M1 Informatique
Etape 5 : Nachos et la gestion des fichiers

Vincent Danjean, Guillaume Huard, Arnaud Legrand,
Vania Marangozova-Martin, Jean-François Méhaut

Année 2010/2011

La gestion des fichiers dans un système est un composant important et complexe. Par exemple, le système de fichiers Unix possède au moins trois niveaux d'indirection avant d'accéder au disque : Une table des fichiers ouverts pour chaque processus, la table système des fichiers ouverts et la table des *i*nodes. De plus, la recherche sur disque de l'information nécessite parfois plusieurs niveaux d'indirection (gros fichiers). Finalement, on utilise des algorithmes d'insertion/retrait et de recherche qui peuvent devenir relativement complexes.

Dans le but d'obtenir un système de fichiers assez simple ou du moins dont l'apprentissage peut se faire sur une courte période, les concepteurs de Nachos ont sacrifié plusieurs concepts qui peuvent affecter le réalisme du système de fichiers. Les principales restrictions de ce système de fichiers sont la dimension maximale des fichiers (37,5KB), l'allocation statique d'espace à la création du fichier, un répertoire à un seul niveau (pas de sous-répertoires), l'absence de tampons d'entrée/sortie (aucune mémoire cache pour le disque) et le manque de robustesse. Le principal avantage tiré de ces restrictions est que le code du système de fichiers est très petit.

Le système de fichiers de Nachos (avec ses restrictions) est présentement fonctionnel. Vous trouverez une version exécutable du système Nachos qui permet de tester le système de fichier dans le répertoire `nachos/code/filesys` sous le nom `nachos`. Les commandes acceptées par ce programme sont -f pour formater le disque, `-cp` pour copier un fichier du système Unix vers le système de fichiers Nachos, `-l` pour lister le contenu du répertoire, `-r` pour détruire un fichier. Des informations plus complètes sont disponibles dans les fichiers `nachos/code/threads/main.cc` et `nachos/code/filesys/fstest.cc`.

Le système de fichiers de Nachos possède un répertoire à un seul niveau. Lors du démarrage du système, Nachos charge en mémoire l'entête du répertoire. Le répertoire contient au plus 10 entrées contenant chacune trois informations : un booléen `inUse` qui indique si l'entrée est utilisée ou pas, une chaîne de caractère contenant le nom du fichier et un pointeur vers l'entête du fichier. L'entête du fichier contient la taille en octets et en secteurs du fichier et un certain nombre de pointeurs vers des blocs de données.

Le système de fichiers est évidemment emmagasiné sur disque. Nachos fournit donc un simulateur de disque qui accepte les requêtes de lectures et écritures d'un secteur. Le simulateur disque calcule le temps nécessaire pour effectuer la requête selon l'état actuel du disque et, après le temps requis, signale la fin de l'entrée-sortie via une interruption. Les données du disque sont enmagasinées dans un Unix. Nachos fournit aussi un pilote d'interruption qui se charge de démarrer l'entrée-sortie et de bloquer le thread jusqu'à ce que la fin de l'entrée-sortie soit signalée (les entrées-sorties sont synchrones au niveau du pilote).

Dans sa version initiale, le système de fichiers Nachos implante une gestion de disque simple et uniforme. Ainsi, la plupart des structures de données tiennent sur un secteur (entêtes et blocs de données). La gestion de l'espace libre est faite par un vecteur de bits stockée sur le secteur 0. L'entête du répertoire est stockée sur le secteur 1.

L'implantation du système de fichiers est faite dans plusieurs classes C++. Ces classes sont :

– le disque (`Disk`)
Cette classe implante la structure d'un disque et fournit les opérations de lecture et écriture d'un secteur. La taille du disque est initialement définie à 128K. Le code est disponible dans `code/nachos/machine/disk(.h/.cc)`.

– le pilote (`SyncDisk`)
Cette classe définit la structure et les routines qui implantent le pilote du disque et fournissent des entrées/sorties synchrones sur le disque. Le code de cette classe se retrouve dans `nachos/code/filesys/synchdisk (.h/.cc)`.

– les entêtes (`FileHdr`)
Cette classe décrit la structure de l'entête d'un fichier. Elle définit toutes les informations contenues dans l'entête et les opérations pour la manipuler. Le code de cette classe se retrouve dans les fichiers `nachos/code/filesys/filehdr.(.h/.cc)`.

– le répertoire (`Directory`)
Cette classe définit la structure du répertoire et les fonctions qui permettent de le manipuler. Le code de cette classe se retrouve dans les fichiers `nachos/code/filesys/directory(.h/.cc)`.

