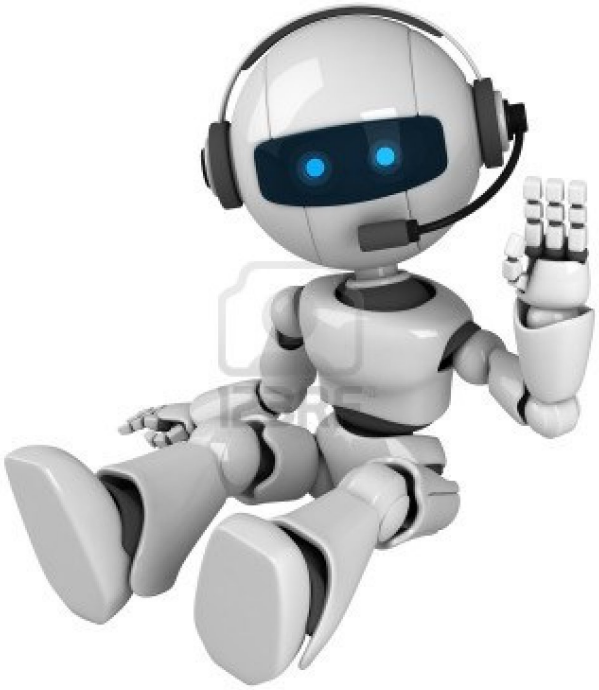
# NachOS : Multithreading
# Année 2013-2014

par Jerôme Barbier, Augustin Husson et David Levayer

26 mars 2014



Rapport généré avec LaTeX

## Table des matières

# 1 Mise en place des threads utilisateurs

On va tâcher de permettre à l'utilisateur de créer des threads. Pour cela on met en place les appels systèmes suivants :

- int UserThreadCreate(void f(void *arg), void *arg) : cette fonction permettra de créer un thread et d'exécuter la fonction f dans ce thread Retourne -1 en cas d'erreur de création de thread.
- void UserThreadExit() : Cette procédure permettra de détruire le thread qu'on a créé précédement

A l'appel système UserThreadCreate, on récupère classiquement les paramètres dans les registres 4 et 5, on appel ensuite la fonction *do_UserThreadCreate*. Cette fonction est disponible dans *userthread.cc*. On va maintenant expliciter ce que contient ce fichier.

## 1.1 Fichier userthread.cc

Dans cette partie ce fichier comporte 3 fonctions :

- extern int do_UserThreadCreate(int f, int arg) : cette fonction est appelée par l'appel système *UserThreadCreate*
- static void StartUserThread(int f) : cette procédure est appelée par la fonction mère do_UserThreadCreate
- void do_UserThreadExit() cette procédure est appelée par l'appel système *UserThreadExit*

### do_UserThreadCreate

Comme on l'a dit, cette fonction est appelée après un appel système. Dans un premier temps, elle va créer un thread :

```
Thread* newThread = new Thread("threadUser");
```

Afin de pouvoir lui allouer le même espace d'adressage que le processus père, il va falloir faire un fork du programme principal. Le problème est que la méthode *Fork* de la classe *Thread*, ne permet pas de passer en paramètre les arguments de la fonction f. (On rappel que f est la fonction que souhaite exécuter l'utilisateur dans un thread). C'est pourquoi on met en place la structure suivante :

```
struct Serialisation{
  int function; // adresse du pointeur de fonction
  int arg; // adresse du pointeur des arguments
};
```

Cette structure est mise dans le fichier *userthread.cc* afin qu'elle reste locale au thread utilisateur.

On crée donc cette structure et on l'initialise avec l'adresse de la fonction *f*, et avec l'adresse des arguments *arg*.

```
Serialisation* save = new Serialisation;
save->function = f;
save->arg = arg;
```

On finit par faire l'appel à la méthode *Fork*

```
newThread->Fork(StartUserThread,(int)save);
```

Le fork fait un appel à la procédure StartUserThread. On va donc maintenant parler de cette procédure

**StartUserThread**

Cette procédure permet d'initialiser correctement le thread qu'on a créé dans la fonction précédente. C'est à dire entre autre lui allouer une pile différente de celle du processus père.

Dans un premier temps, on s'attache à récupérer notre fonction *f* et ses paramètres *args*. Ce qui se fait avec la ligne ci-dessous :

```
Serialisation* restor = (Serialisation*) f;
```

On initialise ensuite les registres du thread, ce qui se fait classiquement en mettant des 0 partouts :

```
for(int i=0;i<NumTotalRegs;i++)
 {
        machine->WriteRegister(i,0);
 }
```

On va maintenant positionner le pc sur l'adresse de la fonction *f*. Ceci est fait afin qu'au lancement du thread, la première ligne exécutée soit celle correspondante à *f*.

```
machine->WriteRegister(PCReg,restor->function);
```

Bien évidemment pour que f s'exécute correctement, il est nécessaire de lui fournir ses paramètres qu'elle devra retrouver dans le registre 4.

```
machine->WriteRegister(4,restor->arg);
```

la machine nachOS possède un pointeur *pcNext* qui est placé à l'instruction qui suit celle pointée par *pc*. La prochaine instruction se trouve classiquement 4 octets après pc. D'où :

```
machine->WriteRegister(NextPCReg,restor->function+4);
```

Il reste maintenant à positionner le pointeur de pile :

```
machine->WriteRegister(StackReg,currentThread->space->BeginPointStack());
```

Bien évidemment on expliquera par la suite (i.e dans le fichier addrspace.cc) comment fonctionne la méthode *BeginPointStack*

Maintenant que tous les registres ont été correctement initialisés, il est temps de lancer le programme!!

```
machine->Run();
```

## 1.2 Fichier addrspace.cc

Afin de positionner le pointeur de pile du nouveau thread, il a fallu modifier le fichier *addrspace.cc et addrspace.h* qui permettent de gérer l'espace mémoir des threads/processus. La principal modification est l'ajout d'un attribut de type *BitMap*

**L'attribut BitMap**