Présentation de Nachos

Vincent Danjean, Guillaume Huard
Vania Marangozova, Jean-François Méhaut

http://www-id.imag.fr/Laboratoire/Membres/Marangozova-Martin_Vania/teaching/NACHOS/NachosInformations.html

Laboratoire ID-IMAG

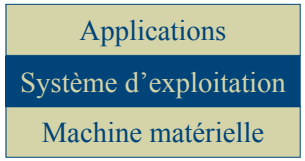
Année 2005/2006

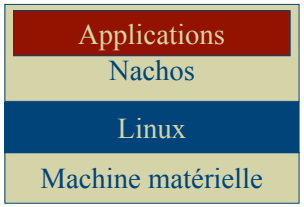
Objectifs du système Nachos

- Comprendre le fonctionnement interne des systèmes d'exploitation (OS)
- Manipuler un grand logiciel
- Programmer les mécanismes de base
  - Gestion de processus (multi-threading, multi-programmation)
  - Gestion d'entrées/sorties (synchronisation des pilotes, ajout d'appel système)
  - Gestion mémoire
  - Gestion fichiers

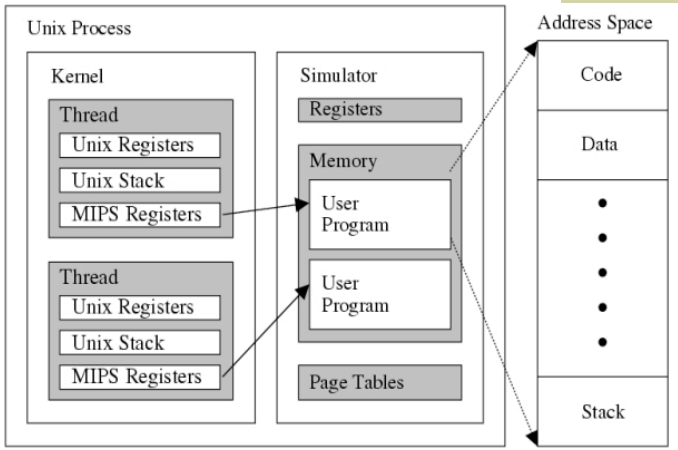
Environnement de travail

- Le système d'exploitation Nachos
  - Système pédagogique, développé à Berkeley (T. Anderson)
  - Minimaliste => simple
  - Simulé => déterministe, simple à modifier, à tester
    - Processeur MIPS (cross-compilateur)





Architecture générale Nachos



Composants internes de Nachos

- *Noyau*
  - Threads, primitives de synchro, ordonnnacement,…
- *Pilotes* (*drivers*) des périphériques émulés
  - Console, disque, réseau,…
- *Mémoire virtuelle*
  - Table des pages, adresse physique, virtuelle,…
- *Gestion de fichiers*
  - Tables des fichiers, gestion du disque physique,…

Ordonnancement

- Politique FIFO (First In First Out)
- Pas de priorité
- Pas de temps-partagé (time-slicing, quantum)
- Changement de contexte
  - Blocage sur une primitive de synchronisation
  - Interruption
- Synchronisation interne
  - Sémaphore, verrous, conditions

Étapes du projet

- Étape 1 : Premiers pas dans Nachos (CSE)
- Étape 2 : Entrées/Sorties console (CSE)
- Étape 3 : Multi-threading (Projet)
- Étape 4 : Gestion mémoire (Projet)
- Etape 5 : Gestion Fichiers (Projet)
- Etape 6 : Réseau (Projet)

Étape 1 : Premiers pas

- Installation, mise en place de CVS, test
- Lecture guidée du code source
  - C++
- Traçage de l'exécution, débogage, gdb
  - Démarrage simulateur, structures de base
  - Exécution d'un programme utilisateur
  - Ordonnancement processus dans Nachos

Étape 2 : Entrées/sorties console

- Pilote d'entrées/sorties minimaliste
  - Écrire un caractère à l'écran
    - Écrire avant d'être sûr que l'écriture soit terminée?
  - Lire un caractère depuis le clavier
    - Lire avant d'être averti de la disponibilité d'un caractère?
  - ➔ Synchronisation des opérations
  - Écrire/Lire des chaînes de caractères

