PROGRAMMATION SYSTEME

PROJET : IMPLÉMENTATION DE CANAUX DE DISCUSSION ENTRE CLIENTS CHOISIS ALEATOIREMENT

DEVAUX Manon - DRIANT Arthur - DURTESTE Gabin - GERARD Hugues

SOMMAIRE

I- Introduction

A l'occasion de notre projet PSE dont le but est d'implémenter les différentes connexions entre un serveur et ses clients, nous avons décidé de coder des canaux de discussion entre deux clients choisis aléatoirement parmi les différentes personnes connectées. Nous nous sommes inspirés du site Omegle qui fonctionne sur le même principe.

Omegle est une plateforme sur internet permettant à chacun de discuter avec quelqu'un qui parle la même langue. Un bouton est disponible pour couper la discussion et rechercher un nouveau partenaire. D'autres fonctionnalités sont également implémentées :

- Discussion avec des personnes qui ont le même centre d'intérêt
- Discussion avec la webcam de l'ordinateur allumé afin d'avoir un échange vidéo
- Une section pour les adultes

L'exemple d'Omegle nous paraissait d'être un bon point de départ pour mettre en pratique nos connaissances acquises en PSE ce semestre. Nous avons donc décidé d'implémenter ce système en gardant les fonctions de base : des discussions entre deux personnes choisies aléatoirement parmi les personnes connectées et un moyen de changer de partenaire de discussion.

Cette fonctionnalité a été implémentée de la manière suivante : si l'utilisateur écrit « NEXT! » dans sa console, lui et son interlocuteur sont automatiquement renvoyés à l'accueil et recherchent alors chacun un nouveau partenaire.

Nous avons aussi voulu rajouter la possibilité de choisir son destinataire en recherchant son pseudo.

II – Fonctionnalités implémentées et choix techniques

A) Mettre en place la communication client/client

Tout d'abord, la création d'une discussion entre deux clients nécessite que les deux puissent communiquer entre eux par l'intermédiaire du serveur. On prend comme point de départ les programmes serveur.c et client.c issus de l'avant dernier TP. Dans l'état actuel des choses le client peut envoyer des données au serveur mais le serveur n'est pas en mesure de le faire.

```
5
6   void *lecture(void *arg);
7
8   int main(int argc, char *argv[]) {
9       int sock, ret;
10      struct sockaddr_in *adrServ;
11      int fin = FAUX;
12      char ligne_ecr[LIGNE_MAX];
13
14      int lgEcr;
15
16      if (argc != 3)
17          erreur("usage: %s machine port\n", argv[0]);
18
19      printf("%s: creating a socket\n", CMD);
20      sock = socket (AF_INET, SOCK_STREAM, 0);
21      if (sock < 0)
22          erreur_IO("socket");
23
24      printf("%s: DNS resolving for %s, port %s\n", CMD, argv[1], argv[2]);
25      adrServ = resolv(argv[1], argv[2]);
26      if (adrServ == NULL)
27          erreur("adresse %s port %s inconnus\n", argv[1], argv[2]);
28
29      printf("%s: adr %s, port %hu\n", CMD,
30                  stringIP(ntohl(adrServ->sin_addr.s_addr)),
31                  ntohs(adrServ->sin_port));
32
33      printf("%s: connecting the socket\n", CMD);
34      ret = connect(sock, (struct sockaddr *)adrServ, sizeof(struct sockaddr_in));
35      if (ret < 0)
36          erreur_IO("connect");
37
38      DataSpec lecteur;
39      ret = pthread_create(&lecteur.id, NULL, lecture, &sock);
40      if (ret != 0)
41          erreur_IO("creation thread");
42
43      while (!fin) {
44          printf("> ");
45          if (fgets(ligne_ecr, LIGNE_MAX, stdin) == NULL)
46              erreur("saisie fin de fichier\n");
47
48          lgEcr = ecrireLigne(sock, ligne_ecr);
49          if (lgEcr == -1)
50              erreur_IO("ecrire ligne");
51
52          //printf("%s: %d bytes sent\n", CMD, lgEcr);
53
54          if (strcmp(ligne_ecr, "fin\n") == 0)
55              fin = VRAI;
56      }
57
58      if (close(sock) == -1)
59          erreur_IO("close socket");
60
61      exit(EXIT_SUCCESS);
62  }
```

Pour permettre au client d'afficher les messages reçus par le serveur, il suffit de créer un autre thread, qui appelle ici la fonction « lecture » prenant en argument le canal sur lequel les messages seront envoyés par le serveur.
Dès lors, il suffit de faire une boucle infinie contenant l'instruction « lireLigne(sock, ligne) ; » afin de récupérer tout ce qui est écrit dans ce canal puis l'afficher dans la console via un printf.

