IT202
Rapport intermédiaire

April 17, 2015

Équipe : Anas Hakkal, Vincent Jeanjean, Erwan Le Masson, Bakari Nouhou, Maxime Paillassa
Encadrant : Mathieu Faverge

Introduction

Ce projet de deuxième année au département informatique de l'ENSEIRB-MATMECA a pour objectif l'implémentation d'une bibliothèque de thread dans l'espace utilisateur possédant la même interface que `phtread`.

Avancement du projet

Le premier objectif de ce projet est l'implémentation d'une version mono-cœur sans préemption. Entièrement implémentée, cette première version doit être capable d'exécuter les tests fournis.

À l'heure actuelle, une implémentation a été proposée, cependant elle ne permet de valider qu'un seul test. En particulier, les fonctions `thread_create` et `thread_yield` ont donné des résultats satisfaisants sur des tests réduits. L'attente d'un thread n'a pas été remis en cause par les tests mais le changement de contexte entre plusieurs thread ne fonctionne pas.

L'origine des problèmes n'a pas été identifiée, la manipulation de plusieurs contextes rendent l'étape de débuguage très délicate. Il est peut être nécessaire de revoir nos méthodes de validation avec, par exemple, un makefile plus évolué et l'ajout d'un certain nombre de tests unitaires.

L'objectif pour les prochaines séances est en priorité de passer les tests avant de viser des fonctionnalités plus avancées.

Choix d'implémentation

À l'initialisation de la bibliothèque, deux listes globales de thread sont créées. Elle contiennent les threads en attentes ainsi que les threads actifs respectivement. Le thread courant est toujours en tête de la liste active. Leur implémentation utilise les listes génériques *BSD queue*.

La structure `thread_s` est utilisée pour représenter une entrée dans la liste. Elle contient le contexte du thread, une référence vers un thread en attente et une valeur de retour. L'adresse d'une instance sert d'identifiant, c'est pourquoi le type `thread_t` est en réalité un pointeur vers une structure. L'état d'un thread est codé par la liste dans laquelle il se trouve et sa position dans la liste.

Un appel à `thread_yield` inverse simplement le thread courant et un autre dans la liste, puis change de contexte. Pour se mettre en attente, un thread place sa référence dans la structure du thread à attendre et se place dans la liste des threads en attente. Lorsqu'un thread s'arrête, il conserve sa valeur de retour et se rend inaccessible depuis la liste des threads actifs. Si un thread était en attente, il est réveillé et un appel à `thread_yield` est effectué.

Le thread principal est ajouté à la liste des threads actifs par défaut à au chargement de la bibliothèque pour pouvoir manipuler son contexte comme

n'importe quel autre. Les codes d'initialisation et de destruction des variables globales sont placés dans des fonctions possédant les attributs *constructor* et *destructor* de *gcc*. Pour assurer un appel systématique à la fonction `thread_exit`, une fonction de couverture a été ajoutée.

Cette implémentation permet de ne pas réveiller un thread encore en attente mais ne fait aucune garantit quant à la répartition du temps d'exécution. De plus, elle ne résout pas certain problèmes de libération mémoire. Il semble aussi nécessaire de revoir l'ordre dans lequel les threads sont échangés dans la liste active pour éviter une famine.

Conclusion

La première phase du projet est implémentée mais n'est toujours pas fonctionnelle. Bien qu'en partie temporaire, il n'est pas envisageable de commencer à changer le code pour respecter les objectifs finaux, il est donc essentiel de corriger cette première étape.