Programmer en C++ avec : PostgreSQL & Visual C++

Par :
Mohamed Hédi ZAHER
hedi@atu2l.org

Auteur : Mohamed Hédi ZAHER

décembre 2002 – première version

Table des matières

1 - Prologue

Les qualités intrinsèques de PostgreSQL comme Système de Gestion de Bases de Données [S.G.B.D.] font qu'on l'adopte souvent lorsqu'on exige ou on ne peut se passer de ces deux plus que sont l'aspect orienté objet et le service PostGIS de Bases de Données Géographiques.

La redondance des cas de figures où des développeurs sont amenés à se servir de PostgreSQL dans des plates-formes propriétaires font penser à la symbiose entre cet SGBD et les Environnement de Développement Intégrés [E.D.I.] .

L'objectif de ce document est de traiter le cas de la programmation C++ avec Visual C++ comme E.D.I. et PostgreSQL. Inutile de rappeler l'inexistence de support « commercial » pour ce thème.

Ce document ne traitera pas l'utilisation d'intermédiaires du type pilote ODBC mais bien la programmation C++ en mode natif de la communication avec la base de données.

2 – Problématique

Pour répondre à l'objectif de ce document, on traitera un cas de figure simple et concret qui permet d'aborder la problématique de la cohabitation Visual C++ et PostgreSQL d'une manière pragmatique.

Le but est qu'une lecture de ce document permettra à un développeur ayant une première expérience en Visual C++ de s'attaquer immédiatement à la programmation avec PostgreSQL en mode natif pour exploiter ses spécificités.

Sans s'attarder sur les particularités du S.G.B.D. ni celles de l'E.D.I. , on se propose de développer une première application.

L'objectif est considéré atteint dès qu'un programme écrit en C++ avec Visual C++ arrive à établir une connexion en mode natif avec une base de données PostgreSQL, et échanger des requêtes SQL ; moment à partir duquel les difficultés ne résident plus dans la cohabitation des deux outils, mais dans la propre utilisation de chacun parmi eux.

Le développement se fait en utilisant une seule machine sous un système d'exploitation Microsoft Windows [95/98/Me/NT4.0/2000/XP]. L'application générée tourne dans un environnement similaire.

3 - Processus de développement

On suppose déjà installés sur une machine dédiée un système d'exploitation du type Windows ainsi que l'E.D.I. Visual C++ 6.0.

Le processus de développement de notre application passe par les étapes :

1. Installer Cygwin, un émulateur Unix sous Windows.
2. Installer CygIPC, un utilitaire nécessaire à l'exécution de PostgreSQL sous Cygwin.
3. Installer PostgreSQL.
4. Mettre en œuvre LibPQ++, une bibliothèque qui interface PostgreSQL pour une utilisation native en C++.
5. Importer LibPQ++ dans un projet Visual C++.
6. Utiliser LibPQ++ pour attaquer en natif PostgreSQL.

Chacune de ces étapes sera détaillée en un mode pas à pas.

4 - Cygwin

Cygwin est un environnement Unix pour Windows développé par Redhat. Cygwin est compatible avec toutes les versions de Windows depuis Windows 95 -inclus - à l'exception de Windows CE. Il couvre donc 95-98-Me-NT 4.0-2000-XP.

Il s'agit de deux composantes :

1. Un noyaux sous forme d'un DLL : cygwin1.dll. Ce DLL émule une couche Unix et fournit l'essentiel des fonctionnalités de l'API Unix.
2. Un ensemble d'outils, portés depuis Unix, qui fournissent le *'look & feel '* de GNU/LiNUX. Il s'agit essentiellement d'une collection de *shell*, compilateur, éditeur,…

L'installation de Cygwin nécessite le téléchargement du logiciel et l'exécution de l'assistant de l'installation. Le dit assistant change souvent. Les différences mineures qui atteindraient la procédure d'installation ne devraient pas altérer le bon fonctionnement de l'ensemble.

1. Chercher la version la plus récente sur www.cygwin.com et cliquer sur le lien *'install now'*.
2. Choisir d'enregistrer le fichier setup.exe dans un répertoire local temporaire.
3. Après téléchargement, lancer le dit fichier.
4. Un écran promotionnel se présente, cliquer sur *'next'*.
5. Choisir entre une installation depuis Internet et un téléchargement selon votre connexion. On traitera le cas d'un téléchargement. Cliquer alors sur *'Download from Internet'*.
6. Choisir un répertoire local temporaire pour télécharger les fichiers.
7. Choisir le type de connexion qui vous correspond.
8. Choisir la localisation du miroir qui vous convient.
9. Choisir les paquetages à installer. Une installation complète est recommandée.
10. A la fin du téléchargement, fermer et relancer setup.exe.
11. *'Next'*.
12. Choisir *'Install from Local Directory'* pour une installation depuis les paquetages téléchargés. Préciser le répertoire temporaire où sont enregistrés les fichiers déjà téléchargés.