On obtient alors ce code :

```
64
65  void *lecture(void *arg)
66  {
67      int *sock_ = (int *)arg;
68      int sock = *sock_;
69      char ligne[LIGNE_MAX];
70      int lgEcr;
71
72      while (VRAI) {
73              lgEcr = lireLigne(sock, ligne);
74              if (lgEcr == -1)
75                  erreur_IO("lire ligne");
76
77              else if (lgEcr == 0)
78                  erreur_IO("Fin de la session.");
79
80              printf("%s\n\n> ", ligne);
81          }
82  }
```

B- Récupération des informations du destinataire

Après avoir configuré la communication entre deux clients, il faut être en mesure de récupérer les informations du destinataire. On modifie donc la structure DataSpec de cette manière :

```
4 /* module datathread : donnees specifiques */
5
6 /* donnees specifiques */
7 typedef struct DataSpec_t {
8   pthread_t id;               /* identifiant du thread */
9   int libre;                  /* indicateur de terminaison */
10 /* ajouter donnees specifiques après cette ligne */
11   int tid;                   /* identifiant logique */
12   int canal;                 /* canal de communication */
13   int tid_dest;              /* ID du destinataire */
14   int last_tid_dest;          /*ID du destinataire précédent */
15   sem_t sem;                 /* semaphore de reveil */
16   char pseudo[LIGNE_MAX];    /* pseudo de la session */
17   int tid_request;            /*determine si le client a une demande de connexion*/
18   bool pret;
19 } DataSpec;
20
```

On a alors rajouté les éléments suivants :

- Le pseudo de l'utilisateur permettant de l'identifier auprès des autres clients
- Le tid du destinataire (son affectation est détaillée un peu plus tard dans le rapport). Le choix de rajouter le tid est stratégique, cela permet de facilement envoyer des informations au destinataire et de récupérer le pseudo de ce dernier grace au tableau global dataSpec.
- Le tid du dernier destinataire permettant de savoir avec quel client il a discuté en dernier et de ce fait lui éviter d'être à nouveau connecté avec lui lors de sa prochaine connexion (après un NEXT!).

- Le tid_request aurait dû permettre de récupérer les données du client cherchant à se connecter à un autre. L'idée était la suivante : si un client1 veut discuter avec un autre client2 en particulier, le champ « tid_request » de ce dernier est modifié pour contenir le tid du client1 demandant la communication. On peut de ce fait récupérer son pseudo beaucoup plus rapidement sur le thread du client2 pour lui demander s'il souhaite lui-aussi communiquer avec le client1. Tout cela sera expliqué en détail plus loin.
- Le booléen « pret » sert seulement à ce qu'un client qui est en pleine déconnexion ne soit pas reconnecté avec quelqu'un avant qu'il soit retourné à l'accueil. Il est utilisé dans la fonction « threadworker » sur laquelle on va se pencher tout de suite :

La fonction threadWorker est la fonction appelée lors de la création d'un thread. C'est en quelque sorte l'initialisation d'un client. En effet, dans cette fonction va être appelée la fonction get_pseudo(int canal, char pseudo[LIGNE_MAX]) dans laquelle le client va devoir rentrer son pseudo. Comme le pseudo est l'identificateur de chaque client, il doit être unique.

```
void get_pseudo(int canal, char pseudo[LIGNE_MAX])
{
    bool fin = false;
    char ligne_renv[LIGNE_MAX];

    strcpy(ligne_renv, "Rentrez votre pseudo SVP : ");
    int lgLue  = ecrireLigne(canal, ligne_renv);

    while (!fin)
    {
        lgLue = lireLigne(canal, pseudo);
        verif_lgLue(lgLue);

        fin = true;
        for (int i=0 ; i<NB_WORKERS ; i++)
        {
            if (strncmp(pseudo, dataSpec[i].pseudo, LIGNE_MAX) == 0)
            {
                strcpy(ligne_renv, "Erreur, pseudo déjà utilisé. Veuillez en choisir un autre : ");
                ecrireLigne(canal, ligne_renv);
                fin = false;
            }
        }
    }
```

C'est également dans la fonction threadWorker que va être établie chaque connexion entre deux clients.

