Projet Système : Gestion de threads

Thibaud Cheippe, Lux Benjamin, Boutin Guillaume, Papa Gueye

23 avril 2012

Introduction

Le but de ce projet est de coder en C une bibliothèque de threads en espace utilisateur. Nous avons commencé par nous familiariser avec les mécanismes de changement de contexte en exécutant le programme fourni en exemple, puis nous l'avons modifié pour mieux comprendre le comportement des threads lors des changements de contexte. Nous avons implémenté une structure de données pour gérer les threads, et avons codé les premières fonctions de manipulation de threads.

Base de donnée

Structure

La liste des threads est implémentée de façon simple, il s'agit d'une liste d'élément chaînés. Nous avons choisi de mettre un pointeur sur le dernier élément de la liste pour des raisons pratiques. (cela sera expliqué plus loin).

La liste contient des structures threads :

```
struct thread{
  ucontext_t* context;
  void ** retval;
  int priority;
};
```

Le champs retval est utilisé pour stocker la valeur de retour du thread.

Le champs priorité n'est pas utilisé pour l'instant, il est prévu (à la base) pour modifier la priorité d'exécution des différents threads.

Sélection et insertion de threads

Au lieu d'implémenter directement une gestion des priorités, les threads sont sélectionnés en tête (afin d'être exécutés), et ajoutés en queue. On utilise donc le procédé fifo.

Fonctions sur les Threads

Avancement actuel

À l'heure actuelle nous avons implémenté la totalité des fonctions de l'interface *thread.h* proposée.

thread_self

Cette fonction retourne juste le pointeur vers notre structure. L'adresse en mémoire de celle ci étant unique, celà fait un identifiant unique pour chaque thread (une fois casté en int).

thread_create

Cette fonction créer un nouveau thread en allouant la place nécessaire à un *struct thread* ainsi qu'à une pile de la même taille que le programme courant. On remplit le *u_context* avec le context courant puis on appelle *makecontext()* avec une fonction qui se chargera d'apeller la fonction passé en paramètre puis stockera son résultat et enfin détruira le thread en désalouant la mémoire consomée par le *u_context*.

thread_yield

Elle remplace le thread courrant avec celui qui est retirer de la pile avec *get_lower_priority_thread()*. Dans le cas ou la pile est vide, *i.e.* il n'y a pas d'autre thread que le thread courant, cette fonction ne fait rien.

thread_join

Cette fonction se contente de vérifier si le thread passé en argument est terminé, si c'est le cas on récupère la valeur de retour stocker avant la destruction du thread.

thread_exit

Cette fonction termine et désaloue le thread courant.

Travail à venir

Cet ensemble de fonction nous servira à manipuler les threads. Nous allons coder des programmes de test pour simuler le fonctionnement de ces threads au fur et à mesure des différents changements de contexte.