УДК 621.002
В.М. Мельник
Луцький національний технічний університет

ОСОБЛИВОСТІ СТВОРЕННЯ СОКЕТІВ З ВИКОРИСТАННЯМ РЕСУРСІВ ПЕРШОЇ БІБЛІОТЕКИ ДЛЯ ПЛАТФОРМИ WIN32 ТА ПІДХОДИ ДО ПОДІБНОЇ РЕАЛІЗАЦІЇ В UNIX

Проаналізовано алгоритм та інтерфейс створення потокового сокета з використанням можливостей першої бібліотеки для платформи Win32. Проаналізовано запропонований алгоритм та реалізовано створення сокета з застосуванням ресурсів UNIX засобами мови С.
Ключові слова: сокет, алгоритм, ресурс, платформа Win32, з'єднання, передача даних, клієнт, сервер.

Постановка проблеми. На сьогоднішній день на базі сокетів спроектовано величезну кількість програм для роботи з мережею, написано багато бібліотек, які слугують «обгортками» для сокетів, щоб абстрагуватися від деталей реалізації передачі даних [1-4]. Програмний інтерфейс платформи Win32 та створена системна бібліотека для роботи з мережею надає можливість працювати з базовою технологією передачі даних – програмними інтерфейсами взаємодії «клієнт-сервер». На базі сокетів працюють найпоширеніші протоколи 7-го (прикладного) рівня моделі OSI, такі як ftp, http, smtp, pop3 і т.д.

Однак слід зазначити, що **метою** постало питання аналізу алгоритму і підходів створення сокетів на базі платформи win32 і функцій першої версії бібліотеки як завжди застосовних. Перевагою послужило те, що всі вони можуть бути використані і в наступних версіях, так як підтримуються розробниками. Заодно, реалізація та порівняння вищезгаданих підходів і алгоритму побудови сокетів здійснювалася і можливостями ресурсів операційної системи UNIX на базі рідної їй мови програмування С.

Аналіз інтерфейсу реалізації та кроків алгоритму. Нагадаємо різницю алгоритмів передачі даних між клієнтом і сервером. Згідно з [2] основна частина клієнта алгоритмічно відрізняється з'єднанням з сервером та відсиланням йому даних (інструкцій). Перед цим проходять споріднені процедури заповнення відповідної структури параметрами сервера в обох випадках. Ініціалізація бібліотеки потрібна для завантаження необхідної її версії, а на сервер кладеться відповідальність за прослуховування порту. Клієнт зі своєї сторони з'єднується з сервером, відсилає йому інструкції і чекає відповіді. Після цього обидві сторони закривають з'єднання. Подібним чином це відбувається і в UNIX та Linux [5], тільки на базі реалізації їх типових можливостей.

Проаналізуємо функції та структури, необхідні для реалізації алгоритму передачі даних [2]. Представимо функцію завантаження бібліотеки:

int WSAStartup(WORD *wVersionRequested***,** LPWSADATA *lpWSAData***);**

Параметр wVersionRequested(in) – версія бібліотеки, для завантаження, де молодший байт визначає основну версію, старший – доповняльну. Параметр lpWSAData(out) – інформація про завантажену бібліотеку. Функція повертає нуль, або код помилки [6].

Після завантаження версії бібліотеки необхідно створити сокет. Всі функції для організації роботи клієнта і сервера визначені в межах першої бібліотеки [7-9]: функція socket(), структура sockaddr_in, функції bind(), connect(), listen(), accept(), send(), recv(), closesocket() та shutdown() для закриття з'єднання. Найпростіша функція для створення сокета може мати наступний прототип:

SOCKET socket(int *af***,** int *type***,** int *protocol*);

Параметр af(in) – визначає сімейство адрес, що використовуватимуться при передачі даних через сокети. Найчастіше використовується значення параметру AF_INET = 2. Інші значення поки що розглядати не будемо. Параметр type(in) – визначає тип створюваного сокета, де найчастіше використовується значення SOCK_STREAM і свідчить про використання протоколу із сімейства TCP при передачі даних. Для організації з'єднання з використанням сімейства протоколів UDP використовується значення SOCK_DGRAM. Параметр protocol(in) визначає протокол використання для передачі даних. В нашому випадку використовуємо значення IPPROTO_TCP. Нагадаємо, що функція повертає дескриптор створеного сокета.

