Systemy informatyczne w medycynie

Ćwiczenie 2

Instrukcja do laboratorium

semestr letni 2022

ODBC - obsługa bazy danych z poziomu języka c++

1. Konfiguracja systemu komputerowego

Krytyczną i za razem jedną z trudniejszych części tworzenia medycznego systemu informatycznego jest konfiguracja komputerów, na których ten system będzie uruchomiony. Trudność bierze się z faktu, że proces ten często wymaga analizowania naraz wielu aspektów działania systemu operacyjnego (np. działanie kodu może zależeć od tego co znajduje się w plikach bibliotek, to co jest w bibliotekach może zależeć od kolejności ich instalacji i tak dalej i tak dalej). W efekcie podczas stawiania pierwszych kroków programista często napotyka na błędy. Umiejętność samodzielnego radzenia sobie z błędami jest najważniejszym doświadczeniem, które student powinien wynieść z pracy nad projektem.

1.1. Możliwe błędy i działanie w ich wypadku

Należy być świadomym faktu, że praktycznie każdy błąd został już co najmniej raz popełniony przez kogoś kto napisał o tym w Internecie. Ogólna zasada postępowania w przypadku błędów:

- Skopiować treść błędu z terminala
- Wkleić treść w wyszukiwarce Google
- Przejrzeć kilka pierwszych wyszukanych propozycji (dobrze, gdy są powiązane ze stroną stackoverflow.com)
- Wybrać najbardziej sensowne rozwiązanie i spróbować ją zastosować
- Jeżeli rozwiązania jakiegoś problemu nie ma w Internecie oznacza to, że najprawdopodobniej ten problem nie istnieje, a my próbujemy zrobić coś, co zupełnie nie ma sensu. Dlatego w takim wypadku należy spróbować zastanowić się, czy na pewno wiemy, co chcemy zrobić.

Przykład:

Często występującym błędem przy instalacji oprogramowania jest problem z zajętością aplikacji apt. Alert błędu wygląda wtedy w następujący sposób:

```
sim20@sim20-VirtualBox:~$ sudo apt install biuld-essential
E: Could not get lock /var/lib/dpkg/lock-frontend - open (11: Resource temp
orarily unavailable)
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend),
is another process using it?
```

Gdy wyszukamy w Internecie treść błędu już na pierwszej proponowanej stronie dowiemy się, że najczęściej przyczyną tego błędu jest działająca w tle aplikacja “daily-update”, będąca unixowym odpowiednikiem “Windows update”. Czasami po uruchomieniu komputera wyszukuje ona w Internecie najnowszych aktualizacji i stara się je instalować, niestety rezerwując sobie przy tym użycie aplikacji apt. Aby się upewnić można wylistować, które aplikacje używają obecnie apt:

```
ps aux | grep -i apt
```

```
sim20@sim20-VirtualBox:~$ ps aux |grep -i apt
root       729  0.0  0.0   4624   744 ?        Ss   20:37   0:00 /bin/sh /usr/lib/apt/apt.sys
temd.daily update
root       760  0.0  0.0   4624  1548 ?        S    20:37   0:00 /bin/sh /usr/lib/apt/apt.sys
temd.daily lock_is_held update
sim20     2733  0.0  0.0  22752   968 pts/0    S+   21:04   0:00 grep --color=auto -i apt
```

Rozwiązaniem tego problemu jest odczekanie kilku minut aż proces daily-update się zakończy, bądź uruchomienie ponownie Linuxa. Można również w ustawieniach systemowych wyłączyć automatyczne wyszukiwanie aktualizacji, co pozwoli uniknąć tego problemu w przyszłości.

