**Теоретичні відомості до лабораторної роботи  
"Взаємодія розподілених процесів через механізм сокетів"**дисципліна "Комп’ютерні мережі"

[Сокети в TCP/IP 1](#_heading=h.30j0zll)

[Блокуючі та неблокуючі сокети 2](#_heading=h.1fob9te)

[API WinSock 2](#_heading=h.3znysh7)

[Діаграми мережевих протоколів взаємодії розподілених систем 14](#_heading=h.2et92p0)

[Варіанти індивідуальних завдань 17](#_heading=h.2jxsxqh)

**Сокети в TCP/IP**

Сокети, зокрема Windows Sockets (або API WinSock), є програмним інтерфейсом до протоколу TCP (який забезпечує дотримування послідовності повідомлень, надійне з’єднання) та протоколу UDP (лише проста дейтаграмна розсилка, ненадійне з’єднання) стеку протоколів TCP/IP. Всі реалізації механізму сокетів (включно з WinSock) базуються на *Berkley Sockets API*, тому є практично у всіх ОС (включно всіма варіантами MS Windows) ідентичними, навіть на рівні імен функцій API. Відмінності є лише на рівні розширень. Наприклад, в Unix-платформах сокети є частиною ядра ОС, а в Windows це окрема підсистема, яка, в разі використання, має бути активована/дезактивована окремими спеціальними функціями, яких немає в *Berkley Sockets API*, відповідно сокети ОС Windows будуть розширенням до *Berkley Sockets*.

Технологія сокетів дуже нагадує роботу із послідовними файлами (для ОС Windows більш точно – іменовані *канали*): сокет потрібно відкрити, зчитувати (прослуховувати) з нього, писати і в кінці роботи закрити. Що передається через сокет "знає" лише програма, - для інтерфейсу сокетів це лише група байт (повідомлення), і яка його структура, призначення тощо має бути передбачена протоколом взаємодіючих процесів.

Механізм сокетів, як взаємодію мінімум двох мережевих процесів, зручно описувати в термінології архітектури клієнт/сервер. Так програма (хост), яка ініціює сесію з іншою програмою - *сервером*, виступає *клієнтом*, і мета взаємодії визначається клієнтом через його запити до програми-сервера (далі просто клієнт і сервер). В часі сервер має бути розгорнутим раніше клієнтів, інакше на клієнтах має бути реалізований цикл перевірки наявності в мережі потрібного сервера.

IP-адресація передбачає наявність у хоста однієї (і більше) IP-адрес, кожна з яких має 65536 портів (16-бітне число без знаку). З портами асоціюються сервіси (серед стандартних: е-пошта, Web (HTTP), обмін файлами (FTP) тощо)[[1]](#footnote-1). Але і до конкретного порта IP-адреси може приєднатися будь-яке число клієнтів. От окремій сесії для клієнта і буде свій сокет, хоча їх можна створювати і більше. Наприклад, Web-сервер для протоколу HTTP (обмін гіпертекстовими документами), має прослуховувати порт 80 своєї IP-адреси (для FTP 21-й порт, для пошти SMTP 25-й). Ініціалізація клієнтом сокета на порт 80 Web-сервера означає, що Web-сервер, в разі згоди на сесію, буде створювати персональний сокет з даним клієнтом. І таких сокетів одночасно може бути відкрито тисячі (це вже залежить від апаратного ресурсу сервера). Тобто на сервері, який прослуховує порт, має бути створений сокет, якщо з’явився клієнт.

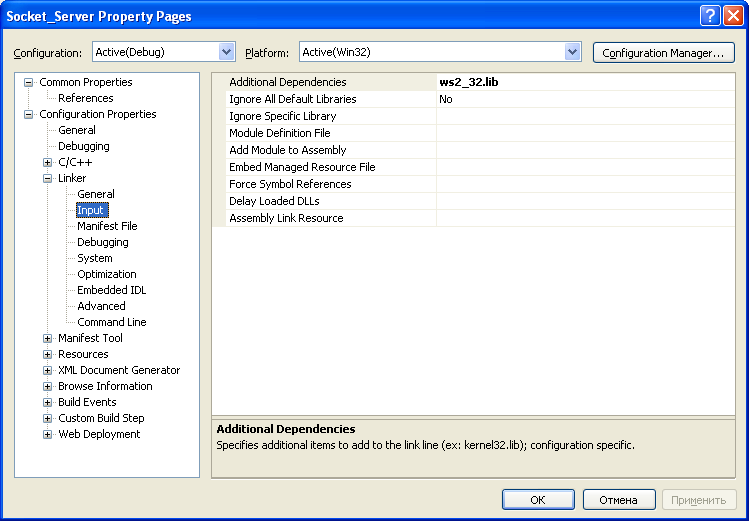
***Блокуючі та неблокуючі сокети***

Коли програма (клієнт чи сервер) виконує операцію зчитування повідомлення (функція recv), то результат її завершення непередбачуваний у часі. Поки функція не поверне отримане повідомлення, програма виконуватися далі не може, тобто блокується. Робота з сокетами в такому режимі називається *синхронні* або *блокуючі* сокети. Відповідно, якщо блокування не допускати (а це важливо для подіє-орієнтованих Windows-програм), то це будуть *асинхроні* або *неблокуючі* сокети. Для асинхронних сокетів потрібно використовувати або механізм підзадач для блокуючих функцій, або спеціально розроблені для ОС Windows функції (класи), в яких можна перевіряти результат завершення дії без блокування.