Занесення даних, що визначають основні параметри підключення, здійснюється в спеціальну структуру, яка може приймати вигляд:

```
struct sockaddr_in
{
    short sin_family;
    unsigned short sin_port;
     struct in_addr sin_addr;
     char sin_zero[8];
};
```

де параметр *sin_family* визначає сімейство адрес і набуває значення AF_INET, параметр *sin_port* визначає порт IP, параметр *sin_addr* визначає адресу IP та параметр *sin_zero*, що визначає розмір структури sockaddr і не заповнюється.

Після заповнення структури необхідно асоціювати її з сокетом сервера для прийняття даних. Функція bind може набувати наступного вигляду:

int bind(SOCKET *s*, const struct sockaddr* *name*, int *namelen*);

Параметр **s**(in) – це сокет, що асоціюється зі структурою, параметр name(in) – сама структура, з якою асоціюється сокет, а параметр namelen(in) – розмір попереднього параметра в байтах. Функція bind() повертає нуль, а при помилці – її код.

На сервер кладеться відповідальність прослуховувати порт, що він і здійснює за допомогою функції listen(). Перший параметр (s(in)) асоціює сокет, який переводиться в режим прослуховування, а другий (backlog(in)) встановлює максимальну кількість підключень до сервера. Функція listen() повинна повернути нуль або код помилки.

При виявленні клієнта, який хоче підключитись до сервера, сервер повинен прийняти підключення через реалізацію функції accept(). Перший її параметр асоціює сокет підключення, а інші параметри – вихідні, нас не стосуються (встановлюються NULL). Функція повертає дескриптор сокета, що прийняв підключення при нормальному завершенні або код помилки.

У разі підключення клієнта до сервера, останній може отримувати від клієнта дані або інструкції для дій за допомогою функції recv(), поданої нижче:

int recv(SOCKET *s*, char* *buf*, int *len*, int *flags*);

Перший параметр представляє сокет, через який передаються дані. Другий і третій параметри відносяться до буфера даних (buf(out)) та його величини (len(in)) в байтах. Параметр flags(in) можна встановити в нуль. Функція повинна повернути кількість прийнятих байт або код помилки.

Після виконання операцій з'єднання можна закрити через функцію shutdown():

int shutdown(SOCKET *s*, int *how*);

де параметр **s**(in) – сокет, а параметр how(in) можна виставити в SD_BOTH для нашого випадку. У разі успіху функція повертає нуль, а у разі помилки – код помилки.

Останнім кроком потрібно закрити сокет, використавши функцію closesocket(), єдиним параметром якої є значення повернення функції socket(). Функція також повертає нуль, а в разі помилки – її код.

За умовою алгоритму реалізації зі сторони клієнта необхідно також виконати три перших етапи: завантажити бібліотеку, створити сокет та заповнити даними структури, що асоціюються із сокетом. Клієнт повинен також з'єднатися з сервером, використовуючи функцію connect():

int connect(SOCKET *s*, const struct sockaddr* *name*, int *namelen*);

Функція приймає вхідні параметри: **s**(in) – сокет, через який клієнт з'єднується з сервером, name(in) – ім'я сокета в структурі sockaddr, до якого буде під'єднуватися клієнт та розмір структури, в байтах namelen(in)

Після з'єднання клієнт може відправляти серверу інструкції (дані), використовуючи функцію send(), що може бути представлена прототипом:

int send(SOCKET *s*, const char* *buf*, int *len*, int *flags*);

Параметри мають такі ж значення, як і в раніше розглянутій функції recv().

На відповідному етапі спілкування з сервером клієнту потрібно отримувати результат від сервера. Проте клієнт також може бути сповіщений про будь-який крок виконання дій через вищерозглянуті функції. Після завершення всіх операцій з'єднання, сокет потрібно закрити.

Слід зазначити, що всі вищенаведені функції беруться із першої версії бібліотеки [5,7], але перевага в тому, що всі вони можуть бути використані і в наступних версіях, так як підтримуються їх розробниками. Звичайно, в другій версії можна використовувати більш удосконалені функції з більшими можливостями, які відрізняються префіксом WSA та кількістю параметрів.

