Лабораторная работа N 4

КЛИЕНТ-СЕРВЕРНЫЕ ВЗАИМОДЕЙСТВИЯ ПОСРЕДСТВОМ СОКЕТОВ В РЕЖИМЕ TCP-СОЕДИНЕНИЯ

Цель работы

Практическое освоение механизма сокетов. Построении TCP-соединений для межпроцессного взаимодействия программ Клиента и Сервера в модели "Клиент-сервер".

Содержание работы
Ознакомиться с заданием к лабораторной работе.
Ознакомиться с понятиями сокета, моделей взаимодействия между процессами на основе сокетов, структурами данных и адресацией, используемых при работе с сокетами, основными шагами при организации взаимодействия процессов в сети в режиме TCP-соединения.
Выбрать и изучить набор системных вызовов, обеспечивающих решение задачи.
Для указанного варианта составить на языке Си++ две программы: программу сервера и программу клиента, реализующие требуемые действия.
Отладить и оттестировать составленную программу, используя инструментарий ОС.
Защитить лабораторную работу, ответив на контрольные вопросы.

Методические указания к лабораторной работе

Существует две модели взаимодействия между процессами в сети: модель соединений с протоколом TCP (Transmission Control Protocol), и модель дейтаграмм с протоколом UDP (User Datagram Protocol). В данной лабораторной работе используется первая из названных моделей.

Ниже приводятся основные шаги и необходимые системные вызовы для выполнения основных этапов при работе с сокетами в режиме TCP-соединения.

1. Адресация и создание сокета

Совокупная информация об адресе, порте программы-адресата (абонента), модели соединения, протоколе взаимодействия составляет т.н. сокет (конечная абонентская точка), формально представляющий собой структуру данных. Существует несколько видов сокетов:

обобщенный сокет (generic socket), определяется в файле <sys/socket.h>:

struct sockaddr {

u_char sa_family; /* Семейство адресов (домен) */

char sa_data[]; }; /* Адрес сокета */

Сокеты для связи через сеть, определяется в файле <netinet/in.h>:

struct sockaddr_in {

u_char sin_len; /* Длина поля sockaddr_in (для FreeBSD) */

u_char sin_family; /* Семейство адресов (домен) */

u_short sin_port; /* Номер порта */

struct in_addr sin_addr; /* IP-адрес */

char sin_zero[8]; }; /* Поле выравнивания */

где struct in_addr {

n_int32_t s_addr}.

Создается сокет при помощи системного вызова socket().

#include <sys/socket.h>

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

Параметр domain - домен связи, в котором будет использоваться сокет (значение AF_INET - для домена Internet (соединение через сеть), AF_UNIX - домен, если процессы находятся на одном и том же компьютере);

Параметр type определяет тип создаваемого сокета (значение SOCK_STREAM - для режима соединений, SOCK_DGRAM - для режима дейтаграмм);

Параметр protocol определяет используемый протокол (в случае protocol = 0 по умолчанию для сокета типа SOCK_STREAM будет использовать протокол TCP, а сокета типа SOCK_DGRAM - протокол UDP).

При программировании TCP-соединения должны быть созданы сокеты (системный вызов socket()) и в программе сервера и в программе клиента, при этом в обеих программах сокеты связываются с адресом машины, на которую будет установлена программа сервера. Но, если в программе сервера для определения IP-адреса в структуре сокета может быть использована переменная INADDR_ANY, то в программе клиента для занесения в структуру сокета IP-адреса машины сервера необходимо использовать системный вызов inet_addr().

Сетевые вызовы inet_addr() и inet_ntoa() выполняют преобразования IP-адреса из формата текстовой строки "x.y.z.t" в структуру типа in_addr и обратно.

#include <arpa/inet.h>

in_addr_t inet_addr (const char *ip_address);

char * inet_ntoa(const struct in_addr in);

Для того чтобы процесс мог ссылаться на адрес своего компьютера, файле <netinet/in.h> определена переменная INADDR_ANY, содержащая локальный адрес компьютера в формате in_addr_t.

2. Связывание

Системный вызов bind() связывает сетевой адрес компьютера с идентификатором сокета.

#include<sys/types.h>

#include<sys/socket.h>

int bind (int sockfd, const struct sockaddr *address, size_t add_len);
sockfd - дескриптор файла сокета, созданным с помощью вызова socket(),

address - указателем на обобщенную структуру адреса сокета, к которой преобразуется структура sockaddr_in, в случае передачи данных через сеть.

size_t add_len - размер указанной структуры адреса сокета.

В случае успешного завершения вызова bind() он возвращает значение 0. В случае ошибки, например, если сокет для этого адреса уже существует, вызов bind() возвращает значение -1. Переменная errno будет иметь при этом значение EADDRINUSE.

Операция связывания выполняется только в программе сервера.

3. Включение приема TCP-соединений

После выполнения связывания с адресом и перед тем, как какой-либо клиент сможет подключиться к созданному сокету, сервер должен включить прием соединений посредством системного вызова listen().

