

NACHOS

Lapauze Johann / Noirot Gil / Tronc Raphaël / Vey Julien
RICM2 2008/2009

Sommaire

2

- **Partie 1: Premier pas**

- Objectifs
- Implémentations
- Résultats

- **Partie 2 : Entrées/Sorties**

- Objectifs
- Implémentations
- Résultats

- **Partie 3 : Multi-Threading**

- Objectifs
- Implémentations
- Résultats

- **Partie 4: Gestion de mémoire**

- Objectifs
- Implémentations
- Résultats

- **Partie 5 : Gestion de fichiers**

- Objectifs
- Implémentations
- Résultats

- **Partie 6 : Réseau**

- Objectifs
- Implémentations
- Résultats

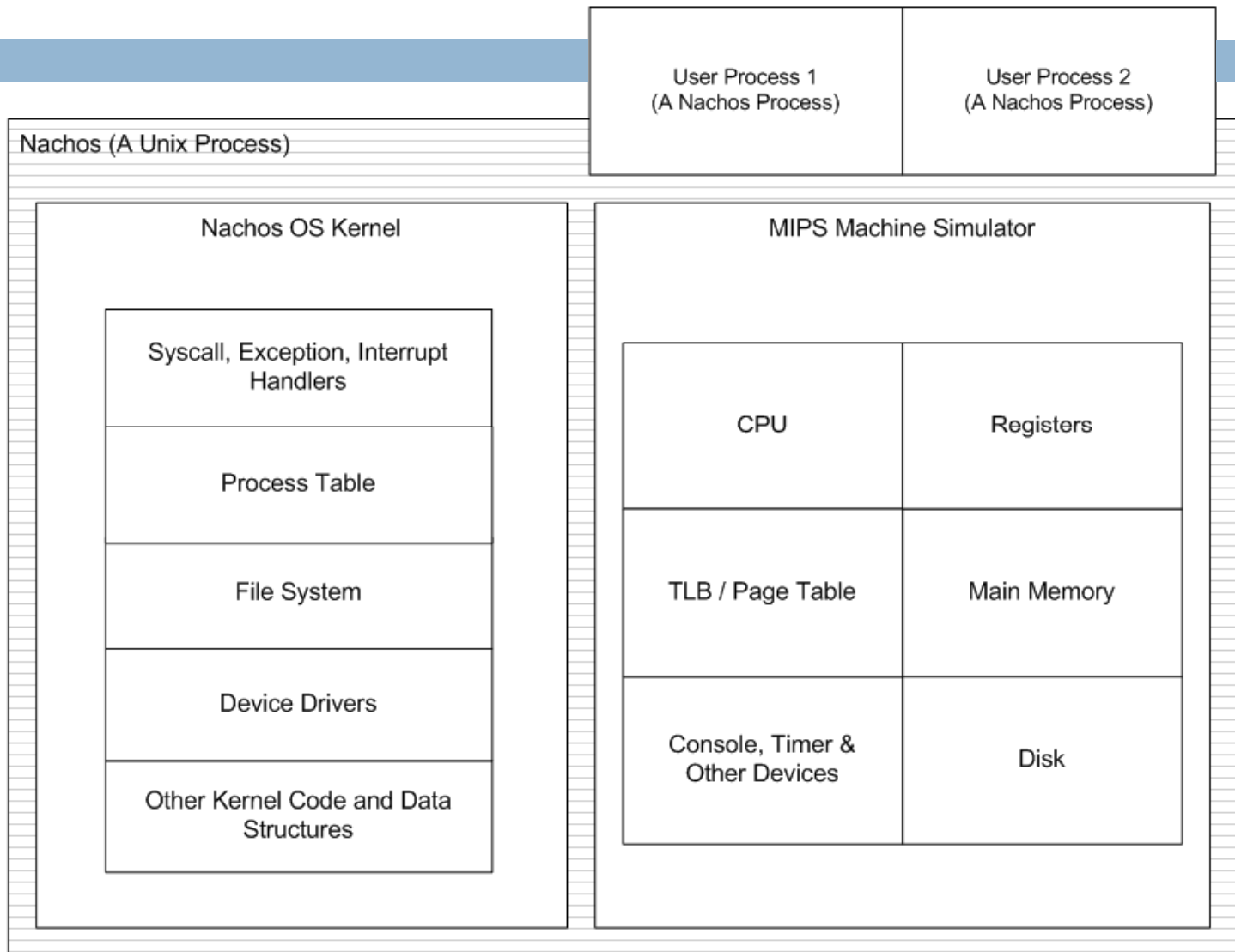
1 Premier pas

3

Objectifs

Implémentations

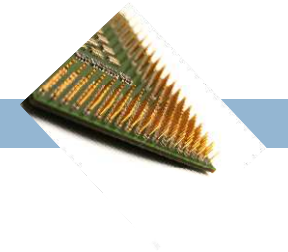
Résultats



1 Premier pas

4

Nachos ↔ O.S



Objectifs

Implémentations

Résultats

- **CPU: Registres, Horloge, UAL**
 - Registers[40], OneTick(), OneInstruction
- **Ordonanceur**
 - Scheduler, peut être lancer en FIFO ou avec préemption
- **Mémoire principale**
 - MainMemory [1024*128] ; 131ko
- **Disque dur**
 - SynchDisk 32+ 1024*128 ; 131 ko
- **Système de fichier**
 - FileSystem ; 10 entrées max dans chaque répertoire
- **IRQ**
 - Interrupt;
- **Réseau**
 - PostOffice ; 10 mailbox représentant des ports.

1 Premier pas

5

Exécution d'un programme utilisateur

Objectifs

Implémentations

Résultats

- **StartProcess(Executable)**
 - Initialisation du système
 - Programme dans la mémoire MIPS
 - (*page virtuelle, page physique*)
 - Initialisation des registres
 - Attribution d'une stack
 - Chargement du contexte
 - Lancement du programme
 - *Appel système par interruptions. OP_SYSCALL*
 - Fin par un cleanup()
 - *Delete l'ensemble des composants*
 - *Exit(0)*

2 Entrées/Sorties

6

Objectifs

Implémentations

Résultats

- **Console asynchrone**
 - Détection de fin de fichier
 - Mise en place de < 'c' >

- **Console synchrone**
 - Implémentations

- **Appels système**
 - PutChar / GetChar
 - PutString / GetString
 - PutInt / GetInt

- **Halt**
 - Appel implicite
 - Valeur de retour

2 Entrées/Sorties

7

Console asynchrone

Objectifs

Implémentations

Résultats

- **E/S gérés par Interrupt->Shedule**
 - **Création d'une console:**
 - `new Console()`
 - `interrupt->Schedule(ConsoleReadPoll, ..);`