Більш простою для взаємодії на рівні віддалених процесів в UNIX є схема організації спілкування клієнта і сервера за допомогою датаграм з використанням протоколу UDP [10].

Процес-сервер повинен спочатку налаштувати адресу сокета. При цьому сокет може бути прив'язаний до конкретного мережевого інтерфейсу чи до комп'ютера в цілому, тобто в повній адресі сокета може бути або вказана IP-адреса конкретного мережевого інтерфейсу, або дана вказівка операційної системи, що інформація може надходити через будь-який мережевий інтерфейс, що є в наявності. Процес-клієнт повинен спочатку здійснити ті ж самі підготовчі дії, а потім передати повідомлення, вказавши IP-адресу мережевого інтерфейсу і номер порту сервера.

Для створення сокета потрібно підключити бібліотеки *<sys/types.h> та <sys/soket.h>*. Кожній із вищенаведених дій відповідає конкретний системний виклик. Як і у платформі Win32, в UNIX створення сокета здійснюється за допомогою системного виклику soket(). Для прив'язки сокета до IP-адреси і номера порту застосовується системний виклик bind(). Для очікування отримання інформації, її читання і, за необхідності, визначення адреси відправника застосовується системний виклик recvfrom(). За відправку датаграми відповідає системний виклик sendto(). Для транспортних протоколів TCP/IP існує два види сокетів: UDP-сокет для роботи з датаграмами і TCP-сокет – це потоковий сокет, який і буде нас інтересувати. Для реалізації алгоритму в тому ж руслі системний виклик створення сокета буде мати вигляд:

int socket(int domain, int type, int protocol);

Параметр *domain* визначає сімейство протоколів передачі інформації. Для двох вищерозглянутих сімейств цей параметр приймає такі значення: *PF_INET* – для TCP/IP і *PF_UNIX* – для внутрішніх протоколів UNIX Domain. Параметр *type* визначає семантику обміну інформації. Будем користуватись двома способами і значеннями для параметра *type*: 1) *SOCK_STREAM* – для зв'язку за допомогою віртуального з'єднання; 2) *SOCK_DGRAM* – для обміну через повідомлення. Параметр *protocol* специфікує конкретний протокол для вибраного сімейства та способу зв'язку і приймає значення у разі, коли присутні декілька протоколів, а у нашому випадку приймає 0. У випадку успішного завершення виклик повертає файловий дескриптор як посилання на створений вузол комунікації при всіх подальших мережевих з'єднаннях, а у разі помилки ним повертається від'ємне значення.

Коли створений сокет, потрібно налаштувати його адресу, використавши системний виклик bind(), перший параметр якого містить дескриптор сокета для налаштування адреси. Другий і третій параметри задають цю адресу. Виклик bind() може бути поданий прототипом:

*int bind(int sockd, struct sockaddr *my_addr, int addrlen);*

де параметр *sockd* являється дескриптором комунікаційного вузла, значення повернення виклику *socket()*. Параметр *my_addr* – адрес структури прив'язки нашого сокета, має тип вказівника на структуру-шаблон *struct sockaddr*, яка повинна бути заповнена перед викликом. Параметр *addrlen* – це фактична довжина структури, адреса якої передається першим параметром.

Для роботи з TCP/IP використаємо дещо подібну структуру адреси сокета, що описана у файлі *<netinet/in.h>* [2]:

```
struct sockaddr_in
{
    short sin_family;          /*вибране сімейство протоколів – завжди AF_INET*/
    unsigned short sin_port; /*16-бітовий номер порту в мережевому порядку байта*/
    struct in_addr sin_addr; /*адрес мережевого інтерфейсу*/
    char sin_zerro[8];         /*поле не використовується, але повинно завжди бути
                                        заповнене нулями*/
}
```

Поле *sin_family* – задає сімейство протоколів і для нашого випадку прийме значення *AF_INET*. Віддалена частина IP-адреси міститься в структурі *struct in_addr*, а *sin_port* містить номер порту в мережевому порядку байт. Ситуація відрізняється від попередньої тим, що існує два варіанти задання порту: фіксований порт за бажанням користувача і порт, призначений довільно операційною системою.