#include<sys/socket.h>

int listen (int sockfd, int queue_size);
sockfd - дескриптор файла сокета, созданным с помощью вызова socket(),
queue_size - число запросов на соединение с сервером, которые могут стоять в очереди.

Данная операция выполняется только в программе сервера.

4. Прием запроса на установку TCP-соединения

Когда сервер получает от клиента запрос на соединение, он создает новый сокет для работы с новым соединением. Первый же сокет используется только для установки соединения. Дополнительный сокет для работы с соединением создается при помощи вызова accept(), принимающего очередное соединение.

#include<sys/types.h>

#include<sys/socket.h>

int accept (int sockfd, struct sockaddr *address, size_t *add_len);

sockfd - дескриптор сокета, для которого ведется прием соединений;

address - указатель на обобщенную структуру адреса сокета с информацией о клиенте; так как связь использует соединение адрес клиента знать не обязательно и допустимо задавать параметр address значением NULL;

add_len - размер структуры адреса, заданной параметром address, если значение address не равно NULL.

Возвращаемое значение соответствует идентификатору нового сокета, который будет использоваться для связи. До тех пор, пока от клиента не поступил запрос на соединение, процесс, выдавший системный вызов accept() переводится в состояние ожидания.

Данная операция выполняется только в программе сервера.

5. Подключение клиента

Для выполнения запроса на подключение к серверному процессу клиент использует системный вызов connect().

#include<sys/types.h>

#include<sys/socket.h>

int connect (int csockfd, const struct sockaddr *address, size_t add_len);
csockfd - дескриптор файла сокета клиента, созданным с помощью вызова socket();
address - указателем на обобщенную структуру адреса сокета, к которой преобразуется структура sockaddr_in, в случае передачи данных через сеть;
size_t add_len - размер указанной структуры адреса сокета.

В случае успешного завершения вызова connect() он возвращает значение 0. В случае ошибки, системный вызов connect() возвращает значение -1, а переменная errno идентифицирует ошибку.

Данная операция выполняется только в программе клиента.

6. Пересылка данных

Для сокетов типа SOCK_STREAM дескрипторы сокетов, полученные сервером посредством вызова accept() и клиентом с помощью вызова socked(), могут использоваться для чтения или записи. Для этого могут использоваться обычные вызовы read() и write(), либо специальные системные вызовы send() и recv(), позволяющие задавать дополнительные параметры пересылки данных по сети. Синхронизация данных при работе с сокетом аналогична передаче данных через программный канал.

#include<sys/types.h>

#include<sys/socket.h>

ssize_t recv (int sockfd, void *buffer, size_t length, int flags);

ssize_t send (int sockfd, const void *buffer, size_t length, int flags);
socfd - дескриптор сокета, через который читаются или записываются данные;
buffer - буфер, в который они помещаются или откуда отсылаются через сокет;
length - размер буфера;
flags - поле дополнительных опций при получении или передаче данных.

В случае успешного чтения/записи системные вызовы send() и recv() возвращают число прочитанных/отосланных байт, или -1 в случае ошибки; в случае разорванной связи (клиент разорвал TCP-соединение) вызов recv() (или read()) возвращают нулевое значение; если процесс пытается записать данные через разорванное TCP-соединение посредством write() или send(), то он получает сигнал SIGPIPE, который можно обработать, если предусмотрена обработка данного сигнала.

В случае flags = 0 вызовы send() и recv() полностью аналогичны системным вызовам read() и write().

Возможные комбинациями констант параметра flags системного вызова send():
MSG_PEEK Процесс может просматривать данные, не "получая" их;
MSG_OOB Обычные данные пропускаются. Процесс принимает только срочные данные, например, сигнал прерывания;
MSG_WAITALL Возврат из вызова recv произойдет только после получения всех данных.

Возможные комбинациями констант параметра flags системного вызова recv():
MSG_OOB Передать срочные (out of band) данные;

MSG_DONTROUTE При передаче сообщения игнорируются условия маршрутизации протокола более низкого уровня. Обычно это означает, что сообщение посылается по прямому, а не по самому быстрому маршруту (самый быстрый маршрут не обязательно прямой и может зависеть от текущего распределения нагрузки сети).

Данные операции выполняются и в программе сервера и в программе клиента.

7. Закрытие TCP-соединения

Закрываются сокеты так же, как и обычные дескрипторы файлового ввода/вывода, - при помощи системного вызова close(). Для сокета SOCK_STREAM ядро гарантирует, что все записанные в сокет данные будут переданы принимающему процессу.

Данные операции выполняются и в программе сервера и в программе клиента.

Варианты заданий

1. Организовать взаимодействие типа клиент - сервер. Клиент делает запрос серверу на выполнение какой-либо команды. Сервер выполняет эту команду и возвращает результаты клиенту.

2. Организовать взаимодействие типа клиент - сервер. Клиент делает запрос серверу о передаче файлов с определенным расширением из указанной директории. Сервер сканирует указанную директорию и отправляет клиенту список файлов, удовлетворяющих запросу.

3. Организовать взаимодействие типа клиент - сервер. Сервер при подключении к нему нового клиента высылает список IP-адресов уже подключенных клиентов. А остальным клиентам рассылается сообщение в виде IP-адреса о том, что подключился такой-то клиент.

