NachOS - Etape 1

1 Commandes

1. cd nachos/code
2. make clean
3. make

2 Test

1. ./build/nachos-userprog -x build/halt Machine halting!

   Ticks: total 19, idle 0, system 10, user 9 Disk I/O: reads 0, writes 0 Console I/O: reads 0, writes 0 Paging: faults 0 Network I/O: packets received 0, sent 0

   Cleaning up...

   $\Longrightarrow$ Initialisation du système nachos et exécution du programme utilisateur MIPS: halt (il demande l'arrêt du système).

2. ./build/nachos-userprog

   No threads ready or runnable, and no pending interrupts. Assuming the program completed. Machine halting!

   Ticks: total 10, idle 0, system 10, user 0 Disk I/O: reads 0, writes 0 Console I/O: reads 0, writes 0 Paging: faults 0 Network I/O: packets received 0, sent 0

   Cleaning up...

3 Lire le code

3.1 Initialisation du système

Fonction main du fichier threads/main.cc: point d'entrée du programme.

1. Observation de la transformation d'un exécutable en un système d'exploitation sur lequel il existe un unique thread noyau.
2. Distinguer ce qui correspond a une exécution réelle en mode système de ce qui correspond a une simulation de matériel.

Allocation du simulateur Dans la fonction main.cc on trouve l'intialisation de la structure de données: $(void)Initialize(argc, argv)$;. C'est l'appel a la fonction Initialize du fichier system.cc dans lequel il y a l'initialisation du nachos et la routine de nettoyage. La fonction Initialize fait l'initialisation globale de la structure des données en faisant attention aux flags. En général, cette partie est fait pour tout appel a la fonction:

```
  DebugInit (debugArgs);            // initialize DEBUG messages
    stats = new Statistics ();      // collect statistics
    interrupt = new Interrupt;      // start up interrupt handling
    scheduler = new Scheduler ();       // initialize the ready queue
    if (randomYield)                // start the timer (if needed)
        timer = new Timer (TimerInterruptHandler, 0, randomYield);

    threadToBeDestroyed = NULL;

    // We didn't explicitly allocate the current thread we are running in.
    // But if it ever tries to give up the CPU, we better have a Thread
    // object to save its state.
    currentThread = new Thread ("main");
    currentThread->setStatus (RUNNING);

    interrupt->Enable ();
    CallOnUserAbort (Cleanup);  // if user hits ctl-C
```

Toutes les classes initialisés et alloués sont toutes les structures des données utilisés par nachos: (apparaissant au début du fichier)

```
 Thread *currentThread;           // the thread we are running now
Thread *threadToBeDestroyed;     // the thread that just finished
Scheduler *scheduler;            // the ready list
Interrupt *interrupt;            // interrupt status
Statistics *stats;               // performance metrics
Timer *timer;                    // the hardware timer device,
                                        // for invoking context switches
```

Le premier thread noyau Toute création d'un thread fait appel a la fonction du fichier Thread.cc:

```
 Thread::Thread (const char *threadName)
{
    name = threadName;
    stackTop = NULL;
    stack = NULL;
    status = JUST_CREATED;
#ifdef USER_PROGRAM
    space = NULL;
#endif
}
```

Ici, un bloc de contrôle va lui être initialisé, on va suavegardé son nom, le début du bloc, son statut et si c'est un thread du programme utilisateur son space (au début sera null). Le thread main ne possède pas d'espace d'adressage car il est un thread noyau. L'idée de la création de ce thread main est que si on laisse tomber le CPU, c'est mieux d'avoir un object thread dans lequel on puisse sauver son état. Je crois que tous les autres threads noyaux peuvent être crées

de ma même façon, mais par contre en appelant la fonction fork, qui permet l'exécution du thread.

3.2 Exécution d'un programme utilisateur

La mémoire MIPS est la mémoire simulée associée au processeur MIPS simulé.

```
#ifdef USER_PROGRAM
    machine = new Machine (debugUserProg);       // this must come first
#endif
```

C'est la mémoire du programme utilisateur et les registres. Pour plus d'information on regarde les fichiers machine.cc et machine.h. Lors de l'initialization de machine, un tableau de registre est initilisé à 0, et de même pour la mémoire. La création de machine permet l'initialisation de la simulation du hardware pour permettre l''exécution des programmes utilisateurs.

StartProcess() est appelée avec le nom du fichier donné en paramètre. Voici le code, qui permet d'exécuter le pgm utilisateur, ouvrir l'exécutable, le charger en mémoire et puis "sauter" vers celui-ci:

```
void
StartProcess (char *filename)
{
    OpenFile *executable = fileSystem->Open (filename);
    AddrSpace *space;

    if (executable == NULL)
      {
          printf ("Unable to open file %s\n", filename);
          return;
      }
    space = new AddrSpace (executable);
    currentThread->space = space;

    delete executable;          // close file

    space->InitRegisters ();    // set the initial register values
    space->RestoreState ();     // load page table register

    machine->Run ();            // jump to the user progam
    ASSERT (FALSE);             // machine->Run never returns;
    // the address space exits
    // by doing the syscall "exit"
}
```

On observe le code de la fonction Run qui se trouve dans le fichier Machine/mipssim.cc

L'appel système Halt On observe le fichier Machine/interrupt.cc pour la fonction Halt(). Dans le fichier mipssim.cc on peut observer la fonction OneInstruction qui code un appel système $OP_SYSCALL$:

```
case OP_SYSCALL:
        RaiseException(SyscallException, 0);
        return;
```

4 Utilisation du système NachOS