H-Nachos : La présentation

Projet d'une conception de système d'exploitation

R. Soumard, C. Semanaz, W. Ayari, J. Neola

28 janvier 2021

Unité de Formation et de Recherche en Informatique, Mathématiques et Mathématiques Appliquées de Grenoble – UGA

Plan

Présentation de l'équipe

Jahna Neola : Mise en place du rapport et des slides

Wassim Ayari : Bonus

Clery Semanaz : Lead Programmer

Romain Soumard : Chef projet, gestion github

Organisation du travail

Travail du Lundi au samedi inclus, de 10 à 18H

Matin : Lecture des énoncés, répartition du travail

Soirée : Rédaction du rapport ou des logs

Workflow github

Créations et répartition des issues

Test des fonctionnaliés

Pull request et inclusion des modifications

Pull request additionnel pour la documentation et les fix

Choix d'implémentation

Organisation des threads

Problème

Comment passer simplement la fonction à exécuter et ses paramètres lors de la création d'un thread utilisateur ?

Comment garder une trace des threads utilisateurs qui s'exécutent ?

Solution choisie

Étendre la classe Thread avec de nouveaux attributs et créer un tableau de thread dans l'espace d'adressage du processus en court afin de pouvoir les manipuler à tout moment.

Organisation des threads

Avantages

Centralisation du code dédié aux threads dans une seule classe.

Manipulation et passage des paramètres plus aisés.

Facilite le comptage et la gestion des threads.

Gestion de la pile grandement simplifiée.

Inconvénients

Les paramètres de do_UserThreadCreate et StartUserThread deviennent inutiles.

Une solution trop simple par rapport à ce qui était attendu ?

Partage de la pile

Problèmes

Quelle taille de pile pour un processus ?

Quelle taille de pile pour ses threads ?

Comment partager la pile de manière satisfaisante ?

Solutions choisies

Calculer la taille de la pile en fonction du nombre de threads utilisateur maximum.

Étendre la classe Thread avec un attribut index et se servir du tableau pour gérer la pile.

Partage de la pile

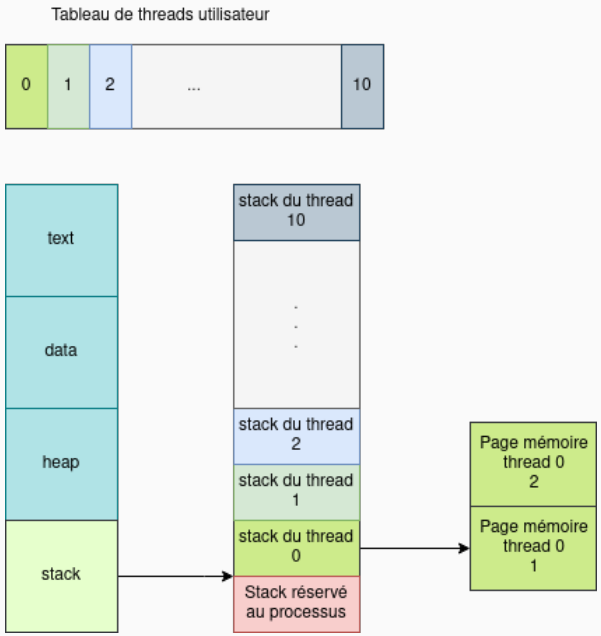

Figure 1 : Gestion de la pile

Taille de la pile d'un processus :
$(NB\_MAX\_THREAD + 1) \times 2 \times PageSize$ octets

Emplacement de la pile d'un thread :
$adresse\_debut\_pile - (Index + 1) \times 2 \times PageSize$

On garde 2 pages en début de pile pour le processus lui-même.

Attente d'un thread

Problème

Méthode UserThreadJoin() propose d'attendre la terminaison d'un thread spécifique

Solution choisie

Utilisation d'un sémaphore interne à chaque thread auquel les autres peuvent s'abonner

Attente d'un thread

Un thread possède :

- Un lock
- Un compteur dénotant le nombre de threads en attente sur ce thread-ci.

Attente d'un thread

En appelant la fonction UserThreadJoin(), le thread courrant fera prendre au thread cible prendre le verrou pour le mettre en attente (en incrémentant bien sûr le compteur des threads en attente).

Terminaison d'un thread

Dans notre implémentation :

- Terminer un thread revient également à terminer tous les threads en attentes sur celui-ci.
- Le thread doit se terminer sans que l'utilisateur n'ai à appeller la fonction UserThreadExit().

Terminaison d'un thread

Pour terminer un thread :

- Chaque thread en attente dans le thread courant va lacher le verrou.
- On supprime le thread courant du tableau des threads et on appelle Finish().

Terminaison d'un thread

Pour terminer un thread sans appelle explicite de Exit() :

- Ajout d'un attribut pour y mettre l'adresse de UserThreadExit() dans la structure contenant les arguments de la fonction à exécuter par le thread.
- On empile dans le registre r6 l'appel système de UserThreadExit (lors de la création du thread dans Start.S)
- On écrit finalement dans RetAddrReg l'adresse de UserThreadExit() pour qu'elle soit exécutée.