**Для лабораторної достатньо використовувати блокуючі сокети** (щоб не ускладнювати програмування). Це досягається сценарієм роботи, коли на кожне повідомлення ведучого має бути відповідь ведомого, або точно оговорена кількість повідомлень (так зване *вікно*) ведучого, після чого передаюча сторона переходить на прийом.

***API WinSock***

Функції API WinSock для роботи із сокетами складаються із функції WSAStartup ініціалізації інтерфейсу WinSock та функцій *Berkley Sockets API* (accept, bind, closesocket, connect, listen, recv, send, shutdown, socket та ряд допоміжних функцій роботи із значеннями IP-адрес, портів тощо).

Програми клієнта та сервера мають мати заголовочний файл: 

#include <winsock2.h>  
та лінкуватися з бібліотекою:

* + **ws2\_32.lib** при розробці в Visual Studio (див. рис. для VS/2005, вікно властивостей проекта);
  + **ls2\_ws32.a** при використанні Dev-CPP.

Сценарій для клієнта та сервера щодо використання функцій WinSock:

|  |  |  |  |
| --- | --- | --- | --- |
| **Клієнт** | | **Сервер** | |
| 1 | Ініціалізація інтерфейсу – WSAStartup() | 1 | Ініціалізація інтерфейсу – WSAStartup() |
| 2 | Створення сокету – socket()**.** Заповнення параметрів сокета. | 2 | Створення сокету – socket(). Заповнення параметрів сокета. |
| 3 | З’єднатися із сервером – connect(). | 3 | Встановлення асоціації локальної адреси із сокетом – bind(). |
| 4 | Відсилати повідомлення – send(). Приймати повідомлення – recv(). | 4 | Прослуховувати порт на з’єднання – listen(). |
| 5 | Закриття сокету – closesocket(). | 5 | Вибір клієнтів із черги на приєднання до сервера – accept(). (Для кожного нового клієнта створює окремий сокет.) |
| 6 | Відсилати повідомлення – send(). Приймати повідомлення – recv(). |
| 7 | Закриття сокету – closesocket(). |

Розглянемо специфікації основних мінімально необхідних функцій (їх прототипи вже є в winsock2.h).

Функція **WSAStartup**.

|  |  |
| --- | --- |
| int **WSAStartup**(WORD ***wVersionRequested***, LPWSADATA ***lpWSAData***); | |
| Виконує ініціалізацію інтерфейсу сокетів (потрібна лише в Windows, в Unix-платформах всі функції роботи із сокетами представлені в ядрі ОС). Ця функція повинна передувати використанню всіх наступних функцій API. | |
| **Аргументи:** | |
| *wVersionRequested* | версія інтерфейсу, яка задає основну та допоміжну його редакцію. Зараз використовується версія 2.2, отже задавати потрібно значення 0x0202. |
| *lpWSAData* | вказівник на структуру типу:  typedef struct WSAData {  WORD wVersion;  WORD wHighVersion;  char szDescription[WSADESCRIPTION\_LEN+1];  char szSystemStatus[WSASYS\_STATUS\_LEN+1];  unsigned short iMaxSockets;  unsigned short iMaxUdpDg;  char FAR\* lpVendorInfo;  }; |
| **Повертає значення:** | |
| 0 | успішна ініціалізація інтерфейсу |
| <> 0 | код помилки |
| **Приклад:** | |
| WSADATA WSA\_Socket;  SOCKET s;    //ініціалізуємо роботу програми із сокетами  if(WSAStartup(0x0202,&WSA\_Socket)!=0) {  printf("%s","\nSocket\_Server()> open WSAStartup is invalid!");  getch(); return 1;  } | |

Функція створення сокета **socket**

|  |  |
| --- | --- |
| SOCKET **socket(**int ***af*,** int ***type*,** int ***protocol*);** | |
| Виконує | |
| **Аргументи:** | |
| *af* | родина адрес. Потрібно брати за значення PF\_INET |
| *type* | тип сокета: SOCK\_STREAM - захищене з’єднання (через протокол TCP), SOCK\_DGRAM - незахищене, через обмін незв’язаними між собою дейтаграм (через протокол UDP).  Для лабораторних потрібне з’єднання із відслідковуванням послідовностей (захищене), тому потрібно брати значення SOCK\_STREAM. |
| *protocol* | задає протокол, має бути значення DEFAULT\_PROTOCOL. |
| **Повертає значення:** | |
| 0 | сокет не створений. Інші помилки уточнюються через **int** **WSAGetLastError(void);** яка повертає код помилки |
| дескриптор нового сокета | в разі успішного завершення. Зверніть увагу, що створена структура для сокета (а дескриптор вказує на цю структуру) є пустою і потребує заповнення! |
| **Приклад:** | |
| WSADATA WSA\_Socket;  SOCKET s;    //ініціалізуємо роботу програми із сокетами  if(WSAStartup(0x0202,&WSA\_Socket)!=0) {  printf("%s","\nSocket\_Server()> open WSAStartup is invalid!");  getch(); return 1;  }  s= socket(PF\_INET, SOCK\_STREAM, DEFAULT\_PROTOCOL); if (s==0) { /\* сокет не творено \*/ … exit;} | |

