RAPPORT DE PROJET

Unité d'enseignement Programmation en C

8 DECEMBRE 2023

ANTOINE PROMIS
ARNAUD BONNET

Table des matières

1. Organisation du code

Tout d'abord, ce projet est plus ou moins une simulation du fonctionnement d'une entreprise. Il y a 3 fichiers principaux qui correspondent chacun à un « acteur » majeur du projet. On retrouve le fichier Client.c qui simule un client qui vient demander un service à une entreprise. Le fichier Master.c qui correspond au chef d'un projet dans une entreprise et qui traite les demandes du clients et distribue les tâches aux employés. Ces employés sont donc représentés par le fichier Worker.c.

Chacun des trois types de fichiers créé un processus. Le client est créé avec la commande ./client suivie de l'ordre qu'il veut envoyer et le communique au master. Le master est quand à lui créé avant le client avec la commande ./master. Le master traite l'ordre que le client lui a donné et le transmet aux workers qui s'occupent de réaliser la demande du client une fois le résultat obtenu, les workers envoient une réponse qui a pour but de dire au master que l'ordre à été réalisé puis ils envoient au master le résultat obtenu. Ce résultat est ensuite envoyé au client qui s'arrête après avoir obtenus sa réponse. Ce n'est pas le cas du master et des workers qui eux attendent un nouvel ordre de la part d'un client. Pour s'arrêter, ils auront donc besoin d'un ordre leurs disant de le faire.

Il y a également des fichiers qui ont une utilité pratique comme le fichier utils.c qui fournis quelques programmes utiles au projet ou le fichier myassert.c qui permet de créer ses propres vérification des retours de fonctions. Il y a également des fichiers client_master.h et master_worker.h qui permettent la mise en commun de certaines variables et fonctionnalités entre plusieurs fichiers (entre le client et le master pour l'un et entre le master et les workers pour l'autre). Enfin il y a des scripts fournis pour nous aider et pour gagner du temps, le script rmsempipe.sh permet de détruire les sémaphores et les tubes si le master ne le fait pas correctement ou s'arrête brutalement. Le script pstree affiche la structure en forme d'arbre binaire crée par les workers et le script test_client.sh permet de vérifier rapidement que toutes les fonctionnalités sont utilisables correctement.

2. Protocoles de communication

Il y a plusieurs types de communication utilisés ainsi que des système pour réguler le nombre de client et l'ordre d'exécution .

2.1 Les tubes nommés

Il y a deux tubes nommés utilisés dans ce projet. Ils permettent la communication entre le client et le master. Un tube fait la communication dans un sens (le master envoie des informations au client), nous avons décidé de l'appeler COM_TO_CLIENT, le deuxième s'appelle COM_FROM_CLIENT et permet la communication dans le sens inverse (du client vers le master). Les tubes nommés ont l'avantage d'être facilement accessible si on connait leurs nom et leurs emplacement.

2.2 Les tubes anonymes

Il y a ensuite des couples de tubes anonymes, ils permettent la communication entre le master et les workers et entre workers également. Il s'agit de tubes sans noms auxquels on accède grâce à un identifiant. Ces tubes permettent de communiquer entre un processus père et son fils grâce à une duplication des entrées des tubes puis fermetures des entrées inutiles. Cette duplication se fait à l'aide d'un fork suivi d'un exec. Il existe à chaque fois un tube dans un sens et un dans l'autre pour que les échanges soient possibles. Le client n'a accès à aucun de ces tubes et ne communique donc qu'avec le master.

2.3 Un tube anonyme commun

Enfin il existe un tube anonyme pour que les workers puissent envoyer des informations directement au master si les processus pères n'ont pas besoin de traiter ces données. Tout les workers peuvent envoyer des informations et le master peut les récupérer mais pas l'inverse.

2.4 Les sémaphores

Enfin pour réglementer le nombre de clients au même moment, on introduit un système de sémaphores qui ne permet d'avoir qu'un seul client au même moment. Et un deuxième sémaphore qui permet de mettre le master en attente d'un nouveau client.

3. Problèmes

Il n'y a aucun problème dans le fonctionnement du projet si ce n'est que dans certains cas nous nous sommes rendu compte que notre affichage final était peut être un peut léger et qu'il faut peut être afficher plus d'informations. Nous avons testé le projet avec le script test_client.sh qui ne renvoie aucune erreur, les résultats attendus sont présents lors de l'exécution. Enfin si on teste le projet avec un analyseur de mémoire comme valgrind, il n'y a pas de fuites mémoires.

Il reste cependant un dernier problème, nous avons utilisé git pour gérer les versions du projet, il y a des commits et des push réguliers mais nous n'avons pas placé notre fichier .git au bon endroit. Le vérificateur d'archives indique donc un warning.