 - `readAvail = new Semaphore ("read avail", 0);`
 - `writeDone = new Semaphore ("write done", 0);`
 - `readAvail->P ();` // wait for character to arrive
 - `ch = console->GetChar ();`
 - **Détection de fin de fichier :**
 - **Mise en place de < 'c'>**
 - `console->PutChar ('<'); writeDone->P ();`
 - `console->PutChar (ch); writeDone->P ();`
 - `console->PutChar ('>'); write Done ->P();`

2 Entrées/Sorties

8

Console synchrone

Objectifs

Implémentations

Résultats

□ SynchronGetChar()

```
verrouGet->P(); //mutex
readAvail->P ();
char ch;
ch = console->GetChar ();
verrouGet->V();
return ch;
```

□ SynchronGetString(char *s, int n)

```
verrouGet->P();
char ch = 'a';
int i = 0;
while( (ch != EOF && ch != '\0'
        && ch != '\n') && i<n ){
    readAvail->P ();
    ch = console->GetChar ();
    s[i] = ch;
    i++;
}
verrouGet->V();
```

2 Entrées/Sorties

9

Appels systèmes

Objectifs

Implémentations

Résultats

- **Appels systèmes**
 - => mode noyau
 - => exécuté par linux
 - => en Assembleur (x86)
 - Argument placés automatiquement dans les registres R4 et R5

- **PutInt, putChar et GetInt, GetChar**
 - Implémentations similaire.
 - Lecture/Ecriture (Put/Get) dans la mémoire mips à l'adresse R4

- **GetString et PutString**
 - PutString : besoin de charger **ce qui a** en mémoire MIPS avec copyFromMachine à l'adresse donné par R4
 - GetString : besoin de charger **dans** la mémoire MIPS avec copyToMachine à l'adresse donné par R4
 - copyFromMachine : machine->ReadMem
 - copyToMachine : machine->WriteMem

2 Entrées/Sorties

10

Objectifs

Implémentations

Résultats

- **Halt automatique**
 - interrupt->Halt ()
 - Éteint nachos proprement, et imprime les statistics
 - Rajouté dans le main apres l'appel a SynchConsoleTest

- **Valeur de retour**
 - Registre 2

.

3 Multi-Threading

11

Objectifs

Implémentations

Résultats

- **Structure de donnée d'un Thread:**
 - Status: JUST_CREATED,RUNNING,READY, BLOCKED
 - Pointeur de bas de pile: stack
 - Pointeur haut de pile: stackTop
 - Contenu des registres: machineState[]

- **Si Thread utilisateur:**
 - Contenu registres utilisateur: userRegisters
 - Espace d'adressage du thread : space

.

