



Національний університет

водного господарства

та природокористування

Міністерство освіти і науки, молоді та спорту України

Національний університет водного господарства та
природокористування

Кафедра електротехніки та автоматики

043-27

Методичні вказівки та завдання

до виконання контрольної роботи

з дисципліни

“Програмування багаторівневих систем управління”

студентами за напрямом підготовки 6.050202 “Автоматизація та
комп’ютерно-інтегровані технології” заочної форми навчання

Рекомендовано методичною
комісією за напрямом
підготовки “Автоматизація та
комп’ютерно-інтегровані
технології”.

Протокол №1 від 19.10.2010



Національний університет

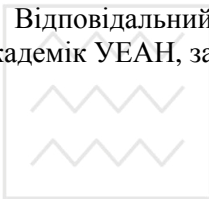
водного господарства

та природокористування

Методичні вказівки та завдання до виконання контрольної роботи з дисципліни “Програмування багаторівневих систем управління” студентами за напрямом підготовки 6.050202 “Автоматизація та комп’ютерно-інтегровані технології” заочної форми навчання / А.П. Сафоник. – Рівне: НУВГП, 2010. – 34 с.)

Упорядник: А.П. Сафоник, ст.в. кафедри електротехніки та автоматики.

Відповідальний за випуск: Б.О. Баховець, к.т.н., професор, академік УЕАН, завідувач кафедри електротехніки та автоматики.



водного господарства
та природокористування

© Сафоник А.П., 2010
© НУВГП, 2010



I. МЕТА ВИКОНАННЯ КОНТРОЛЬНОЇ РОБОТИ

Для того щоб навчитись використовувати теоретичні знання при розробці прикладних програмних пакетів – багаторівневих систем в реальному масштабі часу, а також транслювати, компонувати і відлагоджувати програми, як на мовах високого так і низького рівнів, методика вивчення дисципліни “Програмування багаторівневих систем управління” передбачає виконання контрольної роботи, завдання якої охоплюють принципи будови багаторівневих систем, роботу мікропроцесорної техніки в реальному масштабі часу, мікропроцесорні системи, операційні системи реального часу, програмні протоколи міжкомп’ютерного обміну, зв’язок мікропроцесорних систем з технологічними об’єктами в реальному масштабі часу.

В результаті виконання контрольної роботи, студент повинен навчитись:

- розробляти багаторівневі системи у вигляді прикладних програм;
- принципам роботи мікропроцесорної техніки в реальному масштабі часу;
- розробляти мікропроцесорні системи;
- застосовувати протоколи міжкомп’ютерного обміну інформацією;
- застосовувати зв’язок мікропроцесорних систем з технологічними об’єктами в реальному масштабі часу.

II. ЗАВДАННЯ НА КОНТРОЛЬНУ РОБОТУ

Завдання на контрольну роботу є індивідуальним для кожного студента групи і обирається згідно вказівок, наведених нижче.

Контрольна робота виконується у вигляді пояснювальної записки. Записку оформляють на листах формату А4.

При виконанні контрольної роботи необхідно дотримуватись таких вимог:

- на титульній сторінці вказуються зверху назви навчального закладу, факультету і кафедри;
- посередині – “Контрольна робота з дисципліни “Програмування багаторівневих систем управління”;
- нижче з правої сторони потрібно написати “Контрольну роботу виконав...” і далі прізвище, ім’я і по батькові, номер залікової книжки;
- прізвище та ініціали керівника проекту;
- внизу сторінки проставляють рік виконання;
- на другій сторінці записують завдання до роботи.

Пояснювальна записка має бути представлена у друкованому вигляді. Разом з друкованим виконанням має бути представлений електронний варіант роботи і супровідних матеріалів (пояснювальна записка, проект...).

Перше завдання – реферат обсягом 7-10 сторінок. Темі вибираються згідно списку студентів групи (можливе корегування за попередньою домовленістю з викладачем).



Друге завдання – розробка, опис та програмування одного з рівнів багаторівневої системи. Варіанти обираються за останньою цифрою залікової книжки.

Завдання №1.

1. Алгоритмізація розв'язку задач в реальному масштабі часу: фільтрація.
2. Алгоритмізація розв'язку задач в реальному масштабі часу: планування завдань.
3. Алгоритмізація розв'язку задач в реальному масштабі часу: розрахунок статичних характеристик.
4. Алгоритмізація розв'язку задач в реальному масштабі часу: порівняльна характеристика алгоритмів.
5. Коротка характеристика SCADA-систем (для об'єктно-орієнтованого програмування систем реального часу).
6. SCADA-система Citect.
7. Коротка характеристика операційної системи реального часу RTEMS.
8. Промислова система керування у жорсткому реальному часі LabVIEW Real-Time
9. Коротка характеристика операційної системи реального часу QNX.
10. Системи реального часу. Характерні риси інтерфейсів СРЧ, організації обчислювального процесу.
11. Співвідношення ОСРЧ і ОС універсального призначення. Особливості складу, інтерфейсів, організації обчислювального процесу. Специфіка реального часу.
12. Типовий склад ОС РЧ. Мікроядро. Параметризуємість конфігурації ОС РЧ.
13. Архітектура апаратних засобів СРЧ. Процесор/пам'ять/пристрій, безперервний потік команд/подій. Рівнева організація/інтеграція апаратних засобів.
14. Керування пам'яттю в ОС РЧ. Моделі пам'яті, механізми розподілу пам'яті. Віртуальна пам'ять в ОС РЧ.
15. Багатозадачність в ОС РЧ. Процес/задача/потік. Стан процесу/задачі/потіку. Користувач, сеанс. Планування і диспетчеризація. Методи диспетчеризації.
16. Синхронна й асинхронна взаємодія. Події, сигнали, переривання. Особливості обробки системних викликів з оброблювачів переривань (негайне виконання сервісу, затримане і відкладене виконання сервісу).
17. Особливості мережної передачі даних на прикладі фізичного рівня і рівня доступу CAN (Controller Area Network). Мережева передача з прив'язкою до часу (time-triggered protocol).
18. Засоби забезпечення відказостійкості СРЧ. Статичні і динамічні методи забезпечення виконання додатків – Rate Monotonic Algorithm, Earliest Deadline First).
19. Служба часу в ОС РЧ. Системний і астрономічний час. Основні задачі служби часу. Реалізація періодичного режиму роботи.



20. Задачне і подійне керування обчислювальним процесом. Що не витісняється, що витісняється, кругове (round-robin) планування.

Завдання №2.

1. Використовуючи одну з мов об'єктно-орієнтованого програмування (наприклад Builder, Delphi...) написати програму для прийому та передачі даних через СОМ-порт.

У розробленому додатку забезпечити:

- 1). Прийом та передачу даних через СОМ-порт.
- 2). Графічний інтерфейс програми.
- 3). Можливість вибору порту для передачі.
- 4). Можливість налаштування режиму передачі.

2. Використовуючи одну з мов об'єктно-орієнтованого програмування (наприклад Builder чи Delphi) написати програму для прийому та передачі даних через СОМ-порт використовуючи потоки.

У розробленому додатку забезпечити:

- 1). Прийом та передачу даних через СОМ-порт.
- 2). Графічний інтерфейс програми.
- 3). Можливість вибору порту для передачі.
- 4). Можливість налаштування режиму передачі.

3. Використовуючи одну з мов об'єктно-орієнтованого програмування (наприклад Builder чи Delphi) написати програму для сервера клієнт-серверної системи обліку газу.

У розробленому додатку забезпечити:

- 1). Створення бази даних.
- 2). Графічний інтерфейс програми.
- 3). Можливість під'єднання до бази даних по мережі (наприклад Ethernet).
- 4). Використовуючи одну з мов об'єктно-орієнтованого програмування

(наприклад Builder чи Delphi) написати програму для клієнта клієнт-серверної системи обліку газу.

У розробленому додатку забезпечити:

- 1). Запис отриманих даних у базу даних.
- 2). Графічний інтерфейс програми.
- 3). Можливість під'єднання до бази даних по мережі (наприклад Ethernet).

