УДК 621.002
В.М. Мельник
Луцький національний технічний університет

ОСОБЛИВОСТІ СОКЕТНОЇ ВЗАЄМОДІЇ МІЖ ПРОЦЕСАМИ В UNIX ТА LINUX

Розглянуто підходи міжпроцесної взаємодії в UNIX з використанням файлових сокетів та інтерфейсу для їх реалізації. Подано можливості використання парних сокетів як каналів для взаємодії споріднених процесів. Запропоновано можливості програмної реалізації мережевих неблокуючих сокетів для взаємодії між клієнтом і сервером.

Постановка проблеми. На сьогоднішній день сокети займають одне із найвизначніших місць в мережевому програмуванні [1-4]. Особливо це стосується найбільш провідних операційних систем, які є досить визнаними з усіма їхніми перевагами для застосування в мережі. В даний момент хотілося б говорити про операційні системи сімейства UNIX та Linux як мережеві, в яких сокетна взаємодія між віддаленими процесами займає важливе місце. Як і інші засоби міжпроцесної взаємодії, сокети вперше були реалізовані саме на платформі UNIX версії 4.2BSD. Слід також добавити, що концепція сокетів, як одного з найбільш цікавих і досконалих засобів обміну даними між процесами в мережі, стала настільки прогресивною, що сьогодні всі новітні системи підтримують, в крайньому випадку, хоча б деяку частину сокетів. Причини їх успіху полягають в простоті підходів та універсальності.

Програми, які обмінюються даними за допомогою сокетів, можуть працювати в одній так і в різних системах, що використовують для обміну даними як спеціальні об'єкти системи, так і мережевий стек. Як і канали зв'язку, сокети використовують простий інтерфейс, що базується на звичайних файлових функціях, таких як read() і write(). При відкритті сокета програма UNIX отримує дескриптор файла, що дозволяє організувати подальшу роботу з ними, включаючи вищезгаданий інтерфейс файлових функцій чи системних викликів. Але на відміну від каналів зв'язку, сокети дозволяють передавати дані в обох напрямках: в синхронному режимі, так і в асинхронному.

Аналіз останніх досліджень. Насьогодні більшість програмістів взяли за мету використання різного роду бібліотек високого рівня для роботи з сокетами та їх програмної реалізації [2]. Тим, хто програмує сокети для операційної системи Windows, досить знайомими є мережеві сокети, які, як правило, організовують обмін даними за допомогою протоколів сімейства TCP/IP. Проте в операційних системах сімейства UNIX є наявні й інші види сокетів, які спеціально передбачені для організації комунікації і обміну даними між локальними процесами [3,4]. Слід все-таки виразити думку, що високорівневі бібліотеки не дозволяють задіяти всю потужність і можливу багатосторонність сокетів. Найочевидніший приклад багатосторонності має прояв в файлових сокетах. Отже, ***метою даної статті*** слугують з'ясування окремих програмних особливостей сокетної взаємодії для організації комунікації та передачі даних між спорідненими процесами.

Основна частина. Спочатку потрібно зупинитись на файловому просторі імен. Сокети в файловому просторі імен (*file namespace*), які також називають UNIX-сокети, використовують в якості адрес через імена файлів спеціального типу. Важливою особливістю таких сокетів є те, що з'єднання за допомогою них локального і віддаленого додатків неможливе і при умовах, якщо і файлова система, в якій створений сокет, доступна віддаленій операційній системі. Для наглядного прикладу створимо сокет и зв'яжемо його з файлом socket.soc.

```
sock = socket(AF_UNIX, SOCK_DGRAM, 0);
if(sock < 0)
{
```

© В.М. Мельник

```
    perror("socket is failed!");
    return EXIT_FAILURE;
}
srvr_name.sa_family = AF_UNIX;
strcpy(srvr_name.sa_data, "socket.soc");
if(bind(sock, &srvr_name, strlen(srvr_name.sa_data) +
        sizeof(srvr_name.sa_family)) < 0)
{
    perror("bind failed!");
    return EXIT_FAILURE;
}
```

Константи і функції, які необхідні для роботи з сокетами в файловому просторі імен, об'явлені в файлах <sys/types.h> і <sys/socket.h>. Так як і файли, сокети в програмах представлені дескрипторами, які можна отримати за допомогою функції socket().

