Курс «Информационные сети и телекоммуникации»

Лабораторная работа № 4 «Многоадресная рассылка (multicast)»

Задание на лабораторную работу

Обязательная программа

Разработку предлагается вести на основе программы, написанной на ЛР № 1.

1. Сделать выполнение приема и отправки данных параллельными

Необходимо реализовать программу, ведующую прием сообщений непрерывно, и при этом позволяющую пользователю ввести и отправить сообщение в любой момент. Программа:

1.1. Создает дейтаграммный сокет для работы в сетях IPv4 по протоколу UDP.
1.2. Запрашивает у пользователя адрес и порт для указания как собственных и выполняет привязку сокета.
1.3. Запрашивает у пользователя адрес и порт для отправки на них сообщений и сохраняет эти данные в переменной-структуре `sockaddr_in`.
1.4. Создает поток для приема сообщений, который в бесконечном цикле принимает сообщение и отображает его на экране по прибытии вместе с адресом и портом отправителя.
1.5. В бесконечном цикле (в основном потоке):
  1.5.1. Запрашивает у пользователя текст сообщения.
  1.5.2. Если введенный текст — `/quit`, прерывает цикл.
  1.5.3. Иначе отправляет сообщение на заданный в п. 1.3 адрес и порт.
1.6. Закрывает ранее созданный сокет.

2. Изменить программу, задействовав вместо одноадресной рассылки (unicast) многоадресную (multicast) с использованием фиксированной группы multicast.

2.1. Необходимые заголовочные файлы: `<winsock2.h>` вместо `<winsock.h>`, `<ws2tcpip.h>` для многоадресной рассылки.
2.2. В пункте 1.3 требуется отныне и впредь вводить адреса класса D.
2.3. До начала передачи данных (перед пунктом 1.4) выполнять настройку сокета функцией `setsockopt()`:

2.3.1. Отключить доставку пакетов multicast обратно источнику, если он находится в той же группе multicast, куда отправлен пакет (уровень `IPPROTO_IP`, параметр `IP_MULTICAST_LOOP`).

2.3.2. Указать отправлять пакеты multicast через сетевой интерфейс, которому принадлежит адрес, указанный в пункте 1.2 (уровень `IPPROTO_IP`, параметр `IP_MULTICAST_IF`).

2.3.3. Присоединиться к группе multicast, адрес которой указан в п. 1.3 (уровень `IPPROTO_IP`, параметр `IP_ADD_MEMBERSHIP`).

2.4. Вместе с пунктом 1.6 необходимо покидать группу многоадесной рассылки (параметр `IP_DROP_MEMBERSHIP` уровня `IPPROTO_IP`).

2.5. Проверить правильность работы многоадресной рассылки.

2.5.1. **В случае, если доступны две машины,** программам на них следует присоединиться к одной группе multicast (с указанием одинакового номера порта) и отправить по сообщению. Каждая программа должна получать сообщения, отправленные другой, и не получать своих. **Внимание:** машины должны быть в одной сети!

2.5.2. **В случае, если доступна только одна машина,** тестирование можно осуществить так:

- оба экземпляра программы должны присоединяться к общей группе multicast (например, 226.0.0.1);
- первый экземпляр должен получать сообщения на порт *N*, а отправлять на порт *(N+1)*, второй — наоборот.

Нескольким экземплярам программы следует присоединиться к одной группе multicast (с указанием одинакового номера порта) и отправить по сообщению. Каждая программа должна получать все сообщения, включая собственные.

3. Обеспечить поддержку псевдонимов пользователей

Необходимо добавить возможность пользователям устанавливать псевдонимы, которыми будут подписыватсья их сообщения вместо адресов и портов, специальным сообщением-командой.

3.1. Перед пунктом 1.4 требуется запросить у пользователя желаемый псевдоним `name` и отправить сообщение вида **`/nick`** `name`, а затем некоторое время (например, 1 с) дожидаться возможного сообщения о том, что данный псевдоним уже используется.

3.2. Если за время ожидания было получено сообщение **`/taken`** `name`, где `name` — тот же псевдоним, который хотел занять пользователь, следует напечатать сообщение об этом и вернуться к пункту 3.1. В противном случае (сообщение не получено) следует отправить сообщение **`/taken`** `name` и начать обмен сообщениями (пункт 1.4 и далее).

3.3. Всем программам в любой момент, кроме времени ожидания в пункте 3.1:

3.3.1. При получении сообщения **/nick** `name`, если `name` совпадает с псевдонимом, который удалось занять в пункте 3.2, необходимо отправить сообщение **/taken** `name` источнику сообщения **/nick** `name`.

3.3.2. При получении сообщения **/taken** `name` необходимо сохранить информацию, что псевдоним `name` соответствует аресу и порту отправителя пакета.

3.4. При печати сообщений вместо адреса и порта отправителя требуется печатать псевдоним отправителя, если он известен из пункта 3.3.2.