Тепер потрібно зв’язати створений сокет із реальною адресою та портом. для цього спочатку заповнюємо реальними значеннями структуру sockaddr\_in, а потім клієнт зв’язує її з сокетом функцією connect(), а сервер bind(). Далі просто приклад заповнення полів змінної service, яка в подальшому профігурує в функціях connect(), та bind(). Нехай це будуть IP-адреса 127.0.0.1, а порт 14900.

sockaddr\_in service;  
service.sin\_family = PF\_INET; //може бути також AF\_INET  
service.sin\_addr.s\_addr = inet\_addr("127.0.0.1");  
service.sin\_port = htons(14900);

Функція створення сокета **closesocket**

|  |  |
| --- | --- |
| SOCKET **closesocket(**SOCKET ***s*);** | |
| Виконує | |
| **Аргументи:** | |
| *s* | закриваємий сокет. |
| **Повертає значення:** | |
| 0 | успішне закриття |
| <> 0 | код помилки |
| **Приклад:** | |
| WSADATA WSA\_Socket;  SOCKET s;    //ініціалізуємо роботу програми із сокетами  if(WSAStartup(0x0202,&WSA\_Socket)!=0) {  printf("%s","\nSocket\_Server()> open WSAStartup is invalid!");  getch(); return 1;  }  s= socket(PF\_INET, SOCK\_STREAM, DEFAULT\_PROTOCOL);  … … … … … if (closesocket(s)!=0) { /\* сокет не вдається закрити \*/ … exit;} | |

Функція асоціації локальної адреси із сокетом **bind**

|  |  |
| --- | --- |
| int **bind(**SOCKET ***s*,** const structsockaddr\* ***name*,** int *namelen***);** | |
| Виконує | |
| **Аргументи:** | |
| *s* | сокет, через який очікуємо приєднання клієнтів. |
| *name* | структура sockaddr\_in, з якою асоціюється адреса та порт для сокета.. |
| *namelen* | довжина структури сокета (в байтах). |
| **Повертає значення:** | |
| 0 | успішне закриття |
| <> 0 | код помилки |
| **Приклад:** | |
| (приклад запозичено із Microsoft Document Eplorer (MSDN))  #include <stdio.h>  #include "winsock2.h"  void main() {  //----------------------  // Initialize Winsock  WSADATA wsaData;  int iResult = WSAStartup(MAKEWORD(2,2), &wsaData);  if (iResult != NO\_ERROR)  printf("Error at WSAStartup()\n");    //----------------------  // Create a SOCKET for listening for  // incoming connection requests  SOCKET ListenSocket;  ListenSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  if (ListenSocket == INVALID\_SOCKET) {  printf("Error at socket(): %ld\n", WSAGetLastError());  WSACleanup();  return;  }  //----------------------  // The sockaddr\_in structure specifies the address family,  // IP address, and port for the socket that is being bound.  sockaddr\_in service;  service.sin\_family = PF\_INET;  service.sin\_addr.s\_addr = inet\_addr("127.0.0.1");  service.sin\_port = htons(27015);  //----------------------  // Bind the socket.  if (bind( ListenSocket,  (SOCKADDR\*) &service,  sizeof(service)) == SOCKET\_ERROR) {  printf("bind() failed.\n");  closesocket(ListenSocket);  return;  }  … … … …  return;  } | |

Тепер сервер має прослуховувати порт за допомогою функції listen.

Функція прослуховування сокета (для виявлення бажаючих приєднатися клієнтів) **listen**

|  |  |
| --- | --- |
| int **listen(**SOCKET ***s*,** int ***backlog*);** | |
| Виконує | |
| **Аргументи:** | |
| *s* | сокет, який прослуховується. |
| *backlog* | задає максимальну кількість підключень клієнтів до сервера. |
| **Повертає значення:** | |
| 0 | успішне закриття |
| <> 0 | код помилки |
| **Приклад:** | |
| (приклад запозичено із Microsoft Document Eplorer (MSDN))  #include <stdio.h>  #include "winsock2.h"  void main() {  //----------------------  // Initialize Winsock  WSADATA wsaData;  int iResult = WSAStartup(MAKEWORD(2,2), &wsaData);  if (iResult != NO\_ERROR)  printf("Error at WSAStartup()\n");  //----------------------  // Create a SOCKET for listening for  // incoming connection requests.  SOCKET ListenSocket;  ListenSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  if (ListenSocket == INVALID\_SOCKET) {  printf("Error at socket(): %ld\n", WSAGetLastError());  WSACleanup();  return;  }  //----------------------  // The sockaddr\_in structure specifies the address family,  // IP address, and port for the socket that is being bound.  sockaddr\_in service;  service.sin\_family = AF\_INET;  service.sin\_addr.s\_addr = inet\_addr("127.0.0.1");  service.sin\_port = htons(27015);  if (bind( ListenSocket,  (SOCKADDR\*) &service,  sizeof(service)) == SOCKET\_ERROR) {  printf("bind() failed.\n");  closesocket(ListenSocket);  return;  }  //----------------------  // Listen for incoming connection requests  // on the created socket  if (listen( ListenSocket, 1 ) == SOCKET\_ERROR) //можливе лише одне підключення  printf("Error listening on socket.\n");  printf("Listening on socket...\n");  WSACleanup();  return;  } | |

Підключення клієнта реєструється функцією accept.