B) Les choisir aléatoirement

Dans notre modèle, les clients sont connectés ensemble de manière aléatoire. Dès lors il est important de préciser que l'on travaille dans un modèle de workers statiques. On cherche donc un nombre aléatoire entre 0 et le nombre de clients possibles – 1 (taille du tableau de

workers noté ici NB_WORKERS), ce nombre est l'indice du tableau et correspond à un worker. Pour être sélectionné, ce nombre doit respecter 5 conditions :

```
while (!connected)
{
    dest_id = rand_a_b(0, NB_WORKERS);

    if (dataTh->tid_request != -1)  //SI quelqu'un essaie de se connecter à cette personne, on écoute sa réponse
    { ...
    }

    lockMutexBoucle();
    if (dest_id != dataTh->tid && dataSpec[dest_id].tid_dest == -1 &&
        dataSpec[dest_id].canal != -1 && dest_id != dataTh->last_tid_dest &&
        dataSpec[dest_id].pret == true)
        // Si le dest_id n'est pas le canal lui même, si ce n'est pas un client déjà en communication
        // et si le canal dest est en ligne et si ce n'est pas le canal avec lequel on était à l'instant en communication
```

- dest_id ne doit pas être le canal lui-même (inutile pour un canal d'être connecté à lui-même).
- dest_id ne doit pas être un canal déjà en communication.
- dest_id doit évidemment être en ligne (cette condition est implictement incluse dans le deuxième, mais on a préféré l'expliciter pour plus de clarté)
- dest_id ne doit pas être le canal avec lequel on vient de communiquer, donc que l'on vient de quitter en faisant un « NEXT! »
- Enfin, c'est la condition qui a nécessité le plus de travail et de réflexion : le canal doit être « prêt » à se connecter.

En effet, lors de nos tests nous nous sommes rendus compte que, lorqu'un canal en attente dans l'accueil détectait un canal qui venait de faire un « NEXT! », il se connectait plus rapidement que ce dernier ne se déconnectait. Il a fallu donc faire attention à cela en créant le champ « pret » dans le fichier DataSpec.h. Le client devient alors « prêt » à se connecter lorsqu'il a correctement rejoint l'accueil.

Si toutes ces conditions sont respectées, les champs « tid_dest » sont initialisés et l'on sort de la boucle. Les champs du canal destinataire sont également initialisés (toujours à l'aide de mutex), le thread destinataire ne pourrait donc pas vérifier toutes les conditions préalablement citées. C'est pourquoi on a fait un deuxième « if » vérifiant simplement si le canal destinataire était prêt pour discuter avec l'autre client.

```
else if (dataTh->tid_dest != -1 && dataTh->pret == true) //S'il a déjà été affecté par son destinataire
{
    printf("ON EST ICIII POUR %s\n", dataTh->pseudo);
    connected = true;
    dataTh->pret = false;
    dataSpec[dataTh->tid_dest].pret = false;
    strcpy(ligne_renv,"");
    ecrireLigne(dataTh->canal, ligne_renv);
    strcpy(ligne_renv,"Vous êtes maintenant en ligne avec ");
    strcat(ligne_renv, dataSpec[dataTh->tid_dest].pseudo);
    ecrireLigne(dataTh->canal, ligne_renv);
```

Une fois qu'un canal est prêt pour la communication, il sort donc de la boucle while et la fonction « SessionClient » est appelée.

Cette fonction permet deux choses :
- Transmettre les messages du client 1 au client 2

- Créer des « commandes » permettant par exemple à un client de quitter l'application ou changer de partenaire.

```
else if (strcmp(ligne, "FIN!") == 0) {
        printf("%s: fin client\n", CMD);
        changement_partenaire(dataTh);
        fin_chat = true;
        fin_session = true;
 }
else if (strcmp(ligne, "NEXT!") == 0) {
        printf("%s: changement de partenaire\n", CMD);
        changement_partenaire(dataTh);
        fin_chat = true;
}
```

