Rapport de Projet de 3.6: Système d'exploitation
Projet Simulateur

ADAM Philibert - BOUVERON Matthieu

Attendus du projet

Dans le cadre de ce projet, il était attendu de mettre en place un système client/serveur permettant à la fois d'un côté une modélisation d'un espace «Océan» et de l'autre la modélisation de structures «bateaux» qui évoluent dans l'espace. Différentes voies de communication doivent donc être mises en place pour communiquer les informations relatives aux déplacements, ainsi que les messages publiés par les différents bateaux à destination des autres.
Il est également attendu une visualisation succincte de l'espace de navigation avec la position des obstacles et des bateaux qui y circulent.

Fonctionnalités actuellement mises en place

- Connexion au serveur
- Communication et enregistrement du nom du bateau dans une liste de clients connectés au serveur
- Création d'un système de discussion commun à tous les clients

Fonctionnement

- Utiliser la commande `$make` pour compiler les fichiers `MainServer.c` et `MainClient`.
- Dans un terminal, démarrer le serveur en utilisant la commande `$./MainServer`
- Dans d'autres terminaux, connecter des clients en utilisant la commande `$./MainClient [addresseServeur] [nickname]`
- Les clients sont alors connectés au channel de tchat et peuvent communiquer entre eux.
- Pour quitter: taper `$Quit` dans le terminal d'un client, il sera déconnecté.
- Quand tous les clients sont déconnectés, n'importe quelle entrée clavier entraîne la déconnexion du serveur.

Architecture du code

Notre code est constitué de deux fichiers principaux, *MainClient.c* et *MainServer.c,* qui gèrent respectivement les connections et traitement des côtés serveur et client aux différents moyens de communication. Ces deux fichiers font appel, *via* le *header inclusions.h* et les différentes inclusions qu'il contient, à des fonctions définies pour le traitement des *sockets* dans le fichier *sock.c*.

Le fichier *MainServer.c*

On commence par créer des variables correspondant à:

- un buffer, qui stocke les chaînes de caractères avant envoi ou après réception sur le *socket* de discussion (*chat*)
- le nombre de clients connectés
- une liste des clients connectés, dans la limite d'un nombre maximal défini au préalable

On met ensuite en attente deux *sockets*: un qui sert à la détection et à l'enregistrement des nouveaux clients qui veulent se connecter aux différents services (actuellement seul le *chat* est fonctionnel), l'autre qui sert aux transferts de messages entre les bateaux *via* le serveur.

Un mécanisme reposant sur la fonction *select()* et ses fonctions associées permet d'attendre que des données soient disponibles dans les différents *sockets* pour déclencher les traitements correspondants:

- Si on détecte quelque chose sur le *socket* de connection (*socketConnection*), cela signifie qu'un nouveau client se connecte. On l'enregistre alors dans la liste des clients connectés et on lui ouvre l'accès aux différents services. On récupère également ses données de connections (les noms des différents *sockets* de service qui lui correspondent et le nom qu'il communique lors de sa connexion).
- Si on détecte quelque chose sur un des *sockets* de service de *chat* (*client.sChat*), cela signifie qu'un client a envoyé un message à diffuser aux autres clients. On expédie donc ce message aux autres clients.
- Si on détecte une entrée clavier, on sort de la boucle de traitement. On ferme alors toutes les connexions éventuellement encore ouvertes avec des clients, on ferme les connexions globales et on met fin au programme.

Le fichier *MainClient.c*

Tout d'abord, si l'utilisateur essaie de lancer ce programme sans donner les paramètres nécessaires au traitement, à savoir l'adresse du serveur et le nom de son bateau, on lui indique la syntaxe à suivre. Ces paramètres sont récupérés dans *argv[]*. Dès que ces paramètres sont passés, on se connecte aux *sockets* de connexion et de discussion du serveur, placés en attente, et on envoie le nom du bateau sur *socketConnection* pour l'enregistrement.

De même que précédemment, on utilise le mécanisme *select()* pour détecter les différentes entrées de données. Il n'est toutefois pas nécessaire ici d'écouter le *socket* de connection.

- Si on détecte une entrée sur *socketChat*, c'est qu'il s'agit d'un message émis par un autre bateau. On le récupère donc et on l'affiche sur le terminal. A noter: le cas où le serveur se déconnecte de manière impromptue est traité et occasionne la déconnexion du client.
- Si on détecte une entrée sur le clavier différente de «Quit», on la stocke dans le buffer et on l'envoie au serveur pour diffusion. Si le mot «Quit» est rentré, on sort de la boucle d'écoute et on met fin aux connections.

Perspectives

Les parties *connexion* et *discussion* fonctionnent très bien et ont été comprises en détail. De plus, les concepts d'écriture modulaire ont été assimilés. On voit donc qu'il reste beaucoup de tâches à accomplir avant de considérer le projet comme totalement terminé vis-à-vis du cahier des charges.

Les communications des autres données nécessaires à l'évolution du système modélisé sur d'autres *sockets* ne devrait pas poser de problèmes, si ce n'est l'utilisation de *threads* pour paralléliser les traitements afin de ne pas être bloqué par des actions sans rapport direct avec celle voulue.

L'affichage de la carte a déjà été réfléchi (voir le fichier *ocean.c*), bien qu'il ne fonctionne pas encore. De même, quelques structures ont été approuvées au niveau de groupe de test pour standardiser les différentes communications. Ces structures sont décrites dans *struct.h*, même si elles n'ont pas encore été mise en place dans notre code. Les deux autres binômes les ont implémentées.