5. Використовуючи одну з мов об'єктно-орієнтованого програмування (наприклад Builder чи Delphi) написати програму для клієнт-серверної системи обліку газу.

У розробленому додатку забезпечити:

- 1). Запис та читання даних по мережі.
- 2). Графічний інтерфейс програми.



6. Використовуючи одну з мов об'єктно-орієнтованого програмування (наприклад Builder, Delphi...) написати програму для прийому та передачі даних через СОМ-порт.

У розробленому додатку забезпечити:

- 1). Прийом та передачу даних через СОМ-порт.
- 2). Графічний інтерфейс програми.
- 3). Можливість вибору порту для передачі.
- 4). Можливість налаштування режиму передачі.

7. Використовуючи одну з мов об'єктно-орієнтованого програмування (наприклад Builder чи Delphi) написати програму для прийому та передачі даних через СОМ-порт використовуючи потоки.

У розробленому додатку забезпечити:

- 1). Прийом та передачу даних через СОМ-порт.
- 2). Графічний інтерфейс програми.
- 3). Можливість вибору порту для передачі.
- 4). Можливість налаштування режиму передачі.

8. Використовуючи одну з мов об'єктно-орієнтованого програмування (наприклад Builder чи Delphi) написати програму для серверного додатку системи приємо-передавача.

У розробленому додатку забезпечити:

- 1). Створення бази даних.
- 2). Графічний інтерфейс програми.
- 3). Можливість під'єднання до бази даних по мережі (наприклад Ethernet).

9. Використовуючи одну з мов об'єктно-орієнтованого програмування (наприклад Builder чи Delphi) написати програму для клієнтського додатку системи приємо-передавача.

У розробленому додатку забезпечити:

- 1). Запис отриманих даних у базу даних.
- 2). Графічний інтерфейс програми.
- 3). Можливість під'єднання до бази даних по мережі (наприклад Ethernet).

10. Використовуючи одну з мов об'єктно-орієнтованого програмування (наприклад Builder чи Delphi) написати програму для клієнта клієнт-серверної системи обліку газу.

У розробленому додатку забезпечити:

- 1). Запис та читання даних по мережі.
- 2). Графічний інтерфейс програми.



III. Теоретичні відомості і методичні рекомендації до розв'язання завдань

Порядок виконання завдання 1, 6

1. Перш за все, необхідно запустити середовище Borland C++ Builder. Для цього зазвичай необхідно зайти в меню ПУСК → Программы → Borland C++ Builder 6 → C++ Builder 6 (рис. 1)

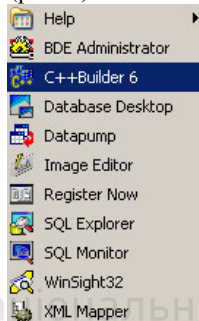


Рис. 1. Запуск C++ Builder із головного меню.

2. Після запуску середовища перед нами з'явиться 5 вікон:

- Головне меню середовища (рис. 2)



Рис. 2. Головне меню середовища.

- Вікно дерева об'єктів (рис. 3)

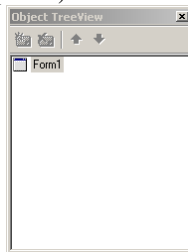


Рис. 3. Вікно дерева об'єктів.

- Вікно подій (рис. 4)

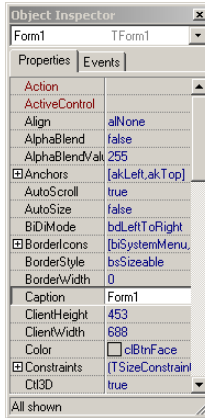


Рис. 4. Вікно подій.

- Вікно з макетом форми програми, яку ми розробляємо (рис. 5)

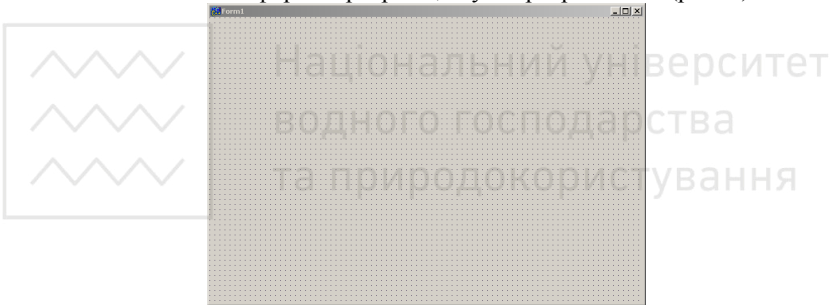


Рис. 5. Форма програми.

- Вікно з текстовим редактором, який призначений для написання коду програм додатку (рис. 6)

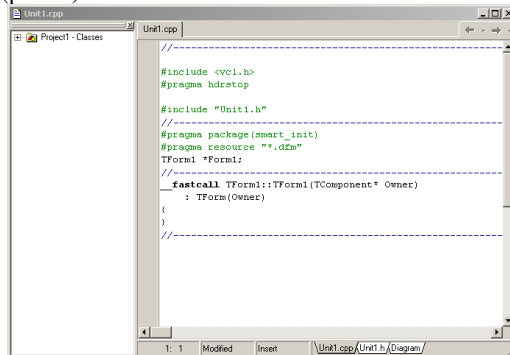


Рис. 6. Редактор для програмування.



3. Переіменуюмо заголовок основної форми програми. Для цього перейдемо у вікно **Object Inspector**, в полі **Caption** введемо «COM-порт» (рис.7).



Рис. 7. Назва заголовку форми.

4. Перейдемо до розробки графічного інтерфейсу. Для забезпечення можливості налаштування параметрів послідовного порту вручну необхідно розташувати графічні елементи **TEdit**, **TLabel**, **TButton**, приблизно так як показано на рисунку 8.

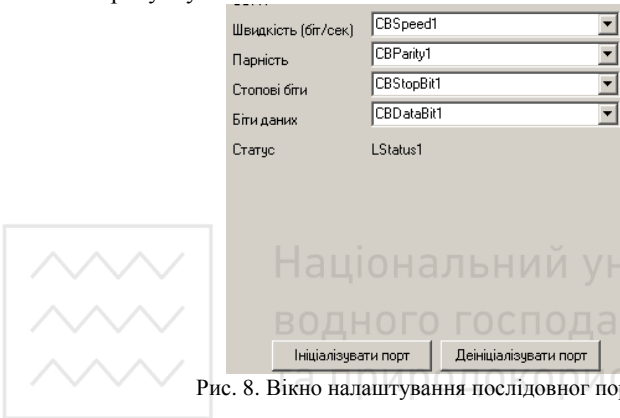


Рис. 8. Вікно налаштування послідовного порту.