4. Организовать взаимодействие типа клиент - сервер. Клиент при входе в связь с сервером должен ввести пароль. Разрешено сделать три попытки. Если пароль не верен, сервер должен блокировать IP-адрес клиента на 5 минут.

5. Организовать взаимодействие типа клиент - сервер. Клиенты подключается к первому серверу, и передают запрос на получение определенного файла. Если этого файла нет, сервер подключается ко второму серверу и ищет файл там. Затем либо найденный файл пересылается клиенту, либо высылается сообщение, то такого файла нет.

6. Организовать взаимодействие типа клиент - сервер. К серверу одновременно может подключиться только один клиент. Остальные клиенты заносятся в очередь, и им высылается сообщение об ожидании освобождения сервера.

7. Организовать взаимодействие типа клиент - сервер. Клиент отсылает строку серверу. Сервер отсылает данную строку на другие сервера, список которых хранится в файле, а там уже осуществляется поиск файлов содержащих данную строку. Результаты поиска отсылаются клиенту.

8. Эмуляция DNS сервера. Клиент подсоединяется к серверу, IP которого хранится в файле dns.url и делает ему запрос на подключение к серверу "Имя сервера". DNS-сервер имеет список, хранящийся в файле о соответствии имен серверов и IP-адресов. Если в списке нет "имени сервера" запрошенного клиентом, то сервер DNS подключается последовательно к другим серверам, хранящимся в файле dns.url и т.д. Если сервер не найден, клиенту возвращается соответствующее сообщение.

9. Организовать чат. К серверу подключаются клиенты. При подключении клиента сервер спрашивает имя, под которым клиент будет известен в соединении. Сервер хранит IP-адреса подключаемых клиентов и их имена. Все сообщения каждого клиента рассылаются

остальным в виде ""имя клиента" - сообщение". Сообщения рассылаются сервером всем клиентам также при вхождении в связь нового клиента, и выходе какого-либо клиента

10. Организовать взаимодействие типа клиент - сервер. Клиент передает имя файла, а сервер поблочно копирует файл клиенту. Файл, имя которого передается, располагается на сервере.

11. Организовать взаимодействие типа клиент - сервер. Клиент передает серверу строку, а сервер возвращает строку, зашифрованную любым криптоалгоритмом (можно даже просто XOR). По возможности, реализовать потоковый режим шифрования.

12. Организовать взаимодействие типа клиент - сервер. Сетевая игра. В роли клиента выступает любая простейшая игра (морской бой, шашки, шахматы и т.д.).

13. Организовать взаимодействие типа клиент - сервер. Серверу передаются файлы, которые раскладываются по папкам в зависимости от типа файла. Сопоставление типа файла и папки, в которую он должен быть положен, хранится в файле на сервере.

14. Необходимо разработать клиент/серверное приложение, в котором сервер каждые 10 секунд распространяет некоторое текстовое сообщение, например, о погоде, всем *промежуточным* клиентам, зарегистрированным в группе 233.0.0.1, порт 1502 с помощью UDP. Текст сообщения хранится в текстовом файле на сервере. Промежуточный клиент фильтрует полученные сообщения и в случае изменения содержимого отображает его в консоли. Для *конечного* клиента промежуточный клиент выступает сервером. Конечный клиент присоединяется к промежуточному и получает тексты последних пяти отфильтрованных сообщений (с помощью протокола TCP/IP). Необходимо снабдить приложение конечного клиента графическим интерфейсом.

15. Две стороны обмена играют в "угадай число". Для установления соединения одной из сторон достаточно знать адрес и порт другой. Изначально каждая из них загадывает натуральное число, и на запросы стороны отвечает "меньше", "больше", "равно", сравнивая число в запросе с текущим значением своего числа. Получив ответ, каждая сторона изменяет свое число на единицу. Обе стороны задают запросы параллельно, повторяя запросы с некоторым интервалом в случае неполучения ответа. Протокол должен обеспечивать совпадение текущих значение чисел за конечное число шагов.

16. Организовать взаимодействие типа клиент - сервер. Клиент передает имя файла, а сервер копирует файл клиенту в несколько потоков. Файл, имя которого передается, может располагаться где угодно (сервер или папка с открытым доступом).

17. Организовать взаимодействие типа клиент - сервер. Генерация пароля на удаленной машине. Параметр-адрес машины исполнителя, функция генерации (любая функция). Возвращаемое значение пароль.

18. Организовать взаимодействие типа клиент - сервер. Передача на сервер массива чисел, сортировка массива (Шелла) и возвращение его обратно клиенту. Возвращаемое значение массив чисел и количество итераций.

19. Организовать взаимодействие типа клиент — сервер. Эмуляция работы почтового сервера. Сопоставление адресов получателя и IP — на сервере. Сообщения должны доставляться получателю, если он не в сети, после того как он снова появится в онлайне.

20. Организовать взаимодействие типа клиент - сервер. Клиент передает имя файла, а сервер побитово копирует файл клиенту. Файл, имя которого передается, располагается на сервере.