

**Практическая работа №2 по курсу**  
**«Технология разработки программного обеспечения»**  
**Тема работы: «Разработка клиент-серверного приложения»**

1. Цель и задачи работы.

1.1. Цель работы: Разработка клиент-серверного приложения, вычисляющего длину минимального пути в неориентированном графе.

1.2. Задачи работы:

- получение знаний об этапах разработки программного обеспечения;
- получение базовых знаний по разработке приложений в ОС Linux;
- получение знаний и практических навыков реализации межпроцессных взаимодействий с помощью механизма сокетов;
- получение знаний по работе с графами;
- разработка клиент-серверного приложения.

2. Требования к работе

2.1. Разработать приложение типа «клиент-сервер».

2.1.1. Функции клиентской части приложения:

- устанавливать соединение с сервером;
- получать от пользователя входную информацию, содержащую описание графа в соответствии с индивидуальным заданием.

Способы ввода информации:

а) с клавиатуры;

б) чтение из файла;

- проверять корректность полученной информации;
- передавать информацию серверу;
- получать ответы от сервера и выводить на консоль.

2.1.2. Функции серверной части приложения:

- ожидать запросов от клиентов. Минимальное количество клиентов, с которыми сервер может одновременно работать: 3;
- принимать от клиента информацию, содержащую описание графа;
- проверять корректность входной информации;
- вычислять длину минимального пути в графе между заданными вершинами;
- возвращать клиенту ответ, содержащий длину и список вершин минимального пути.

2.2. Требования к описанию графа:

2.2.1. Количество вершин: не менее 6

2.2.2. Количество ребер: не менее 6

2.2.3. Тип графа: неориентированный,

– для 2-го курса: с единичными весами ребер

– для 3-го курса: с произвольными неотрицательными весами ребер

2.2.4. Способ задания графа – в соответствии с индивидуальным заданием

2.2.5. Алгоритм поиска на графе:

– для 2-го курса: алгоритм Дейкстры

– для 3-го курса: произвольный

2.3. Требования к объему данных, передаваемых между клиентской и серверной частями приложения: обеспечить оптимальный объем передаваемых данных.

2.4. Исходный код программы должен быть снабжен комментариями, поясняющими значение системных вызовов, функций, макросов, переменных, блоков и строк.

2.5. Приложение должно быть способно функционировать, как при запуске клиентской и серверной частей приложения на одном компьютере, так и на разных сетевых узлах. В качестве параметра клиентская часть приложения должна принимать IP-адрес и порт серверной части. При запуске приложения на одном компьютере клиентская часть должна использовать в качестве IP-адреса сервера адрес 127.0.0.1 (адрес loopback-интерфейса операционной системы).

2.6. Завершение работы клиентской части – по ключевому стоп-слову, например, exit. Окончание работы клиентской части приложения не должно приводить к остановке серверной части.

2.7. Завершение работы серверной части: по сигналу от пользователя.

2.8. Протокол взаимодействия клиентской и серверной частей приложения: TCP и UDP (должны поддерживаться оба протокола). Используемый протокол задается параметром при запуске приложения. Порт на стороне сервера произвольный.

2.9. При реализации взаимодействия по протоколу UDP реализовать механизм, обеспечивающий надежную доставку полезной нагрузки. Данный механизм должен обеспечивать следующую логику:

2.9.1. подтверждать прием сообщения с полезной нагрузкой путем отправки специального служебного сообщения (при этом подтверждать получение служебного сообщения не надо);