В конструкторі форми напишемо наступний код, який буде ініціалізувати графічний інтерфейс:

```
AnsiString SpeedValues[14];
SpeedValues[0]="300";
SpeedValues[1]="600";
SpeedValues[2]="1200";
SpeedValues[3]="2400";
SpeedValues[4]="4800";
SpeedValues[5]="9600";
SpeedValues[6]="14400";
SpeedValues[7]="19200";
SpeedValues[8]="38400";
SpeedValues[9]="56000";
SpeedValues[10]="57600";
SpeedValues[11]="115200";
SpeedValues[12]="128000";
SpeedValues[13]="256000";
for(int i=0; i<14; ++i)
{
    Form1->CBSpeed1->Items->Add(SpeedValues[i]);
}
Form1->CBSpeed1->Text = SpeedValues[5];
```



```
AnsiString Parity[5];
Parity[0]="Чет";
Parity[1]="Нечет";
Parity[2]="Her";
Parity[3]="Маркер";
Parity[4]="Пробел";
for(int i=0; i<5; ++i)
{
    Form1->CBParity1->Items->Add(Parity[i]);
}
Form1->CBParity1->Text = Parity[2];
AnsiString StopBits[3];
StopBits[0] = "1";
StopBits[1] = "1.5";
StopBits[2] = "2";
for(int i=0; i<3; ++i)
{
    Form1->CBStopBit1->Items->Add(StopBits[i]);
}
Form1->CBStopBit1->Text = StopBits[0];
AnsiString DataBits[5];
DataBits[0] = "4";
DataBits[1] = "5";
DataBits[2] = "6";
DataBits[3] = "7";
DataBits[4] = "8";
for(int i=0; i<5; ++i)
{
    Form1->CBDataBit1->Items->Add(DataBits[i]);
}
Form1->CBDataBit1->Text = DataBits[4];
Form1->LStatus1->Caption = "Порт COM1 не ініціалізований";
```

5. Запрограмуємо кнопку «Ініціалізувати порт». Під ініціалізацією порту розуміємо наступне:

- визначення кількості наявних в системі послідовних портів (процедура описана на 7 – 8 сторінці методичних вказівок до лабораторної роботи №4);
- реалізація алгоритму вибору необхідного вільного порту;
- отримання дескриптора порту за допомогою функції **CreateFile** (описана на сторінці 8 методичних вказівок до лабораторної роботи №4);
- перевірка правильності отриманого дескриптору (необхідно перевірити чи не є дескриптор порту рівним **INVALID_HANDLE_VALUE**, якщо так, то необхідно прийняти відповідні міри по звершенню роботи програми, та інформуванні користувача);
- налаштувати порт використовуючи структуру **DCB** (процедура описана на 12 сторінці методичних вказівок до лабораторної роботи №4);
- виконати очищення черги послідовного порту за допомогою функції **PurgeComm** (13 сторінка до лабораторної роботи №4).



Перед тим як писати даний код, в заголовочному файлі форми, в розділі **public**, необхідно розмістити наступний рядок:

```
HANDLE hPort1;
```

Повністю код буде виглядати таким чином (приблизно):

```
TRegistry *Reg = new TRegistry;
TStringList *Ts = new TStringList;
AnsiString port;
AnsiString SpeedValue;
AnsiString DataBits;
AnsiString Parity;
AnsiString StopBits;
DCB dcb;
Reg->RootKey = HKEY_LOCAL_MACHINE;
AnsiString key = "HARDWARE\\\\DEVICEMAP\\\\SERIALCOMM";
Reg->OpenKey(key,false);
Reg->GetValueNames(Ts);
port = Ts[0];
SpeedValue = Form1->CBSpeed1->Text;
DataBits = Form1->CBDataBit1->Text;
Parity = Form1->CBParity1->Text;
StopBits = Form1->CBStopBit1->Text;
hPort1 = CreateFile(port.c_str(),GENERIC_READ|
GENERIC_WRITE,0,NULL,OPEN_EXISTING,
FILE_FLAG_OVERLAPPED,NULL);
if (port == "COM1")
{
    if ((hPort1 == INVALID_HANDLE_VALUE)||
        (hPort1 == NULL))
    {
        MessageBox(NULL,"Помилка ініціалізації COM1","Помилка",MB_ICONERROR);
        return;
    }
    if (!GetCommState(hPort1,&dcb))
    {
        MessageBox(NULL,"Помилка отримання параметрів
COM1","Помилка",MB_ICONERROR);
        return;
    }
}
if (SpeedValue=="110") dcb.BaudRate=CBR_110;
if (SpeedValue=="300") dcb.BaudRate=CBR_300;
if (SpeedValue=="600") dcb.BaudRate=CBR_600;
if (SpeedValue=="1200") dcb.BaudRate=CBR_1200;
if (SpeedValue=="2400") dcb.BaudRate=CBR_2400;
if (SpeedValue=="4800") dcb.BaudRate=CBR_4800;
if (SpeedValue=="9600") dcb.BaudRate=CBR_9600;
if (SpeedValue=="14400") dcb.BaudRate=CBR_14400;
if (SpeedValue=="19200") dcb.BaudRate=CBR_19200;
if (SpeedValue=="38400") dcb.BaudRate=CBR_38400;
if (SpeedValue=="56000") dcb.BaudRate=CBR_56000;
if (SpeedValue=="57600") dcb.BaudRate=CBR_57600;
```



```
if (SpeedValue=="115200") dcb.BaudRate=CBR_115200;
if (SpeedValue=="128000") dcb.BaudRate=CBR_128000;
if (SpeedValue=="265000") dcb.BaudRate=CBR_265000;
if (DataBits=="4") dcb.ByteSize=4;
if (DataBits=="5") dcb.ByteSize=5;
if (DataBits=="6") dcb.ByteSize=6;
if (DataBits=="7") dcb.ByteSize=7;
if (DataBits=="8") dcb.ByteSize=8;
if (Parity=="Чет") dcb.Parity=EVENPARITY;
if (Parity=="Нечет") dcb.Parity=MARKPARITY;
if (Parity=="Her") dcb.Parity=NOPARITY;
if (Parity=="Маркер") dcb.Parity=ODDPARITY;
if (Parity=="Пробел") dcb.Parity=SPACEPARITY;
if (StopBits=="1") dcb.StopBits=0;
if (StopBits=="1.5") dcb.StopBits=1;
if (StopBits=="2") dcb.StopBits=2;
if (port == "COM1")
{
    if (!SetCommState(hPort1,&dcb))
    {
        MessageBox(NULL,"помилка встановлення параметрів
COM1","Помилка",MB_ICONERROR);
        return;
    }
    if (!PurgeComm(hPort1,PURGE_TXCLEAR | PURGE_RXCLEAR))
    {
        MessageBox(NULL,"Помилка очищення черги
COM1","Помилка",MB_ICONERROR);
        return;
    }
    Form1->LStatus1->Caption="Порт COM1 ініціалізовано";
}
}
```

6 Далі необхідно запрограмувати кнопку для деініціалізації послідовного порту (кнопка «Деініціалізувати порт»). Мета деініціалізації полягає в скиданні дескриптора відкритого порту. Код приведений нижче:

```
if (port == "COM1")
{
    CloseHandle(hPort1);
    Form1->LStatus1->Caption="Порт COM1 деініціалізовано";
}
}
```

7. Перейдемо до реалізації процедури запису/зчитування даних з порта. Для цього спочатку реалізуємо графічний інтерфейс користувача, який надасть змогу проводити дану процедуру. Розмістимо та скомпонуємо (на вибір програміста) наступні елементи графічного інтерфейсу на формі програми: **TMemo** (даний елемент потрібен для відображення даних отриманих з COM-порту), **TEdit** (даний елемент потрібен для введення даних, які будуть передаватись через послідовний порт), а також 2 елементи **TButton** (елементи управління процесом прийому та передачі).



8. Реалізуємо тепер програмно процедуру запису/зчитування (детально вона описана на сторінці 15-17 методичних матеріалів до лабораторної роботи №4). Для організації зчитування даних з послідовного порту необхідно виконати наступні дії:

- перевести порт в режим прийому функцією **SetCommMask**;
- виконати процедуру зчитування за допомогою функції **ReadFile**.

Повністю даний код буде виглядати наступним чином (записати необхідно в вікно редактора, яке відкриється подвійного клацання мишкою по елементу **TButton**, який керує зчитуванням даних з порту):

```
if (!SetCommMask (hPort1, EV_RXCHAR))
{
    ShowMessage("ПОМИЛКА ВСТАНОВЛЕННЯ МАСКИ ПОРТУ");
};
COMSTAT CommStat;
DWORD dwError;
DWORD dwRead;
BYTE *Buf;
OVERLAPPED OverRead;
int i;
AnsiString STR;
OverRead.hEvent = CreateEvent(NULL, True, False, NULL);
if (OverRead.hEvent == Null)
    {ShowMessage("Помилка при спробі зчитування порту");}
if(!ClearCommError(hPort1,&dwError,&CommStat)) {
    MessageBox(NULL,"Помилка очищення порту", "Помилка",MB_ICONERROR);}
dwRead = CommStat.cbInQueue;
if (dwRead > 0)
    {
        Buf = new BYTE[dwRead];
        if (!ReadFile(hPort,Buf,dwRead,&dwRead,&OverRead))
            {
                MessageBox(NULL,"Помилка читання порту", "Помилка",MB_ICONERROR);
            }
    }
// В chReadBuff знаходяться прочитані байти
// Далі йде обробка отриманих байтів
for(i=0; i<dwRead; ++i)
    {
        STR=STR+char(Buf[i]);/* IntToStr(short(Buf[i])) + " "; */
    }
Form1->Memo1->Lines->Add(STR);
delete Buf;
}
```

//тут Мемо1 – об'єкт **TMemo** який використовується для відображення даних отриманих з СОМ-порту.