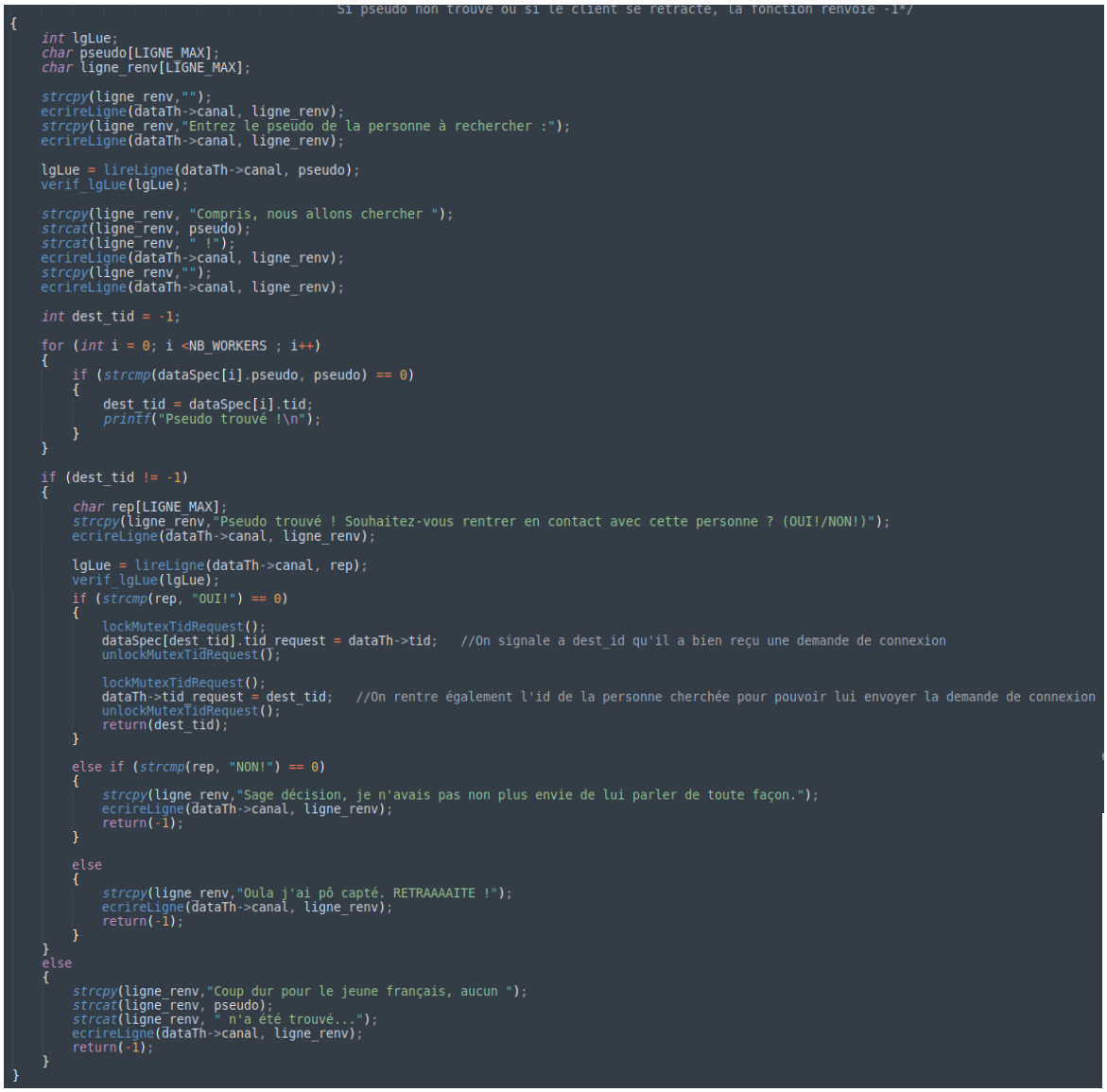
C) Changer de client

On veut permettre à l'utilisateur de changer d'interlocuteur dès qu'il le souhaite et cela en une commande : « NEXT! ». Dès que cette commande est entrée, le serveur appelle la fonction « changement_destinataire » qui permet de rénitialiser les champs « tid_dest » et mettre à jour les champs « last_tid_dest » des deux clients connectés. Nous avons fait face à un problème : la fonction « lireLigne » est bloquante.
De ce fait, le thread du client 2 n'ayant pas entré « NEXT! » attend de recevoir un message. Il faut alors que le thread du client 1 voulant changer de destinataire prévienne le client 2 que la connexion a été interrompue et qu'il doit retourner à l'accueil. Cela est fait via la fonction « ecrireLigne » et détecté grace à la condition dans la fonction « sessionClient » :

```
if (dataTh->tid_dest == -1)
{
    strcpy(ligne_renv, "Vous voilà retourné à l'accueil.");
    ecrireLigne(dataTh->canal, ligne_renv);
    fin_chat = true;
}
```