Функція підключення нового клієнта **accept**

|  |  |
| --- | --- |
| SOCKET **accept(**SOCKET ***s*,** const structsockaddr\* ***addr*,** int\* ***addrlen*);** | |
| Виконує | |
| **Аргументи:** | |
| *s* | сокет, через який очікували приєднання клієнтів. |
| *addr* | потрібно брати значення NULL.. |
| *addrlen* | потрібно брати значення NULL. |
| **Повертає значення:** | |
| 0 | сокет не створений. Інші помилки уточнюються через **int** **WSAGetLastError(void);** яка повертає код помилки |
| дескриптор нового сокета (для нового клієнта) | в разі успішного завершення. З цим новим сокетом сервер і має працювати далі. |
| **Приклад:** | |
| (приклад запозичено із Microsoft Document Eplorer (MSDN))  include <stdio.h>  #include "winsock2.h"  void main() {  //----------------------  // Initialize Winsock.  WSADATA wsaData;  int iResult = WSAStartup(MAKEWORD(2,2), &wsaData);  if (iResult != NO\_ERROR)  printf("Error at WSAStartup()\n");  //----------------------  // Create a SOCKET for listening for  // incoming connection requests.  SOCKET ListenSocket;  ListenSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  if (ListenSocket == INVALID\_SOCKET) {  printf("Error at socket(): %ld\n", WSAGetLastError());  WSACleanup();  return;  }  //----------------------  // The sockaddr\_in structure specifies the address family,  // IP address, and port for the socket that is being bound.  sockaddr\_in service;  service.sin\_family = AF\_INET;  service.sin\_addr.s\_addr = inet\_addr("127.0.0.1");  service.sin\_port = htons(27015);  if (bind( ListenSocket,  (SOCKADDR\*) &service,  sizeof(service)) == SOCKET\_ERROR) {  printf("bind() failed.\n");  closesocket(ListenSocket);  return;  }  //----------------------  // Listen for incoming connection requests.  // on the created socket  if (listen( ListenSocket, 1 ) == SOCKET\_ERROR)  printf("Error listening on socket.\n");  //----------------------  // Create a SOCKET for accepting incoming requests.  SOCKET AcceptSocket;  printf("Waiting for client to connect...\n");  //----------------------  // Accept the connection.  while(1) {  AcceptSocket = SOCKET\_ERROR;  while( AcceptSocket == SOCKET\_ERROR ) {  AcceptSocket = accept( ListenSocket, NULL, NULL );  }  printf("Client connected.\n");  ListenSocket = AcceptSocket;  break;  }  WSACleanup();  return;  } | |

Тепер сервер може працювати з клієнтом, посилаючи повідомлення функцією send та приймаючи повідомлення функцією recv.

Функція з’єднання клієнта із сервером (через заданий сокет) **connect**

|  |  |
| --- | --- |
| int **connect(**SOCKET ***s*,** const structsockaddr\* ***name*,** int *namelen***);** | |
| Виконує | |
| **Аргументи:** | |
| *s* | сокет, через який клієнт приєднується до сервера. |
| *name* | структура sockaddr\_in, з якою асоціюється адреса та порт для сокета.. |
| *namelen* | довжина структури сокета (в байтах). |
| **Повертає значення:** | |
| 0 | успішне приєднання. |
| <> 0 | код помилки |
| **Приклад:** | |
| (приклад запозичено із Microsoft Document Eplorer (MSDN))  #include <stdio.h>  #include "winsock2.h"  void main() {  //----------------------  // Initialize Winsock  WSADATA wsaData;  int iResult = WSAStartup(MAKEWORD(2,2), &wsaData);  if (iResult != NO\_ERROR)  printf("Error at WSAStartup()\n");  //----------------------  // Create a SOCKET for connecting to server  SOCKET ConnectSocket;  ConnectSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  if (ConnectSocket == INVALID\_SOCKET) {  printf("Error at socket(): %ld\n", WSAGetLastError());  WSACleanup();  return;  }  //----------------------  // The sockaddr\_in structure specifies the address family,  // IP address, and port of the server to be connected to.  sockaddr\_in clientService;  clientService.sin\_family = AF\_INET;  clientService.sin\_addr.s\_addr = inet\_addr( "127.0.0.1" );  clientService.sin\_port = htons( 27015 );  //----------------------  // Connect to server.  if ( connect( ConnectSocket, (SOCKADDR\*) &clientService, sizeof(clientService) ) == SOCKET\_ERROR) {  printf( "Failed to connect.\n" );  WSACleanup();  return;  }  printf("Connected to server.\n");  WSACleanup();  return;  } | |

Тепер клієнт може працювати з сервером, посилаючи повідомлення функцією send та приймаючи повідомлення функцією recv. Звичайно ж, для клієнта буде послати серверу повідомлення про мету сеансу.