Перший параметр цієї функції – це домен, до якого належить сокет. Домен сокета визначає тип з'єднання. Якщо домен визначений константою AF_UNIX, то він відповідає сокетам для файлового простору імен. Другий параметр функції socket() визначає тип сокета, а значення SOCK_DGRAM вказує, що це датаграмний сокет [1]. Такого типу сокети здійснюють ненадійні з'єднання для передачі даних по мережі і підтримують передачу даних у вигляді широких повідомлень. Ще інший, тип сокетів, що часто використовується – це SOCK_STREAM, який відповідає потоковим сокетам, що реалізують з'єднання за принципом «точка-точка» і надійну передачу даних. Однак, в просторі файлових імен і потокові і датаграмні сокети надійні. Третій параметр функції socket() дозволяє вказати протокол, що буде використовуватися для передачі даних. Його залишимо рівним нулю, пам'ятаючи, що при помилці функція socket() повертає -1.

Після отримання дескриптора сокета слід викликати функцію bind(), яка організовує зв'язок для сокета з заданою адресою в програмі-сервері. Першим параметром функції bind являється дескриптор сокета, а другим – вказівник на структуру sockaddr і змінну srvr_name, яка містить адресу, на яку реєструється сервер. Третій параметр цієї функції містить довжину структури, яка містить адресу реєстрації сервера. Замість загальної структури sockaddr для сокетів UNIX в файловому просторі імен можна використовувати спеціалізовану структуру sockaddr_un. Поле sockaddr.sa_family дозволяє вказувати сімейство адресів, якими потрібно скористатися. В нашому випадку – це сімейство адрес файлових сокетів UNIX AF_UNIX. Сама адреса сімейства AF_UNIX (поле sa_data в спеціалізованій структурі) являє собою звичайне ім'я файла сокета. Після виклику bind() програма-сервер стає доступна для з'єднання через задану адресу, тобто ім'я файла.

При обміні даними з датаграмними сокетами будемо використовувати не файлові функції write() і read(), а спеціальні функції recvfrom() і sendto(). Вони також можуть і застосовуватися при роботі з потоковими сокетами, але у відповідному прикладі ми використаємо файлові функції read() і write(). Для читання даних із датаграмного сокета застосуємо функцію recvfrom(), яка за замовчуванням блокує програму до тих пір, поки на вході не появляться нові дані.

```
bytes=recvfrom(sock, buf, sizeof(buf), 0, &rcvr_name, &namelen);
```

З викликом функції recvfrom(), ми передаємо їй вказівник на ще одну структуру типу sockaddr, в якій функція повертає дані про адресу клієнта, що подав запит на з'єднання. У випадку файлових сокетів цей параметр не несе корисної інформації. Останній параметр функції recvfrom() – це вказівник на змінну, якій буде повернена довжина структури з адресою. Якщо інформація про адресу клієнта нас не цікавить, то можемо передати значення NULL в двох останніх параметрах. По завершені роботи з сокетом закриваємо його, використовуючи файлову функцію close(). Перед виходом із програми-сервера потрібно видалити файл сокета, який був створений функцією socket(). Це можна реалізувати через використання функції unlink() [3].

© В.М. Мельник

Якщо програма-сервер є простою, то програма-клієнт є ще простішою. В ній відкривається сокет за допомогою функції socket() і дані передаються серверу за допомогою функції sendto():

```
srvr_name.sa_family = AF_UNIX;
strcpy(srvr_name.sa_data, SOCK_NAME);
strcpy(buf, "Hello, Unix sockets!");
sendto(sock, buf, strlen(buf), 0, &srvr_name, strlen(srvr_name.sa_data) +
    sizeof(srvr_name.sa_family));
```

Перший параметр функції sendto() представляє дескриптор сокета. Другий і третій параметри подають адресу буфера для передачі даних і його довжину. Четвертий параметр призначений для передачі додаткових прапорів. Два останні параметри несуть інформацію про адресу сервера і його довжину, відповідно. Якщо для роботи з датаграмними сокетами викликати функцію connect(), як наведено нижче, то можна не вказувати адреси призначення кожен раз, а тільки один раз в якості параметра функції connect()). Перед викликом функції sendto() треба заповнити структуру sockaddr, а саме, змінну srvr_name даними про адресу сервера. Після закінчення передачі даних слід закрити сокет, використавши функцію close(). Якщо запустити програму-сервер, а за нею і програму-клієнт, то сервер роздрукує тестову лінійку інформації, яка передана клієнтом.