- 2.9.2. контролировать доставку отправленного сообщения с полезной нагрузкой путем ожидания и приема подтверждающего служебного сообщения;
- 2.9.3. если сообщение с полезной нагрузкой отправлено, а подтверждение не приходит в течение 3 с – обеспечить повторную отправку сообщения с полезной нагрузкой;
- 2.9.4. если сообщение с полезной нагрузкой было отправлено 3 раза, а подтверждение не пришло, то:
  - 2.9.4.1. для клиентской части выводить сообщение «Потеряна связь с сервером» и завершить работу;
  - 2.9.4.2. для серверной части выводить сообщение «Потеряна связь с клиентом» и продолжать работу.
- 2.10. Язык написания программы: C/C++.
- 3. Ход выполнения работы:
  - 3.1. Провести анализ требований к разрабатываемой программе. При возникновении вопросов, уточнений – обратиться к заказчику (преподавателю).
  - 3.2. Разработать спецификацию к программе (раздел 1 отчета), содержащую:
    - 3.2.1. общее описание задачи (назначение программы, архитектура, ОС, язык реализации);
    - 3.2.2. описание интерфейса;
    - 3.2.3. функциональные требования;
    - 3.2.4. требования к тестированию;
  - 3.3. Разработать проект приложения. В рамках проектирования должны быть разработаны и приведены в отчете (раздел 2):
    - 3.3.1. Перечень модулей, входящих в клиентскую и серверную части, а также функциональные возможности, реализуемые перечисленными модулями.
    - 3.3.2. Алгоритмы работы клиентской и серверной частей приложения при функционировании по протоколу UDP. Схемы алгоритма должны быть выполнены согласно ГОСТ 19.701-90.
    - 3.3.3. Форматы структур данных, передаваемых между клиентской и серверной частями приложения.
  - 3.4. Реализовать приложение на одном из заданных языков программирования и отладить приложение.
  - 3.5. Провести тестирование приложения.
  - 3.6. Провести статический анализ исходного кода приложения.

### 3.7. Написать отчет по работе. Оформление отчета – согласно Правилам оформления студенческих работ СПбПУ.

#### 4. Рекомендации по выполнению работы

- 4.1. В рамках изучения межпроцессных взаимодействий с помощью механизма сокетов: изучить системные вызовы `socket()`, `bind()`, `listen()`, `accept()`, `connect()`, `recv()`, `send()`, `shutdown()`.
- 4.2. Для выполнения требования работы не менее, чем с тремя клиентами одновременно, изучить следующие подходы к мультиплексированию ввода/вывода:
- использование системного вызова `select()` и работа со всеми клиентами в рамках одного процесса;
  - использование системного вызова `fork()` и работа с каждым клиентом в рамках отдельного процесса;
  - использование библиотеки `pthread()` и работа с каждым клиентом в рамках отдельного потока (рекомендовано к реализации для 3-го курса).
- 4.3. Определить форматы запроса к серверу и ответа сервера с использованием структур, содержащих числовые типы данных: `char`, `unsigned char`, `short`, `int` и т.д. (на усмотрение разработчика с учетом требования к оптимальности объема передаваемых через сеть данных).  
Возможные варианты:
- используются символьные типы данных: `char[]`
  - используются числовые типы данных: `short`, `int`, `long`, `double` ...
- 4.4. Для сборки программы использовать компилятор `gcc` или аналогичный.  
Примеры использования компилятор `gcc`:
- ```
user@ubuntu:~$ gcc calc_client.c -o calc_client
user@ubuntu:~$ gcc calc_server.c -o calc_server
```
- Указанные команды запускают на компиляцию программу, записанную в соответствующем файле (например, `calc_client.c`) и, в случае отсутствия ошибок, сохраняют результат в указанном после ключа `-o` исполняемом файле (например, `-o calc_client`).
- 4.5. Для запуска серверной части приложения в фоновом режиме использовать символ «&»:
- ```
user@ubuntu:~$ ./calc_server &
```
- 4.6. Для запуска клиентской части приложения в интерактивном режиме использовать следующий формат:
- ```
user@ubuntu:~$ ./calc_client <IP-адрес> <protocol>
<номер_порта>
```

- 4.7. Для остановки серверной части приложения использовать команду отправки сигналов процессу `kill <pid >`, где `pid` – process id, идентификатор процесса сервера, например:
- ```
user@ubuntu:~$ kill 8453
```

