BELLA Jean-Paul
LABERGE Marc-Antoine

IFT630 - Devoir 3

PARTIE I

Démarches et recherches:

Pour réaliser ce travail, une connaissance préalable sur le fonctionnement d'un serveur a été nécessaire. Nous nous sommes renseigner sur la mise en place d'un serveur communiquant à l'aide de socket car cela semblait être la solution majoritairement utilisée par la communauté et nous permettant donc d'avoir facilement accès à de l'aide sur des forums ou grâce à des tutoriels.

Dans un premier temps, nous avons établi un serveur communiquant avec un unique client afin de bien maîtriser cette nouvelle notion. Assez rapidement, nous nous sommes alors entrepris à l'incorporation d'une commande "fork" insérée dans une boucle afin de créer tous les clients et le serveur dans des processus distincts. Une première difficulté est apparue avec ce découpage lorsqu'il fallait repérer le processus parent. En effet, les processus enfant passaient systématiquement par l'état "parent" avant de devenir enfant pour le reste du déroulé de l'algorithme. Nous avons résolu ce problème en ajoutant un condition ne permettant qu'au dernier processus d'être serveur.

Une fois les clients et le serveur lancés, il fallait s'assurer de la bonne communication entre ces derniers. C'est alors qu'un nouvel outil de parallélisme est nécessaire: le mutex. Les communications avec le serveur étant simultanées, un mutex sur la sortie standard a du être mis en place pour éviter le chevauchement des chaînes de caractères rendus par les clients. A présent, nous avons plusieurs clients communiquant simultanément à un unique serveur. C'est alors que nous avons décidé d'introduire le multithreading. Afin de représenter une utilisation concrète du multithreading, nous avons simulé un petit restaurant. Le restaurant possède trois fournisseurs (= 3 clients) qui apportent fruits, légumes et viandes. Le gestionnaire des ressources du restaurant (serveur) prend note des différentes arrivées, puis met à jour les 3 registres (fichiers "fruits.txt", "vegetables.txt", "meats.txt") en veillant à bien séparer les aliments (3 threads distinctes qui écrivent sur des fichiers distincts). De cette façon, le multithreading permet de s'affranchir de la nécessité de fermer et d'ouvrir en boucle les différents fichiers après écriture, puisqu'un flux est dédié à chacun des fichiers.

Le client dynamique est lancé à partir de l'exécutable "client". Malgré notre volonté de permettre l'écriture de ce client, nous ne parvenons qu'à connecter le client au serveur, mais la communication entre ces derniers est rendue difficile certainement par l'ouverture d'un autre file descriptor pour le socket du nouveau client. Cependant, les signaux sont bien détectés par le client comme l'énoncé le précisait.

Mode de déploiement:

Pour lancer le server et ses clients, il suffit d'entrer un terminal placé dans le répertoire où se trouve ce même document la commande suivante:

clang++ server.cpp -o server && ./server 8080 localhost

OU

gcc++ server.cpp -o server && ./server 8080 localhost

Pour lancer le client en mode interactif, il faut ouvrir un nouveau terminal et lancer la commande suivante:

clang++ tcpClient.cpp -o client && ./client 8080 localhost

OU

gcc++ tcpClient.cpp -o client && ./client 8080 localhost

Références:

Communication par socket Serveur-Client

http://www.bogotobogo.com/cplusplus/sockets_server_client.php

Utilisation de fork pour la création d un serveur avec un processus par client

https://www.youtube.com/watch?v=BIJGSQEipEE
https://www.geeksforgeeks.org/create-n-child-process-parent-process-using-fork-c/