Процедура запису даних з порту схожа за принципом до зчитування, відмінність полягає у використанні функції **WriteFile**.

```
BYTE    chWriteBuff[1024];
OVERLAPPED  Sync;
DWORD    dwWrite;
```



```
AnsiString str_buf;  
int i;  
str_buf = Edit1->Text; //.c_str()-48;  
for(i=0;i<str_buf.Length();i++)  
{  
    chWriteBuff[i]=char(str_buf[i+1]);  
}
```

```
Sync.hEvent=CreateEvent(NULL,TRUE,FALSE,NULL);  
if (Sync.hEvent==NULL)  
{  
    ShowMessage("ПОМИЛКА СТВОРЕННЯ ПОДІЇ");  
};  
if (!WriteFile(hPort1, &chWriteBuff, str_buf.Length(), &dwWrite, &Sync))  
{  
    if (GetLastError() != ERROR_IO_PENDING)  
    {  
        ShowMessage("ПОМИЛКА ЗАПИСУ ДАНИХ");  
    }  
}
```

9. Створений проект необхідно відкомпілювати і запустити на виконання. Для компіляції модуля необхідно вибрати пункт меню **Project | Compile Unit** (або натиснути клавіші **Alt-F9**). Цей метод зручний для віднаходження синтаксичних помилок в написаному коді, і відбувається без запуску програми в режим виконання. Для того щоб запустити програму не в середовищі програмування необхідно її зібрати, для цього виконують команду **Project | Build**. Для відлагодження внутрішніх помилок в програмі, які пов'язані з синтаксисом програми, а з логікою і внутрішньою організацією алгоритму, зручно запускати програму з середовища розробки, для цього існує пункт меню **Run | Run (F9)**, в даному режимі можна переглядати стан змінних програми, викликати відладник, ставити контрольні точки в коді, а також переглядати асемблерний код програми.

Порядок виконання роботи і опрацювання результатів завдання 2, 7

Відриємо програму створену в минулому проекті. Для цього виконаємо команду **File | Open**, і в стандартному діалозі виберемо проект який створили минулого разу.

Створимо модуль для потоку. Для цього виберіть пункт меню **File | New | Other** для відкриття вікна створення нового модуля (рис. 9).

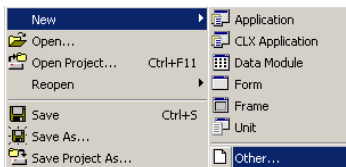


Рис. 9. Відкриття вікна створення нового модуля.



Знайдіть у цьому вікні на вкладці New пункт **Thread Object**. Виділіть його й натисніть кнопку ОК (рис. 10).

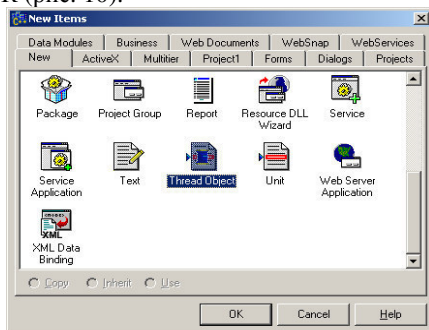


Рис.10. Вибір пункту **Thread Object**.

У результаті з'явиться вікно, показане на рис. 11. У цьому вікні потрібно вказати ім'я потоку який створюється . Назвемо потік – TReadThread.



Рис. 11. Задаємо ім'я потоку.

В результаті Builder C++ автоматично згенує 2 файли Unit2.cpp і Unit2.h . Перекочатись між ними можна натискаючи на вкладки як показано на рисунку 12.

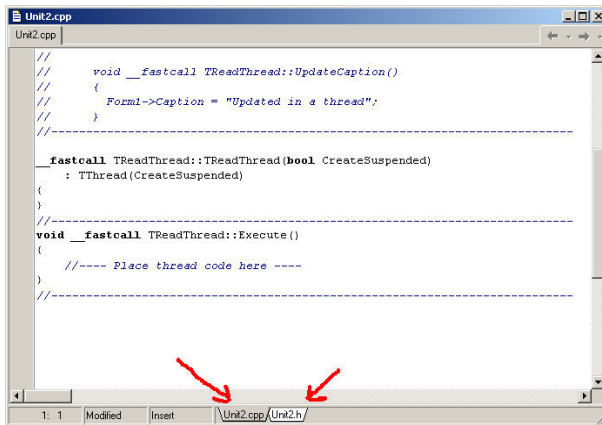


Рис.12. Вікно з текстом програми потоку.



Збережемо дані файли зі стандартними іменами (Unit2.cpp і Unit2.h) і перейдемо до розгляду коду.

Перейдемо на вкладку Unit2.h і розглянемо код:

```
//-----  
#ifndef Unit2H  
#define Unit2H  
//-----  
#include <Classes.hpp>  
//-----  
class TReadThread : public TThread  
{  
private:  
protected:  
    void __fastcall Execute();  
public:  
    __fastcall TReadThread(bool CreateSuspended);  
};  
//-----  
#endif
```

Перед нами клас **TReadThread** наслідуваний від стандартного класу **TThread**. **Builder** автоматично створив дві функції:

- 1) `__fastcall TReadThread(bool CreateSuspended);` – це функція-конструктор, яка використовується при створення нового об'єкту класу, поки на неї не потрібно звертати увагу, але до неї ще повернемося;
- 2) `void __fastcall Execute();` - в тілі даної функції ми прописуємо код який буде виконувати потік.

Перейдемо до розділу **private** і пропишемо змінні та методи:

```
DWORD dwRead;  
BYTE *Buf;  
void __fastcall DoRead(void);
```

Перейдемо тепер до файлу Unit2.cpp (права вкладка на рис12.):

```
__fastcall TReadThread::TReadThread(bool CreateSuspended)  
: TThread(CreateSuspended)
```

```
void __fastcall TReadThread::Execute()  
{
```

Перш за все нам необхідно підключити модуль форму основної програми, для цього необхідно написати **#include "Unit1.h"** одразу після **#include "Unit2.h"** (код виділений червоним шрифтом в наступному лістингу).