– les fichiers ouverts (`OpenFile`)
Cette classe permet de lire et écrire des données dans un fichier. Elle traduit les lectures et écritures dans un fichier en lectures et écritures de secteurs sur le disque. Pour y parvenir, elle utilise l'entête du fichier et la position courante dans le fichier. Le code de cette classe se retrouve dans `nachos/code/filesys/openfile(.h/.cc)`.

– le système de fichiers (`FileSys`)
Cette classe définit l'interface du système de fichiers. Elle fournit les opérations telles la création, la destruction et l'ouverture de fichiers. Le code de cette classe se retrouve dans `nachos/code/filesys/filesys(.h/.cc)`.

L'interaction entre ces différents objets est assez simple. Lors de l'initialisation du système, Nachos lit sur le disque les informations concernant l'espace libre et le répertoire. Ces informations sont directement disponibles respectivement sur les secteur 0 et 1. Lors de l'ouverture d'un fichier, le système de fichiers obtient l'information nécessaire dans le répertoire et retourne un pointeur vers un objet de type `OpenFile`. Nachos utilise ensuite ce pointeur pour modifier le fichier (lecture, écriture ou positionnement dans le fichier). La fermeture d'un fichier est implicite en Nachos. Elle se fait lors de la destruction de l'objet de type `OpenFile`.

Nachos fournit un système de fichiers possédant bien des limitations et restrictions. L'objectif de cette étape est de compléter avec de nouvelles fonctionalités. Les parties I, II et III sont obligatoires. Les parties IV, V et VI sont obligatoires pour les groupes de 5, sont optionnelles pour les autres groupes.

Avant de débuter les différentes parties de cette étape, nous vous conseillons de lire et analyer l'ensemble des fichiers du système de fichiers Nachos.

Partie I. Implantation d'une hiérarchie de répertoires

Nachos fournit donc actuellement un répertoire à un seul niveau et tous les fichiers du disque se trouvent donc dans ce répertoire. L'implantation de répertoires hiérarchiques implique qu'un répertoire puisse contenir des fichiers ET des sous-répertoires. De nouvelles fonctions doivent donc être fournies afin de créer et détruire un répertoire, changer de répertoire et imprimer le contenu du répertoire courant.

La création d'une hiérarchie de répertoire pose de nouveaux problèmes. En particulier, on doit pouvoir se déplacer aisément dans la hiérarchie, l'ajout d'une méthode pour mémoriser le répertoire courant, la création de répertoires spéciaux pour parcourir la hiérarchie dans les deux directions (`.` et `..`) et finalement

la possibilité d'utiliser des noms de chemin (`path names`).

Afin d'obtenir des résultats uniformes et de faciliter le développement, le nombre d'entrées dans un sous-répertoire est limité à 10 (constante `NumDirEntries` du fichier `filesys.cc`. Deux de ces entrées sont réservées pour les répertoires `.` et `..`. De plus, seuls des noms relatifs au répertoire courant seront utilisés (pas de nom de chemin ou *path name*. Enfin, la destruction d'un répertoire ne se fait que si celui-ci est vide.

Partie II. Implantation de la table système des fichiers ouverts

L'implantation actuelle du système de fichiers ne fournit aucune faÁon simple pour maintenir plusieurs fichiers ouverts simultanément. Nachos doit avoir un objet différent pour chaque fichier ouvert ou encore fermer implicitement un fichier avant d'en ouvrir un autre.

Il s'agit donc d'intégrer à Nachos un mécanisme simple pour maintenir une plusieurs fichiers ouverts simultanément. Le nombre maximum de fichiers sera limité à 10.

Partie III. Support des accès concurrents

Comme cela a été vu dans l'étape 3, le noyau Nachos support le concept de processus léger (thread). Ainsi, on peut avoir plusieurs fonctions du noyau qui s'exécutent simultanément. Le système de fichiers Nachos, même avec les amélioration déjà apportées, ne supporte pas les accès concurrents. Ainsi, si deux programmes ou deux threads s'exécutant simultanément accèdent au même fichier ou au même répertoire, cela peut avoir des conséquences fâcheuses allant de la corruption des données jusqu'à la perte de fichiers.

Le système de fichiers doit donc être modifié pour être capable de contrôler les accès simultanés. Une solution simple consiste à interdire à un thread ou programme en exécution d'ouvrir ou de détruire un fichier déjà ouvert. Dans un premier temps, les accès concurrents aux répertoires n'ont pas à être contrôlés.

La mise en place des supports concurrents implique la modification de la classe définissant les threads Nachos (le code de cette classe se trouve dans `nachos/code/threads.(h/cc)`). Vous devez la modifier pour y mettre l'information sur les fichiers et répertoires ouverts par le thread. La solution normale consiste à implanter une table des fichiers ouverts pour chacun des threads, en plus de la table des fichiers ouverts du système (2 niveaux de tables). Cette organisation présente l'avantage d'indiquer clairement les fichiers ouverts par le thread et d'empêcher un thread d'accéder par erreur aux fichiers ouverts par d'autres threads.