Перший варіант вимагає в якості порту додатне, заздалегідь відоме число і для протоколу UDP використовується для налаштування адрес сокетів і передачі інформації. Другий варіант вимагає вказати замість значення порту 0, і тоді операційна система прив'язує сокет до вільного номера порту. Системні виклики sendto() і recvfrom() мають дещо інші параметричні подання:

```
int sendto(int sockd, char *buff, int nbytes, int flags, struct sockaddr *to, int addrlen);
int recvfrom(int sockd, char *buff, int nbytes, int flags, struct sockaddr *from, int addrlen);
```

Параметр *sockd* – це дескриптор сокета, значення повернення викликом socket(), через який приймається чи відправляється інформація. Параметр *buff* – адреса області пам'яті, з якої буде

братися інформація для передачі чи розміщуватися прийнята інформація. Параметр *nbytes* – це кількість байт, яка повинна бути передана/прийнята, починаючи з адреси пам'яті *buff*. Параметр *to/from* – посилання на структуру, що містить адресу сокета отримувача чи відправника, яку перед запуском виклику потрібно заповнити чи обнулити. Параметр *addrlen* містить фактичну довжину структури, адреса якої передається в якості параметра *to/from*. Якщо параметр *to/from* має значення *NULL*, то і параметр *addrlen* може приймати *NULL*. Параметр *flags* визначає режими використання системних викликів і в нашому випадку буде приймати значення 0. У разі успішного завершення системні виклики повертають кількість реально відісланих чи прийнятих байт, а при помилці – від'ємне значення.

В UNIX для визначення IP-адрес на комп'ютері можна скористатися утилітою */sbin/ifconfig*, яка видає всю інформацію про мережеві інтерфейси, сконфігуровані в обчислювальній системі. За приклад для протоколу Ethernet можна взяти таке її подання [4,10]:

Eth0 Link encap: Ethernet HWaddr 00:90:27:A7:1B:FE
inet addr: 192.168.253.12B cast:192.168.253.255
Maks: 255.255.255.0
UP BROADCAST NOTRAILERS RUNNING MULTICAST MTU:1500
Metric: 1 RX packets: 122556059 errors: 0 dropped: 0
Overruns: 0 frame: 0 TX packets: 116085111 errors: 0
Dropped: 0 overruns: 0 carrier: 0 collisions: 0
Txqueuelen: 100 RX bytes: 2240402748 (2136.6 Mb)
TX bytes: 3057496950 (2915.8 Mb) Interrupt: 10
Base address: 0x1000
lo Link encap: Local Loop back
inet addr: 127.0.0.1. Maks: 255.0.0.0.
UP LOOPBACK RUNNING MTU: 16436 Metric: 1
RX packets: 403 errors: 0 dropped: 0 overruns: 0 frame: 0
TX packets: 403 errors: 0 dropped: 0 overruns: 0
carrier: 0 collisions: 0 txqueuelen: 0
RX bytes: 39932 (38.2 Kb) TX bytes: 39932 (38.2 Kb)

В даному випадку мережевий інтерфейс *eth0* використовує протокол Ethernet. Фізична 48-бітова адреса, зашита в мережевій карті, – *00:90:27:A7:1B :FE*. Його IP-адреса 192.168.253.12. Мережевий інтерфейс *lo* – це локальний інтерфейс, який через спільну пам'ять емулює роботу мережевої карти для взаємодії процесів на одній машині за повними мережевими адресами. Наявність цього інтерфейсу дозволяє налаштовувати мережеві програми на машинах, що не мають мережевих карт, а його IP-адреса однакова.

Встановлення логічного з'єднання здійснює системний виклик connect()**.** Для клієнта під час роботи з TCP-сокетами він приховує всередині себе налаштування сокета на вибраний системою порт і довільний мережевий інтерфейс (а саме виклик bind() з нульовим значенням порту і IP-адресою *INADDR_ANY*).

int connect(int socket, struct sockaddr *servaddr, int addrlen);

Параметр *sockd* – дескриптор створеного комунікаційного вузла, тобто значення повернення виклику socket(). Параметр *servaddr* являє адресу структури для інформації про повну адресу сокета сервера і має тип вказівника на структуру-шаблон *struct sockaddr*, яка повинна бути конкретизована відносно сімейства протоколів, що використовується, і заповнена перед викликом. Параметр *addrlen* повинен містити фактичну довжину структури, адреса якої передається до його установки чи до встановленого в системі часу – *timeout*. Виклик *connect()* повертає 0 при нормальному завершенні і від'ємне значення при помилці.