Étape 3 : Multi-threading

- Programmes utilisateurs à plusieurs threads
  - Compréhension gestion threads Nachos
  - Appels système de création/terminaison de threads
  - Retour sur pilote console pour exclusion mutuelle
  - Terminaison automatique threads

Étape 4 : Gestion mémoire

- Pagination de la mémoire
  - Compréhension mémoire virtuelle Nachos
  - Mise en place d'une mémoire paginée
    - Stratégies d'allocation des pages mémoire
  - Mise en place de la multi-programmation
  - Implémentation d'un shell
  - Implémentation de primitives malloc/free

Etape 5: Gestion de fichiers

- Augmenter la taille maximale des fichiers
  - Blocs d'indirection, structure de type inode
- Avoir des fichiers de taille variable
  - Allocation dynamique de blocs et secteurs
- Gérer la structure de répertoire
  - Structure répertoire, hiérarchie de répertoire,…
- Optimiser les accès disque
  - Ordonnancer les demandes

Déroulement du travail

- Nachos disponible sur mandelbrot
  - Possibilité d'installation sur vos PCs, besoin de compilation d'un cross-compilateur
- Documents décrivant les différents étapes
  - Questions pour compréhension, définition d'actions à programmer (bonus)
  - Accès à

    http://www-id.imag.fr/Laboratoire/Membres/Marangozova-Martin_Vania/teaching/NACHOS/NachosInformations.html
- Travail en binôme

Introduction à C++

Historique du langage C++

- Créateur du langage
  - Bjarne Stroustroup (AT&T Laboratories)
- Historique
  - 1980 : C with classes
  - 1983 : C++
- Objectifs
  - Garder les avantages du C
    - Efficacité du code généré + parc logiciel énorme (UNIX)
  - Bénéficier d'une meilleure qualité de programmation
    - C++ : Extension de C aux objets

Généralités

- C++ : langage à objets à base de classes :
  - Encapsulation partielle
  - Héritage multiple
  - Généricité
  - Etc.
- Rappel: Classe = données-structures + traitements
  - Données propres à chaque objet (instance)
  - Traitements communs à toutes les instances d'une classe
- Remarque
  - Les classes n'ont d'existence qu'à la compilation

Spécification d'une classe

- Exemple : une classe Compteur
  - Fichier **Compteur.h**

```
Class Compteur {
  public :
      void raz () ;
      void inc () ;
      int value () ;
  private :
      int val ;
} ;
```

Implantation d'une classe

- Compteur.cc

```
void Compteur:raz ()
{
  val = 0 ;
}
void Compteur:inc ()
{
  val++ ;
}
int value ()
{
  return val ;
}
```

Instanciation

- Déclaration statique d'une instance de Compteur

```
{
 Compteur c ;

 c.raz () ;
 c.inc () ; c.inc () ;
 printf (« %d\n », c.value()) ;
 // Affiche 2
}
```

- Remarque
  - Erreur si on accède directement à **c.val**

Instanciation (2)

- Version dynamique

```
{
 Compteur *c ;
 c = New Compteur ;
 c->raz () ;
 c->inc () ; c->inc () ;
 printf (« %d\n », c->value()) ;
 // Affiche 2
 delete c ;
}
```

- Opérateurs `new` et `delete`
  - A utiliser en remplacement de `malloc` et `free`
  - Gestion des objets complexes

Gestion mémoire dynamique

- En C, on écrivait

```
int * zone ;

 zone = (int *) malloc (5 * sizeof(int)) ;
 // Allocation d'un tableau de 5 entiers

 free (zone) ;
 // Liberation espace occupé par zone
```

- En C++, on écrit :

```
int * zone ;
 zone = new int [5] ;

 delete [] zone ;
```

Pseudo-variable `this`

- Permet d'obtenir un pointeur sur l'objet courant
- Utilisable uniquement dans une méthode d'objet

```
class C {
  public :
     void positionnerx (int x)
  private
     int x ;
} ;
```

```
void C:positionnerx (int x)
{
 this->x = x;
}
```

Initialisation automatique: les constructeurs

- Initialisation automatique du compteur

```
class Compteur {
  public :
      Compteur () ;
      void raz () ;

        ...

Compteur:Compteur()
{
 raz () ;
}
```