13. Choisir le répertoire destination de l'installation. Ce répertoire devient alors racine de l'environnement Unix.
    A partir de là, ce répertoire sera désigné CYGWINHOME/ .
14. S'assurer que *'Local Package Directory'* pointe le répertoire contenant les paquetages téléchargés.
15. Sélectionner les paquetages à installer.
16. Fin de l'installation.

Pour vérifier le succès de l'installation, lancer le 'shell' de Cygwin via le menu démarrer et taper :
ls
Se présente alors la liste du contenu du répertoire CYGWINHOME/ .

Plusieurs miroir d'installation de Cygwin commence à proposer PostgreSQL. Dans ce cas, la mise en œuvre du S.G.B.D. devient intuitive et facile. Sauter le chapitres : 6 - PostgreSQL si c'est le cas.

Un support en anglais pour Cygwin est disponible à :
www.cygwin.com/ml/cygwin/
Un support en anglais dédié à Cygwin et PostgreSQL est disponible aussi à :
http://archives.postgresql.org/pgsql-cygwin/

5 - CygIPC

CygIPC est un utilitaire pour Cygwin requis pour l'exécution de PostgreSQL. CygIPC fournit la mémoire partagée, les sémaphores et le support de la communication inter-processus *via* les messages pour l'environnement d'émulation Unix Cygwin.

CygIPC est disponible sur plusieurs miroir dont :

http://www.neuro.gatech.edu/users/cwilson/cygutils/V1.1/cygipc/

Le binaire exécutable y est aussi. Il faut noter que les versions de CygIPC antérieures à 1.04 ne fonctionnent pas correctement.

Pour mettre en œuvre CygIPC :

1. Télécharger la dernière version du binaire, par exemple cygipc-1.10-1.tar.gz2, et enregistrer la dans le répertoire /CYGWINHOME.
2. Lancer le *shell* de Cygwin, extraire le contenu de l'archive téléchargé dans le répertoire CYGWINHOME/. Ceci placera le fichier ipc-deamon.exe, entre autres, dans CYGWINHOME/usr/local et ou ses sous-répertoire.

   Plusieurs paquetages de CygIPC sont distribués sous forme de .tar.gz2. Ils sont alors compressés avec bzip2, utilisant un algorithme plus puissant que gzip et zip, et produisant alors des archives moins encombrants en général.

   Bzip fait partie de la distribution officielle de Cygwin. bunzip2.exe serait alors à CYGWINHOME/bin.

   Pour installer alors le binaire pré-compilé de CygIPC, taper dans le shell de Cygwin :

   ```
   cd /
   bunzip2 –c <chemin vers l'archive>/<l'archive> | tar -xvf
   ```

   Dans notre cas, ceci revient à :

   ```
   bunzip2 –c /cygipc-1.10-1.tar.gz2 | tar –xvf
   ```

3. Tester la bonne installation de CygIPC en tapant dans le *shell* :

   ```
   ipc-daemon &
   ```

   Si tous va bien, le *shell* retourne le numéro de processus attribué à CygIPC, sans afficher de messages d'erreurs.

   Noter que l'instruction *'&'* ordonne Cygwin de lancer la commande en arrière plan, et permet de garder la main sur le *shell*.

6 - PostgreSQL

L'installation de PostgreSQL pourrait être incluse dans celle de Cygwin [*cf.* chap 4]. Ce chapitre détaillera une installation à part entière. Attention, ceci n'est pas valable pour une mise à jour, si l'installation de Cygwin incluait un version antérieure de PostgreSQL.

L'installation de PostgreSQL sous Cygwin ne diffère en rien de celle sous un LiNUX ou Unix-like standard. L'objet de ce chapitre est de détailler le nécessaire pour une utilisation avec Visual C++ par la suite. Le fichier INSTALL inclus dans la racine des archives de PostgreSQL explicite plus les procédures d'installations et mises à jour.

