M1 MOSIG
Etape 6 : Nachos Network

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

Année 2010/2011

Note *Before starting this project, you should make sure that your NachOS implements mutex and condition synchronization structures. It is not necessary to export those to the user level.*

The goal of this project is to implement advances network services.

As message trasfer is at the basis of network function. NachOS offers basic mechanisms fr message transfers. You should develop these mechanisms in order to obtain a file trasfer protocol or a process migration mechanism.

Partie I. Study of NachOS Network Mechanisms

Each NachOS machine (simulated by a Linux process) is a network node. Network nodes communicate through a network simulated by Unix sockets. You can test the basic network functions in the following way :

– Launch two terminals
– Go to the `code/network` directory. Launch the following commands in the terminals (attention, you have at most 3 seconds to do so)

For machine 0 :

```
$ ./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...
$
```

Pour la machine 1, on a le résultat suivant :

```
$ ./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...
$
```

The network is implemented in 5 nachos files :

– `machine/network.h` et `machine/network.cc` These filesimplement the mechanisms t emulate the physical network. The classes' interface is very close to the one of the console class. The only difference is that the transfer unit is not a character but a packet. The network implements a message transfer mechanism. The messages are ordered i.e they are sent and received in the same order. The transmission is unreliable i.e messages can be lost. The messages have a fixed size. The routing is implemented by the network mechanism.
– `network/post.h` et `network/post.cc`. This class implements a communication abstraction built on top of the network : the letter box. One machine may have multiple letter boxes which serve for sending and receiving messages. The posting and getting of a message are synchronous operations. The letter box is actually a synchronized list of messages `SynchList`. It is synchronized as it can be accessed by multiple threads.
– `network/nettest.cc` This file gives a small example for the use of the communication routinesand of the letter boxes.

Action I.1. *Interpret the results obtained with* `code/network/nettest.cc`*.*

Action I.2. *Change* `nettest.cc` *in order to transfer* 10 *data messages and* 10 *acknowledgements between machines*0 *and* 1*.*

Action I.3. *Create a new test program in order to generate a ring topology of n machines. Machine* 0 *will be the one to initialize the network by sending a special token message to machine*1*. The latter will keep the token a certain amount of time and then transfer it to machine* 2*. Machine* 2 *will also keep the token some time before sending it to machine* 3*, etc. The test should stop when the token goes through the complete ring.*

Action I.4. *Launch again your tests but change the parameter configuring of te reliability of the network. The perameter gives the probability to transfer a message. If it is* 1*, it will mean that the network is totally reliale. Test different values for the parameter and interpret the behavior of your programs.*

Small example for machine 0,

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

For machine 1 :

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

Partie II. Reliable Transfer of Fixed-size Messages

Starting from the provided non reliable communication layer, you should now implement a new communication interface that is reliable. For each message, the sender machine will wait for an acknowledgement during a limited time period (`TEMPO`). If the acknowledgment is not received during this period of time, the message should be send again. The maximum number of re-emissions is set to `MAXREEMISSIONS`. If a message canot be resent, the sender thread is notified.

Action II.1. *You should start by analyzing NachOS code in order to understand how to manage timeouts.*

Action II.2. *It is forbidden to change the code of* `machine/network.cc`*. Your modifications should be added to the* `network` *directory. You should propose a new communication interface based on the non reliable one. We propose to use the logic of the existing TCP/IP protocol.*

Action II.3. *Test your programs with the new communication interface.*

Partie III. Transfer of Variable-size Messages

The goal now is to remove the size limit for transferred messages. To do so, a message should be cut in pieces of the same size. As the transfer is totally ordered, the pieces of the same mesage do not need to be numbered.

Action III.1. *Define and implement the new classes allowing the management of variable-size messages.*

Action III.2. *Test your implementation.*

Partie IV. File Transfer

File transfer is based on the client-server architecture. A client machine will send a file or receive a file from a remote machine. A server process should execute o the remote machine.

Action IV.1. *Define the protocol between the client and the server machines. To send a file you may use either your relaible communication layer or the interface for sending variable-size messages.*

Action IV.2. *Implement and test your protocol.*

Action IV.3. *Calculate the trasfer debit between the client and the server machines (divise the size of the file per the transmission time) The result should be in KB/s.*

Partie V. BONUS : Process Migration

A process migration allows the migration of an executing process from one NachOS machine to another. This may be used for load balancing, for example.

To migrate a process, you should copy all of its address space to the remote machine. We propose that the decision for process migration is taken by the process itself. Typically the programm will call a special system call in order to start the migration.

Action V.1. *Define and implement process migration.*

One of the problems to manage during migration is file access. A process may have opened several files before its migration. After its migration, you should ensure that the process continue to be able to access these files. Multiple solutions are possible. One solutions is to also migrate the files. Another is to access files remotely.

Action V.2. *Implement one solution for managing open files for a migrating process.*