1.2. Instalacja kompilatora i bibliotek

Instalacja gcc, będącego kompilatorem języka c (kompilator jest niezbędny do programowania):

```
sudo apt-get install build-essential
```

Instalacja może potrwać kilka minut. Aby sprawdzić, czy odbyła się pomyślnie, można wyświetlić dostępną wersję gcc przy pomocy komendy:

```
gcc -v
```

Efekt `gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)`

Instalacja bibliotek programistycznych w języku c do obsługi PostgreSQL:

```
sudo apt-get install libpq-dev
```

```
sudo apt-get install postgresql-server-dev-10
```

1.3. Instalacja ODBC

ODBC (ang. **O**pen **D**ata**B**ase **C**onnectivity – otwarte łącze baz danych) jest interfejsem pozwalający programom łączyć się z systemami zarządzającymi bazami danych (w naszym wypadku z PostgreSQL). Instalacja odbywa się z pomocą terminala:

```
sudo apt-get install unixodbc unixodbc-bin unixodbc-dev
```

Instalacja dodatkowych modułów odbc specjalnie do postgresa:

```
sudo apt-get install odbc-postgresql
```

1.4. Konfiguracja ODBC

Konfiguracja ODBC polega na przygotowaniu dwóch plików konfiguracyjnych *odbc.ini* i *odbcinst.ini*, które w przypadku Ubuntu powinny znajdować się w folderze *etc*. Plik odbcinst.ini przechowuje ścieżkę do zainstalowanych na komputerze sterowników i w przypadku dobrze przeprowadzonej instalacji powinien był zostać automatycznie uzupełniony:

```
[PostgreSQL ANSI]
Description=PostgreSQL ODBC driver (ANSI version)
Driver=psqlodbca.so
Setup=libodbcpsqlS.so
Debug=0
CommLog=1
UsageCount=1

[PostgreSQL Unicode]
Description=PostgreSQL ODBC driver (Unicode version)
Driver=psqlodbcw.so
Setup=libodbcpsqlS.so
Debug=0
CommLog=1
UsageCount=1
```

Plik odbc.ini zawiera informacje o bazie danych, z którą ma łączyć się program poprzez odbc i należy wypełnić go samodzielnie według szablonu:

```
[alias]

Driver = PostgreSQL Unicode

Description = cokolwiek

Servername = localhost

Port = 5432

UserName = nazwaUżytkownika

Password = hasłoUżytkownika

Database = nazwaBazyDanych
```

Przykładowo:

```
[testODBC]
Driver = PostgreSQL Unicode
Description = Test DB
Servername = localhost
Port = 5432
UserName = sim20
Password = sim2020
Database = test
```

Edytowanie plików znajdujących się w folderze etc jest utrudnione, ponieważ można to robić jedynie z poziomu administratora. W tym celu najłatwiej uruchomić edytor tekstowy przy użyciu terminala. Przechodzimy do folderu etc przy pomocy komendy:

```
cd /etc
```

Następnie uruchamiamy edytor jako administrator komendą:

```
sudo gedit odbc.ini
```

```
sim20@sim20-VirtualBox:~$ cd /etc
sim20@sim20-VirtualBox:/etc$ sudo gedit odbc.ini
[sudo] password for sim20:
```

2. Pierwszy program w c++

Pusty plik tekstowy z rozszerzeniem cpp tworzymi i jednocześnie otwieramy w edytorze tekstu za pomocą komendy:

```
gedit program1.cpp &
```

Dzięki znakowi & na końcu komendy program gedit otworzy się w trybie "w tle" co umożliwi nam jednoczesną pracę w terminalu.

```
#include<iostream>
using namespace std;
int main(){
        cout<<"Scio me nihil scire"<<endl;
        return 0;
}
```

Zapisujemy plik [ctrl+s] i przechodzimy do terminala. Kompilujemy kod przy pomocy kompilatora g++ (jest to moduł kompilatora gcc dla c++) komendą o następującej formie:

```
g++ [kodProgramu.cpp] -o [plikWyjsciowy]
```

W naszym wypadku:

```
g++ program1.cpp -o program1
```

W efekcie w folderze Lab2 powstanie plik wywoływalny aplikacji o nazwie program1:

```
sim20@sim20-VirtualBox:~/Desktop/Lab2$ g++ program1.cpp -o program1
[2]+  Done                    gedit program2.cpp
sim20@sim20-VirtualBox:~/Desktop/Lab2$ ls
program1  program1.cpp
```

Aby uruchomiony stworzony program należy wywołać komendę w postaci:

./nazwaProgramu

Powinniśmy uzyskać następujący efekt:

```
sim20@sim20-VirtualBox:~/Desktop/Lab2$ ./program1
Scio me nihil scire
sim20@sim20-VirtualBox:~/Desktop/Lab2$
```

3. Komunikacja z bazą danych z poziomu języka c++

Gdy już przetarliśmy pierwsze szlaki z programowaniem w c++ możemy spróbować napisać bardziej złożony program, który umożliwi nam komunikację z bazą danych. Do celów demonstracyjnych założona zostanie prosta baza danych o nazwie *lab2*. Baza stworzona zostanie z poziomu użytkownika *sim20*, któremu przy tworzeniu zostało przypisane hasło *sim2020*:

```
sim20@sim20-VirtualBox:~/Desktop/Lab2$ createdb Lab2
sim20@sim20-VirtualBox:~/Desktop/Lab2$ psql -d Lab2
psql (11.8 (Ubuntu 11.8-1.pgdg18.04+1), server 10.12 (Ubuntu 10.12-0ubuntu0.18.0
4.1))
Type "help" for help.

Lab2=# CREATE TABLE kontynenty (id serial, nazwa varchar(20) not null);
CREATE TABLE
Lab2=# INSERT INTO kontynenty (nazwa) VALUES ('Azja');
INSERT 0 1
Lab2=# INSERT INTO kontynenty (nazwa) VALUES ('Europa');
INSERT 0 1
Lab2=# INSERT INTO kontynenty (nazwa) VALUES ('Ameryka');
INSERT 0 1
Lab2=# SELECT * FROM kontynenty;
 id |  nazwa
----+---------
  1 | Azja
  2 | Europa
  3 | Ameryka
(3 rows)

Lab2=#
```

Następnie dodany został nowy moduł w pliku odbc.ini, który umożliwi łączenie się z bazą Lab2 z poziomu języka c++:

```
[Lab2 ODBC]
Driver = PostgreSQL Unicode
Description = ODBC do Lab2
Servername = localhost
Port = 5432
UserName = sim20
Password = sim2020
Database = Lab2
```

Przejrzystą dokumentację funkcji biblioteki ODBC można znaleźć pod tym adresem:

- https://docs.microsoft.com/en-us/sql/odbc/reference/syntax/odbc-api-reference?view=sql-server-ver15

Poniżej przedstawiono przykładowy kod programu korzystającego z biblioteki ODBC w celu wywołania instrukcji `select`. Przy niektórych linijkach kodu umieszczono komentarz wyjaśniający jaki jest ich cel. Czerwonymi prostokątami oznaczono fragmenty kodu, zawierające informacje do jakiej bazy program ma się podłączyć i jakie zapytanie w języku sql ma wywołać, a także obsługę tego zapytania. Reszta kodu odpowiedzialna za przygotowywanie połączenia z bazą jest pewnego rodzaju szablonem i może wyglądać bardzo podobnie niezależnie od tego do jakiej bazy się podłączamy i co chcemy z nią zrobić.

