L3 – Cours Système – Travaux dirigés
Techniques de débogage appliquées à Nachos

Emmanuel Agullo, Eddy Caron

Vendredi 17 février 2006

1 Options de débogage de Nachos

1.1 Messages système

Nachos permet tout d'abord d'afficher une grande quantité de messages système.

```
./nachos -d +
```

On peut voir les *ticks* de horloge interne de Nachos, les commutations entre threads, la gestion des interruptions, etc.

Il est possible de n'afficher qu'une partie de ces informations. Au lieu du `+` qui correspond à activer l'ensemble des options de débogage, on peut mettre :

`t` pour ce qui a trait aux threads système Nachos,

`a` pour ce qui a trait à la mémoire MIPS,

`m` pour ce qui a trait à la machine Nachos,

`t` pour ce qui a trait à la gestion des threads,

`i` pour ce qui a trait aux interruptions,

`n` pour ce qui a trait au réseau Nachos,

`f` pour ce qui a trait au filesystem Nachos,

`d` pour ce qui a trait aux disques Nachos.

Certaines parties de Nachos (disques, . . .) ne sont pas manipulées pour l'instant. Le flag correspondant n'affichera alors aucun message supplémentaire. Mais on peut ajouter vos messages pour ces flags, voire même ajouter de nouveaux flags.

1.2 Exécution cycle par cycle en espace utilisateur

L'option `-s` permet d'exécuter pas à pas un programme utilisateur en stoppant l'exécution après chaque instruction du programme, et en affichant l'ensemble des registres de la machine virtuelle.

2 Débogage dans le noyau avec Gdb

2.1 Exécution pas à pas dans le noyau

On peut tracer Nachos au plus bas niveau, dans le noyau, en utilisant `gdb`. Par défaut, on compile avec l'option `-g` (voir les `Makefile`). Ensuite, si on veut exécuter la fonction `main` pas à pas, on place un *breakpoint* comme suit.

```
gdb nachos
[...]
(gdb) break main
Breakpoint 1 at 0x804d6fa: file main.cc, line 84.
(gdb) run
Starting program: nachos
[...]
Breakpoint 1, main (argc=1, argv=0x8046db0) at main.cc:84
84          DEBUG('t', "Entering main");
(gdb)
```

Ensuite, on progresse avec les commandes `s` (atomic *step*), `n` (*next* instruction in the current fonction), `c` (*continue*), etc.

On peut également placer un *breakpoint* à un endroit précis d'une fonction en précisant le fichier et la ligne.

```
(gdb) break ../userprog/addrspace.cc:90
```

2.2 Trouver ce qui plante

Enfin lors d'un plantage, `gdb` permet de trouver facilement ce qui a causé l'erreur. Pour cela, on peut lancer l'exécution normale de nachos dans `gdb`, en précisant les arguments de lancement lors de l'appel à `run`.

```
gdb nachos
[...]
(gdb) run -x ../test/halt
```

Une fois que l'erreur s'est produite, elle s'affiche relativement explicitement. On peut alors connaître l'ensemble des fonctions qui ont été successivement appelées sur la pile grâce à `where`, ce qui fournit par ailleurs le fichier et la ligne ou se trouve l'appel, et les arguments utilisés.

```
(gdb) run -x test/fork
Starting program: nachos -x test/fork

Program received signal SIGSEGV, Segmentation fault.
0x0804dbda in do_UserFork () at addrspace.cc:490
490       *(char*)p = 0;
(gdb) where
#0  0x0804dbda in do_UserFork () at addrspace.cc:490
#1  0x0804f91b in ExceptionHandler (which=SyscallException)
```

```
    at exception.cc:258
#2  0x08050ea1 in Machine::RaiseException (this=0x805add0,
    which=SyscallException, badVAddr=0)
    at ../machine/machine.cc:109
#3  0x08052b5d in Machine::OneInstruction (this=0x805add0,
    instr=0x80644b0) at ../machine/mipssim.cc:587
#4  0x08051390 in Machine::Run (this=0x805add0)
    at ../machine/mipssim.cc:48
#5  0x0804dad8 in StartUserThread (dummy=0)
    at addrspace.cc:476
#6  0x080535d4 in ThreadRoot ()
#7  0x00000000 in ?? ()
```

Ensuite on peut afficher le contenu des variables globales ou locales en utilisant `print` (ou `p`). On peut également changer remonter ou descendre dans la pile pour étudier les variables différentes fonctions empilées.

```
(gdb) p p
$1 = 0
(gdb) up
#1  0x0804f91b in ExceptionHandler (which=SyscallException)
    at exception.cc:258
258             int res = do_UserFork();
(gdb) up
#2  0x08050ea1 in Machine::RaiseException (this=0x805add0,
    which=SyscallException, badVAddr=0)
    at ../machine/machine.cc:109
109         ExceptionHandler(which);
(gdb) p this
$2 = (Machine *) 0x805add0
```

À noter que les temporaires `$1`, `$2`, ... peuvent être réutilisés ensuite comme raccourcis.

```
(gdb) p $2
$3 = (Machine *) 0x805add0
(gdb) p $2->pageTableSize
$4 = 32
```

`gdb` fournit beaucoup d'autres fonctionnalités intéressantes...