Функції передачі повідомлення **send** та прийому повідомлення **recv**

|  |  |
| --- | --- |
| int **send(**SOCKET ***s*,** const char\* ***buf*,** int ***len*,** int ***flags*);** int **recv(**SOCKET ***s*,** const char\* ***buf*,** int ***len*,** int ***flags*);** | |
| Виконує | |
| **Аргументи:** | |
| *s* | сокет, через який клієнт/сервер посилає/приймає повідомлення. |
| *buf* | буфер даних.. |
| *len* | довжина буфера (в байтах). |
| *flags* |  |
| **Повертає значення:** | |
| кількість прийнятих/відісланих байт | успішне виконання. |
| < 0 | код помилки, які уточнюються через функцію  **int** **WSAGetLastError(void);** яка повертає код помилки |
| **Приклад:** | |
| (приклад запозичено із Microsoft Document Eplorer (MSDN))  #include <stdio.h>  #include "winsock2.h"  void main() {  //----------------------  // Initialize Winsock  WSADATA wsaData;  int iResult = WSAStartup(MAKEWORD(2,2), &wsaData);  if (iResult != NO\_ERROR)  printf("Error at WSAStartup()\n");  //----------------------  // Create a SOCKET for connecting to server  SOCKET ConnectSocket;  ConnectSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);  if (ConnectSocket == INVALID\_SOCKET) {  printf("Error at socket(): %ld\n", WSAGetLastError());  WSACleanup();  return;  }  //----------------------  // The sockaddr\_in structure specifies the address family,  // IP address, and port of the server to be connected to.  sockaddr\_in clientService;  clientService.sin\_family = AF\_INET;  clientService.sin\_addr.s\_addr = inet\_addr( "127.0.0.1" );  clientService.sin\_port = htons( 27015 );  //----------------------  // Connect to server.  if ( connect( ConnectSocket, (SOCKADDR\*) &clientService, sizeof(clientService) ) == SOCKET\_ERROR) {  printf( "Failed to connect.\n" );  WSACleanup();  return;  }  //----------------------  // Declare and initialize variables.  int bytesSent;  int bytesRecv = SOCKET\_ERROR;  char sendbuf[32] = "Client: Sending data.";  char recvbuf[32] = "";  //----------------------  // Send and receive data.  bytesSent = send( ConnectSocket, sendbuf, strlen(sendbuf), 0 );  printf( "Bytes Sent: %ld\n", bytesSent );  while( bytesRecv == SOCKET\_ERROR ) {  bytesRecv = recv( ConnectSocket, recvbuf, 32, 0 );  if ( bytesRecv == 0 || bytesRecv == WSAECONNRESET ) {  printf( "Connection Closed.\n");  break;  }  printf( "Bytes Recv: %ld\n", bytesRecv );  }  WSACleanup();  return;  } | |

**Діаграми мережевих протоколів взаємодії розподілених систем**

Є декілька усталених способів графічного представлення сценаріїв взаємодії процесів, які все ж будуть зводитися до відображення:

1. порядку взаємодії (хто передає, хто приймає),
2. характеру поточної взаємодії (ініціалізація певної фази сценарію, проведення чи завершення) з відповідними процедурами,
3. дій при виникненні інших особливих ситуацій (тут можуть виділятися загальні для всіх фаз особливі ситуації та лише характерні для конкретної фази).

Розглядаючи взаємодію процесів в мережі в рамках архітектури клієнт/сервер часто використовуються діаграми (рис. 2), що складається із 3-х смуг часу (в смислі послідовностей подій, а не астрономічного часу) для *клієнта*, *постачальника сервісу* (мережа, наприклад) та *сервера*. Як правило, в таких діаграмах показують лише



Ініціатор дії буде мати певне позначення дії у своїй смузі та по горизонталі такого рядка у інших смугах повинно бути пусто, тобто зразу видно ініціатора дії. Від дії ініціатора має йти стрілка на смугу приймаючої сторони; відповідні дії приймаючої сторони мають вже бути рядками нижче від дії ініціатора. На стрілці може бути підпис, навіть якщо він буде і на середній смузі – він стосується лише стрілки, яка точно показує хто і кому передає. Для мережевих протоколів такою передачею є команда в протокольних полях пакета/дейтаграми чи повідомленні, що передається по сокету. Отже передача команд (подій тощо) між наявними трьома сторонами (клієнт, постачальник сервісу, сервер) зображати пропонується так:

|  |  |
| --- | --- |
|  | Клієнт передав серверу якусь команду з параметрами. Після цього сервер щось має виконати обов’язково за сценарієм. (У подальшому ми введемо конкретні команди.) |
|  | На команду від клієнта сервер після ряду обов’язкових дій передав клієнту команду (наприклад, результат запиту від клієнта). У клієнта також прописана якась реакція. |
|  | На команду клієнта сервер дає 7 команд. Кількість може бути і змінною, відомою на рівні сценарію (протоколу). |
|  | Постачальник сервісу передав однакову команду клієнту та серверу. |

Синтаксис позначень на діаграмі станів:

|  |  |  |
| --- | --- | --- |
| **Позначення** | **Назва та призначення** | **Приклад** |
| текст | будь-який текст, який може бути коментарем чи описом певної дії. Також може бути оператор(и) якоїсь мови програмування. |  |
| команда(парам1…) | команда, може мати параметри.  У прикладі параметром команди txtfiltr є рядок (літерал txt, відповідно взятий в парні лапки). |  |
| <текст> | тег дії, заданої внутрішнім текстом |  |
| ?<текст> | умовний тег (типу *if-then*), якщо з нього не виходить стрілка, то його вкладення розміщують з відступом правіше. |  |
| !<текст> | тег особливої дії, яка виникає ззовні певної фази сценарію, або виникає припинення поточної фази. |  |
| {текст} | Мітка, також назва фази/етапу сценарію. Для невеликих дистанцій перехід вгору/вниз у своїй смузі може бути показаний стрілкою (як на блок-схемах, але лише в своїй смузі. Для "дальніх" переходів використовується аналог оператора goto: *<перейти на {етап 1}>*). |  |

Для прикладу опишемо сценарій наступної розподіленої задачі. Клієнт переписує із вхідного текстового файлу у вихідний файл лише рядки, що задовольняють певному критерію, який є на сервері. Клієнт передає кожний рядок до сервера, а останній передає клієнту для отриманого рядка результат – підходить чи не підходить. Отже потрібно розробити сценарій обміну командами, передбачивши також можливі позаштатні ситуації. Для команд потрібно задати формати та їх параметри.