Всі рядки котрі розпочинаються з символу `//` закоментовані, тобто на них не звертаємо уваги. Поки нас цікавить функція **Execute()** (виділена жирним шрифтом). Як вже зазначалось вище, в тілі функції (між { }) ми пишемо код, який буде виконувати наш потік (тобто вставимо сюди код який ми писали для кнопки «Зчитати з порту» у попередньому завданні):



```
HANDLE hPort;
COMSTAT CommStat;
DWORD dwError;
DWORD dwRead;
BYTE *Buf;
OVERLAPPED OverRead;
int i;
AnsiString STR;
hPort = Form1->GetPortHandle();
if (!SetCommMask (hPort, EV_RXCHAR))
{
    ShowMessage("ПОМИЛКА ВСТАНОВЛЕННЯ МАСКИ ПОРТУ");
};
OverRead.hEvent = CreateEvent(NULL, True, False, NULL);
if (OverRead.hEvent == Null)
    {ShowMessage("Помилка при спробі зчитування порту");}
if(!ClearCommError(hPort,&dwError,&CommStat)) {
    MessageBox(NULL,"Помилка очищення порту", "Помилка",MB_ICONERROR);}
dwRead = CommStat.cbInQue;
if (dwRead > 0)
{
    Buf = new BYTE[dwRead];
    if (!ReadFile(hPort,Buf,dwRead,&dwRead,&OverRead))
    {
        MessageBox(NULL,"Помилка читання порту", "Помилка",MB_ICONERROR);
    }
}
// В chReadBuff знаходяться прочитані байти
// Далі йде обробка отриманих байтів
for(i=0; i<dwRead; ++i)
{
    STR=STR+char(Buf[i]);/* IntToStr(short(Buf[i])) + " "; */
}
Synchronize(DoRead);
delete Buf;
}
```

В результаті файл Unit2.cpp буде виглядати наступним чином (код який потрібно дописати виділений жирним шрифтом):

```
_fastcall TReadThread::TReadThread(bool CreateSuspended)
: TThread(CreateSuspended)
{
}
//-----
void _fastcall TReadThread::Execute()
{
    HANDLE hPort;
    COMSTAT CommStat;
    DWORD dwError;
    OVERLAPPED OverRead;
    int i;
    AnsiString STR;
```




```
void __fastcall TReadThread::DoRead(void)
{
    AnsiString STR;
    for(int i=0; i<dwRead; ++i)
    {
        STR=STR+char(Buf[i]);/* IntToStr(short(Buf[i])) + " "; */
    }
    Form1->Memo1->Lines->Add(STR);
    delete Buf;
}
//-----
```

Form1->Memo1->Lines->Add(STR) – даний рядок додає в поле **Memo1** основної форми додає символ з масиву **Buf**, в котрий ми записуємо дані отримані з порту.

На даному етапі робота з потоком завершена, перейдемо до основної програми.

Для підключення потоку до тіла основної програми виконаємо наступні дії:

1. Підключимо модуль потоку до коду основної програми, та об'явимо об'єкт класу **TReadThread** як показано на рисунку 13:

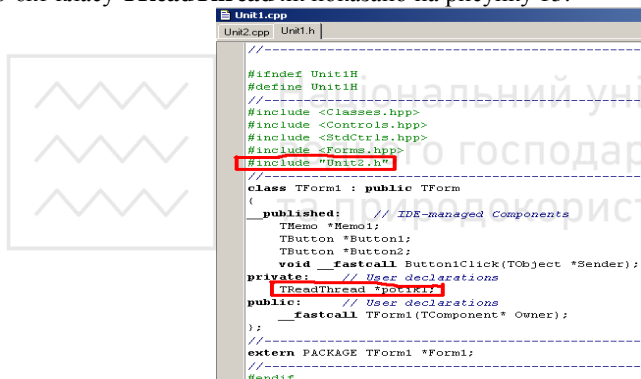


Рис.13. підключення модуля потоку до основної програми.

2. Двічі клацнемо по кнопці яка відповідає за «Зчитування з порту».

3. В кінці вікна яке відкриється напишемо наступний код

```
potik1 = new TReadThread(true);
```

```
potik1->Resume();
```

```
potik1->Priority = tpLower;
```

4. Для зупинки потоку клацнемо 2 рази по кнопці «Деініціалізація порту»

5. У відкритому вікні напишемо наступний код:

```
potik1->Terminate();
```

6. Вкінці файлу Unit1.cpp допишемо:

```
HANDLE __fastcall TForm1::GetPortHandle(void)
```

```
{
    return this->hPort;
}
```



7. У файл Unit1.h у розділі оголошення **public** запишемо наступне
HANDLE __fastcall GetPortHandle(void);

Пояснення:

potik1 = new TReadThread(true); - створюємо новий об'єкт потоку.

potik1->Resume(); - запускаємо потік на виконання.

potik1->Terminate(); - зупинка та знищення потоку.

potik1->Priority = tpLower – задання пріоритету для потоку.

Порядок виконання і опрацювання результатів завдання 3, 8

Додамо до серверної частини програмний код, який буде автоматично підключати ODBC-драйвер з створеною нами базою даних (див. лабораторна робота №6). Для цього до серверної частини проекту додамо наступні компоненти **TOpenDialog** та **TButton**, які будуть відповідати за підключення бази даних. Процедура відкриття буде містити наступний код:

```
{
AnsiString name;
AnsiString dir;
if ((OpenDialog1->Execute()))
{
dir=OpenDialog1->FileName;
name = "database1";
DBModule1->Database1->AliasName=name;
//memo1.Lines.add(dir);
createODBCforMDB(&name,&dir);
}
}
//-----
```

Як бачимо, процедура відкриття (підключення) бази даних містить в собі процедуру **createODBCforMDB(&name,&dir)**, яку нам потрібно описати:

```
void __fastcall TForm3::createODBCforMDB(AnsiString *name,AnsiString *path)
//створюємо ODBC аліас для MDB файла
{
TRegistry *r;
r = new TRegistry;
//створюємо аліас
r->OpenKey("Software\ODBC\ODBC.INI\ODBC Data Sources",true);
r->WriteString(*name,"Microsoft Access Driver (*.mdb)");
r->CloseKey();
//налаштуємо
r->OpenKey("Software\ODBC\ODBC.INI\"+*name,true);
r->WriteString("DBQ",*path);
r->WriteString("Driver","C:\WINDOWS\system32\odbcjt32.dll");
r->WriteInteger("DriverId",25);
r->WriteInteger("SafeTransactions",0);
r->WriteString("UID","");
r->CloseKey();
r->OpenKey("Software\ODBC\ODBC.INI\"+*name+"\Engines\Jet",true);
r->WriteString("ImplicitCommitSync","");
r->WriteString("UserCommitSync","");
r->WriteInteger("Threads",3);
```

```

r->CloseKey();
//оновлюємо реєстр
r->Free();
}

```

При цьому у **public** має бути описана створена процедура:

```
public void __fastcall createODBCforMDB(AnsiString *name,AnsiString *path);
```

На цьому робота з базою даних завершена.

Перейдемо до розробки сервера. Основними задачами, які ми ставимо перед серверним додатком – збір інформації через мережу з клієнтів, її збереження в базі даних, та видача інформації при запитах клієнтів.

Створимо новий проект (процедура створення нового проекту описана в лабораторних роботах 4 і 5), збережемо його на диску і дамо назву **server.bpr**. На основній формі проекту розмістимо компоненти: **TMemo1**, **TButton1**, **TButton2**. Перейменуємо **TButton1** в Start, а **TButton2** в Stop (дані кнопки будуть відповідно запускати та зупиняти сервер). Результат на рисунку 14.

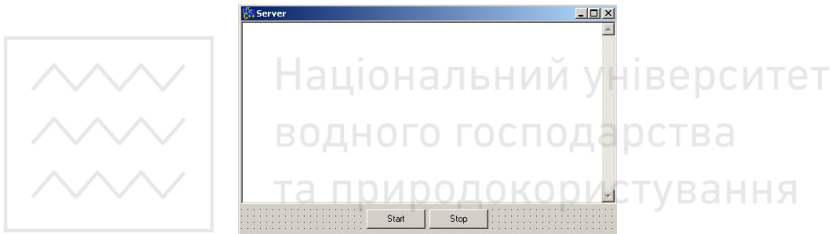


Рис. 14. Вікно сервера.

Для роботи в локальній мережі та з базами даних в середовищі C++ Builder використовуються не візуальні компоненти, які доцільно розташовувати на так званих модулях даних, щоб не загроможувати основну форму.

Створимо модуль, який буде відповідати за комунікацію в мережі. Для цього виберемо пункт **File | New | Other | Data Module**, як показано на рисунку 15.

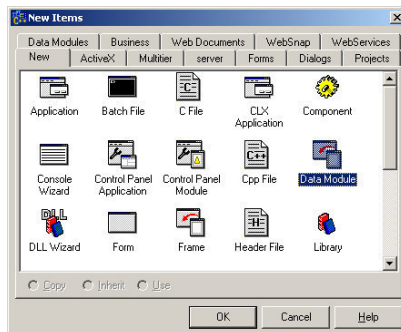


Рис. 15. Створення модуля даних.



