Полное руководство по сетевому программированию для разработчиков игр. Часть 1 (скучная). (3 стр)

Систематизируем то, что получилось

("Ээээ... Слышь, братан?! А о чем вы тута щас, типа, разговариваете, в натуре, я не понял?!")

Жизненный цикл нашего приложения пока не очень долго длится... Мы всего лишь создаем сокет, проверяем, все ли с ним ок, и затем уничтожаем его, попутно убирая мусор. Однако это основа основ всего нашего дальнейшего обучения. Каркас, который у нас получился, может быть использован в любом приложении, работа которого так или иначе связана с передачей данных на расстоянии. Да, мы пока еще ничего не знаем о параметрах, которые передаем функции socket(), и их значении и влиянии на работу приложения, но мы теперь знаем концептуальные вещи, которые помогут нам легко и быстро разобраться во всем. Теперь самое время взглянуть на общий код всего нашего каркаса. Сразу оговорюсь, что я намеренно не включил в него код для обработки ошибок. В случае возникновения ошибок мы просто сигнализируем о них и завершаем программу, со статусом завершения, равным 1. Я не стал пока создавать объектную структуру для нашего каркаса, потому что на данный момент нам нужна максимальная простота и ясность. К тому же я еще не строил наш каркас в Windows-приложение, поэтому компилировать листинг 1.04 win надо как консольное приложение.

```
// Listing 1.04 win

#pragma comment (lib, "ws2_32.lib");  // ищем нужную библиотеку

#include <iostream>
#include <winsock2.h>        // winsock2.h: typedef u_int SOCKET

using namespace std;
```

```
WORD     winsock_version;      // запрашиваемая версия winsock-интерфейса
WSADATA   winsock_data;      // сюда записываются данные о сокете
int     winsock_error;      // для проверки ошибок
SOCKET sd;            // наш дескриптор сокета

winsock_version = MAKEWORD (2, 0);    // задаем версию winsock

winsock_error = WSAStartup (winsock_version, &winsock_data);
if (winsock_error != 0)
{
  // здесь мы еще не можем использовать WSAGetLastError(),
  // потому что winsock // еще не инициализирован
  // говорим пользователю, что возникли проблемы и выходим
  cerr << "Could not initialize winsock" << endl;
  exit (1);
}
else
{
  // если инициализация прошла успешно, то пора создавать сокет
  sd = socket (PF_INET, SOCK_DGRAM, 0);
  if (sd == INVALID_SOCKET)
  {
    winsock_error = WSAGetLastError();
    cerr << "Could not create a socket. Error code: "
    << winsock_error << endl;
    exit (1);
  }
}

cout << "The socket application is up and running" << endl;

winsock_error = closesocket (sd);
if (winsock_error == SOCKET_ERROR)
```

```
      cerr << "Could not close the socket properly" << endl;
      exit (1);
    }

    winsock_error = WSACleanup ();
    if (winsock_error == SOCKET_ERROR)
    {
      winsock_error = WSAGetLastError();
      cerr << "Could not perform cleaning up" << endl;
      exit (1);
    }

    return 0;
  }
```

```c
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>

int main ()
{
  int sd = socket (PF_INET, SOCK_DGRAM, 0);
  if (sd == -1)
  {
    perror ("socket");
    exit (1);
  }

  printf ("The socket application is up and running\n");

  int error = close (sd);
  if (error == -1)
  {
    perror ("close");
    exit (1);
  }
  return 0;
}
```

В качестве домашнего задания можешь написать функцию, которая будет обрабатывать ошибки должным образом. Хммм... Я могу дать один совет. Всегда (!!!) проверяй значения, возвращаемые функциями, работающими с сетью. Дело в том, что сетевой код является самой нестабильной частью приложения, вследствие архитектуры сетевых протоколов и Internet в целом. Например, ты никогда не можешь быть уверен в том, что вовремя игры по сети у твоего соперника не отключится электричество в доме (или по нему не нанесут ядерный удар, хе-хе). В этом случае твое приложение не должно впадать в ступор, ожидая данных от него, а грамотно рассказать тебе, что произошло, что может быть причиной произошедшего, и предложить варианты дальнейших действий.

#OSI, #сокеты

18 мая 2003 (Обновление: 20 янв 2011)

Комментарии [4]