Введемо команду txtfiltr для фільтрації через критерій на сервері рядків вхідного файлу. Власне, на сервері можуть бути реалізовані будь-які інші команди, але наша ідентифікуватиметься за іменем txtfiltr. Введемо ще другу команду abort без параметрів для припинення сеансу клієнт/сервер будь-коли, наприклад, в "середені" використання команди txtfiltr.

Параметри та їх використання для команди txtfiltr нехай будуть такі (всі текстові літерали):

|  |  |
| --- | --- |
| **Параметр** | **Призначення** |
| "open" | Клієнт повинен отримати дозвіл на роботу з командою txtfiltr, пославши серверу команду txtfiltr("open"). Сервер або погоджується, тоді відповідає командою txtfiltr("yes"), або відмовляє, відсилаючи команду txtfiltr("no"). |
| "yes" | Сервер командою txtfiltr("yes") дає згоду працювати з командою txtfiltr. |
| "no" | Сервер командою txtfiltr("no") відмовляє (в даний момент) працювати з командою txtfiltr. Клієнт може пізніше знову ініціалізувати роботу з txtfiltr. |
| "abort" | Посилаючи команду з цим параметром клієнт або сервер негайно припиняють роботу з командою txtfiltr. Звичайно, це означає завершення всіх пов’язаних з командою процесів, файлів тощо. |
| "txt" | Клієнт посилає команду txtfiltr("txt",<рядок>) для перевірки вказаного другим параметром рядка. Сервер має відповісти або txtfiltr("true"), або txtfiltr("false"). |
| "true" | Сервер відповідає txtfiltr("true"), якщо рядок у команді txtfiltr("txt",<рядок>) від клієнта задовольняє критерій. |
| "false" | Сервер відповідає txtfiltr("false"), якщо рядок у команді txtfiltr("txt",<рядок>) від клієнта не задовольняє критерій. |

Наш сценарій буде складатися із двох етапів: ініціалізації роботи з командою txtfiltr (рис.3) та її використання (рис. 4).





Зверніть увагу, що в сценаріях розписують, як правило, лише події стосовно взаємодії сторін, а деталі стосовно окремих процедур на клієнті чи сервері стосуються вже реалізації програми. Наприклад, наш клієнт переписує вхідний файл у вихідний з перевіркою рядків на сервері. Зрозуміло, що для роботи з файловою системою належить відкривати/закривати файли тощо. І в різних системах програмування це буде по-своєму. Тому це залишають "на реалізацію" в конкретній системі без опису всіх таких деталей в сценаріях.