Системний виклик listen() застосовується на TCP-сервері для переведення TCP-сокета в пасивний стан і формування черги для приєднаних сокетів, після з'єднання в стані не повністю і повністю встановленого зв'язків. Його параметри: дескриптор TCP-сокета і глибина черги:

int listen(int sockd, int backlog);

Виклик listen() потребує попереднього налаштування адреси сокета за допомогою виклику bind() та повертає значення 0 при нормальному завершенні і -1 – при виникненні помилки.

Системний виклик accept() дозволяє серверу виявити повну адресу клієнта, що встановив з'єднання:

int accept(int sockd, struct sockaddr *cliaddr, int *client);

Параметр *sockd* – дескриптор створеного і налаштованого сокета, переведеного в пасивний стан за допомогою виклику listen(). Виклик accept() також потребує попереднього налаштування адреси сокета за допомогою виклику bind(). Параметр *cliaddr* слугує для отримання адреси логічно з'єднаного клієнта і повинен містити вказівник на структуру адреси. Параметр *clilen* містить вказівник на змінну цілого типу, яка після повернення з виклику буде містити фактичну довжину адреси клієнта. Перед викликом вона повинна містити максимально допустиме значення такої довжини. Якщо параметр *cliaddr* приймає *NULL*, то параметр *clilen* також може приймати *NULL*. Значення повернення такі ж, як і у попередньому виклику.

На даному кроці аналізу уже слід відзначити гнучкість операційної системи UNIX та її можливості для створення сокетів через співмірні алгоритми.

Результати реалізації. В межах реалізації наших представлень наведемо приклад коду клієнт-серверної програми для передачі повідомлень по локальній мережі. На базі Win32 сервер містить дві функції: одна створює сокет, робить прослуховування та приймає підключення, інша – приймає дані, причому перша в окремому потоці (для уникнення зависання програми під час прийняття даних) викликає іншу.

```
//функція створення сокета, прослуховування та підключення
DWORD WINAPI ServerThread(LPVOID lpParam)
{
    SOCKET sServerListen, sClient;
    sockaddr_in localaddr, clientaddr;
    HANDLE hThread;
    DWORD dwThreadId;
    INT iSize;
    sServerListen = socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
    if(sServerListen == SOCKET_ERROR) return 0;
    localaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    localaddr.sin_family = AF_INET;
    localaddr.sin_port = htons(5050);
    if(bind(sServerListen,(struct sockaddr*) &localaddr, sizeof(localaddr))==SOCKET_ERROR) return 1;
    listen(sServerListen, 4);
    while (1)
    {
          iSize = sizeof(clientaddr);
          sClient = accept(sServerListen, (struct sockaddr *)&clientaddr,&iSize);
          if(sClient == INVALID_SOCKET) break;
          hThread = CreateThread(NULL, 0, ClientThread (LPVOID sClient), 0, &dwThreadId);
          if(hThread == NULL) break;
          CloseHandle(hThread);
     }
     closesocket(sServerListen);
     return 0;
}
//функція прийому даних
DWORD WINAPI ClientThread(LPVOID lpParam)
{
     SOCKET sock=(SOCKET)lpParam;
     TCHAR szRecvBuff[10240], szSendBuff[10240], szTimeString[64];
     UINT ret;
     while(1)
     {
         ret = recv(sock, szRecvBuff, 10240, 0);
         if (ret == 0) break;
        else if (ret == SOCKET_ERROR) break;
        szRecvBuff[ret] = '\0';
        SendDlgItemMessage(hWndDlg, IDC_EDITRECV, WM_SETTEXT, 0, (LPARAM) szRecvBuff);
     }
     return 0;
```

}

Спочатку ми створюємо сокети sServerListen, sClient, нитку-потік (далі нитка) виконання hThread з її параметрами та об'являємо структури localaddr і clientaddr типу sockaddr_in. Створення потокового сокета обумовлює застосування в якості першого параметра значення AF_INET, а два інші нам відомі з опису функції socket(). При умові успішного створення сокета проходить заповнення структури localaddr та її полів, використовуючи функції мережевого порядку байт htonl() і htons() для реалізації з'єднання через функцію bind(). В якості першого параметра виступає сокет sServerListen, а інші два параметри уже відомі нам з попередніх кроків. Сервер реалізує прослуховування listen(sServerListen, 4) з максимальним підключенням, рівним 4.