Pour mettre en œuvre PostgreSQL sous Cygwin :

1. Se procurer les sources de la dernière version de PostgreSQL dans le cdrom d'une distribution de LiNUX ou à :
   www.postgresql.org
2. Décompresser l'archive dans CYGWINHOM/, pour ce faire taper dans le *shell :*
   unzip -c postgresq-7.2.3.tar.gz | tar -xvf
   ce qui crée le répertoire CYGWINHOME/postgreSQL-7.2.3/
3. Se rendre dans ce répertoire en tapant :
   cd postgresql-7.2.3

gmake, l'utilitaire make de GNU est utilisé. Comme dans Cygwin, gmake est souvent disponible avec la commande make du *shell*. Sous d'autres Unix-like, la commande est gmake. Aucune différence fondamentale n'existe. Il est conseillé d'utiliser la version 3.76.1 ou postérieure de gmake.
Pour vérifier la version de gmake deCygwin, taper :
make --version

Taper successivement dans le *shell* :

4. ./configure

   Cette commande exécute le scripte de configuration du fichier configure. Les valeurs incluses par défault suffisent en général.

   Vérifier simplement l'existence de la ligne :
   --with-CXX

Si elle n'est pas disponible, ou si elle est remplacée par --without-CXX, la rétablire. Il s'agit de la commande ordonnant la compilation et l'installation de l'interface PostgreSQL pour C++.

5. make

Cette command construit PostgreSQL. Elle prend entre 5 et 30 minutes d'éxécution. La dernière ligne retournée doit être :

All of PostgreSQL is successfully made. Ready to install.

6. make check

Cette commande doit effectue les tests de non régression et doit retourner une valeur concluante.

7. make install

Cette commande crée les répertoire nécessaire et installe les fichiers de PostgreSQL. Elle installe aussi les fichiers d'entête, dits *headers*, nécessaires au développement d'application côté client.

Dans un contexte où on désirerait développer coté serveur, éventuellement des fonctions en C par exemple, il serait conseillé d'installer tous les fichiers d'entête en tapant :

make install-all-headers

8. make clean
   make distclean

La première commande supprime les fichiers temporaires créés lors de l'installation, en épargnant les fichiers de configuration crées par ./configure. La seconde supprime tous et restore le répertoire source à son état originel. Elle est conseillée si, après installation, on décide changer des paramètres de configuration dans le cas d'une configuration erronée par exemple.

9. Pour désinstaller complètement PostgreSQL, taper :
   make uninstall

10. Ne pas oublier de rappeler au système comment trouver les nouvelles bibliothèques installées en tapant :
    LD_LIBRARY_PATH=/usr/local/pgsql/lib
    export LD_LIBRARY_PATH
    C'est particulièrement nécessaire lors de l'obtention de messages d'erreurs du type :
    psql: error in loading shared libraries
    libpq.so.2.1: cannot open shared object file: No such file or directory
    Si le système le permet, préférer :
    /sbin/ldconfig /usr/local/pgsql/lib

7 – LibPQ++

Libpq++ est la bibliothèque C++ client de PostgreSQL. Elle est couramment utilisée dans le développement client PostgreSQL en C++ sous les Unix-like.

Libpq++ est disponible en téléchargement à l'emplacement :
http://developer.postgresql.org/cvsweb.cgi/pgsql-server/src/interfaces/libpq%2b%2b/Attic/ ,
ou avec la distribution de PostgreSQL.

Bien que PostgreSQL soit indisponible en natif sous Windows, sa biblithèque C++ libpq++, ainsi que son terminal interactif psql peuvent être compilés et exécutés normalement sous les systèmes d'exploitation propriétaires de Microsoft.

Win32.mak, un makefile spécifique est inclus dans la distribution de la source de libpq, est particulièrement destiné pour Visual C++ et apparaît incompatible avec d'autres E.D.I. . Une compilation manuelle est possible et des essais sont en cours pour une adaptation à Borland C++Builder.

Pour mettre en œuvre Libpq++ sous Windows, l'utilitaire *nmake* de Visual C++ est utilisé en mode ligne de commande.

Pour ce faire :

1. Récupérer la distribution de la source de Libpq++ en la téléchargeant.

2. Se rendre dans le répertoire de la source.

3. Taper :
   nmake -f win32.mak ,
   ce qui suppose que les binaires de Visual C++ sont pointés par la variable de l'environnement PATH.

   Si le système répond par une erreur semblable à « fichier ou command introuvable » prendre le soin de taper :
   <chemin de VC++>\nmake –f win32.mak ,
   ou <chemin de VC++> pointe jusqu'au répertoire des binaires contenant nmake.