3 Multi-Threading

12

Objectifs

Implémentations

Résultats

- **Pile des threads noyau :**
 - Dans l'espace d'adressage de l'environnement => Mémoire LINUX

- **Pile des threads utilisateurs:**
 - Dans la mémoire virtuelle MIPS

.

3 Multi-Threading

13

Objectifs

Implémentations

Résultats

□ Création d'un thread noyau:

□ New Thread(« Name »)

- Pile=NULL
- Status = Just_Created

□ Fork(function,argument):

- Appel à StackAllocate(&function,argument):
 - Allocation de la mémoire pour la pile 4Ko
 - Sommet de pile = « Sentinelle » :STACK_FENCEPOST
 - PC=T hreadRoot(InitialPC, InitialArg, WhenDonePC, StartupPC)

□ Suppression d'un Thread:

□ Finish:

- threadToBeDestroyed = this
- Appel à sleep => status = BLOCKED

Lors du prochain appel à Scheduler::Run

Le thread pointé par threadToBeDestroyed sera détruit.

3 Multi-Threading

14

Objectifs

Implémentations

Résultats

□ La classe Scheduler:

□ ReadyToRun appelé par :

- Semaphore::V()
- Thread::Fork()
- Thread::Yield()
- Status = READY
- Ajout en fin de liste des threads READY

□ FindNextToRun appelé par:

- Yield()
- Sleep()
- Permet de récupérer le premier prêt pour lui donner le processeur

3 Multi-Threading

15

Objectifs

Implémentations

Résultats

- **SaveUserState**
 - `userState = machine->register`
 - Registre processeur MIPS
 - Pointeur de Pile
 - Table des Pages

- **RestoreUserState**
 - `machine->register = userState`

- **SaveState**
 - `pageTable`
 - `pageTableSize`

- **RestoreState**
 - Restore la table des pages.

3 Multi-Threading

16

Objectifs

Implémentations

Résultats

- **Analyse de trace de l'exécution**
 - Lecture du fichier (Open)
 - Fichier executable (en Noff)
 - Section Text
 - Section Data
 - Section bss
 - Taille Addrspace = Somme sections

.

3 Multi-Threading

17

Objectifs

Implémentations

Résultats

- **Création d'un AddrSpace:**
 - Dans le constructeur AddrSpace
 - NumPages calculé avec size
 - Recalcule de size
 - Initialisation PageTable
 - Correspondance virtuelles – physique
 - InitRegister: initialisation des registres
 - Ecriture des registres d'états
 - RestorState

3 Multi-Threading

18

Objectifs

Implémentations

Résultats

- **Lancement: Run()**
 - Execution instruction par instruction
 - Gestion de l'horloge OneTick

- **Arret: Halt() ou Exit()**
 - Run boucle à l'infini

.

3 Multi-Threading

19

Objectifs

Implémentations

Résultats

- **Création d'un Thread**
 - ▣ Utilisation d'une bitMap protégée
 - ▣ Utilisation d'une structure de stockage
 - ▣ Appel de Fork() par un new_Thread

- **Fonction appelée par Fork**
 - ▣ Récupération des paramètres
 - ▣ Initialisation des registres
 - ▣ Enregistrement des registres d'états

.

3 Multi-Threading

20

Objectifs

Implémentations

Résultats

- **Arrêt d'un thread**
 - Libère la place dans la bitMap
 - Appel à Finish()

- **Synchronisation**
 - Variable dans AddrSpace
 - A la création le premier prend un sémaphore
 - A la suppression le dernier lâche le verrou

3 Multi-Threading

21

Objectifs

Implémentations

Résultats

- **Plusieurs threads par Processus**
 - Synchronisation deux sémaphores
 - Un sémaphore pour le Put
 - Un sémaphore pour le Get

 - BitMap: garantit la place pour de nouveau thread.

- **Terminaison automatique**
 - Sauvegarde de la valeur de retour

.