В зацикленні (очікуванні) спрацьовує функція accept(), першим параметром якої є сокет sServerListen, другим і третім – вказівник на структуру (struct sockaddr *) &clientaddr, та її розмір &iSize. Змінна sClient повинна отримати дескриптор сокета. Якщо він отриманий, то створюється нитка hThread через використання функції CreateThread() з її відповідними параметрами, всередині якої здійснюється виклик функції ClientThread() для прийому переданих даних клієнтом.

В функції ClientThread фіксується сокет передачі та організовується необхідна пам'ять для її подальшої роботи. Функція recv(sock, szRecvBuff, 10240, 0) організовує отримання даних від клієнта, що поміщаються в буфер szRecvBuff[ret]. Після цього відправляється сервером повідомлення через функцію SendDlgItemMessage() і управління переходить в першу функцію, яка закриває нитку та сокет.

Реалізація UNIX-програми для TCP-сервера.

```
/*Приклад простого TCP-сервера для сервісу echo.*/
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet.h>
#include <arpa/inet.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
void main();
{
	int sockfd, new sockfd;/*дескриптори для слухового і приєднаного сокетів*/
	int clilen;		/*довжина адреси клієнта*/
	int n;			/*кількість прийнятих символів*/
	char line[1000];	/*буфер для прийому інформації*/
	struct sockaddr_in servaddr, cliaddr;/*структури для розміщення повних адрес сервера і
                                                                клієнта*/
	/*Створюєм сокет TCP.*/
	if((sockfd=socket(AF_INET, SOCK_STREAM, 0))<0)
	{
		perror(NULL);
		exit (1);
	}
	/*Заповнюємо структуру для адреси сервера: сімейство протоколів TCP/IP, номер порту
	51000. Обнулюємо структуру перед заповненням.*/
	bzero(&servaddr, sizeof(servaddr));
	servaddr.sin_family=AF_INET;
	servaddr.sin_port=htons(51000);
	servaddr.sin_addr.s_addr=htonl (INADDR_ ANY);
	/*Налаштовуємо адресу сокета.*/
	if(bind(sockfd, (struct sockaddr*) &servaddr, sizeof (servaddr))<0)
	{
		perror (NULL);
		close (sockfd);
		exit (1);
	}
```

```
/*Переводимо створений сокет в пасивний стан. Глибину черги описуємо значенням 5.*/
if(listen(sockfd, 5)<0)
{
        perror (NULL);
        close (sockfd);
        exit (1);
}
/*Основний цикл сервера.*/
while(1)
{
        /*В першому clilen заносимо максимальну довжину адреси клієнта.*/
        clilen=sizeof(cliaddr);
        /*Очікуємо повністю налаштованого зв'язку. В структурі cliaddr буде повна адреса клієнта, а в змінній clilen – його фактична довжина. Через дескриптор буде відбуватись спілкування з клієнтом. Другий і третій параметри приймають NULL.*/
        if((newsockfd=accept(sockfd (struct sockaddr*) &cliaddr, &clilen)) <0)
        {
                perror(NULL);
                close(sockfd);
                exit (1);
        }
        /*В циклі приймаємо інформацію від клієнта доти, поки не виникне помилка (виклик read() поверне від'ємне значення) або клієнт не закриє зв'язок (read() поверне 0). Максимальну довжину даних від клієнта обмежимо 999 символами. В операціях читання і запису скористаємося дескриптором приєднаного сокета.*/
        while(n=read(newsockfd, line, 999))>0)
        {
                /*прийняті дані відправляємо назад*/
                if((n=write(newsockfd, line, strlen (line+1))) > 0)
                {
                        perror (NULL);
                        close (newsockfd);
                        exit (1);
                }
        }
        /*Якщо при читанні виникла помилка, завершуємо роботу!*/
        if(n<0)
        {
                perror(NULL);
                close(sockfd);
                close(newsockfd);
                exit (1);
        }
        /*Закриваємо дескриптор приєднаного сокета і виходимо очікувати нового зв'язку.*/
        close (newsockfd);
}
}
```