**Варіанти індивідуальних завдань**

|  |  |
| --- | --- |
| **№ вар.** | **Варіант** |
| **1** | **Редактор текстових рядків.**  Після приєднання клієнта, сервер передає йому набір текстових рядків (не менше 5). Клієнт і сервер відображають ці рядки у своїх вікнах. У клієнті можна змінювати символи, вставляти нові чи видаляти, не порушуючи кількість рядків та їх максимальний розмір. Будь-яка зміна у рядках клієнта (заміна символа, видалення, вставка нового) передаються серверу і синхронно відображаються у його вікні. |
| **2** | **Передача файлів.**  Клієнт передає серверу довільний файл (із збереженням його оригінального імені). Сервер зберігає передані файли в своєму каталозі. Клієнт та сервер мають відображати хід передачі/прийому: кількість переданих/прийнятих байт, відсоток виконання операції та поточну швидкість передачі. Клієнт в кінці передачі має відобразити час виконання операції. |
| **3** | **Віддалена консоль.**  Клієнт передає на сервер певну кількість команд (командного процесора ОС), які на клієнті можуть вводитися вручну чи братися із текстового файла. Сервер, після отримання останньої команди, виконує весь пакет команд. Результати команд сервер перенаправляє з консолі командного процесора на текстовий файл результатів. Файл результату передається клієнту. (З клієнта краще передавати команди, що не потребують діалогу з оператором.) |
| **4** | **Спрощений чат.**  Дві програми на різних хостах дають можливість користувачам підтримувати спільний відкритий чат. Кожна репліка містить ім'я автора, текст повідомлення та, можливо, ім'я адресату повідомлення. У вікнах програм мають бути поля для вводу реплік та  (Сервер, будучи подібним до клієнта функціонально, запускаємо першим.) |
| **5** | **Електронна "дошка об'яв".**  На сервері вікно розбите на області із текстом (можна також із зображеннями) - об'яви. З клієнтської програми потрібно задавати значення в будь-якій області об'яв. На сервері має бути можливість відкривати/закривати доступ до кожної області. Якщо з клієнта поступає текст для закритої області, то має бути відповідна репліка для користувача. |
| **6** | **Диспетчер файлів.**  Сервер дає можливість клієнтській програмі мати панель файлів і папок файлової системи сервера (доступної клієнтам). Це має бути щось подібне, але простіше за дизайном, до панелі Total Commander, Far. Реалізація навігації у панелі необов'язкова, графіка - мінімальна. |
| **7** | **Контейнер іменованих об'єктів.**  На сервері зберігаються іменовані об'єкти клієнтів: цілі числа, символьні рядки, дампи структур тощо, наприклад, як база параметрів клієнтів. Для зберігання/зміни на сервері об'єкта потрібно передати трійку <*Ім'я*; *Тип*; *Значення*>. Для зчитування потрібне лише ім'я, а сервер має дати відповідь трійкою <*Ім'я*; *Тип*; *Значення*>. Клієнт може сторити, змінити, зчитати, видалити об'єкт на сервері та отримати перелік імен об'єктів. |
| **8** | **Перекладач.**  Клієнт посилає слово(а) на сервер, які вводить користувач. Сервер повертає або словарну статтю, або відмову. Якщо клієнт відсилає декілька слів, то сервер повертає переклад кожного слова (без узгодження відмінків) на позиції оригінала, взявши із словарної статті самий перший варіант. (На сервері достатньо реалізувати "словник" із якось десятка слів.) |
| **9** | **Калькулятор.**  Сервер реалізує калькулятор на чотири арифметичні операції. З клієнта за одне звернення до сервера передається введена користувачем операція та два її операнди. В кінці сеансу сервер передає до клієнта min, max та середнє операндів,  кількість виконаних дій. |
| **10** | **Контексний пошук у файлі(ах).**  На сервері зберігається файл(и) (достатньо лише текстових). Користувач в клієнті задає контекст (лише із символів клавіатури), клієнт відсилає останній до сервера для виконання контексного пошуку (достатньо обмежитися фіксованим файлом/файлами). Результати пошуку порядково відсилаються до клієнта. |
| **11** | **Програмуємий будильник.**  1) Користувач з клієнта задає інтервал в секундах відображення часу на сервері (за змовчанням це 1 секунда). Сервер в зазначені інтервали передає до клієнта свій час, а клієнт його відображає. 2) Користувач з клієнта може виставити до 3-х таймерів (будильників), частина з яких прямого обліку часу, частина зворотнього. Сервер, крім системного часу, передає поточні значення для запущених таймерів та сигнал (чи транспорант) при спрацьовуванні таймера. (Клієнт лише відображає дані сервера, але сам він ніякий час не рахує.) |
| **12** | **Гра "хрестики-нолики".**  Гра "хрестики-нолики" на полі не менше 7х7 клітинок. Один гравець - на сервері, другий - на клієнті. Клієнт в ході гри може її завершити, почати нову гру, завершити сеанс. Після завершення сеансу сервер до клієнта передає статистику гри (ігор): кількість партій, кількість виграшів та програшів. |
| **13** | **Підбір пороля.**  Клієнт підбирає пароль, який зберігається на сервері. При запуску сервера задається його пароль. Сервер передає клієнту лише довжину пароля. З боку клієнта має бути можливість ручного підбору користувачем та автоматичного підбору на протязі заданого проміжку часу в секундах. Клієнт відображає номер поточної спроби та її вигляд. При вгадуванні пароля процедура автоматичного підбору зупиняється. Якщо був заданий проміжок часу на автобідбір, то сервер повертає кількість виконаних спроб. |
| **14** | **Порівняння/корекція файлів.**  Спочатку на клієнті та сервері зберігаються однакові текстові файли. На сервері можлива зміна у файлі. Клієнт чи по команді користувача, чи через задані інтервали часу корегує свій файл, згідно змін на сервері. За кожним зверненням клієнта сервер передає лише одну зміну у форматі <*Номер\_рядка*; *Нове\_значення\_рядка*>, але клієнт ці дії виконує у циклі (без спеціального переходу до наступного зчитування дією з боку користувача). |
| **15** | **Сортування рядків.**  Клієнт передає на сервер введені користувачем (можна передбачити також із файлу) текстові рядки. Сервер виконує сортування їх та повертає відсортованими до клієнта. Ключ сортування задається користувачем як стартова позиція та довжина ключа. |