В файловому просторі імен сокети подібні на іменовані канали за однією прикметою, для ідентифікації сокетів використовуються файли спеціального типу. Серед сокетів існує і аналог неіменованих каналів – це сокети-пари (*socket pairs*), або *парні сокети*. То ж як і неіменовані канали, парні сокети створюються парами і не володіють іменами. Область застосування парних сокетів така ж, як і неіменованих каналів, – для взаємодії меж батьківським и новостворением процесами. Як і в випадку неіменованого каналу, один із дескрипторів використовується одним процесом, а інший – другим. В якості прикладу використання парних сокетів розглянемо програму sockpair.c, яка створює два процеси за допомогою функції fork(). В якості прикладу для обміну англійським вітанням дочірні процеси програми sockpair.c використовують парні сокети.

```
//програма sockpair.c
#include <sys/types.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#define STR1 "How are you?"
#define STR2 "I'm Ok, thank you."
#define BUF_SIZE 1024
int main(int argc, char **argv)
{
   int sockets[2];
   char buf[BUF_SIZE];
   int pid;
   if(socketpair(AF_UNIX, SOCK_STREAM, 0, sockets) < 0)
   {
      perror("socketpair() failed");
      return EXIT_FAILURE;
   }
   pid = fork();
   if(pid != 0)
   {
      close(sockets[1]);
      write(sockets[0], STR1, sizeof(STR1));
```

© В.М. Мельник

```
        read(sockets[0], buf, sizeof(buf));
        printf("%s\n", buf);
        close(sockets[0]);
    }
    else
    {
       close(sockets[0]);
       read(sockets[1], buf, sizeof(buf));
       printf("%s\n", buf);
       write(sockets[1], STR2, sizeof(STR2));
       close(sockets[1]);
    }
}
```

Парні сокети створюються функцією socketpair(), яка містить чотири параметри такі ж, як і у функції socket(), а четвертий параметр містить масив із двох змінних, які отримують повернені дескриптори. Дескриптори сокетів, повернені функцією socketpair(), уже готові до передачі даних і зразу можна застосовувати до них файлові функції read() і write(). Після виклику fork() кожен процес отримує два дескриптори, один із яких він повинен закрити, використовуючи функцію close().

З точки зору програмування парних сокетів можна поставити питання: "Чому ці функції відносяться до сокетів?" Адже при роботі з ними не використовуються ні адреси, ні модель клієнт-сервер. Все ж можна помітити, що функції socketpair() передаються значення домена і типу сокета; то ж як формально, так і з точки зору реалізації в системі дійсно використовуються сокети. Відмітимо, що задавання домена в функції socketpair() лишнє, оскільки система підтримує для неї тільки сокети в домені AF_UNIX. З визначення вище це є зовсім логічне обмеження, тому що, як говорилося, парні сокети не мають імен, і призначення їх – для обміну даними між спорідненими процесами.

Особливо важливим типом сокетів являються *мережеві сокети*. Значення, яке приймають мережеві сокети в UNIX-системах, відоме [2]. Використання мережевих сокетів робить процес масштабування проекту стійким, хоча в мережевих сокетів існують і недостатки. Якщо навіть сокети використовуються для обміну даними на одній і тій же машині, то дані передавання повинні пройти всі рівні мережевого стеку. Це, як правило, негативно впливає на швидкодію і навантаження на систему.

В якості прикладу розглядався комплекс із двох додатків: клієнта і сервера, які використовують мережеві сокети для обміну даними. З цією метою нижче наведемо тільки деякі необхідні для нашої необхідності програмні уривки. Найперше, потрібно отримати дескриптор сокета:

```
sock = socket(AF_INET, SOCK_STREAM, 0);
if(sock < 0)
{
     printf("socket() failed: %d\n", errno);
     return EXIT_FAILURE;
}
```

Першим параметром передається константа AF_INET, яка вказує, що сокет, який відкривається, повинен бути мережевого типу. Значення другого параметра вимагає потокового сокета. В наступному, як і для випадку сокета в файловому просторі імен, викликається функція bind():

```
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = INADDR_ANY;
serv_addr.sin_port = htons(port);
if(bind(sock, (struct sockaddr*) &serv_addr,
  sizeof(serv_addr)) < 0)
```

© В.М. Мельник

```
{
    printf("bind() failed: %d\n", errno);
    return EXIT_FAILURE;
}
```

Змінна serv_addr, – це структура типу sockaddr_in, тип якої призначений для зберігання адрес в форматі Інтернету. Основна відмінність sockaddr_in від sockaddr_un полягає в наявності параметра sin_port, призначеного для зберігання значення порта. Функція htons() переписує двобайтове значення порта так, щоб порядок байтів відповідав плану порядку, прийнятому в Інтернет (див. нижче). В якості сімейства адрес вказуємо значення аргументу AF_INET (сімейство Інтернету), а в якості самої адреси – спеціальну константу INADDR_ANY. Завдяки цій константі програма-сервер зареєструється на всіх адресах тієї станції, на якій вона виконується.