Gestion de la pagination

Problème

Comment profiter de la pagination ?

Solution choisie

Utilisation de la méthode reatAtVirtual() permet l'encapsulation

Problème

Comment maintenir, gérer, compter et synchroniser les processus en cours d'execution ?

Solution choisie

Utilisation d'un tableau pour stocker une référence vers les processus créés.

La création d'un processus : ForkExec(char* executable)

Problème

Comment créer un processus ainsi que son espace d'adressage ?

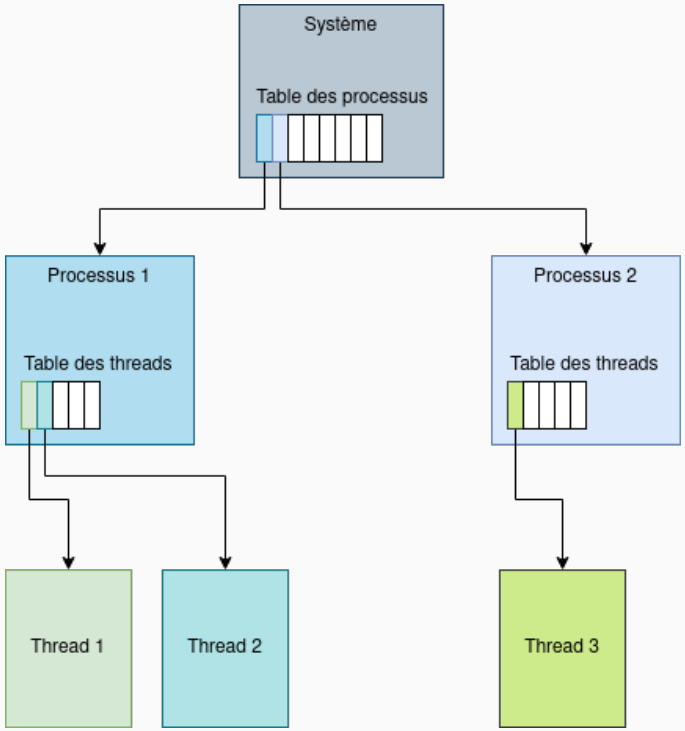

Figure 2 : Le tableau de processus

Synchronisation de la terminaison des processus

Problème

Comment terminer un processus sans interrompre l'exécution des potentiels autres processus en cours d'exécution ?

Solution choisie

Utilisation de notre tableau de processus pour connaitre les threads encore en cours d'exécution.

Notre gestion des processus

Inconvénient

- Le tableau de processus n'était pas indispensable
- Une redondance des donées peu couteuse

Avantage

- Une gestion claire et compréhensible
- Permet de futurs évolutions

Hiérarchie de dossiers

Question

Quel statut pour le répertoire racine ?

Choix

Un répertoire comme un autre sauf que son répertoire parent équivaut à lui-même.

Ouvrir plusieurs fichiers

Problème

Comment ouvrir plusieurs fichiers pour le système ? pour un thread ?

Configuration choisie

Le système connait tous les fichiers ouverts. Les threads ne connaissent que ceux qu'ils utilisent.

Ajout des verrous et conditions

Problèmes

Comment implémenter des verrous ?

Comment implémenter des Conditions ?

Comment mettre en oeuvre des attentes limitées ?

Solutions choisies

Verrous : Utiliser un sémaphore limité à un token.

Conditions : Utiliser une liste de threads bloqués.

Attentes limités : Utiliser les interruptions du Timer et ajouter une nouvelle liste de threads endormis dans l'ordonnanceur.

Ajout des verrous et conditions

Avantages

Verrous : Pratiquement déjà implémenté.

Attente : Possibilité de compter et garder trace des threads bloqués sur l'objet.

Attente limité : Une estimation relativement précise du temps écoulé pour le réveil d'un thread.

Inconvénients

Verrous : Pas l'implémentation la plus simple ou la plus efficace.

Attentes limités : Beaucoup de difficulté pour gérer l'effacement des threads dans les deux listes à la fois en fonction de l'origine du réveil (timer ou Signal).

Évolutions possibles

Lecture partagée

Optimisation

La lecture sur fichier n'a pas besoin d'être exclusive

Idée

Utiliser la table thread des fichiers ouverts ainsi qu'un boolean (isRead) et compter les lecteurs.

Envoi de messages de longueur variable en réseau

Optimisation

Non testé dû à un manque de temps.

Idée

Implémentation d'un système similaire à TCP/IP.

Limites de H-NachOS

Limites et erreurs

Limites

ForkExec(char* filename) ne marche pas avec le système de fichier.

Création de l'espace d'adressage d'un processus dans une fonction "launcher".

Solutions

Tests plus approfondis pour découvrir l'origine du problème.

Déplacer la création de l'espace d'adressage dans la fonction qui créé le processus, puis tester.

Conclusion

Au cours de ce projet nous avons pu :

- apprendre et comprendre le fonctionnement interne d'un système d'exploitation.
- travailler en équipe à la réalisation d'un projet logiciel.
- approfondir nos connaissances des outils de développement.

Voici venue l’heure de la démonstration tant attendue.

Questions?