4. Sont alors générés les fichiers :
   interfaces\libpq++\Release\libpq++.dll
   la bibliothèque de liaison dynamique, ou DLL, d'interfaçage,

   interfaces\libpq++\Release\libpqdll++.lib
   à importer pour lier les programmes développés à libpq++.dll, et
   interfaces\libpq++\Release\libpq++.lib
   la bibliothèque statique d'interfaçage.

5. Le seul fichier nécessitant réellement une installation pour être utilisé est libpq++.dll. Sur la machine servant au développement, tout comme les machines sur lesquelles les applications développées tourneront, il doit être placé dans son emplacement dédié. Il s'agit généralement du répertoire WINNT\SYSTEM32 pour les systèmes sous Windows NT4.0/2000/XP, et du répertoire WINDOWS\SYSTEM pour Windows 95/98/Me.
   Si le fichier est installé par la suite via un assistant d'installation d'application lors de la distribution, prendre le soin d'effectuer un test de version. La ressource VERSIONINFO incluse dans le *DLL* est prévu pour ce fait. Une version plus récente ne doit pas être écrasée.

6. Pour poursuivre le développement sur la machine servant à cette compilation de libpq++, ajouter les sous-répertoires src\includes et src\interfaces\libpq++ du dossier source de libpq++ dans le chemin d'inclusion de la configuration du compilateur. Sinon, les copier dans le répertoire *includes* de Visual C++.

7. Les bibliothèques ainsi générées peuvent être utilisées pour développer en C++ sous Windows. Pour faire appel réellement à libpq++.dll, il suffit d'inclure libpqdll++.lib dans le projet de l'E.D.I. utilisé.

8. Particulièrement sous Visual C++, ajouter libpqdll++.lib dans le projet en cliquant avec le bouton droit. Choisir ajouter.

8 - Programmation en C++ avec libp++

En supposant ouvert un projet d'application en C++ avec Visual C++ 6.0, et inclus dans ce projet l'import de la bibliothèque dynamique via libpq++dll.lib, ou la bibliothèque statique libpq++.lib , ce chapitre détaillera un petit programme en C++, faisant justement appel à libpq++ pour communiquer en natif avec une base de données PostgreSQL.

Cette application est inspirée de celle distribuée par Attic, et servant comme exemple d'initiation, avec le copyright :
Copyright (c) 1994, Regents of the University of California .

Ce programme effectue une connexion avec une base de données sous PostgreSQL *via* libpq++. Il se met en boucle pour saisir de l'utilisateur des requêtes SQL, qu'il communique au backend PostgreSQL. Il affiche alors les résultats retournés par la base de données, ou des messages d'erreurs. En tapant < E N T R E E > sans saisir de requêtes, le programme se termine.

Pour terminer la première application en C++ :

1. Créer dans le projet de visual C++ un fichier source, s'il n'en contient pas un.
2. Dans la partie entête du fichier, ajouter facultativement :

```
/*****************************************************************
*       no1.cpp : Premier programme en C++ exploitant libpq++     *
*                                                                 *
*       Petite boucle interactive où des requêtes peuvent être saisies *
*       et adressées au backend.                                  *
*       copyright© 2002, ATu2L.org                               *
*       hedi@atu2l.org                                            *
*                                                                 *
*       d'après un programme distribué par ATTIC:                 *
*       Copyright (c) 1994, Regents of the University of California *
*****************************************************************/
```

3. Dans la partie des *includes*, ajouter impérativement les deux imports :

```
#include <iostream.h>
```

```
#include <libpq++.h>   /* remplacer par #include "[chemin]/libpq++.h " si VC++ ne reconnaît pas le chemin de libpq++, revoire l'étape 6 du chapitre 7 */
```