Клієнт має функцію для з'єднання з сервером і відправки даних, яка для win32:

```
DWORD APIENTRY ClientProc(LPVOID lpParam)
{
    SOCKET sClient;
    char szBuffer[255];
    int ret, i;
    sockaddr_in server;
    hostent *host = NULL;
    char szServerName[255], szMessage[10240];
```

```
    HWND hWndDlg = (HWND)lpParam;
    SendDlgItemMessage(hWndDlg, IDC_HOST, WM_GETTEXT, 255, (LPARAM)szServerName);
    SendDlgItemMessage(hWndDlg, IDC_EDITSEND,WM_GETTEXT,10240, (LPARAM)szMessage);
    sClient = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if(sClient == INVALID_SOCKET)
    {
         MessageBox(hWndDlg, "Can't create socket", "Error", 0);
         return 1;
     }
         server.sin_family = AF_INET;
         server.sin_port = htons(5050);
         server.sin_addr.s_addr = inet_addr(szServerName);
         if(connect(sClient, (struct sockaddr *)&server, sizeof(server)) == SOCKET_ERROR)
        {
              MessageBox(hWndDlg, "connect failed", "Error", 0);
              return 1;
        }
        ret = send(sClient, szMessage, strlen(szMessage), 0);
        if(ret == SOCKET_ERROR) MessageBox(hWndDlg, "send failed", "Error", 0);
        Sleep(1000);
        char szRecvBuff[255];
        ret = recv(sClient, szRecvBuff, 255, 0);
        if(ret == SOCKET_ERROR) MessageBox(hWndDlg, "recv failed", "Error", 0);
        closesocket(sClient);
        return 0;
}
```

З організацією вихідних даних та об'єктів для створення функції з'єднання клієнта з сервером потрібно задавати в функціях-викликах socket() перший параметр AF_INET. Він повинен відповідати і значенню поля структури struct in_addr в структурі sockaddr_in, екземпляром якої є створений об'єкт server. Функції SendDlgItemMessage() формуються в звичайному для них порядку, враховуючи ім'я сервера та повідомлення і відповідну їм зарезервовану пам'ять. Так, якщо відповідні масиви мали вигляд:

```
char szServerName[255], szMessage[10240];
```

то функції набудуть вигляду [3]:

```
SendDlgItemMessage(hWndDlg, IDC_HOST, WM_GETTEXT, 255, (LPARAM)szServerName);
SendDlgItemMessage(hWndDlg, IDC_EDITSEND, WM_GETTEXT, 10240, (LPARAM)szMessage);
```

Після цього слід перевірити значення повернення функції socket(), яке передалося об'єкту sClient. Якщо сокет створено (не INVALID_SOCKET), то здійснюємо формування параметрів та застосовуємо функції connect() для встановлення з'єднання і send() для відправлення повідомлення.

Після цього потрібно бути готовим клієнту отримати інформацію від сервера. Треба сформувати параметри та функцію recv(), в якій буде виступати сокет sClient для відсилання повідомлення серверу, а другий і третій параметри відповідають імені і пам'яті масиву для отримання інформації від сервера.