D) Choisir le client grâce à son pseudo

Nous avons également voulu offrir la possibilité à un client de choisir son destinataire en rentrant son pseudo. Nous avons donc créé la commande « CHERCHER! » dans « sessionCient » qui appelle la fonction « chercher_pseudo » :



```
                                    Si pseudo non trouvé ou si le client se rétracte, la fonction renvoie -1*/
{
    int lgLue;
    char pseudo[LIGNE_MAX];
    char ligne_renv[LIGNE_MAX];

    strcpy(ligne_renv,"");
    ecrireLigne(dataTh->canal, ligne_renv);
    strcpy(ligne_renv,"Entrez le pseudo de la personne à rechercher :");
    ecrireLigne(dataTh->canal, ligne_renv);

    lgLue = lireLigne(dataTh->canal, pseudo);
    verif_lgLue(lgLue);

    strcpy(ligne_renv, "Compris, nous allons chercher ");
    strcat(ligne_renv, pseudo);
    strcat(ligne_renv, " !");
    ecrireLigne(dataTh->canal, ligne_renv);
    strcpy(ligne_renv,"");
    ecrireLigne(dataTh->canal, ligne_renv);

    int dest_tid = -1;

    for (int i = 0; i <NB_WORKERS ; i++)
    {
        if (strcmp(dataSpec[i].pseudo, pseudo) == 0)
        {
            dest_tid = dataSpec[i].tid;
            printf("Pseudo trouvé !\n");
        }
    }

    if (dest_tid != -1)
    {
        char rep[LIGNE_MAX];
        strcpy(ligne_renv,"Pseudo trouvé ! Souhaitez-vous rentrer en contact avec cette personne ? (OUI!/NON!)");
        ecrireLigne(dataTh->canal, ligne_renv);

        lgLue = lireLigne(dataTh->canal, rep);
        verif_lgLue(lgLue);
        if (strcmp(rep, "OUI!") == 0)
        {
            lockMutexTidRequest();
            dataSpec[dest_tid].tid_request = dataTh->tid;   //On signale a dest_id qu'il a bien reçu une demande de connexion
            unlockMutexTidRequest();

            lockMutexTidRequest();
            dataTh->tid_request = dest_tid;   //On rentre également l'id de la personne cherchée pour pouvoir lui envoyer la demande de connexion
            unlockMutexTidRequest();
            return(dest_tid);
        }

        else if (strcmp(rep, "NON!") == 0)
        {
            strcpy(ligne_renv,"Sage décision, je n'avais pas non plus envie de lui parler de toute façon.");
            ecrireLigne(dataTh->canal, ligne_renv);
            return(-1);
        }

        else
        {
            strcpy(ligne_renv,"Oula j'ai pô capté. RETRAAAAITE !");
            ecrireLigne(dataTh->canal, ligne_renv);
            return(-1);
        }
    }
    else
    {
        strcpy(ligne_renv,"Coup dur pour le jeune français, aucun ");
        strcat(ligne_renv, pseudo);
        strcat(ligne_renv, " n'a été trouvé...");
        ecrireLigne(dataTh->canal, ligne_renv);
        return(-1);
    }
}
```

Si le pseudo entré existe, son tid est stocké dans tid_request et une demande de connexion lui est envoyée. Ce dernier aurait alors seulement à l'accepter et, le cas échéant, renvoyer à l'accueil les deux personnes avec lesquelles elles étaient.

Cependant, après maintes tentatives pour modifier les canaux des deux principaux concernés, les messages n'étaient toujours pas transmis correctement. Nous avons donc fait le choix d'abandonner cette fonctionnalité pour l'instant car elle rendait le programme extrêmemezeqnt plus compliqué.

III – Retour d'expérience

Cette partie restitue nos ressentis sur la conduite de ce projet, les difficultés éventuelles rencontrées durant toute la période de travail. Tout d'abord, nous avons tous pris du plaisir à faire ce projet, car cette programmation possède une application ludique et nous sommes impatients de voir tout le monde l'utiliser.

A travers ce projet, nous avons pu apprendre à utiliser l'outil Git. Celui-ci s'est montré très pratique, notamment lorsque l'on travaille à plusieurs sur un projet. Le fait de pouvoir récupérer les versions précédentes de notre code en cas d'erreur est également très utile. Nous avons conscience que sa maîtrise est un vrai atout pour nos futurs projets de développement et apprendre à l'utiliser dans ce contexte est un vrai plus.

Cependant, nous avons trouvé cet outil très dur à prendre en main et, même après plusieurs dizaines de commits et quelques branches créées, nous ne le maîtrisons pas totalement et avons fait face à des erreurs que l'on a mis beaucoup de temps à résoudre en plus des erreurs auxquelles on a dû faire face dans le code.

IV – Conclusion

Nous avons donc mis au point un tchat connectant deux personnes aléatoirement tout à fait fonctionnel. De plus, il offre la possibilité à l'utilisateur de changer d'interlocuteur dès qu'il le souhaite. Nous aurions souhaité rajouter une recherche par pseudo, mais nous n'y sommes pas arrivés. Nous espérons que ce projet plaira à la classe et qu'ils prendront autant de plaisir à l'essayer que nous à le concevoir.