| **16** | **Гра "морський бій".**  Грає лише користувач на клієнті. Сервер відповідає лише про результат попадання/знищення кораблика. Достатньо на сервері мати єдине статичне представлення поля з кораблями. Клієнт в матриці 10х10 повинен відмічати результати стрільби/попадань (без зайвої графічної реалізації, - достатньо використовувати декілька символів для позначень). Клієнт в ході гри може її завершити, почати нову гру, завершити сеанс. Після завершення сеансу сервер до клієнта передає статистику гри (ігор): кількість партій, кількість стрільб, кількість попадань та промахів. |
| **17** | **Реплікація даних.**  Клієнт і сервер мають однаковий набір змінних (числові, строкові тощо). Ці дані незалежно можуть змінюватися як на сервері, так і на клієнті. Але в певні моменти часу, нехай з ініціативи сервера, виконується синхронізація даних (репліковані дані в розподілених системах - майже завжди, с точністю до моменту синхронізації, співпадаючі дані). Отже, після синхронізації знову в обох програмах однакові дані. При синхронізації кожна з програм передає лише дані (змінні), які були змінені після попередньої синхронізації. Разом із новим значенням потрібно передавати час зміни ("штамп" часу). Якщо однойменні дані в обох програмах були змінені, то за нове значення береться те, що має останній "штамп" часу. У вікні кожної з програм потрібно відображати поточне значення змінних та мати можливість їх редагувати. |
| **18** | **Гра "вгадування 4-значного числа".**  Користувач на клієнті вгадує 4-значне ціле, яке зберігається на сервері. З клієнта передаються 4 цифри, на які сервер дає відповідь із двох цифр: кількість правильних цифр та кількість цифр на своїх місцях. Клієнт в ході гри може її завершити, почати нову гру, завершити сеанс. "Клієнт" може здатися і тоді сервер розкриває загадане число. |
| **19** | **Порівняння файлів.**  На клієнті та сервері спочатку зберігаються ідентичні текстові файли. З часом файли можуть змінюватися (засобами будь-якого текстового редактора), але вони будуть квазіідентичними, якщо: 1) символи верхнього і нижнього регістрів є ідентичними та 2) всі "білі" поля (з одного і більше пропусків) еквівалентні одному пропуску. По команді із клієнта виконується процедура порівняння, користувач на клієнті отримує список рядків, в яких порушена квазіідентичність. |
| **20** | **Вгадування числа за методом Монте-Карло.**  На сервері загадується ціле число в діапазоні 0..2000000. Користувач на клієнті задає час на автопідбір числа клієнтом за методом Монте-Карло (генеруванням випадкового значення). Після завершення інтервалу сервер передає кількість виконаних спроб та, якщо не було вгадування, найбільш близьке число. |
| **21** | **Ping.**  Клієнт ping-ує сервер (посилає повідомлення і вимірює час приходу відповіді) з такою частотою, щоб досягнути заданого навантаження на сегмент мережі (виходячи із поточного завантаження LAN). Повинна бути можливість вар'ювати частотою та розміром сокет-повідомлень. Завантаженість мережі можна контролювати за допомогою "Диспетчера задач Windows", вкладка "сеть". |
| **22** | **Тест "розмір повідомлення/швидкість".**  Побудувати (програмно) для поточного завантаження LAN залежність часу на відсилку серії із 10/50/100/1000/10000 сокет-повідомлень розміром 10/100/1000/10000 байт. Сервер, після отримання відповідної серії повідомлень, передає клієнту час всієї операції. |
| **23** | **Тестування продуктивності мережі.**  Клієнт та сервер почергово обмінюються повідомленнями, розмір яких завжди вибирається випадково в діапазоні від 10 до 50000 байт задану кількість разів (серія вимірювання). Результат вимірювання має таке представлення: сумарна кількість повідомлень, витрачений час, середній розмір повідомлення, середній час на повідомлення, min та max розмір повідомлення в серії. |
| **24** | **Диспетчер розділяємих змінних.**  Клієнт і сервер спільно використовують 10 змінних, які зберігаються на сервері. Змінні можна читати та задавати їм нове значення. Клієнт серією спроб повинен поміняти всім змінним значення. Доступ до змінних регулює (диспетчерезує) сервер. Випадковим чином диспетчер вибирає дозвіл чи заборону на зміну значення для кожної із змінних (так можна імітувати виконання змін ще від якихось інших клієнтів). Якщо змінна заблокована, то сервер посилає відмову клієнту, і той через таймаут, вибраний випадково в діапзоні [0.1, 0.3] секунди може повторити спробу (тобто сервер чергу запитів на зміни не веде). Проведіть серію тестів для визначення часу для проведення по 1 зміні значення для кожної змінної, по 10 та по 100 змін. Також потрібно зберігати кількість виконаних спроб для 1, 10 та 100 редагування для кожної змінної. (Нове значення для змінних предметної цінності не становить і може бути будь-яким.) |
| **25** | **Сканер портів.**  Клієнт сканує порти сервера і дає звіт про відкриті порти на ньому. На сервері має бути реалізована можливість відкривати/закривати певну кількість портів. |
| **26** | **Пошук числа методом *дихотомії* та *золотого перерізу*.**  На сервері задається ціле число в діапазоні [0,1000000], яке має знайти клієнт. Клієнт, знаючи діапазон можливих значень, для пошуку використовує (одночасно чи по черзі) методи *дихотомії* та *золотого перерізу*. На кожну пропозицію числа клієнтом сервер відповідає чи воно менше, дорівнює, чи більше загаданого числа. Після завершення процедур пошуку клієнт видає звіт про кількість ітерацій для кожного із методів та витрачений час. |
| **27** | **Відновлення об'єкта.**  На сервері у прямокутній області відомого для клієнта розміру зберігається певний об'єкт (може бути просто множина точок). Клієнт має N спроб для розпізнавання об'єкта (фігури), це буде правильно вгадана множина точок об'єкта в аналогічній прямокутній області. Вибір точки в області проводити за методом Монте-Карло (вибравши початок координат, наприклад, у нижній лівій вершині, і випадково вибираємо для поточної точки значення ширини та висоти).  Точка буде вгадана, якщо вона попадає в заданий окіл правильної точки об'єкта. На клієнті потрібно відобразити отриманий об'єкт. На сервері має бути відображений оригінальний об'єкт. Можна область з зображенням будувати у текстовому вікні. |

1. Регламентуючий документ RFC 1700. Список закріплених портів за протоколами міститься у файлі *protocol*, список портів служб Windows – у файлі *services* в каталозі <каталог Windows>\system32\drivers\etc (для Windows XP). Ці файли фактично лише інформують користувачів/розробників програм. Якщо ваша програма використовуватиме певний порт(и) постійно, то для відома інших користувачів/розробників є сенс прописати це в вищеназваних файлах на всіх хостах. [↑](#footnote-ref-1)