Літературні терміни little-endians і big-endians визначають порядок байт, що використовується процесором для представлення простих багатобайтних типів (32-бітового цілого). На процесорах Intel порядок байт little-endians «гострокутний», а в системах MacOS X – Power PC Sun SPARC – big-endians [2,3]. Однак протоколи Інтернет використовують «тупокутний» порядок little-endians. У всіх системах, включаючи big-endians рекомендовано використовувати функцію htons(), яка «знає» порядок байт в системі і, по необхідності, приводить його в відповідний для протоколів TCP/IP. В [1,3] використовують термінологію «прямий порядок байт» для little-endian и «обернений порядок байт» для big-endian.

Робота мережевої підсистеми UNIX чи будь-якої іншої полягає в тому, що мережевий сервер повинен вміти виконувати запити багатьох клієнтів (в загальному) одночасно. При цьому в з'єднаннях при використанні потокових сокетів, для кожного клієнта на сервері повинен бути відкритий окремий сокет. Це значить, що не треба встановлювати з'єднання з клієнтом через сокет sock, призначений для прослуховування вхідних запитів (але для мережевих сокетів цього зробити не можна), інакше всі інші намагання з'єднатися з сервером за вказаною адресою і портом будуть заблоковані. Для цього викликаємо функцію listen(), яка переводить сервер в режим очікування запиту у вигляді коду:

```
listen(sock, 1);
```

Другий параметр listen() становить максимальне число з'єднань, які сервер може опрацювати одночасно. Далі викликаємо функцію accept(), яка встановлює з'єднання на запит клієнта:

```
newsock = accept(sock, (struct sockaddr *) &cli_addr, &clen);
if(newsock < 0)
{
    printf("accept() failed: %d\n", errno);
    return EXIT_FAILURE;
}
```

Отримавши запит на з'єднання, функція accept() повертає новий сокет, відкритий для обміну даними з клієнтом, що з'єднався. Сервер як би перенаправляє запит для з'єднання на інший сокет, залишаючи сокет sock вільним для прослуховування запитів на встановлення з'єднання. Другий параметр функції accept() містить інформацію про адресу клієнта, що подав запит на з'єднання. Третій параметр вказує розмір другого. Як і для виклику функції recvfom(), можемо передавати NULL в останніх двох параметрах. Для читання і запису даних сервер використовує файлові функції read() і write(), а для закриття сокетів – функцію close(). В програмі-клієнті потрібно вирішити проблему, якої не було в програмуванні сервера. Слід виконати перетворення доменного імені сервера в його мережеву адресу, тобто застосувати для цього функцію gethostbyname():

```
server = gethostbyname(argv[1]);
if(server == NULL)
```

© В.М. Мельник

```
{
    printf("Host not found\n");
    return EXIT_FAILURE;
}
```

Функція отримує вказівник на рядок з Інтернет-іменем сервера (для прикладу, www.unix.com чи 192.168.1.16) и повертає вказівник на структуру hostent (змінна server), яка містить ім'я сервера в потрібному виді для подальшого використання. Якщо ж необхідно, то виконується приведення доменного імені в мережеву адресу. Далі записуємо в поля змінної serv_addr (структури sockaddr_in) значення адреси и порту:

```
serv_addr.sin_family = AF_INET;
strncpy((char *)&serv_addr.sin_addr.s_addr,
(char *)server->h_addr, server->h_length);
serv_addr.sin_port = htons(port);
```

Програма-клієнт відкриває новий сокет за допомогою функції socket() так же, як і в випадку з сервером (дескриптор сокета, повернений функцією socket() в змінній sock), і викликає функцію connect() для встановлення з'єднання:

```
if(connect(sock, &serv_addr, sizeof(serv_addr)) < 0)
{
    printf("connect() failed: %d", errno);
    return EXIT_FAILURE;
}
```

Тепер сокет готовий до передачі і прийому даних. Програма-клієнт зчитує символи, що вводяться з терміналу. Після натиснення клавіші вводу програма передає дані серверу, очікує зворотне повідомлення сервера і друкує його.

