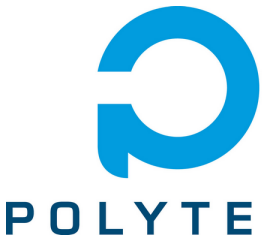

Polytech' Grenoble

RICM 4ème année

NachOS
Etape 4: Mémoire virtuelle

Étudiants:
Elizabeth Paz
Salem Harrache

Enseignant:
Vania Marangozova

Février 2012

1 Git

Pour voir les différences entre l'étape 3 et l'étape 4 vous pouvez lancer un diff avec le tag step3 :

```
git diff step3
```

ou alors directement avec le commit 78799c....

```
git diff 78799ce7a7b788d0f3b501f0d85bab2f21c7190b
```

2 Test de l'étape 4

Dans notre test, on va créer trois processus, qui lancent chacun deux threads qui vont ecrire leurs noms trois fois.

```
Lancement du Test 1 :)
----------------------
./build-origin/nachos-userprog -rs 1 -x ./build/forkprocess
Debut du pere
...
Fin du pere
Debut du fils 0 : lancement des deux threads a et z
Debut du fils 1 : lancement des deux threads b et y
Debut du fils 2 : lancement des deux threads c et x
azbyxczaybxcza
Fin du thread main du fils 0
yb
Fin du thread main du fils 1
xc
Fin du thread main du fils 2
Machine halting!

Ticks: total 652352, idle 9380, system 75070, user 567902
Disk I/O: reads 0, writes 0
Console I/O: reads 0, writes 300
Paging: faults 0
Network I/O: packets received 0, sent 0

Cleaning up...
```

3 Allocation des cadres de pages

Nous avons légèrement modifié la class *frameprovider*. En effet la fonction, celle ci alloue un certain nombre de cadres de façon atomique. Un processus à besoin de N cadres ou ne lance pas.

Listing 1: *code/userprog/frameprovider.cc*

```
int * FrameProvider::GetEmptyFrames(int n) {
    RandomInit(0);
    this->semFrameBitMap->P();
    int * frames = NULL;
    if (n <= this->bitmap->NumClear()) {
        frames = new int[n];
        for(int i=0; i<n; i++) {
            int frame = Random()%NumPhysPages;
            // Recherche d'une page libre
            while(this->bitmap->Test(frame)) {
                frame = Random()%NumPhysPages;
            }
            this->bitmap->Mark(frame);
            bzero(&(machine->mainMemory[ PageSize * frame ]), PageSize );
            frames[i] = frame;
        }
    }
    this->semFrameBitMap->V();
    return frames;
}
```

L'utilisation dans **AddrSpace** est la suivante :

Listing 2: *code/userprog/addrspace.cc*

```
int * frames = frameprovider->GetEmptyFrames((int) numPages);
if (frames == NULL) {
    DEBUG ('p', "Pas suffisamment de memoire !\n");
    return;
}

// first, set up the translation
pageTable = new TranslationEntry[numPages];
for (i = 0; i < numPages; i++) {
    pageTable[i].virtualPage = i;
    // for now, virtual page # = phys page #
    pageTable[i].physicalPage = frames[i];
[...]

delete frames;
```

l'instruction return dans le constructeur avorte la construction de l'objet AddrSpace, dans ***do_ForkExec*** il faut s'assurer que l'objet est correctement intialisé. On s'assure également que le destructeur libère les cadres un par un.

4 Création d'un nouveau processus

La création d'un nouveau processus se déroule en plusieurs etapes :

- Création d'un nouvel espace d'adressage
- Creation d'un nouveau Thread main au quel on assoccie cette espace d'adressage
- Appel de Fork de ce nouveau thread.

5 Terminaison