bouton « consulter la liste des articles » et la liste des articles doit s'afficher dans une boite de dialogue GTK

B. Fonctions liées à la fenêtre de suppression des archives :

Voici la fenêtre du serveur que l'on doit obtenir pour supprimer une archive si la fonction *void fenetreSupressionArchive(GtkWidget* pBouton, gpointer data)* de l'interface serveur est bien implémentée :



Pour tester les fonctions :

- *void deleteArchive(GtkWidget* pBouton, gpointer data)*
  - *int removeArchive(char *article)*
    - *void supprimerArchive(int a)*
      - *void reorganiserListeArchives(int a)*

Il faut cliquer sur le bouton supprimer après avoir saisi le nom de l'archive et vérifier que le tout fonctionne correctement. C'est à dire :

a. Si on supprime une Archive qui est dans la liste des archives alors un message qui confirme la suppression doit apparaitre et l'archive ne doit plus faire parti de la liste des archives.
b. Si on supprime une archive qui n'existe pas alors un message d'erreur doit s'afficher et la liste des archives ne doit pas être modifiée.

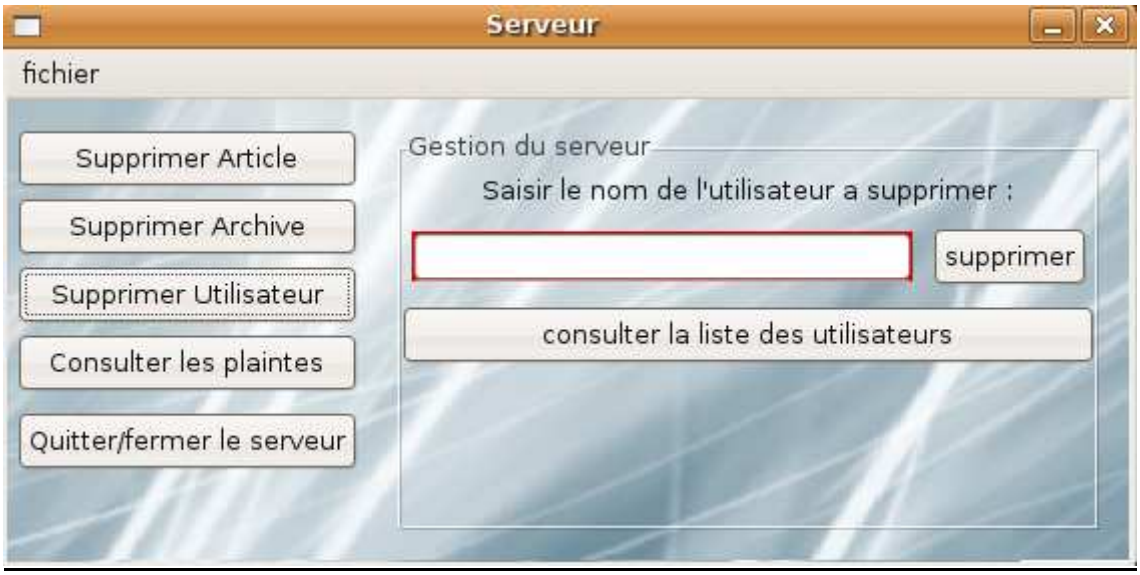
Pour tester les fonctions :

- *void consulterListeArchives(GtkWidget* pBouton, gpointer Fen)*
  - *GtkWidget* listeArchives()*

Il suffit de cliquer sur le bouton « consulter la liste des archives » et la liste des archives doit s'afficher dans une boite de dialogue GTK

C. Fonctions liées à la fenêtre de suppression des Utilisateur :

Voici la fenêtre du serveur que l'on doit obtenir pour supprimer un Utilisateur si la fonction *void fenetreSupressionUser(GtkWidget* pBouton, gpointer data)* de l'interface serveur est bien implémentée :



Pour tester les fonctions :

- *void deleteAUser (GtkWidget* pBouton, gpointer data)*
  - *int removeUser(char *article)*