Зараз зупинимось на *неблокуючих сокетах*. Треба знати, що ними можна не користуватися. Завдяки багатопотоковому (багатопрограмному) програмуванню можна використовувати блокуючі сокети у всіх ситуаціях (при необхідності обробки одночасно кількох сокетів і при перериванні операції над сокетом). За замовчуванням функція socket() створює блокуючий сокет. Щоб зробити його не блокуючим, використаємо функцію fcntl():

```
sock = socket(PF_INET, SOCK_STREAM, 0);
fcntl(sock, F_SETFL, O_NONBLOCK);
```

Тепер будь-який виклик функції read() для сокета sock зразу буде повертати управління. Якщо на вході сокета немає даних для читання, функція read() поверне значення EAGAIN. Для перевірки стану неблокуючих сокетів можна використати функцію select(), яка може перевіряти стан декількох дескрипторів сокетів або файлів. Перший параметр функції – кількість дескрипторів перевірки. Наступні три параметри являють собою набори дескрипторів, які потрібно перевіряти, на готовність до читання, запису і на наявність виняткових ситуацій, відповідно. Функція select() – блокуюча, вона повертає управління, якщо б хоча один з перевіряючих сокетів готовий до виконання відповідної операції. В якості останнього параметра функції select() можна вказати інтервал часу, після якого вона завжди поверне управління. Виклик select() для перевірки наявності вхідних даних на сокеті sock може мати вигляд:

```
fd_set set;
struct timeval interval;
FD_SET(sock, &set);
tv.tv_sec = 1;
tv.tv_usec = 500000;
...
select(1, &set, NULL, NULL, &tv);
if(FD_ISSET(sock, &set)
{
```

© В.М. Мельник

```
    //Наявні дані для читання
}
```

Все, необхідне для роботи функції select() оголошується в заголовковому файлі <sys/select.h>. В приведеному коді FD_SET и FD_ISSET – це макроси для роботи з набором дескрипторів fd_set.

Висновки. Розглянуті підходи міжпроцесної взаємодії в UNIX з використанням сокетів та інтерфейсу для їх використання в файловому просторі імен. Особливістю таких сокетів є те, що з'єднання через них локального і віддаленого додатків неможливе. Все необхідне для роботи з сокетами в файловому просторі імен, об'явлено в файлах <sys/types.h> і <sys/socket.h>. В наведених уривках програм для клієнта і сервера подано параметричні представлення та застосування інтерфейсних функцій для програмної реалізації взаємодії між спорідненими процесами з використанням сокетів як каналів зв'язку через спеціальні файли та можливістю роботи з інформацією передання за допомогою файлових функцій. Проаналізовано заодно механізми використання функцій з парними сокетами, а також проаналізовано переваги і недостатки в їх використанні для організації з'єднання між клієнтом і сервером.

Для мережевих (AF_INET – сімейство Інтернету) сокетів з'ясовано, що дані передавання повинні пройти всі рівні мережевого стеку. Застосування функції htons() призводить до перетворення значення порта згідно Інтернет-порядку байтів. Функція listen() реалізує механізм переводу сервера в режим очікування запиту, а функція accept() повертає новий сокет, відкритий для обміну даними з клієнтом. Для перетворення доменного імені сервера в його мережеву адресу необхідно застосовувати функцію gethostbyname().

З'ясовано спосіб і причини переведення блокуючого сокета в неблокуючий. Функція select() може перевіряти стан декількох дескрипторів одночасно і перевіряти наявність вхідних даних на сокеті.

Література:

1. Стивенс У., UNIX: Разработка сетевых приложений. – СПб.: Питер, 2004. – 880 с.
2. W. R. Stevens, S. A. Rago, Advanced Programming in the UNIX® Environment: Second Edition, Addison Wesley Professional, 2005. – 720 с.
3. Мельник В.М., Стешенко Л.І. Основи операційної системи UNIX / Луцьк. – 2012. – 280 с.
4. Рочкинд М. Программирование для UNIX. 2-е изд. и перераб. и доп. – Пер. с англ. – СПб.; БХВ-Петербург, 2005. – 704 с.: ил.

© В.М. Мельник