Le nombre maximum de fichiers ouverts par un thread sera également limité à 10.

Partie IV. Augmentation de la taille maximale des fichiers

La taille maximale d'un fichier est actuellement de 3,75KB. Cette limitation est imposée par la taille des entêtes (un secteur) et par l'utilisation d'un index à un seul niveau. L'augmentation de la taille à **120KB** limitera désormais la taille d'un fichier à environ celle du disque. Cette augmentation exige l'utilisation d'au moins un niveau d'indirection.

Partie V. Implantation de fichiers de taille variable

Actuellement, dans le système de fichiers, la dimension d'un fichier est spécialisée à sa création. Cela permet d'allouer tout l'espace disque nécessaire au fichier dès sa création et de ne jamais modifier l'entête par la suite. Avec des fichiers de taille variable, la taille initiale d'un fichier est toujours de 0. Le fichier grossit à chaque fois qu'une écriture est faite à un emplacement dépassant la fin actuelle du fichier. Cette modification implique des modifications continuelles de l'entête. Des modifications devront également être apportées à la commande permettant de lister le contenu d'un répertoire. En effet, il faudra maintenant afficher la taille du fichier.

Si vous ne faites pas la partie IV, la taille maximale des fichiers demeure à 3,75KB.

Partie VI. Implantation des noms de chemin (`path name`)

Cette partie est fortement liée à la partie I. Pour tester les noms de chemin, vous aller créer un interpreteur de commandes qui pourra analyser tous les noms passés en paramètre.

Partie VII. Amélioration des accès concurrents

Cette partie est fortement liée à la partie III. Il s'agit maintenant de permettre à deux threads d'ouvrir les mêmes fichiers simultanément. Cependant, chaque thread doit avoir son propre compteur d'emplacement dans le fichier. La synchronisation à mettre en place est de type lecteur/rédacteur. On doit ainsi empêcher deux écritures simultanées ou bien une écriture et une lecture simultanée. Les lectures peuvent se faire simultanément. Vous utiliserz des outils de synchronisation comme les sémaphores, verrous/conditions.

Les accès aux répertoires et entêtes doivent être synchronisés.

Partie VIII. Optimisation des entrées-sorties

On se fixe maintennant l'objectif d'améliorer les performances du système de fichiers. Dex approches seront explorées :

- Optimisation des accès disque
  Les performances du système de fichier peuvent être affectées par les délais de déplacement de la tête de lecture du disque (temps de positionnement) et par le temps de rotation du disque (temps de latence). On peut optimiser cette situation en plaÁant les blocs de données d'un même fichier très près l'un de l'autre (sur le même cylindre ou la même piste) ainsi qu'en ordonnanÁant les requêtes du disque pour minimiser les délais de lecture. Cependant, toutes les requêtes doivent être traitées dans un temps raisonnable.

- Utilisation de tampons d'entrée-sorties
  On peut utiliser des tampons (buffers) en mémoire pour conserver les blocs de données. Ainsi lors d'une lecture, on peut vérifier si le bloc n'est pas déjà dans les tampons. Si c'est le cas, il n'est alors pas nécessaire de faire une lecture disque. Lors d'une écriture, on peut mettre les données dans des tampons et faire le transfert vers le disque plus tard. Cette approche est appelée *w*rite-behind. De plus, les performances peuvent encore s'améliorer si, en lanÁant la lecture du prochain bloc de données d'un fichier aussitôt que la lecture courante termine (au cas où ce bloc serait sur le point d'être lu). On appelle cette aproche le *r*ead-ahead.

  L'espace pour ces tampons doit se limiter à l'espace nécessaire pour stocker 64 secteur du disque (environ 6% de l'espace disque). Cet espace inclut l'espace occupé par les fichiers ouverts, le vecteur de bits... si ceux-ci sont conservés en mémoire.

Partie IX. Robustesse

Les données du disque Nachos peuvent être corrompues si le système s'arrête anormalement (ex. panne de courant, sortie dans le *d*ebugger sans laisser le programme se terminer, ...). Cela se produit principalement pour les opérations d'écriture. Par exemple, la création d'un fichier requiert plusieurs écritures sur le disque : l'écriture de son entête et la version modifiée du répertoire contenant le nouveau fichier. Si le système est interrompu avant que toutes les mises à jour ne soient terminées, l'état du disque peut alors devenir

incohérent. Pour augmenter la robustesse du système de fichier, on doit toujours avoir des données cohérentes sur le disque ou bien être capable de les corriger après un arrêt anormal.