У випадку нормального завершення всіх функцій в процедурі клієнта слід тільки здійснити закриття сокета, застосувавши функцію closesocket(), яка вимагає імені сокета.

```
/*Простий приклад UNIX-програми TCP-клієнта для сервісу echo.*/
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
void main (int argc, char **argv);
{
```

```
int sockfd; /*дескриптор сокета*/
int n;      /*кількість переданих чи прочитаних символів*/
int i;      /*лічильник циклу*/
char sendline[1000], recvline[1000]; /*масиви для лінійок відсилання і прийняття*/
struct sockaddr_in servaddr; /*структура для адреси сервера*/
/*Перевіряємо наявність другого аргументу, при його відсутності завершуємо роботу.*/
if(argc!=2)
{
        printf ("Usage: a.out <IP-address.\n");
        exit (1);
}
/*обнулюємо символьні масиви*/
bzero(sendline, 1000);
bzero(recvline, 1000);
/*створюємо TCP-сокет*/
if((sockfd=socket(PF_INET, SOCK_STREAM, 0)) <0)
{
        perror (NULL); /*друкуємо повідомлення про помилку*/
        exit (1);
}
/*Заповнюємо структуру для адреси сервера: сімейство протоколів TCP/IP, мережевий інтерфейс – із аргументу командної лінійки, номер порту 7. Перед заповненням обнулюємо структуру.*/
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family=AF_INET;
servaddr.sin_port=htons(51000);
if(inet_aton (argv[1], &servaddr.sin_addr)==0)
{
        printf("Invalid IP-address!\n");
        close(sockfd);
        exit (1);
}
/*Встановлюємо логічний зв'язок через створений сокет з сокетом сервера, адресу якого ми занесли в структуру.*/
if(connect(sockfd, (struct sockaddr*) &servaddr, sizeof(servaddr))<0)
{
        perror(NULL);
        close(sockfd);
        exit (1);
}
/*Три рази в циклі вводимо лінійку з клавіатури, відправляємо серверу і читаємо відповідь.*/
for(i=0; i<3; i++)
{
        printf("string =>");
        fflush(stdin);
        fgets (sendline, 1000, stdin);
        if((n=write (sockfd, sendline, strlen (sendline)+1))<0)
        {
                perror("Can\'t write!\n");
                close(sockfd);
                exit (1);
        }
        if((n=read (sockfd, recvline, 999))<0)
        {
                perror("Can\'t read!\n");
                close(sockfd);
                exit (1);
```

```
            }
            printf("%s", recvline);
        }
        /*завершуємо зв'язок*/
        close(sockfd);
}
```

Перед запуском відкомпільованої програми слід пересвідчитися, чи запущений в системі TCP-сервіс echo. Якщо в якості IP-адреси вказати неіснуючу адресу чи адресу вимкненої машини, то програма повідомить про помилку при роботі виклику connect() і доведеться почекати закінчення timeout'а. Якщо логічне з'єднання встановити не вдалося, то відправник про це також отримає повідомлення.

Висновок. Отже, володіючи платформою Win32 та бібліотекою ресурсів для створення програм-сокетів можна вигідно і легко створювати подібні мережеві програми для з'єднання клієнта з сервером. Не виключне застосування ресурсів удосконалених версій бібліотеки, проте переваги в тому, що перша версія її має підтримку у всіх наступних версіях. Слід також відзначити, що подібними підходами можна організовувати створення функцій з'єднання клієнта з сервером і в UNIX, яка дозволяє більш гнучкі і послідовні можливості на базі мови програмування С.

Література:

1. Security Architecture for Open System Interconnection for CCITT Applications. Recommendations X.800 [Текст] / CCITT.– Geneva,1991.– 280 с.
2. Brian Hall. Використання інтернет-сокетів. / Пер. з англ.– 2001.– 68 с.
3. Таймэн Б. FreeBSD 6. Полное руководство [Текст] / Б. Таймэн; Пер. с англ.– М.: ООО «И.Д. Вильямс», 2007.– 1056 с: ил.
4. Стивенс У.И. UNIX: разработка сетевых приложений [Текст] / У.И. Стивенс; СПб.: Питер, 2003.– 1088 с.: ил.
5. В.Є. Карпов, К.А. Коньяков. Основы операционных систем. Лекции и семинары Интернет-университет информационных технологий. [Текст] М.– Intel, 2005.– 536 с.
6. В. Несвижский. Прграммирование аппаратных средств в Windows. С.-Петербург: БХВ-Петербург,– 2004.– 880 с.
7. М.Вахтеров, С.Орлов Четвертый Borland C++и его окружение. / М.: «ИНТЕЛ», 1999.-1200 с.
8. Олифер Н.А., Олифер В.Г. Сетевые операционные системы: [Электронный курс].– режим доступу http://www.citmgu.ru/.
9. Белломо М. UNIX. Наглядний курс освоения операционных систем [Текст] / М. Белломо; М; С.П.; К., 2001.– 336 с.
10. Магда Ю.С. UNIX для студента [Текст] / Ю.С. Магда; Санкт-Петербург, 2007.– 480 с.