Bordeaux INP – ENSEIRB-MATMECA

Systèmes Électroniques Embarqués

Système d'exploitation

MI 207

Rapport de projet

Projet implémentation d'un tchat en C

Par

Vincent GIRONES – Simon LESIEUR

Supervisé par

Guillaume DELBERGUE

Novembre 2019

Sommaire

1 Introduction

Pour ce projet de système d'exploitation, notre but a été d'implémenter un programme de chat en **langage C**. Ce chat est basé sur le principe d'un contrôleur gérant plusieurs clients. Le programme doit utiliser des appels systèmes (des fonctions du shell) et gérer les différents « *pipes* » nécessaires à son fonctionnement.

2 Fonctionnalités gérées par notre programme

2.1 Gestion du nom d'utilisateur

Au lancement du contrôleur puis du client, ce dernier propose à l'utilisateur de se choisir un pseudo qui sera communiqué aux autres utilisateurs par la suite (voir figure ci-dessous, qui montre la gestion du pseudo côté client).

```
//------------------------------ID CLIENT-----------------------------
char nameClient[16];
printf("Entrer votre nom d'utilisateur:");
exit_if((fgets(nameClient, 16, stdin) == NULL),"fgets");
char idClient[40]= "/i";
strcat(idClient,";");
strcat(idClient,pathClientPipe);
strcat(idClient,";");
strcat(idClient,nameClient);
exit_if((write(fileDescServ, idClient, strlen(idClient)+1) == -1), "write");
```

Figure 1 : gestion des pseudos

2.2 Gestion du signal SIGINT

Pour fermer le processus du contrôleur ainsi que celui des clients, nous avons prévu une fonction *handle_sigint* dans le code du contrôleur (voir figure ci-dessous). Quand celui-ci reçoit la commande « ctrl-C », il envoie une fonction spéciale « /c » aux clients, ce qui les ferme et supprime tous les *pipes*, du contrôleur et des clients.

```
//-------gestion des CTRLC et fermeture client-------
void handle_sigint(int sig)
{
  printf("Caught signal %d\n", sig);
  for(int i = 0 ; i<nbOfClient ; i++){
    printf("je tue :%s\ns",clientTab[i].name);
    exit_if((write(clientTab[i].pid, "/c", strlen("/c")+1) == -1),"write");}
  close(fileDescServ);
  system( "rm -rf /tmp/chat/" );
  exit(0);
}
```

Figure 2 : fonction SIGINT

2.3 Commandes /w et /q

Alors qu'un client est lancé avec le contrôleur, il peut taper les commande /w et /q. La commande /w lui permet de savoir combien d'utilisateurs sont connectés (ou afficher si le client est seul) et afficher leurs pseudos (voir figure ci-dessous). La commande /q lui permet de quitter le chat proprement, ce qui supprime son pipe associé et le déconnecte du contrôleur.

```
for(int i = 0 ; i<nbOfClient ; i++){
  if( currentClientId != i){
    printf("Il y a %s\n",clientTab[i].name);
    strcpy(bufferS,"-");
    strcat(bufferS,clientTab[i].name);
    strcat(bufferS,"\n");
    sleep(1);
    exit_if((write(clientTab[currentClientId].pid, bufferS, strlen(bufferS)+1) == -1),"write");}}}
```

Figure 3 : affichage du nombre d'utilisateurs (côté contrôleur)

2.4 Kill des clients et suppression de leur pipes

Quand un client se ferme, il envoie une requête au serveur qui met à jour sa table des clients (voir figure ci-dessous).

```
//-------------CONSIGNE QUIT-----------------
else if(bufferR[1] == 'q')
{
  exit_if((write(clientTab[currentClientId].pid, "/c", strlen("/c")+1) == -1),"write");
  printf("CLOSING CLIENT++++++\n");
  for(int i = currentClientId;i<(nbOfClient-1);i++){
    clientTab[i]=clientTab[i+1];}
  nbOfClient--;
}
```

Figure 4 : gestion de la table des clients

3 Pistes d'amélioration

Durant le développement de ce chat, nous avons rencontré certaines difficultés, principalement dues à un manque de temps pour mettre au point toutes les modifications nécessaires.

- **Gestion des pseudos**
  Notre programme ne génère pas d'erreur (de quelque manière qu'il soit) si un pseudo choisi est déjà présent dans la table des pseudos.
  De plus, il n'est pas possible pour un client de changer de pseudo. Il ne faut normalement que rajouter une commande du même type que les /w et /q mais cela reste à implémenter.

- **Gestion des threads**
  Nous ne sommes pas parvenus à utiliser des threads à la place des processus internes des clients.
- **Contrôleur en tâche de fond**
  Le contrôleur que nous avons implémenté nécessite une console pour fonctionner et ne peut pas fonctionner en fond uniquement.
- **Lancement auto du serveur s'il n'est pas lancé**
  A cause d'une erreur inconnue, nous ne sommes pas parvenus à lancer le serveur à partir d'un client s'il n'est pas déjà lancé (pipe n°0 inexistant). Nous connaissions la structure utilisant l'appel système *fork()* mais nous n'arrivions pas à l'implémenter correctement d'une une fonction à part entière.

4 Conclusion

Ce projet d'implémentation d'un chat nous a permis de développer et appliquer nos connaissances en langage C. De plus, il nous a permis de manipuler plus amplement les appels systèmes ainsi que gérer leurs erreurs possibles (notamment avec la fonction *exit_if*). Certaines améliorations auraient pu être apportées, mais nous sommes parvenus à implémenter un chat fonctionnel qui répond à certains points du cahier des charges.