Version 001

Tutoriel

Écrire un programme fonctionnant en réseau avec la common c++

Auteur : Jean-Louis Gutzwiller.

Dernière modification : 9 septembre 2010

Jean-Louis.Gutzwiller@supelec.fr

Table des matières

1 Introduction 5
2 Utilisation de la common cpp 5
3 Exemples de programmes 5
3.1 Exemple de programme serveur 5
3.2 Exemple de programme client 6
3.3 Téléchargement des exemples 7

1 Introduction

Les fonctions fournies en natif par les différents systèmes d'exploitation ne sont pas toujours compatibles d'un système à l'autre, même si des efforts ont été faits pour que ces fonctions se ressemblent au maximum. Ainsi, l'écriture d'un programme effectuant des opérations en réseau nécessite, si on souhaite faire fonctionner ce programme à la fois sur Linux et sur Windows, d'insérer des directives conditionnelles permettant d'activer des instructions différentes selon que le programme est compilé vers Linux ou vers Windows.

L'utilisation de bibliothèques telles que la common cpp permet d'écrire des programmes portables (pour lesquels le code source est identique pour les deux systèmes).

2 Utilisation de la common cpp

La common cpp étant écrite en C++ (langage objet), votre programme devra être écrit en C++. Vous pouvez utiliser une extension de fichier indiquant que votre programme est en C++ comme « .cc » ou « .cpp ».

Pour la compilation, utilisez le compilateur `g++` et les options indiquant l'utilisation de la common cpp : `` `pkg-config --cflags --libs libccgnu2 libccext2` ``.

Exemple : si votre programme s'appelle « programme.cpp », vous pouvez le compiler avec la ligne de commande suivante :

```
g++  programme.cpp -o  programme -Wall `pkg-config --cflags --libs libccgnu2 libccext2` -g
```

Si vous compilez vers Windows avec la version 4.5.0 du compilateur, je recommande la ligne de commande suivante :

```
g++  programme.cpp -o  programme -Wall `pkg-config --cflags --libs libccgnu2 libccext2` -g
-static-libstdc++ -static-libgcc
```

3 Exemples de programmes

3.1 Exemple de programme serveur

Voici un exemple de programme très simple qui réalise un serveur. À chaque fois qu'un client se connecte, il est supposé que le client émet une chaîne de caractères et qu'il ferme la liaison. Le serveur affiche alors à l'écran la chaîne de caractères reçue.

```
#include <cc++/socket.h>
#include <iostream>
#include <string>
#include <map>
#include <iomanip>
#include <cstdlib>

#define PORT 2000
```

```
// La classe TCP permet d'utiliser la connexion comme
// un stream C++ classique.
class TCP : public ost::TCPStream {
public :
      TCP(void) : ost::TCPStream() {}
      virtual ~TCP(void) {}

      std::ostream& Out(void) {return *tcp();}
      std::istream& In(void)  {return *tcp();}
};


int main(int argc, char * argv[]) {
      char name[100];
      ost::TCPSocket * socket_ecoute;
      TCP * connexion;

      // Création du socket d'écoute :

      sprintf(name, "0.0.0.0:%i", PORT);
      socket_ecoute = new ost::TCPSocket(name);

      // Attention de connexion d'un client :

      while(socket_ecoute->isPendingConnection()) {
            std::string data;
            connexion = new TCP;
            connexion->connect(*socket_ecoute);     // Accepte le client.
            connexion->In() >> data;
            std::cout << data << std::endl;
            delete connexion;
      }
      delete socket_ecoute;
}
```

Notez bien le rôle distinct des objets « socket_ecoute » (de type std::TCPSocket) et « connexion » (de type TCP). Le premier sert à recevoir l'information qu'un client s'est connecté, et le second sert à communiquer effectivement avec ce client. Lorsque la connexion avec ce client est coupée, l'objet « connexion » est détruit, mais l'objet « socket_ecoute » est toujours disponible pour accepter la connexion d'un nouveau client.

3.2 Exemple de programme client

```
#include <cc++/socket.h>
#include <iostream>
#include <string>
#include <map>
#include <iomanip>
#include <cstdlib>

#define PORT 2000
```

```
// La classe TCP permet d'utiliser la connexion comme
// un stream C++ classique.
class TCP : public ost::TCPStream {
public :
      TCP(void) : ost::TCPStream() {}
      TCP(char * name) : ost::TCPStream(name) {}
      virtual ~TCP(void) {}

      std::ostream& Out(void) {return *tcp();}
      std::istream& In(void)  {return *tcp();}
};

int main(int argc, char * argv[]) {
      TCP * socket;
      char nom[100];

      if (argc <= 2) {
            std::cout << "Appel par : client <nom> <message>" << std::endl;
            exit(1);
      }

      sprintf(nom, "%s:%i", argv[1], PORT);
      socket = new TCP(nom);
      if (! socket->isConnected()) {
            std::cout << "Erreur de connexion" << std::endl;
      } else {
            socket->Out() << argv[2];
      }
      delete socket;
}
```

Notez le test « isConnected » qui permet de vérifier à la connexion que le serveur existe ou non. En cas de tentative de connexion sur un serveur qui n'existe pas ou qui ne répond pas, le programme affiche un message d'erreur.

3.3 Téléchargement des exemples

Vous pouvez télécharger ces exemples à l'adresse suivante :

http://www.metz.supelec.fr/metz/personnel/gutzwiller/_TUTOS_98763456_/20100908/Reseau/ReseauTCP.zip