Произвольная программа

4. Добавить возможность читать сообщения нескольких групп multicast

Сокет может одновременно входить в несколько групп многоадресной рассылки. Процедуры отправки и приема сообщений никак не изменятся, однако, получены будут сообщения, адресованные любой из групп, в которые входит сокет. Отправка данных по-прежнему выполняется на конкретный адрес и порт (в одну группу, куда клиент входил изначально).

Необходимо дать пользователю возможность входить в группу multicast командой **/join** *address*, например, **/join** *226.0.0.10*, и покидать группу командой **/leave** *address*, например, **/leave** *226.0.0.10*.

5. Добавить возможность отправки личных сообщений

Требуется добавить в режиме групповой переписки возможность адресной отправки сообщений. При вводе пользователем сообщения специального вида **/to** *host port* `private message text`, например,

/to *10.100.0.42 1234* `Грузите апельсины бочками.`,

сообщение требуется отправлять не в группу multicast, а по указанному адресу и на заданный порт. Вместо адреса и порта следует также предусмотреть указание и псевдонима: **/to** *корейко* `Грузите апельсины бочками`.

6. Добавить команду для определения участников переписки

При получении сообщения **/who** следует не отображать его, как остальные, но автоматически ответить отправителю сообщением **/taken** *name*, где *name* — псевдоним пользователя, который получил команду **/who**.

Указания к выполнению лабораторной работы

Создание потоков

Поток (иногда — *нить*, англ. *thread*) — это участок программы, могущий выполняться одновременно с другими. В любой программе есть, по крайней мере, один поток — основной, в котором выполняется функция `main()`.

Простейший способ создать поток на C++ в Windows — воспользоваться функцией _beginthread() из заголовочного файла `<process.h>`. Достаточно передать ей первый аргумент — указатель на функцию, содержащую код, который необходимо начать исполнять параллельно:

```
#include <process.h>

void runs_in_parallel(void* argument) { … }

int main(int argc, char* argv[]) {
	_beginthread(runs_in_parallel, 0, 0);
	// Выполнение main() не должно заканчиваться —
	// в данной ЛР здесь начинается бесконечный цикл.
	return 0;
}
```

В курсе «Системное ПО» разработка многопоточных программ изучалась подробнее. Предложенное выше решение имеет предельно просто и достаточно для целей работы.

Ожидание ответа определенное время

Ожидание входящего пакета определенное время можно выполнить, настроив сокет функцией `setsockopt()` с параметром `SO_RCVTIMEO` уровня `SOL_SOCKET`. Следует установить желаемое время ожидания (например, 1 с = 1000 мс) и выполнить прием.

Согласно MSDN, если отпущенное время истекло, а пакетов не принято, сокет больше использовать нельзя, а для передачи полезных данных требуется создать новый. **Однако, на практике** можно продолжать пользоваться сокетом после истечения времени ожидания `recvfrom()`. При этом нужно или отключить опцию `SO_RCVTIMEO`, установив её в `0`, или не обрабатывать ошибку с кодом (`WSA`)`EWOULDBLOCK` (10060).

Хранение и поиск в списке псевдонимов и адресов

Предлагается воспользоваться решением на основе ассоциативного массива — структуры данных, подобной массиву, но допускающую использование в качестве индексов (они называются ключами) не только чисел, но и других типов.

Программа ниже иллюстрирует использование стандартного ассоциативного массива.

Шаблон класса `std::map` расположен в заголовочном файле `<map>`:

```
#include <map>
```

В качестве индексов `std::map` допускается использовать только типы, значения которых возможно сравнить оператором «меньше» (`<`). Необходимо «объяснить» компилятору, как сравнивать структуры `sockaddr_in`:

```
bool operator<(const sockaddr_in& x, const sockaddr_in& y) {
    return x.sin_addr.s_addr < y.sin_addr.s_addr;
}

int main(int argc, char* argv[]) {
```

Для хранения связей { адрес и порт, псевдоним } удобно использовать ассоциативный массив, где ключом (индексом) выступал бы адрес и порт (`sockaddr_in`), а значением была бы строка-псевдоним (`std::string`):

```
    std::map<sockaddr_in, std::string> nicknames;
```

При получении пакета можно узнать адрес отправителя в переменной `sender`:

```
    char message[512]
    sockaddr_in sender;
    int size = sizeof(sender);
    recvfrom(s, message, sizeof(message), 0, &sender, &size);
```

Прежде всего, необходимо проверить, нет ли в `nicknames` элемента с ключом `sender`:

```
    if (nicknames.find(sender) != nicknames.end()) {
```

Коль скоро элемент имеется, его значение (псевдоним) можно получить по ключу:

```
        std::cout << nicknames[sender] << ": "
                  << message << std::endl;
    } else {
```

В случае же, когда искомый ключ отсутствует, следует сохранить псевдоним по данному ключу (на самом деле, нужно еще проверять, что `message` начинается с **`/nick`**, а строку-псевдоним извлекать из `message`).

```
        nicknames[sender] = "user";
    }
```

При необходимости можно и удалить элемент по ключу:

```
    nicknames.erase(sender);
}
```

Внимание: демонстрационная программа *не* компилируется и *не* работает (не хватает подключения библиотек, запуска сетевой подсистемы, создания и привязки сокета).