Змінимо назву модуля на **NetModule1** і збережемо файл під назвою **netmodule.cpp**.

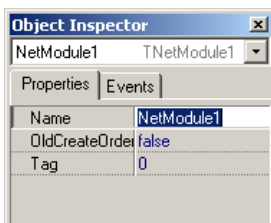


Рис.16. Назва модуля.

Розташуємо на формі модуля компонент **TServerSocket** з вкладки **Internet**. Результат видно на рисунку 17:

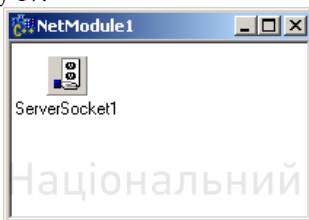


Рис.17. Форма модуля даних.

Створимо модуль, який буде забезпечувати інтерфейс з базою даних. Для цього створимо новий модуль (процедура описана вище) дамо йому ім'я **DBModule1** і збережемо у файлі **dbmodule.cpp**. Розташуємо на модулі наступні компоненти з вкладки **VDE**:

1. **TDatabase** – компонент який забезпечує зв'язок з базою даних, управляє алгоритмом транзакцій.
 2. **TQuery** – компонент, який забезпечує обробку даних в нашій таблиці.
 3. **TUpdateSql** – компонент який дозволяє модифікувати дані в базі даних.
- Форма модуля буде мати наступний вигляд (рис. 18).

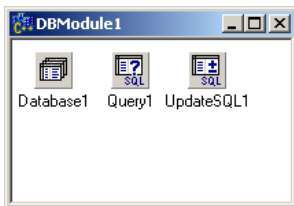


Рис.18. Форма модуля даних.

Тепер прийшла черга задати властивості компонентів. Для компонента **Database1** задамо наступні властивості:



1. **AliasName** – **project** – тут вибираємо псевдонім створеної бази даних.
2. **DatabaseName** – **proj** – даємо базі даних ім'я, яким будемо користуватись для звертання до неї компонентів **Query1**, **UpdateSql**.
Внесені зміни будуть виглядати наступним чином (рис. 19).

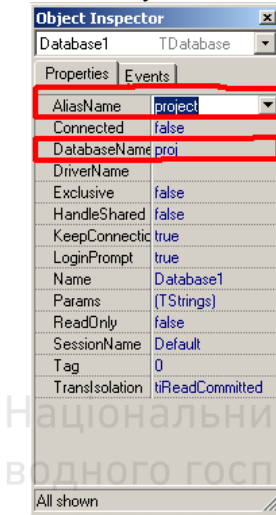


Рис. 19. Властивості компонента Database1.

Тепер налаштуємо компонент **Query1**. Для даного компоненту змінимо лише одну властивість – **UpdateObject**. Її необхідно встановити для того, щоб мати змогу вносити зміни до бази даних (рис. 20).

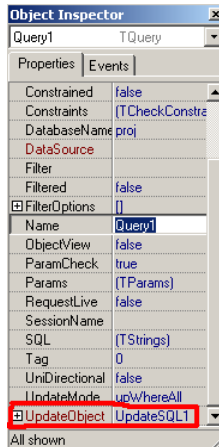


Рис. 20. Налаштування властивостей компонента Query1.



Повернемося до модуля **NetModule1**. Для нього необхідно передбачити події, коли клієнт підключається, відключається від сервера, а також, коли відбувається передача інформації від сервера до клієнта і навпаки. Для перевизначення подій використовується вкладка **Events** вікна **Object Inspector**. Оскільки сервер має відповідати на запити клієнта, то передача інформації від сервера до клієнта буде відбуватись лише по надходженню запиту від клієнта.

Передбачимо спочатку події підключення та відключення клієнта. Для цього нам потрібно перевизначити відповідно наступні методи: **ServerSocket1ClientConnect**, **ServerSocket1ClientDisconnect**. Виділимо мишкою компонент **ServerSocket1**, перейдемо на вкладку **Events** вікна **Object Inspector**, і подвійним клацанням вишки по методам виділеним на рисунку 21 створимо вищезазначені модулі.

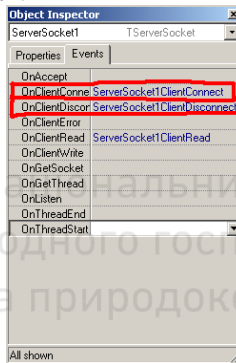


Рис. 21. Перевизначення методів **ServerSocket1ClientConnect** та **ServerSocket1ClientDisconnect**.

Дані методи будуть використовуватися лише для індикації підключення/відключення клієнта, тому внесемо наступний код (код який прописується вручну виділений жирним шрифтом):

```
void __fastcall TnetModule1::ServerSocket1ClientConnect(TObject *Sender,  
    TCustomWinSocket *Socket)  
{  
    Form3->Memo1->Lines->Add(«Клієнт «+Socket->RemoteAddress+» під'єднався.»);  
}  
//-----  
void __fastcall TnetModule1::ServerSocket1ClientDisconnect(TObject *Sender,  
    TCustomWinSocket *Socket)  
{  
    Form3->Memo1->Lines->Add(«Клієнт «+Socket->RemoteAddress+» від'єднався.»);  
}  
//-----
```

Дві данні функції додають строку до поля Memo1, яка сигналізує про підключення чи відключення клієнта. **Socket->RemoteAddress** – властивість

класу `TcustomWinSocket`, яка повертає рядок з IP адресою клієнта котрий підключився.

Для передачі інформації між клієнтом та сервером організуємо кадр інформації, який представляє собою структуру, поля якої можуть містити інформацію про вид запиту, сам запит, а також інформацію з зчитаними даними. Розглянемо код структури.

```
typedef struct
{
    int id;
    int t;
    float p;
    float f;
    char time[50];
    char query[180];
}BUFF;
```

Розпишемо інформаційне навантаження, яке несуть поля структури:

1. `int id;` – в даному полі міститься інформація про вид запиту (1 – вставка даних в таблицю, 2 – отримання даних з таблиці).
2. `int t;` – значення температури.
3. `float p;` – значення тиску.
4. `float f;` – значення витрати.
5. `char time[50];` – час.
6. `char query[180];` – тіло запиту.

Перейдемо до вікна **Object Inspector** та перевизначимо метод **OnClientRead**.

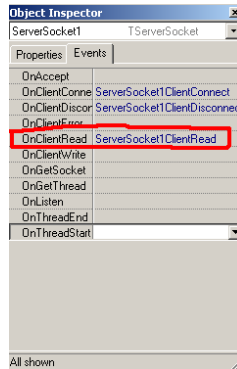


Рис. 22. Перевизначення методу `OnClientRead`.