3 Multi-Threading

22

Objectifs

Implémentations

Résultats

```
#include "syscall.h"

void print(void* c)
{
    int j=0;
    PutString("Thread numero: ");
    PutInt((int)c);
    PutChar('\n');
}

int main()
{
    int i=0;
    UserThreadCreate(print, (void*)1);
    UserThreadCreate(print, (void*)2);
    UserThreadCreate(print, (void*)3);
    PutString("toto\n");
    return 0;
}
```

4 Multi-Processus

23

Objectifs

Implémentations

Résultats

□ Introduction à la pagination

□ Découpage de l'espace d'adressage

- Plusieurs pages de taille fixe
- De même pour la mémoire physique

- Correspondance => PageTable[NbrePages]
 - Fonction Translate
- Lecture => ReadMem
- Ecriture => WriteMem

4 Multi-Processus

24

Objectifs

Implémentations

Résultats

- **Copie du code dans la mémoire**
 - RedAt recopie dans un tableau
 - `executable->ReadAt ((char *) &noffH, sizeof (noffH), 0);`
 - `executable->ReadAt (&(machine->mainMemory[noffH.code.virtualAddr]), noffH.code.size, noffH.code.inFileAddr);`
 - ReadAtVirtual appel RedAt mais ensuite WriteMem

4 Multi-Processus

25

Objectifs

Implémentations

Résultats

- **La classe FrameProvider**

- Commun à toute la machine

- Utilisation d'une BitMap

- Taille = NumPhysPages

- Délivrera la référence à l'adresse physique dans la table des pages

.

4 Multi-Processus

26

Objectifs

Implémentations

Résultats

- **Plusieurs programmes en même temps**
 - Création du nouveau thread
 - Appel à Fork(do_ForkExec)
 - Retour du pid au Processus père
 - Do_ForkExec:
 - Verification de la place disponible
 - Création d'un nouveau space
 - Appel à Run

4 Multi-Processus

27

Objectifs

Implémentations

Résultats

- Verrouillage du père:
 - ▣ Compteur d'enfant pour chaque Thread
 - ▣ Un sémaphore pour chaque Thread
 - ▣ Référence sur le père
 - ▣ Si premier Fork():
 - P() du sémaphore de notre père
 - ▣ Si dernier Fork() à partir:
 - V() du sémaphore de notre père

4 Multi-Processus

28

Objectifs

Implémentations

Résultats

- Implémentation du Join:
 - ▣ Liste Enfant, un élément contient:
 - Pointeur vers un sémaphore
 - Pid du Processus
 - ▣ A l'appel système:
 - Création du sémaphore et P()
 - Ajout à la liste
 - ▣ Lors de la terminaison:
 - Suppression de la liste et V()
 - ▣ Lors d'un Join
 - Récupère sémaphore par rapport au pid
 - P() puis V() puis delete

4 Multi-Processus

29

Objectifs

Implémentations

Résultats

- Terminaison d'un Fork:
 - ▣ Tentative de récupération de notre Sémaphore
 - ▣ Finish

5 Système de Fichiers

30

Objectifs

Implémentations

Résultats

- Fonctionnement général
 - ▣ Un répertoire racine:
 - Pointeurs vers entêtes des fichiers
 - ▣ Les entêtes contiennent:
 - Taille du fichier
 - Nombre de secteurs utilisés
 - Pointeurs vers blocs du disque correspondant

- Attention: Ls, Mkdir, Rmdir, Cat, cp,....
Géré comme des appels systèmes

5 Système de Fichiers

31

Objectifs

Implémentations

Résultats

- Classes et fonctions principales:
 - ▣ `directory` : représente un répertoire
 - ▣ `freeMap`: représente les secteurs libres
 - ▣ `Filhdr` (classe header):
 - `WriteBack`: enregistre les modifications dans le disque
 - `Allocate`: alloue un certain espace disque
 - ▣ `directoryFile`: représente « le disque »
 - ▣ `Directory->FetchFrom(directoryFile)`
recharger en mémoire le répertoire

5 Système de Fichiers

32

Objectifs

Implémentations

Résultats

- Création d'un fichier:
 - ▣ New Directory, FileHeader, BitMap
 - ▣ Directory->FetchFrom(directoryFile)
 - ▣ BitMap->FetchFrom(freeMapFile)
 - ▣ Si secteur suffisant:
 - Ajout dans Directory
 - FileHeader->Allocate
 - Enregistrement des modifications

5 Système de Fichiers

33

Objectifs

Implémentations

Résultats

- Création d'un sous répertoire
 - ▣ Nouveau constructeur
 - 2 entrées table :
 - « . » avec sector le secteur courant
 - « .. » avec sector le secteur de notre père
 - ▣ Ajout dans le répertoire père
 - Si plus de place => Arrêt
 - Si nom déjà utilisé => Arrêt
 - ▣ Enregistrement du nouveau répertoire comme un fichier normal

5 Système de Fichiers

34

Objectifs

Implémentations

Résultats

- Changement de répertoire
 - ▣ A chaque déplacement modification du path
 - ▣ Création d'un parseur pour analyser le « futur » path
 - ▣ Trois cas principaux:
 - /xxxx/yyyy... => déplacement à partir de DirectorySector
 - ../xxxx/yyyy... => déplacement à partir du père du courant
 - Xxxx/yyyy... => déplacement à partir du répertoire courant