Constructeurs paramétrés

- Initialisation automatique du compteur

```
class Compteur {
  public :
      Compteur (int v) {val = v ; }

      ...

   private :
     int val ;
} ;
Compteur c1 (10) ; // ok
Compteur c2 ; // interdit

Compteur *c3 = new Compteur (10) ; // ok
Compteur *c4 = new Compteur ; // interdit
```

Destructeurs

- Libération automatique des objets
- Exécuté
  - delete
  - Sortie du bloc englobant

```
class String {
  public :
      String (char *s) {… new  …  }
      ~String () ;
      …
   private :
     char *str ;
} ;
```

Destructeurs (2)

```
String:~String ()
{
 delete [] str ;
}
```

- Invocation explicite

```
String *ptr ;

  ptr = new String (« coucou ») ;
  …
  delete ptr ;
  // liberation de ptr->str, puis ptr
```

Destructeurs (3)

- Invocation implicite

```
{
 String chaine (« coucou ») ;

  …
  …
}
  // liberation de chaine.str, puis chaine
```

Emulation du matériel

file://nachos/code/machine

Emulateur machine

- code/machine/
  - Emulateur MIPS, programme utilisateur
  - Ne PAS changer les sources (.c) de ce répertoire
  - Autorisé de changer quelques valeurs constantes `NumPhysPages` (dans `machine.h`)
  - Comprendre l'émulateur vous aidera dans la mise au point et la conception des solutions du projet

Machine émulée

- Processeur MIPS
  - Environ 40 registres
- Objet `machine`, classe `Machine`
  - Exportation de méthodes
  - Inspection de l'état
  - Méthodes
    - Exécution d'un programme
    - Accès aux registres
    - Accès à la mémoire

La classe Machine

- Description de la machine physique
  - Contient le code déclenchant l'interprètre (méthode **Run ()**)
  - Communications Kernel/User s'effectuent via mémoire + registres
- Mémoire physique
  - Variable **mainMemory**
  - Par défaut, 32 pages de 128 octets
- Registres
  - Manipulés via **ReadRegister ()** et **WriteRegister ()**
- Memory Management Unit
  - Table de converion (variable **pageTable**) adresse physique/virtuelle

Exemples de méthodes

- **`void Run ()`**
  - Exécution du programme utilisateur chargé en mémoire
- **`int ReadRegister (int num)`**
  - Retourne le contenu du registre `num`
- **`void WriteRegiter (int num, int value)`**
  - Ecriture de `value` dans le regitre `num`

Constantes « Registres » prédéfinies

- **StackReg**
  - Pointeur de pile
- **PCReg**
  - Compteur Ordinal
- **RetAddrReg**
  - Adresse de retour de procédure
- **NextPCReg**
  - Pointeur vers la prochaine instruction

Gestion mémoire

- Utilisation transparente de la MMU
  - Accès mémoire via ReadMem () et WriteMem ()
  - Ces fonctions utilisent la méthode Translate () :

```
Machine::Translate (int virtAddr, int *physAddr,
                    int size, bool writing) ;
```

  - Conversion d'adresse en utilisant **pageTable**
  - Ces fonctions vérifient l'alignement et la protection

Méthode WriteMem ()

```
Bool Machine::WriteMem (int addr, int size, int value)
{
 int physicalAddress ;

 Translate (addr, &physicalAddress, size, TRUE) ;
 switch (size) {
   case 1: …
   case 2: …
   case 4:*(unsigned int *)&machine->mainMemory [physicalAddress]
                  = WordToMachine ((unsigned int) value);
   …

}
```

La classe **Interrupt**

- Objectifs
  - Faire progresser une horloge logique
    - Méthode **OneTick ()**
    - Maintenir des statistiques UserTime/KernelTime
    - Déclencher des interruptions pré-programmées
      - Méthode **Schedule ()**, appelée par les périphériques simulés

```
void Interrupt::Schedule (VoidFunctionPtr handler, int arg,
                    int when, IntType type) ;
```

    - Masquer/Autoriser les interruptions
      - **Méthode SetLevel (IntOn/IntOff)**
    - Eteindre la machine (et donc Nachos) **halt ()**