Il faut cliquer sur le bouton supprimer après avoir saisi le nom de l'utilisateur et vérifier que le tout fonctionne correctement. C'est à dire :

a. Si on supprime un utilisateur qui est dans la liste des utilisateurs alors un message qui confirme la suppression doit apparaitre et l'utilisateur ne doit plus faire parti du répertoire « Users ».
b. Si on supprime un utilisateur qui n'existe pas alors un message d'erreur doit s'afficher et le répertoire « Users » ne doit pas être modifié.
c. Si on supprime un utilisateur qui est connecté alors un message d'erreur s'affiche. En effet on a fait le choix de ne pas pouvoir supprimer un utilisateur connecté car l'inscription est très simple et cela n'aurait donc aucun intérêt. Le but de la suppression d'un utilisateur est donc de supprimer ceux qui n'utilisent plus le forum afin de libérer les Pseudo qu'ils utilisent.

ATTENTION : Supprimer un utilisateur ne supprime pas les messages qu'il a postés.

Pour tester les fonctions :

- *void consulterListeUsers(GtkWidget* pBouton, gpointer Fen)*
  - *GtkWidget* scrollListeUsers()*

Il suffit de cliquer sur le bouton « consulter la liste des utilisateurs » et la liste des utilisateurs doit s'afficher dans une boite de dialogue GTK.

D. Fonctions liées à la gestion des plaintes :

Voici la ce à quoi doit ressembler la fenêtre de lecture des plaintes quand la fonction *void consulterPlainte(GtkWidget* pBouton, gpointer Fen)* est bien codée :

On a en fait deux fenêtres qui sont :

->La fenêtre où apparaissent les plaintes :

La liste des plaintes étant générée par la fonction *GtkWidget* lirePlaintes()*

->La fenêtre du serveur qui permet de supprimer l'ensemble des plaintes :

Pour tester la fonction *void deletePlaintes(GtkWidget* pBouton, gpointer data)* Il suffit de cliquer sur le bouton « Supprimer les plaintes » et après cette opération la liste des plaintes doit donc être vide. C'est-à-dire : après l'utilisation de cette fonction le répertoire « 0 » du répertoire « Articles » doit contenir uniquement le fichier « 0 ».

E. La fonction de déconnexion du serveur :

La fonction *void finServeur(GtkWidget* pBouton, gpointer data)* est appelée lorsque l'on quitte le serveur. Pour déconnecter proprement le serveur elle appelle la fonction *void lockEndSemaphore()* permettant de bloquer tous les sémaphores afin que lorsque le serveur se coupe aucun programme client ne soit en train d'écrire sur le serveur. Cela permet d'éviter les erreurs lors du lancement de la prochaine session du serveur.

Elle appelle aussi la fonction *void supprimeAllUsersConect()* afin de vider correctement la liste des utilisateurs connectés.

F. Les autres fonctions :

a. Gestion des sémaphores :

Les fonctions suivantes permettent de gérer les sémaphores afin d'éviter que les programmes serveur et client plantent durant les opérations de maintenance.

void lockAllSemaphore()
void unlockAllSemaphore()

b. Maintenance du serveur :

La fonction *void archivage()* est appelée avec la fonction *void RemoveUsersArticles()* par la fonction *void *gestionArchivage()* elle-même lancée par la fonction main. Ces fonctions permettent d'archiver les articles trop vieux et de supprimer les articles qui ne sont plus utilisé dans la liste des articles lus par les utilisateurs.

G. Les sémaphores

Pour tester le bon fonctionnement des sémaphores il suffit de faire plusieurs opérations (écriture d'un message, lecture d'un message, suppression d'un Article, archivage) et de s'assurer que le programme ne plante pas. Les sémaphores sont utilisés dans la totalité des fonctions qui font une action sur un ou plusieurs répertoire(s)/fichier(s) du serveur.

V/ Les perspectives de mises à jours pour ce forum :

Dans un futur proche on pourrait envisager :

- Permettre à l'utilisateur de choisir parmi une liste d'images celles qu'il veut en fond d'écran.
- Permettre à l'utilisateur de prendre un avatar
- De donner la possibilité à un administrateur de supprimer un message précis
- De permettre à l'administrateur de bloquer un compte utilisateur et de bloquer son adresse IP pour qu'il ne puisse plus se connecter au serveur
- De donner la possibilité de joindre davantage de types de fichiers (du texte, des vidéos, etc…)