5 Système de Fichiers

35

Objectifs

Implémentations

Résultats

- Changement de répertoire:
 - ▣ Lecture du contenu du disque
 - ▣ `directoryFile = new OpenFile(futur_sector)`
 - ▣ `Directory->FetchFrom(directoryFile)`

 - ▣ Sauvegarde du secteur courant
 - ▣ Sauvegarde du secteur parent
 - ▣ Si « .. » `futur_sector=sector_parent`
 - ▣ Si `./xxxx` `sector_parent=sector`

5 Système de Fichiers

36

Objectifs

Implémentations

Résultats

- Affichage du contenu d'un répertoire
 - ▣ Parseur pour le chemin:
 - Si « » appel à List()
 - Sinon sauvegarde du path courant
 - Déplacement dans le nouveau path
 - List()
 - Retour à l'ancien path

5 Système de Fichiers

37

Objectifs

Implémentations

Résultats

- Suppression d'un répertoire
 - ▣ Vérification que le nom est un répertoire
 - ▣ Vérification que le répertoire est vide
 - ▣ Suppression de l'entrée dans le père
 - ▣ Désallocate des headers.
 - ▣ Libérer l'espace libre
 - ▣ Sauvegarde des états

5 Système de Fichiers

38

Objectifs

Implémentations

Résultats

- Augmentation taille fichier
 - ▣ Capacité de l'entête
 - ▣ Besoin d'indirections (Nouvelle classe)
 - ▣ Allocation de secteurs supplémentaires

5 Système de Fichiers

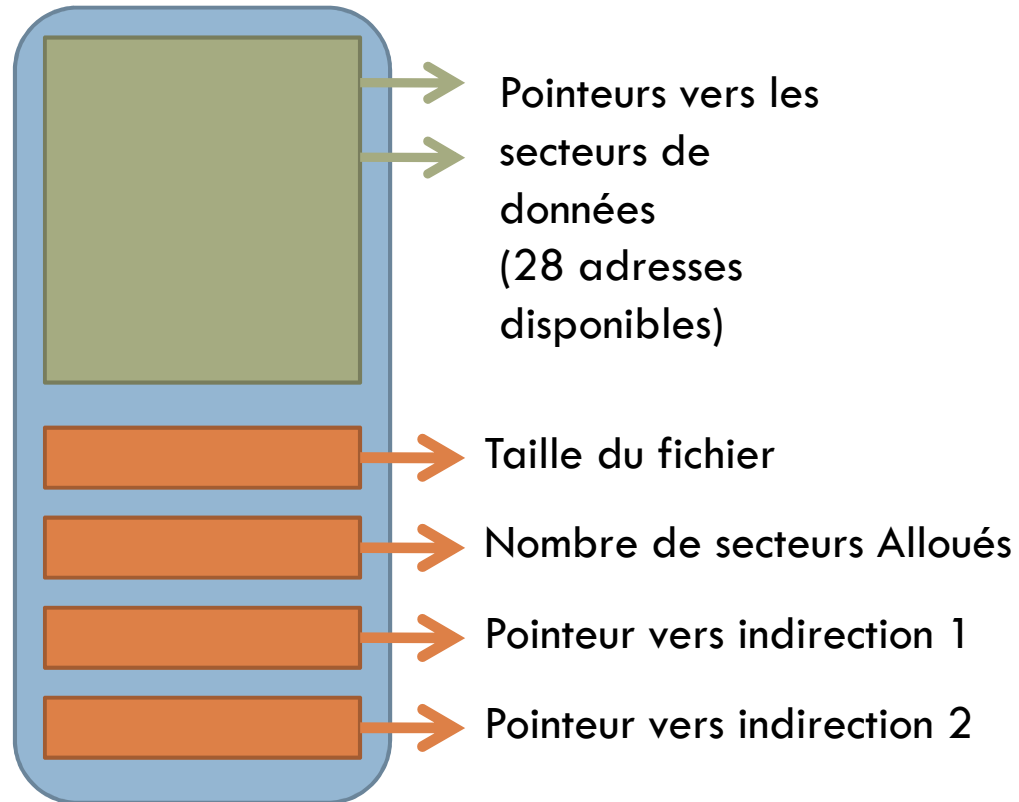
39

Objectifs

Implémentations

Résultats

□ Entête du fichier (Sur un secteur)



