M1 Informatique
Etape 5 : Nachos et le réseau

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

Année 2010/2011

Note *Avant de commencer cette étape, vous devez vous assurer que votre système Nachos dispose des outils de synchronisation verrous et conditions. Il n'est pas nécéssaire que ces outils soient disponibles au niveau utilisateur.*

L'objectif de cette dernière étape du projet est de concevoir et réaliser des mécanismes évolués utilisant le réseau. Le mécanisme élémentaire disponible sur tout réseau est la transmission de messages. La version de base du système Nachos dispose de mécanismes de mécanismes rudimentaires de transmission de messages. L'objectif de cette étape est d'étendre ces mécanismes et de permettre ainsi la construction de services évolués comme le transfert de fichiers ou bien encore la migration de processus.

Partie I. Apprentissage des mécanismes réseau

Chaque machine Nachos (simulé par un processus Linux) est un nœud du réseau. Le réseau (simulé par des sockets Unix) constitue le médium de communication entre les nœuds. Vous pouvez tester les fonctionnalités minimales du réseau de la manière suivante :

- Ouvrez deux fenêtres de votre environnement graphique.
- Positionnez vous dans le répertoire `code/network`. Lancez dans une chacune des fenêtres les commandes suivantes (Attention, vous disposez d'un délai d'au plus trois secondes pour le lancement des deux machines Nachos).

Pour la machine 0, on la résultat suivant :

```
pojoaque $ ./nachos -m 0 -o 1
Got "Hello there!" from 1, box 1
Got "Got it!" from 1, box 1
Machine halting!

Ticks: total 107818290, idle 107818000, system 290, user 0
Disk I/O: reads 2, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 2, sent 2

Cleaning up...
pojoaque $
```

Pour la machine 1, on a le résultat suivant :

```
pojoaque $ ./nachos -m 1 -o 0
Got "Hello there!" from 0, box 1
Got "Got it!" from 0, box 1
Machine halting!

Ticks: total 66289740, idle 66289450, system 290, user 0
Disk I/O: reads 2, writes 0
Console I/O: reads 0, writes 0
Paging: faults 0
Network I/O: packets received 2, sent 2

Cleaning up...
pojoaque $
```

Les composants liés au réseau sont localisés dans 5 fichiers de l'arborescence Nachos :

- `machine/network.h` et `machine/network.cc` Ces fichiers contiennent les mécanismes qui permettent d'émuler le réseau physique. L'interface des classes est tout à fait proche de l'interface console excepté que l'unité de transmission est le paquet et non le caractère pour une console. Le réseau fournit un mécanisme de transmission de messages. Les messages sont ordonnées c'est à dire qu'ils sont reçus dans le même ordre qu'à l'émission. Cela veut dire que les messages ne se doublent pas. Les transmissions sont non fiables (unreliable), c'est à dire que les messages peuvent se perdre pendant leur transfert. Les messages sont de taille fixe. Le problème du routage des messages dans le réseau est complètement pris en charge par le réseau.
- `network/post.h` et `network/post.cc`. Il s'agit d'une abstraction du concept de boîte aux lettres, construite au dessus de la couche réseau. Il peut y avoir plusieurs boîtes aux lettres par machine. Le dépôt et le retrait de message sont des opérations synchrones. Une boîte aux lettres se définit comme une liste de messages synchronisée `SynchList` (liste synchronisée car une boîte aux lettres peut être accédée en concurrence par plusieurs threads).
- `network/nettest.cc` Ce fichier contient un petit exemple d'utilisation des routines de communication et des boîtes aux lettres.

Action I.1. *Interpretez les résultats affichés à partir du contenu du fichier* `code/network/nettest.cc`.

Action I.2. *Modifiez le programme* `nettest.cc` *pour que* 10 *messages de donnée et* 10 *accusés de réception soient échangés entre les machines* 0 *et* 1.

Action I.3. *Nous vous demandons maintenant de créer un nouveau programme de test qui sera constitué de* n *machines reliées par un anneau logique. La machine* 0 *initialisera l'anneau en envoyant un message (jeton) à la machine* 1 *qui le conservera un certain temps, puis le transmettra à la machine* 2 *et ainsi de suite. Le test s'arrêtera lorsque le jeton aura effectué un tour complet de l'anneau.*

Action I.4. *Vous allez maintenant lancer vos programmes de test en spécifiant un nouveau paramètre précisant le niveau de fiabilité des communications. Il s'agit de la probabilité de succès et de non perte des messages. Si le réseau est totalement fiable, la valeur sera* 1. *Une valeur inférieure à* 1 *indique la probabilité pour qu'un message ne disparaisse pas. Interpretez le comportement des programmes de test en spécifiant différentes valeur de fiabilité.*

Un petit exemple sur la machine 0,

```
pojoaque $
pojoaque $ ./nachos -m 0 -o 1 -l 0.5
```