4. Remplacer le corps du programme par :

```
int main() // Procedure principale, point d'entrée au programme
{

    /* 1 : Ouverture de la connexion avec la base de donnée */

    /* Instanciation d'une base de données */
    PgDatabase data("dbname=testbase"); // nom de la base de données = testbase
                                        // à remplacer par le nom d'une base
                                        // existante dans le SGBD PostgreSQL

    /* Vérification du bon établissement de la connexion */
    if ( data.ConnectionBad() )
    {
    // Echec de la connexion
    cout << "Echec de le connexion, BD inexistante ou SGBD en panne..." << endl
    << " message d'erreur retourné : " << data.ErrorMessage() << endl;
    return 1;
    }
    else
    // Succès de la connexion
        cout << "Connexion établie...  saisire vos requêtes:" << endl;

    /* 2: Obtention et traitement des requêtes */

    // Exécution des requètes utilisateur en boucle et en intéractif
    // Arrêt et terminaison du programme si l'utilisateur ne saisisse rien avant
    // de taper <ENTREE>

    ExecStatusType etat;          // état de l'exécution
    string buffer;                // buffer pour les commandes utilisateur
    int fin = 0;                  // booléen qui vaut 1 pour terminer la boucle

    while (!fin)
    {
        cout << "> ";             // prompt
        cout.flush();
        getline(cin, buffer);     // saisie de la requête utilisateur
        if ( buffer != "" )       // test du buffer
                // buffer non vide -> requête utilisateur à exécuter
                if ( (etat = data.Exec( buffer.c_str() )) == PGRES_TUPLES_OK )
                        // test de l'etat de l'exécution de la requête

                        data.DisplayTuples(); // affiche le résultat retourné
                else
                        // requête retournant une erreur
                        cout << "Aucun tuples en retour..." << endl
```

```
                        << "Etat: = " << status << endl
                        << "Erreur retournée: " << data.ErrorMessage() << endl;
                        // affichage de l'erreur
            else
            // pas de requête saisie -> fin de programme et sortie
                  fin = 1;
      }
      return 0;
} // Fin main()
```

5. Sauvegarder le projet et le fichier sources.

6. Compiler, lier.

7. Ne pas oublier de lancer Cygwin, CygIPC et PostgreSQL avec l'option –i avant d'exécuter.

   La commande :
   Postmester –D –i <repertoire de stockage des données>
   lance lance le SGBD en activant la connexion via TCP/IP sur le socket 5432, nécessaire au bon fonctionnement de *psql* et libpq++.

9 - Epilogue

A ce stade, et en passant par les étapes énoncées dans 3 – Processus de développement, une application en C++ tourne correctement sous Windows, en se connectant et exécutant nativement des requête pour PostgreSQL.

Ceci répond largement aux objectifs de ce document.

Il est à noter :

1. PostgreSQL n'est pas destinée à Windows. Les portages réalisés s'apparente au bidouillage. L'exécution en émulation Unix affecte les performances.
   En terme de consommation mémoire, charge CPU, temps de réponse et performance du système de fichier, cette combinaison est loin d'être l'optimale et n'est pas faite pour des serveurs à grande charge. Elle ne sert qu'à des utilisations restreintes, de recherches par exemple,…

2. Libpq++ est en train d'être remplacée par libpqxx, une autre bibliothèque pour interfacer PostgreSQL avec C++. Des portages de libpqxx sous Windows existent, particulièrement réalisées pour Visual C++ 7, ou .Net.
   La mise en œuvre est beaucoup plus complexe, mais aussi bien faisable.
   Les source de libpqxx sont distribuées à :
   http://gborg.postgresql.org/project/libpqxx/download/download.php
   Consulter le fichier INSTALL.txt du sous répertoire win32, inclus dans ces sources, pour plus de détail.
   Des exemples de programmes existent aussi sur Internet.
   Libpq++ et Visual C++ 6 sont de loin les plus répondus, du moins pour le moment, ce qui a fait pencher ce document vers son approche.

3. Egalement, une bibliothèque pour le C existe. Il s'agit de libpq. Son portage sous Windows se fait sensiblement comme libpq++. Une fichier win32.mak est prévu pour cette perspective. La mise en œuvre de libpq avec Visual C++ se fait en remplaçant systématiquement le terme libpq++ par libpq dans le 7ème chapitre de ce document.
   Des illustrations de l'utilisation de libpq sont disponibles à :
   http://www.postgresql.org/idocs/index.php?libpq-example.html

Références

La documentation de l'installation de PostgreSQL-7.2.3 pour Unix :
www.postgresql.org

La documentation en ligne de PostgreSQL :
www.postgresql.org/idocs

La communauté des développeurs PosgreSQL :
developper.posgresql.org

PostgreSQL Database Windows Setup FAQ :
www.ejip.net

l'équipe de développement Gborg :
gborg.postgresql.org

Crédits

Copyright © 2002, ATu2L.org .
Association Tunisienne des Logiciels Libres,
www.atu2l.org
28 Avenue Habib Bourguiba, Tunis
TUNISIE

Mohamed Hédi ZAHER
hedi@atu2l.org