5 Système de Fichiers

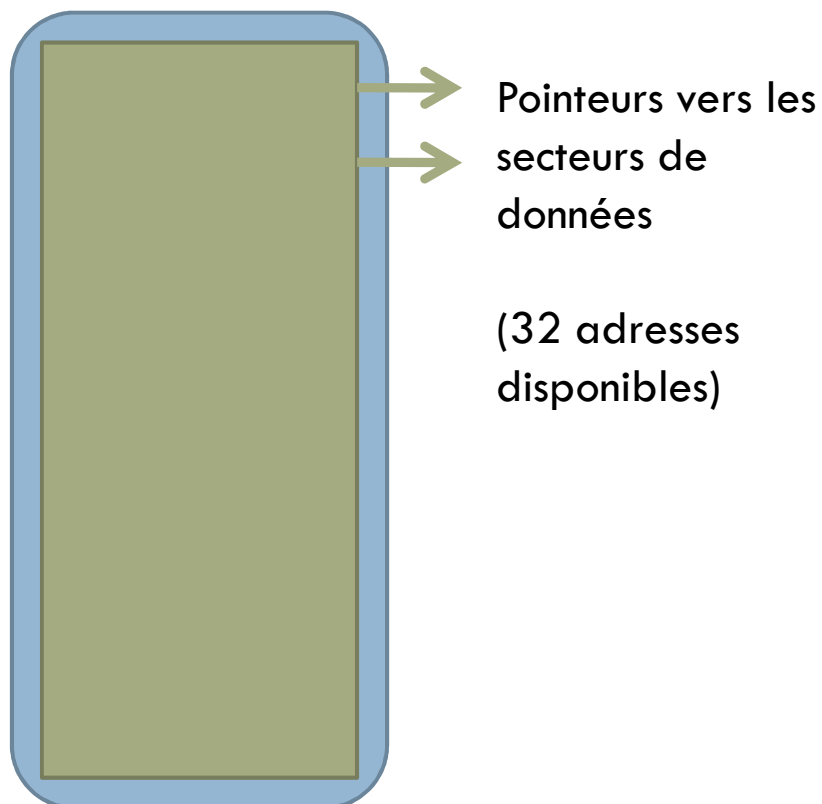
40

Objectifs

Implémentations

Résultats

□ Indirection niveau 1 (Sur un secteur)



5 Système de Fichiers

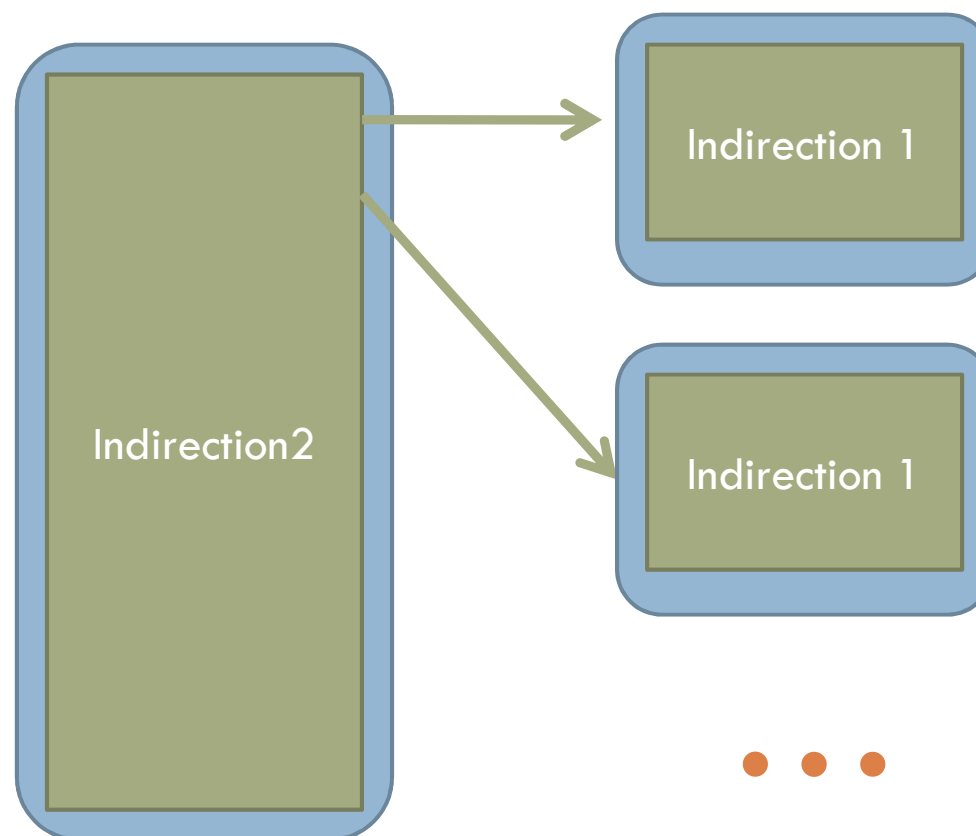
41

Objectifs

Implémentations

Résultats

□ Indirection niveau 2 (Sur un secteur)