Код методу **OnClientRead** буде виглядати наступним чином:

```
void __fastcall TNetModule::ServerSocket1ClientRead(TObject *Sender,
    TCustomWinSocket *Socket)
{
    typedef struct // оголошуємо структуру
    {
```



Національний університет

високого господарства

технологічного використання

```
int id;
int t;
float p;
float f;
char time[50];
char query[180];
}BUFF;
BUFF buff; // структурна змінна
AnsiString sql;
AnsiString t;
AnsiString p;
AnsiString f;
AnsiString time;
Socket->ReceiveBuf(&buff,sizeof(buff)); // отримуємо данні з клієнта
if (buff.id==1)
{
    t = IntToStr(buff.t); //отримуєм данні з запроса
    p = FloatToStr(buff.p);
    f = FloatToStr(buff.f);
    time =AnsiString(buff.time);
    // формуємо запрос до бази даних
    sql = "insert into
table1([temperature],[pressure],[expense],[time])values( '"+t+"', '"+p+"', '"+f+"', '"+time+"' )";
    DBModule1->Database1->StartTransaction(); //запуск транзакації
    DBModule1->UpdateSQL1->SetParams(ukInsert); // будем вставляти дані
    DBModule1->UpdateSQL1->InsertSQL->Clear(); // очищуємо буфер SQL - запиту
    DBModule1->UpdateSQL1->InsertSQL->Add(sql); // додаємо буфер до запиту
    DBModule1->UpdateSQL1->ExecSQL(ukInsert); // виконуєм запит
    DBModule1->Database1->Commit(); // підтвердження закінчення тазакації
}
if (buff.id == 2) // отримали запит від клієнта на вбірку даних з таблиці
{
    typedef struct //формуємо структу, яку будем передавати клієнту
    {
        int from;
        int to;
        int t;
        float p;
        float f;
        char time[50];
    }QUERY;
    QUERY query;
    sql = AnsiString(buff.query);
    DBModule1->Database1->StartTransaction(); // запит до бази даних
    DBModule1->Query1->Close();
    DBModule1->Query1->SQL->Clear();
    DBModule1->Query1->SQL->Add(sql);
    DBModule1->Query1->ExecSQL();
    DBModule1->Query1->Open();
    DBModule1->Database1->Commit();
    DBModule1->Query1->First(); //курсор на перше поле
    for (int i=0; i<DBModule1->Query1->RecordCount;++) // заповнення полів структури і
відправка клієнту
```



Національний університет
одного господарства
та природокористування

```
query.from=i;
query.to=1;
query.t=DBModule1->Query1->FieldByName("temperature")->AsInteger;
query.p=DBModule1->Query1->FieldByName("pressure")->AsFloat;
query.f=DBModule1->Query1->FieldByName("expense")->AsFloat;
strcpy(query.time,DBModule1->Query1->FieldByName("time")->AsString);
DBModule1->Query1->Next();//курсор переходить на слідуюче поле
Socket->SendBuf(&query,sizeof(QUERY));
}
}
if (buff.id == 3)
{
typedef struct
{
int from;
int to;
int t;
float p;
float f;
char time[50];
}QUERY;
QUERY query;
sql = AnsiString(buff.query);
DBModule1->Database1->StartTransaction();
DBModule1->Query1->Close();
DBModule1->Query1->SQL->Clear();
DBModule1->Query1->SQL->Add(sql);
DBModule1->Query1->ExecSQL();
DBModule1->Query1->Open();
DBModule1->Database1->Commit();
DBModule1->Query1->First();
for (int i=0; i<DBModule1->Query1->RecordCount;++i)
{
query.from=i;
query.to=2;
query.t=DBModule1->Query1->FieldByName("temperature")->AsInteger;
query.p=DBModule1->Query1->FieldByName("pressure")->AsFloat;
query.f=DBModule1->Query1->FieldByName("expense")->AsFloat;
strcpy(query.time,DBModule1->Query1->FieldByName("time")->AsString);
DBModule1->Query1->Next();
Socket->SendBuf(&query,sizeof(QUERY));
}
}
}
//-----
```

Далі нам необхідно «примусити» наш сервер слухати мережу, для того щоб отримувати запити від клієнтів. Для цього необхідно написати наступний код для обробки подій натиснення на кнопки **TButton1** та **TButton2** (рис. 23).

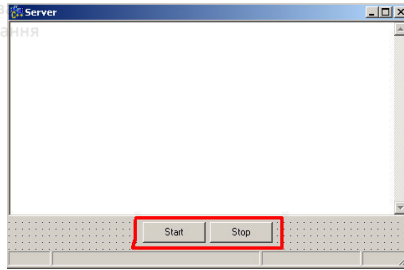


Рис.23. Вікно сервера.

Маємо наступний код:

```
void __fastcall TForm3::GetLoclp(void)
{
    AnsiString hostname;
    AnsiString ip;
    char hn[1024];
    struct hostent *adr;
    WORD wVersionRequested;
    WSADATA wsaData;
    wVersionRequested = MAKEWORD(1, 0);
    WSAStartup( wVersionRequested, &wsaData );
    gethostname(hn,1024);
    adr = gethostbyname(hn);
    short a = (unsigned char)adr->h_addr_list[0][0];
    short b = (unsigned char)adr->h_addr_list[0][1];
    short c = (unsigned char)adr->h_addr_list[0][2];
    short d = (unsigned char)adr->h_addr_list[0][3];
    hostname = AnsiString(hn);
    ip = IntToStr(a)+"."+IntToStr(b)+"."+IntToStr(c)+"."+IntToStr(d);
    Form3->StatusBar1->Panels->Items[1]->Text = "Hostname: "+hostname;
    Form3->StatusBar1->Panels->Items[2]->Text = "IP: "+ip;
    IP=ip;
    HOSTNAME=hostname;
}
//-----
void __fastcall TForm3::Button1Click(TObject *Sender)
{
    NetModule1->ServerSocket1->Active=true;//запускаєм сервер
    DBModule1->Database1->Connected = true;// під'єднуємся до бази даних
    Memo1->Lines->Add("Сервер стартував на "+IP+".");
}
//-----
void __fastcall TForm3::Button2Click(TObject *Sender)
{
    NetModule1->ServerSocket1->Active=false;//зупиняєм сервер
    DBModule1->Database1->Connected = false;//від'єднуємся від бази даних
}
//-----
```

На цьому робота з сервером закінчена.



Порядок виконання завдання № 4, 9

Для розробки клієнтського додатку відкриємо програму, яку розробили в лабораторній роботі №5, оскільки вона забезпечує інтерфейс з СОМ-портом. Для цього додатку необхідно додати можливість роботи в мережі, а також алгоритм обробки даних отриманих через СОМ-порт.

Додамо додатку можливість роботи в мережі, для цього створимо новий модуль даних(процедура описана вище) і розташуємо на формі модуля компонент TClientSocket. Збережемо модуль під назвою netmodule.cpp далі на початку файлу необхідно додати модуль головної форми де описується головне вікно #include "Form1.h", а також у модулі форми (Form1.cpp) необхідно додати #include "netmodule.h" (рис. 24).

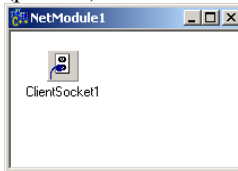


Рис. 24. Клієнтський модуль, який забезпечує роботу в мережі.

Тепер перейдемо до процедури синхронізації з потоком графічного інтерфейсу з потоку зчитування з СОМ-порту. І напишемо наступний код:

На початку файлу додати #include <string.h> #include "form1.cpp" тобто файл для роботи із рядками та файл з графічним інтерфейсом головної форми

```
void __fastcall ReadThread::DoRead(void)
{
    bool col = false;
    typedef struct
    {
        int id;
        int t;
        float p;
        float f;
        char time[80];
        char query[180];
    }BUFF;
    BUFF buff;
    char tmp_buff[30];
    int j=0;
    AnsiString seconds_c;
    AnsiString seconds_p;
    AnsiString str;
    int s_c;
    int s_p;
    bool write_s;
    bool firstTime;
    AnsiString pT, pP, pF;
    for (int i=0; i<dwRead; ++i)
```



```

str += AnsiString(char(Buf[i])); // отримуємо строку із порту
tmp_buff[j] = char(Buf[i]);
++;
if (j>29)
{
j=0;
char t[3];
char p[6];
char f[4];
if (tmp_buff[2] == ' ')
{
t[0] = tmp_buff[3]; col=CheckCol(tmp_buff[3]); //перевірка на достовірність отриманих

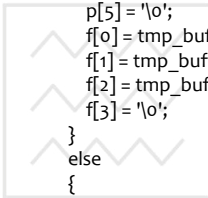
```