Plik source.cpp z kodem tego programu został również zamieszczony w plikach przedmiotu.

```
#include<iostream> //biblioteka portow wejscia/wyjscia
#include<sql.h>
#include<sqlext.h> //dwie biblioteki do obslugi sql
#include<string.h> //bibloteka do obslugi lancuchow znakow

using namespace std;

int main(){ //poczatek glownej funkcji programu
   //laczenie sie z baz:
        //deklaracje zmiennych:
        SQLHENV henv = SQL_NULL_HENV;
        SQLHDBC hdbc = SQL_NULL_HDBC;
        SQLHSTMT hstmt = SQL_NULL_HSTMT; //uchwyty do laczenia sie z odbc
        //funkcje:
        SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
        SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER*)SQL_OV_ODBC3, 0);
        SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);
        SQLSetConnectAttr(hdbc, SQL_LOGIN_TIMEOUT, (SQLPOINTER)5, 0);

        SQLConnect(hdbc, (SQLCHAR*) "Lab2 ODBC", SQL_NTS, (SQLCHAR*) NULL, SQL_NTS, NULL, SQL_NTS);

        SQLAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt);

//obsluga instrukcji wysylanych do bazy danych:
        SQLCHAR strNazwa[20];
        SQLLEN lenNazwa=0, cId=0, lenId=0; //deklaracje zmiennych na ktore bobierane beda dane z bazy

        SQLExecDirect(hstmt,(SQLCHAR*) "SELECT id, nazwa FROM kontynenty", SQL_NTS);
        //wywolywanie dowolnej instrukcji sql, w tym wypadku SELECT

        SQLBindCol(hstmt, 1, SQL_C_USHORT, &cId,      2,  &lenId);
        SQLBindCol(hstmt, 2, SQL_C_CHAR,   &strNazwa,   20, &lenNazwa);
        //podpinanie pod kolumny zwracane przez select wybranych zmiennych (kolumna 1 to id, kolumna 2 to nazwa)

        SQLFetch(hstmt); //pobranie pojedynczego wiersza zwracanego po select

        cout<<"Id: "<<cId<<", Nazwa: "<<strNazwa<<endl; //wyswietlanie na ekranie pobranych wartosci

        return 0;
}
```

Kompilacja i wywołanie programu:

```
sim20@sim20-VirtualBox:~/Desktop/Lab2$ gedit program2.cpp
sim20@sim20-VirtualBox:~/Desktop/Lab2$ g++ program2.cpp -o program2 -lodbc
sim20@sim20-VirtualBox:~/Desktop/Lab2$ ./program2
encoding name too long
Id: 1, Nazwa: Azja
sim20@sim20-VirtualBox:~/Desktop/Lab2$
```

Należy pamiętać, aby przy kompilacji podlinkować bibliotekę ODBC dodatkowym argumentem `–lodbc`.

Jak widać program spowodował wyświetlenie tylko jednego wiersza z tabeli. Wyświetlenie większej liczby wierszy wymagałoby zastosowania w odpowiednim miejscu pętli:

```
//obsluga instrukcji wysylanych do bazy danych:
     SQLCHAR strNazwa[20];
     SQLLEN lenNazwa=0, cId=0, lenId=0; //deklaracje zmiennych na ktore bobierane beda dane z bazy
     SQLRETURN retcode;

     SQLExecDirect(hstmt,(SQLCHAR*) "SELECT id, nazwa FROM kontynenty", SQL_NTS);
     //wywolywanie dowolnej instrukcji sql, w tym wypadku SELECT

     SQLBindCol(hstmt, 1, SQL_C_USHORT, &cId,      2,  &lenId);
     SQLBindCol(hstmt, 2, SQL_C_CHAR,   &strNazwa,   20, &lenNazwa);
     //podpinanie pod kolumny zwracane przez select wybranych zmiennych (kolumna 1 to id, kolumna 2 to nazwa)

     while(retcode != SQL_NO_DATA) {
            retcode = SQLFetch(hstmt); //pobranie pojedynczego wiersza zwracanego po select
            cout<<"Id: "<<cId<<", Nazwa: "<<strNazwa<<endl; //wyswietlanie na ekranie pobranych wartosci
     }
```

```
sim20@sim20-VirtualBox:~/Desktop/Lab2$ g++ program2.cpp -o program2 -lodbc
sim20@sim20-VirtualBox:~/Desktop/Lab2$ ./program2
encoding name too long
Id: 1, Nazwa: Azja
Id: 2, Nazwa: Europa
Id: 3, Nazwa: Ameryka
Id: 3, Nazwa: Ameryka
```

4. Praca własna

W ramach przygotowania do projektu, przed następnymi zajęciami należy:

- Dodać w pliku odbc.ini nowe połączenie z bazą danych stworzoną w ramach laboratorium 1

- Napisać nowy program w c++, przy pomocy, którego będzie możliwe wywołanie komendy `select` z wybranej tablicy ze swojej bazy danych
- Napisać kod umożliwiający z poziomu języka c++ wprowadzenie przy pomocy komendy `insert into` danych do wybranej tablicy w swojej bazie danych