5 Système de Fichiers

42

Objectifs

Implémentations

Résultats

□ Nouvelle taille maximum de fichier

□ Avant : (Pas d'indirection)

30 adresses vers secteurs de données
 $30 * \text{SectorSize} = 3\ 840 \text{ bytes}$

□ Après: (Avec 2 Indirections)

$(28 + 32 + 32 * 32) * \text{SectorSize} = 1\ 38\ 752 \text{ bytes}$

□ Taille du Disque :

$\text{NumSectors} * \text{SectorSize} = 131 \text{ KBytes}$

5 Système de Fichiers

43

Objectifs

Implémentations

Résultats

- Taille de fichier dynamique
 - ▣ Ne plus allouer les secteurs de données à la création du fichier
 - ▣ Ajout d'une méthode « Extend » dans FileHeader et Indirection pour alouer dynamiquement des secteurs

5 Système de Fichiers

44

Objectifs

Implémentations

Résultats

- Algorithme :
 - Estimer le nombre de secteurs nécessaires pour n bytes
 - Déterminer le besoin en indirection
 - Créer les indirections le cas échéant et allouer les secteurs
 - Mettre à jour l'entête de fichier

6 Réseaux

45

Objectifs

Implémentations

Résultats

- **Mécanisme du réseau**
 - Implémentation actuelle du réseau
 - Envoi de 10 messages
 - Anneau de N noeuds
- **Transmission fiable**
 - Utilisation d'une boîte d'envoi
 - 1 thread « Sender » et 1 thread « Listener »
- **Message de longueurs variables**
 - Découpage du message en plusieurs paquets
 - Envoi des paquets à la couche inférieure
 - Regroupement des paquets à la réception

.