Pour la machine 1, on a le résultat suivant :

```
pojoaque $
pojoaque $ ./nachos -m 1 -o 0 -l 0.5
```

Partie II. Transmission fiable de messages de taille fixe

A partir de la couche de communication non fiable qui est fournie, nous vous demandons d'implémenter une nouvelle interface de communication fiabilisée. Le principe est le suivant. Pour chaque message émis, la machine émettrice se met en attente limitée (`TEMPO`) d'un accusé de réception. Si l'accusé de réception n'est pas reçu au bout de la temporisation `TEMPO`, le message est réémis. Le nombre de réémissions pour un message est limité à `MAXREEMISSIONS`. Si un message n'a pu être émis, le thread émetteur reçoit une information précisant que son message n'a pu être transmis. Un message ne doit pas pouvoir être réemis plus d'un certain nombre de fois.

Action II.1. *Vous devez dans un premier temps analyser le code Nachos pour voir comment mettre en œuvre des attentes limitées. Si vous le jugez nécessaire, vous pouvez éventuellement proposer une nouvelle interface permettant de mettre en place des attentes limités.*

Action II.2. *Il est interdit de modifier le code de* `network.cc` *qui figure dans le répertoire* `machine`*. Les compléments pour fournir des communications fiables doivent être ajoutées au répertoire* `network`*. Vous devez donc proposer une nouvelle interface de communication s'appuyant sur l'interface de communication non fiable. La démarche que nous suggérons est similaire à celle retenue par TCP/IP sur le réseau Internet.*

Action II.3. *Modifiez et testez les programmes test en utilisant la classe de communication fiable que vous aurez implémenté.*

Partie III. Transmission de messages de longueur variable

L'objectif de cette partie est de fournir une couche de communication où il n'y a plus de limitation sur la taille des messages transmis. Un message devra donc être découpé en plusieurs messages de taille fixe. Comme la transmission des messages est totalement ordonnée, les différentes fractions d'un même message n'ont pas à être numérotées.

Action III.1. *Définissez et implémentez de nouvelles classes permettant d'envoyer et recevoir des messages de longueur quelconque.*

Action III.2. *Modifiez et testez les programmes test en utilisant des messages de longueur quelconque.*

Partie IV. Transfert de fichier entre deux machines

Il s'agit donc dans cette partie de réaliser un service de transfert de fichiers entre deux machines Nachos. Le principe du transfert de fichier reposer sur le principe du client serveur. Une machine client souhaite transmettre ou recevoir un fichier provenant d'une machine distante. Un processeur serveur doit tourner sur la machine distante.

Action IV.1. *Définissez le protocole d'échange entre la machine cliente (à l'initiative du transfert) et la machine distante. Vous pourrez, soit découper le fichier en différents paquets ou soit vous appuyer sur la couche de communication permettant d'envoyer des messages de longueur quelconque.*

Action IV.2. *Implémentez et réalisez ce protocole d'échange entre deux machines Nachos.*

Action IV.3. *Pour un transfert de fichier, vous calculerez le débit du transfert, c'est à dire le rapport entre le volume du fichier transféré et le temps de la transmission de ce fichier. Le résultat s'exprimera en Kilo Octets par seconde.*

Partie V. Migration de processus

Note *Cette partie est optionnelle pour les groupes normaux (au plus 4 étudiants). Elle est obligatoire pour les groupes de plus de 4 étudiants.*

La migration de processus est une opération permettant à un processus en cours d'exécution d'être déplacé d'une machine Nachos vers une autre machine Nachos. Cela peut permettre de mettre en place une régulation de charge entre différentes machines Nachos. La migration de processus est un service différent d'une exécution distante dans la mesure où le processus est initialisé sur une machine distante et ne changera pas de machine d'exécution. Pour une migration de processus, vous devez copier la totalité de l'espace d'adressage du processus vers la machine distante. Nous vous proposons de réaliser une automigration, c'est à dire que le processus décide par lui même de migrer vers une machine distante.

Action V.1. *Définissez et implémentez la migration de processus.*

Un des problèmes délicats de l'opération de migration concerne les accès au fichier. Avant sa migration, un processus peut avoir ouvert et accédé à un ou plusieurs fichiers. Après sa migration, le processus doit pouvoir continuer à accéder à ces fichiers. C'est ce point qui rend particulièrement difficile les opérations de migration. Plusieurs solutions sont envisageables pour traiter le problème des accès au fichier après migration. La première est de transférer pendant la migration les fichiers ouverts par le processus. Une seconde solution est de traiter les accès au fichier de manière distante, c'est à dire que tous les accès au fichier sont redirigés vers la machine où résidait initialement le processus.

Action V.2. *Cette question est optionnelle. La solution la plus simple semble être de transférer les fichiers ouverts au moment de la migration et de les transférer à la fin de l'exécution du processus ou à la fermeture du fichier. Implémentez cette solution ou une solution que vous jugeriez plus simple à mettre en œuvre.*