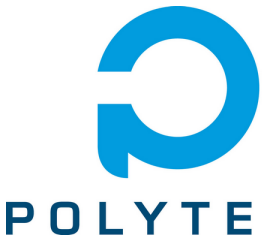

Polytech' Grenoble

RICM 4ème année

NachOS
Etape 3: Multithreading

Étudiants:
Elizabeth Paz
Salem Harrache

Professeur:
Vania Marangozova

février 2012

1 Mise en place des threads *utilisateurs*

Avant de nous interesser au Multithreading, nous allons tous d'abord nous interesser au fonctionnement des threads sous NachOS.

1.1 Fonctionnement des threads sous NachOS

Un thread NachOS comporte une structure de données (ContextBlock) contenant :

status :	JUST_CREATED, RUNNING, READY, BLOCKED
stack :	L'emplacement (bas) de la pile
stackTop :	Sommet de la pile
machineState :	Le contenu de l'ensemble de registres
userRegisters :	Le contenu de chaque registre (Thread user)
space :	L'espace d'adressage (Thread user)

Un thread est tout d'abord créé avec la méthode Thread(). Elle initialise la pile et affecte JUST_CREATED à l'attribut status. Cet état temporaire indique que le thread est créé mais n'est pas encore prêt (READY) à utiliser le processeur. C'est l'appel Fork (différent du Fork Posix) qui va rendre ce thread exécutable. Concrètement, Fork prend en argument un pointeur vers la fonction à exécuter ainsi que son argument. Puis, elle initialise l'ensemble du context block

[...je comprends pas trop ici]

La fonction *saveUserState* : Sert à sauvegarder le contexte d'exécution du programme lors d'une commutation. Les informations telles que l'état des registres ou encore le pointeur de pile sont sauvegardées dans le contextBlock. Ce contexte sera restauré lorsque le Thread est élu avec la méthode *restoreUserState*.

1.2 Description de l'instalation d'un programme utilisateur dans l'environnement NachOS

Le lancement pas à pas permet de mettre en evidence plusieurs étapes lors du lancement d'un programme dans l'environnement Nachos.

```
Initializing address space, num pages 12, size 1536
Initializing code segment, at 0x0, size 384
Initializing data segment, at 0x180, size 32
Initializing stack register to 1520
Starting thread "main" at time 10
```

1.2.1 Création de l'espace d'adressage

La premiere étape est la construction de l'espace d'adressage dans *::AddrSpace*. Le fichier exécutable est stocké dans un objet Noff. Ce dernier contient le text, les données et les variables non initialisées. La taille total de l'espace d'adressage est la somme de ces trois zones au quelle on ajoute la taille de pile utilisateur qui est de 4Ko (1024 * 4).

```
#define UserStackSize       1024
```

Avant de stocker cet espace d'adressage il faut connaitre le nombre de pages necessaires. Le système calcule le nombre de pages puis les initialise pour avoir la correspondance entres adresses virtelles et adresses physiques. Une fois les pages correctement initialisées, on copie les differentes zones en mémoires et ferme le fichier exécutable.

1.2.2 Initialisation des registres

Dans *::InitRegisters*

```
    for (i = 0; i < NumTotalRegs; i++)
      machine->WriteRegister (i, 0);
    machine->WriteRegister (PCReg, 0);
    machine->WriteRegister (NextPCReg, 4);
    machine->WriteRegister (StackReg, numPages * PageSize - 16)
```

Comme on peut le voir, tous les registres sont mis à zéro. Trois registres sont ensuite initialisés : PCReg qui pointe sur la première instruction du programme (0x0). MIPS utilise également un pointeur sur l'instruction N+1, soit à l'adresse 0x4. Enfin le pointeur de pile (StackReg) qui pointe vers la fin de l'espace d'adressage.

[...Pourquoi -16 ?...]

1.2.3 Lancement du programme utilisateur

L'exécution du programme est effectuée par la méthode *::Run*. Celle ci parcours les inscrutions en incrementant l'horloge à l'aide de la fonction OneTick.

1.3 Appel système de création des threads utilisateurs

2 Plusieurs threads par processus

3 Terminaison automatique