Clients de l'objet **Interrupt**

- Timer
  - Ré-armement à chaque fois d'une nouvelle interruption
- Disk
- Console
- Network
- etc…

Contrôleur d'interruption

- Prises en compte ou non des interruptions générées par le matériel
- Objet **interrupt** de la classe **Interrupt**
  - **IntStatus SetLevel (IntStatus level)**
    - Autoriser ou interdire les interruptions
    - Retourne l'état précédent vis à vis des interruptions
  - **Void Enable ()**
    - Autoriser les interruptions
  - **IntStatus getLevel ()**
    - Retourne l'état courant vis à vis des interruptions

Noyau du système

file://nachos/code/thread

Initialisation de Nachos

- Fichiers **main.cc** et system.cc
  - **main.cc** traite les arguments de la ligne de commande Nachos
    - Ex: **nachos -x test**
    - Lancement du système avec exécution du programme test
  - **System.cc** initialise les structures du noyau

Comportement de Nachos

- Il dépend de l'endroit où il est compilé
  - **`code/machine/`** : impossible
  - **`code/threads/`** : Nachos exécute un petit test interne de la multiprogrammation
  - **`code/userprog/`** : Nachos peut exécuter un programme utilisateur
  - **`code/vm/`** : Nachos peut exécuter plusieurs programmes!
  - **`code/filesys/`** : Nachos peut accéder aux fichiers d'un disque

Processus et ordonnancement

- La classe Thread (scheduler.cc, scheduler.h)
  - Gestion de processus indépendants
  - **`Fork()`**, **`Yield()`**, **`Sleep()`**, etc.
- La classe Scheduler: juste une liste de Threads prêts
  - ReadyToRun(Thread *t): Ajoute un thread à la liste
  - Thread *FindNextToRun(void): Retire le premier thread de la liste
  - Run (Thread *t): Alloue le processeur au thread passé en paramètre

Synchronisation

- La classe `Semaphore` (fichiers synch.h, synch.cc)
  - Constructeur
    - **`Semaphore::Semaphore (nom, valeur)`**
  - Opération blocante
    - **`Semaphore::P()`**
  - Opération non blocante
    - **`Semaphore::V()`**

Synchronisation (2)

- La classe Lock (fichiers synch.h, synch.cc)
  - Acquisition du verrou
    - **`void Acquire ()`**
  - Libération du verrou par le détenteur
    - **`void Release ()`**

Synchronisation (3)

- La classe Condition (fichiers synch.h, synch.cc)
  - Blocage sur condition (et relâche de verrou)
    - `void Wait (Lock *c)`
  - Réveil sur condition(et redemande de verrou)
    - `void Signal (Lock *l)`
  - Réveil des threads sur condition (et redemande de verrou)
    - `void Broadcast (Lock *l)`
- Sémantique proche des moniteurs de Hoare

Pilotes de périphériques

Console, disque

file://nachos/code/machine/

Pilote de la console

- Fichiers console.h et console.cc
- Console = matériel asynchrone
  - Une requête retourne immédiatement
- `void PutChar (char ch)`
  - Ecriture du caractère et retour immédiat
  - `writeHandler` est appelé quand l'écriture est faite
- `char Getchar ()`
  - Retourne le caractère si disponible ou EOF
  - `readHandler` est appelé quand la lecture est faite

Pilote disque

- La classe Disk (Fichiers disk.h et disk.cc)
  - Constantes physiques (modifiables…)
    - Taille du secteur 128
    - Nombre de secteurs par piste 32
    - Nombre de pistes: 32
    - Nombres de secteurs: 32 * 32
- Quelques Attributs du disque
  - Pointeur sur le disque (descripteur de fichier UNIX)
  - Handler d'interruption (fin des opérations disque)
  - Dernier secteur accédé

Pilote disque (2)

- Méthodes
  - Envoi d'une requête au disque, une seule requête à la fois, retour immédiat
    - void ReadRequest (int SectorNumber, char *data)
    - void WriteRequest (int SectorNumber, char *data)
  - Calcul de la latence disque
    - int ComputeLatency (int newsector, bool writing)
    - déplacement de la tête+délai rotationnel + transfert