6 Réseaux

46

Mécanisme du réseau

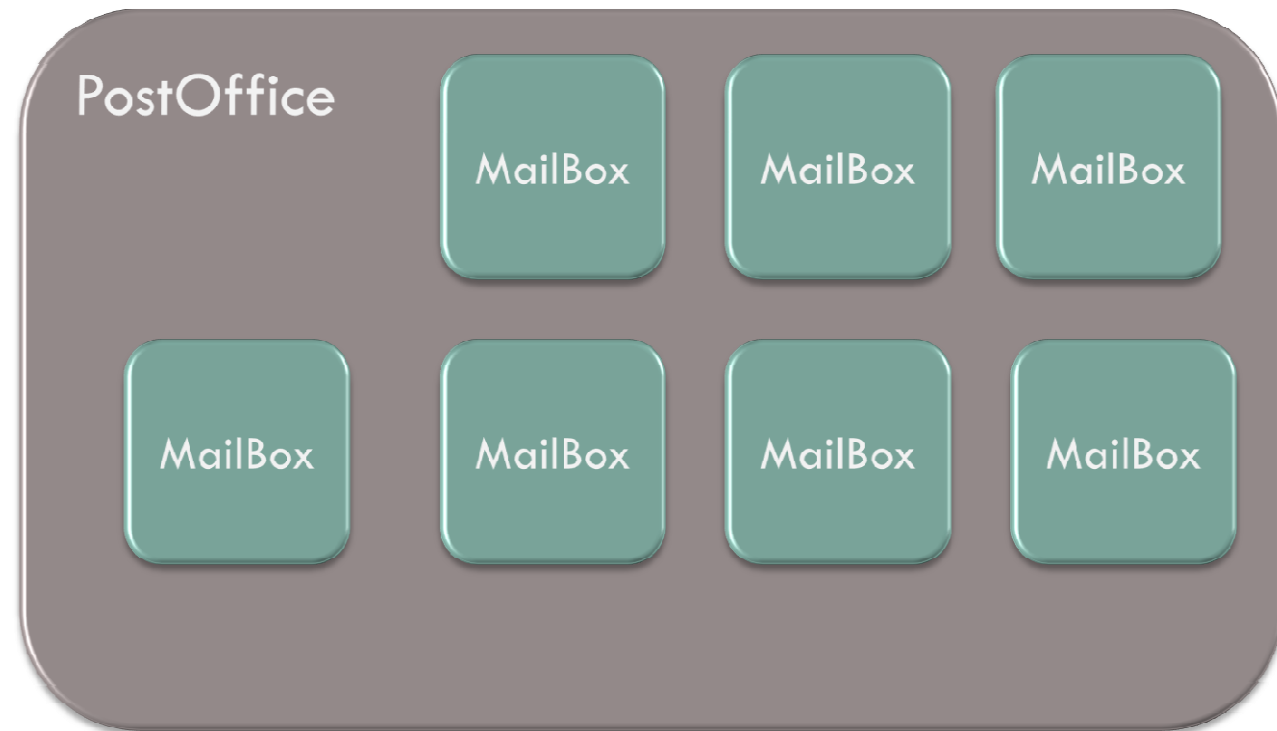
Objectifs

Implémentations

Résultats

□ Implémentation actuelle du réseau

□ Modèle abstrait : La Poste



6 Réseaux

47

Mécanisme du réseau

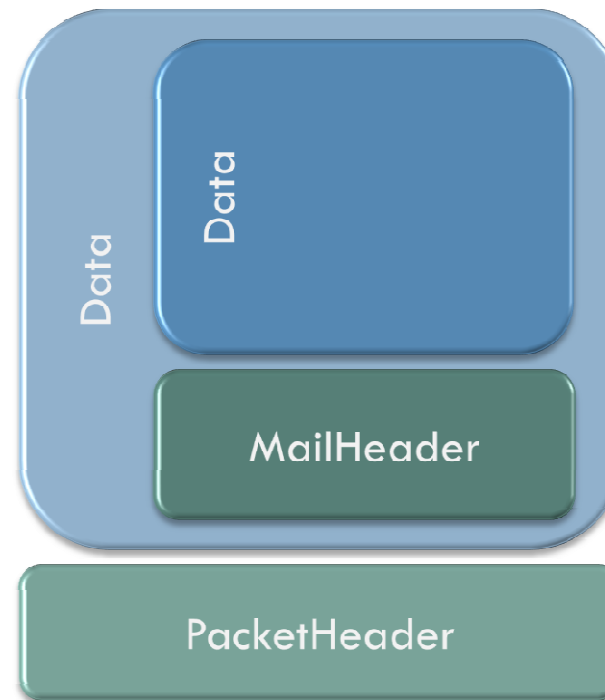
Objectifs

Implémentations

Résultats

□ Implémentation actuelle du réseau

- Modèle réel : modèle de couche



6 Réseaux

48

Mécanisme du réseau

Objectifs

Implémentations

Résultats

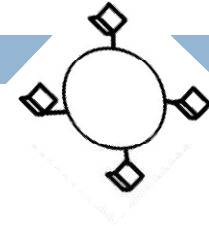
□ Envoi de 10 messages

```
□ while(i<10){  
    //on remplit les MailHeader et PacketHeader  
    ...  
    //on envoie la donnée  
    postOffice->send(outPktHdr,outMailhdr,data);  
    //on réceptionne une autre donnée sur la boîte 0  
    postOffice->receive(0,&inPktHdr,&inMailHdr,buffer)  
}
```

6 Réseaux

49

Mécanisme du réseau



Objectifs

Implémentations

Résultats

□ Anneau de N nœuds

□ Paramètre de *RingTest*: Nbnoeud

□ Pour une machine X :

■ Initiateur? =>(id==0)

- Send(m) à $(id+1)\%Nbnoeud$ sur la mailbox 0 à partir de notre mailbox 1
- Receive sur notre mailbox 0

■ Non initiateur ? => (id!=0)

- Receive sur notre mailbox 0
- Send(m) à $(id+1)\%Nbnoeud$ sur la mailbox 0 à partir de notre mailbox 1

.

6 Réseaux

50

Transmission fiable

Objectifs

Implémentations

Résultats

- **Utilisation d'une boîte d'envoi pour chaque MailBox**
 - Nouvelle primitive : SendReliable(...)
 - 2 types de messages: MSG, ACK
 - Timer de réémission

- **Comment ?**
 - 2 nouvelles threads, « Sender » et « Listener »
 - Une nouvelle méthode WaitForFinish() avant de faire Halt

.

6 Réseaux

51

Transmission de longueurs variable

Objectifs

Implémentations

Résultats

- **Deux nouvelles primitives :**
 - `SendVariable(Pkthdr,MailHdr,data);`
 - rempli des nouveaux champs de MailHdr
 - Découpe « data » en morceaux de taille `MaxMailSize`
 - Envoi ces paquets grâce à `SendReliable(...)`
 - Tag de fin pour le dernier paquet

 - `ReceiveVariable(box,PktHdr,MailHdr,buffer);`
 - récupère les paquets et concatène les différentes données reçus.
 - retourne un pointeur sur les nouvelles données au client

Extensions

52

- Gestion des chemins relatifs et absolus
- Implémentation des fichiers dans le shell
- Gestion de nombreuses exceptions
- ...

Questions et Discussion

53



Commencer le test