даних

```

t[1] = tmp_buff[4]; col=CheckCol(tmp_buff[4]);
t[2] = '\0';
p[0] = tmp_buff[11]; col=CheckCol(tmp_buff[11]);
p[1] = tmp_buff[12]; if(tmp_buff[12]!='.') col=true;
p[2] = tmp_buff[13]; col=CheckCol(tmp_buff[13]);
p[3] = tmp_buff[14]; col=CheckCol(tmp_buff[14]);
p[4] = tmp_buff[15]; col=CheckCol(tmp_buff[15]);
p[5] = '\0';
f[0] = tmp_buff[23]; col=CheckCol(tmp_buff[23]);
f[1] = tmp_buff[24]; if(tmp_buff[24]!='.') col=true;
f[2] = tmp_buff[25]; col=CheckCol(tmp_buff[25]);
f[3] = '\0';
}
else
{

```



```

t[0] = tmp_buff[2]; col=CheckCol(tmp_buff[2]);
t[1] = tmp_buff[3]; col=CheckCol(tmp_buff[3]);
t[2] = '\0';
p[0] = tmp_buff[10]; col=CheckCol(tmp_buff[10]);
p[1] = tmp_buff[11]; if(tmp_buff[11]!='.') col=true;
p[2] = tmp_buff[12]; col=CheckCol(tmp_buff[12]);
p[3] = tmp_buff[13]; col=CheckCol(tmp_buff[13]);
p[4] = tmp_buff[14]; col=CheckCol(tmp_buff[14]);
p[5] = '\0';
f[0] = tmp_buff[22]; col=CheckCol(tmp_buff[22]);
f[1] = tmp_buff[23]; if(tmp_buff[23]!='.') col=true;
f[2] = tmp_buff[24]; col=CheckCol(tmp_buff[24]);
f[3] = '\0';
}
AnsiString T = AnsiString(t);
AnsiString P = AnsiString(p);
AnsiString F = AnsiString(f);
TDateTime tm = Time(); //формуємо поточний час зчитування
seconds_c = FormatDateTime("s", tm);
s_c = StrToInt(seconds_c);
s_p = StrToInt(seconds_p);
if (s_c == s_p)
{
write_s = false;
}

```



Національний університет
водного господарства
та природокористування

```
    }  
    else  
    {  
        write_s = true;  
    }  
    seconds_p = seconds_c;  
    if (col == true)  
    {  
        if (firstTime != true)  
        {  
            if (write_s == true)  
            {  
                str="";  
                firstTime = false;  
                // можна видалитиif(Form1->isNetConnected)  
                // можна видалити{  
                AnsiString time = FormatDateTime("h:m:s",tm);  
                buff.id=1;  
                buff.t = StrToInt(pT);//формуємо структуру  
                buff.p=StrToFloat(pP);  
                buff.f=StrToFloat(pF);  
                strcpy(buff.time,time.c_str());//пересилаємо дані  
                NetModule1->ClientSocket1->Socket->SendBuf(&buff,sizeof(buff));  
                // можна видалити}  
            }  
        }  
        else  
        {  
            if (write_s == true)  
            {  
                str="";  
                pT=T;  
                pP=P;  
                pF=F;  
                firstTime = false;  
                // можна видалитиif(Form1->isNetConnected)  
                // можна видалити{  
                AnsiString time = FormatDateTime("h:m:s",tm);  
                buff.id=1;  
                buff.t = StrToInt(pT);//формуємо структуру  
                buff.p=StrToFloat(pP);  
                buff.f=StrToFloat(pF);//пересилаємо дані  
                NetModule1->ClientSocket1->Socket->SendBuf(&buff,sizeof(buff));  
                // можна видалити}  
            }  
        }  
    }  
    }  
    }  
    delete Buf;  
}
```

Національний університет
водного господарства
та природокористування



```
bool __fastcall ReadThread::CheckCol(char ch)// функція для перевірки правильності даних  
отриманих з порту
```

```
{  
    if (ch != 'o'&&  
        ch != '1'&&  
        ch != '2'&&  
        ch != '3'&&  
        ch != '4'&&  
        ch != '5'&&  
        ch != '6'&&  
        ch != '7'&&  
        ch != '8'&&  
        ch != '9')  
    {  
        return true;  
    }  
    else  
    {  
        return false;  
    }  
}
```

Необхідно додати процедуру обробки інформації яка надходить від сервера на запити клієнта. Для цього перейдемо на модуль **netmodule.cpp** відкриємо **Object Inspector** та перевизначимо метод, як показано на рисунку 25:

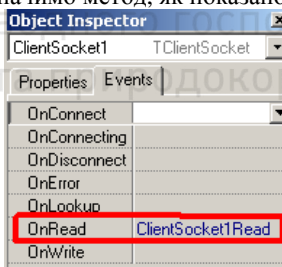


Рис. 25. Перевизначення методу отримання даних с сервера.

Напишемо наступний код:

```
void __fastcall TNetMessage1::ClientSocket1Read(TObject *Sender,  
TCustomWinSocket *Socket)  
{  
    typedef struct  
    {  
        int from;  
        int to;  
        int t;  
        float p;  
        float f;  
        char time[50];  
    }QUERY;  
    QUERY query;  
    Socket->ReceiveBuf(&query,sizeof(QUERY));//отримуємо дані від сервера
```



```
// відбувається обробка запиту з послідовним заповнення елементів графічного інтерфейсу
if (query.to == 1)
{
    Form1->StringGrid1->RowCount=query.from+1;
    Form1->StringGrid1->Cells[0][query.from+1]=AnsiString(query.t);
    Form1->StringGrid1->Cells[1][query.from+1]=AnsiString(query.p);
    Form1->StringGrid1->Cells[2][query.from+1]=AnsiString(query.f);
    Form1->StringGrid1->Cells[3][query.from+1]=AnsiString(query.time);
}
}
//-----
```

Перейдемо до розробки графічного інтерфейсу користувача. Розташуємо на формі такі компоненти:

1. Button1, Button2, Button3 – використовуються відповідно для підключення, відключення, та відправлення запиту до сервера.
2. StringGrid1 – в даному компоненті будуть відображатись дані отримані від сервера.
3. Edit1 – дане поле буде містити IP адресу сервера.

Скомпонуємо ці компоненти на формі найзручнішим чином, а також напишемо обробники подій для кнопок Button1, Button2, Button3:

```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
    if(!Edit1->Text.IsEmpty())
    {
        NetModule1->ClientSocket1->Host=Edit4->Text;// отримуємо IP сервера
        NetModule1->ClientSocket1->Active=true;// під'єднуємося до сервера
    } //-----
}
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    NetModule1->ClientSocket1->Active=false; //Від'єднуємося від сервера
} //-----
void __fastcall TForm1::Button3Click(TObject *Sender)
{
    typedef struct
    {
        int id;
        int t;
        float p;
        float f;
        char time[50];
        char query[180];
    }BUFF;
    BUFF buff;
    buff.id = 2;
    AnsiString query = "select * from table1";//формуємо запит
    strcpy(buff.query,query.c_str());
    NetModule1->ClientSocket1->Socket->SendBuf(&buff,sizeof(BUFF));//відправляємо запит до
сервера
}
}
```

На цьому робота з клієнтською частиною завершена.



Список використаної літератури

1. Михаил Фленов. Библия Delphi Издательство: БХВ-Петербург, 2005 г., 880 стр.
2. Галисеев Г.В. Программирование в среде **Delphi 8** for NET Самоучитель Серия: Самоучитель Издательство: Диалектика, 2004 г 304 стр.
3. Михаил Фленов Программирование в **Delphi** глазами хакера Издательство: БХВ-Петербург, 2003 г., 362 стр.
4. Архангельский А.Я., Программирование в Delphi 7 Издательство: Бином-Пресс, 2003 г., 1152 стр.
5. Павловская Т.А. , Щупак Ю.А. С++. Объектно-ориентированное программирование. ПРАКТИКУМ. – СПб.: Питер,2004
6. Чепмен Дэвис. Освой самостоятельно Visual C++. NET за 21 день.: Пер с англ. – М.: Изд.дом «Вильямс», 2002.– 720 с.
7. В. Кораблев. Самоучитель VISIAL C++.NET. – СПб.: Питер,2004.
8. Х. Дейтел, П. Дейтел. Как программировать на C++. М.: БИНОМ, 2000 г.