Идентификатор процесса серверной части приложения можно узнать, используя команду `ps` или `ps -ax | grep <имя_сервера>`:

```
user@ubuntu:~$ ps
PID   TTY          TIME CMD
3650  pts/0        00:00:00  bash
3651  pts/0        00:00:00  ./calc_server
3653  pts/0        00:00:00  ps
user@ubuntu:~$
user@ubuntu:~$ ps -ax | grep calc_server
3651  pts/0      S+          0:00  ./calc_server
```

- 4.8. Чтобы убедиться, что серверная часть приложения успешно запустилась и ожидает соединения на заданном порту, использовать команду `netstat` с ключами `-lptn`, которая выводит состояние открытых сокетов системы (возможно, эта команда по умолчанию не установлена, необходимо установить ее командой `sudo apt install net-tools`):
- ```
user@ubuntu:~$ netstat -lptn
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it
all.)
```

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address  State       PID/Program name
tcp    0      0 127.0.0.53:53      0.0.0.0:*        LISTEN      -
tcp    0      0 0.0.0.0:22         0.0.0.0:*        LISTEN      -
tcp    0      0 0.0.0.0:8080       0.0.0.0:*        LISTEN      ./calc_server
tcp6   0      0 :::22              :::*             LISTEN      -
```

## 5. Справочный материал

- 5.1. На рис. 1 приведен вариант схемы работы приложения, включающий последовательность системных вызовов и порядок обмена пакетами между клиентской и серверной частями при использовании протокола TCP.

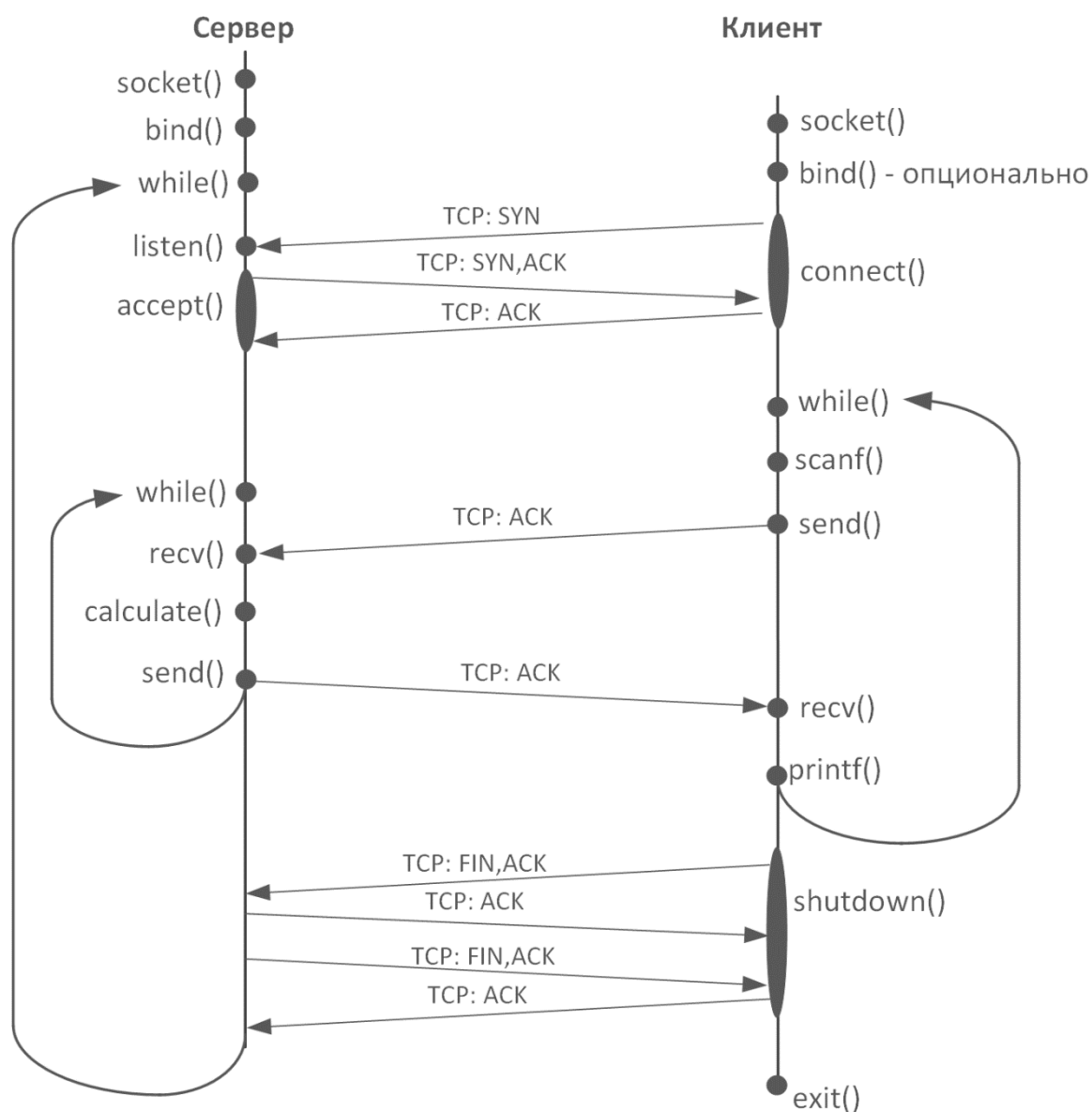


Рис.1 Схема взаимодействия клиентской и серверной частей приложения при использовании TCP

5.2. Системный вызов `socket()` определяет, какой тип сокета необходимо открыть. По условиям задачи необходим сокет, работающий с протоколом IP версии 4. Исходя из этого, использование системного вызова следующее:

- прототип: `int socket(int domain, int type, int protocol);`
- `int domain`: определяет семейство протоколов, которые будут использоваться. Возможные значения: `PF_UNIX`, `PF_INET`, `PF_INET6`, `PF_IPX` и т.д. В нашем случае должно использоваться значение `PF_INET` (`PF=Protocol Family`, `INET=Internet Protocol (IP) версии 4`);
- `int type`: тип сокета. Возможные значения: `SOCK_STREAM`,

SOCK\_DGRAM, SOCK\_RAW и т.д. С помощью этого параметра задается используемый протокол: TCP или UDP;

- `int protocol`: задает конкретный протокол, используемый сокетом. В качестве значения можно использовать 0, в этом случае автоматически будет использован TCP для SOCK\_STREAM и UDP для SOCK\_DGRAM.

5.3. Для отладки взаимодействия клиентской и серверной частями полезно использовать утилиту `tcpdump`, позволяющую наблюдать за обменом пакетами на сетевых интерфейсах. Для этого:

- определить имя loopback-интерфейса системы, отобразив список сетевых интерфейсов, например, командой `ip link show`:

```
user@ubuntu:~$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode
DEFAULT group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode
DEFAULT group default qlen 1000
    link/ether 00:15:5d:f8:b0:27 brd ff:ff:ff:ff:ff:ff
```

В приведенном выводе два интерфейса: `lo` – loopback-интерфейс, `eth0` – физический Ethernet-интерфейс;

- установить утилиту `tcpdump`, если она не установлена (`sudo apt install tcpdump`);
- изучить описание утилиты (`man tcpdump`);
- запустить утилиту, указав в качестве параметра имя loopback-интерфейса `lo`:

```
sudo tcpdump -i lo -n -A
```

выход из утилиты `tcpdump` – комбинация клавиш CTRL+C;

- изучить вывод утилиты `tcpdump`, а также параметры, используемые при запуске (`-i`, `-n`, `-A`).

5.4. Общая схема взаимодействия по UDP приведена на рис. 2;

5.5. Для реализации логики взаимодействия по протоколу UDP можно использовать перевод сокета в неблокирующий режим, при котором операция чтения из сокета завершается ошибкой в случае, если приемный буфер сокета пуст (в блокирующем режиме, который используется по умолчанию, операция чтения из сокета не завершится, пока в приемном буфере не появятся данные для чтения). Для перевода сокета в неблокирующий режим изучить и использовать системный вызов `fcntl()`.

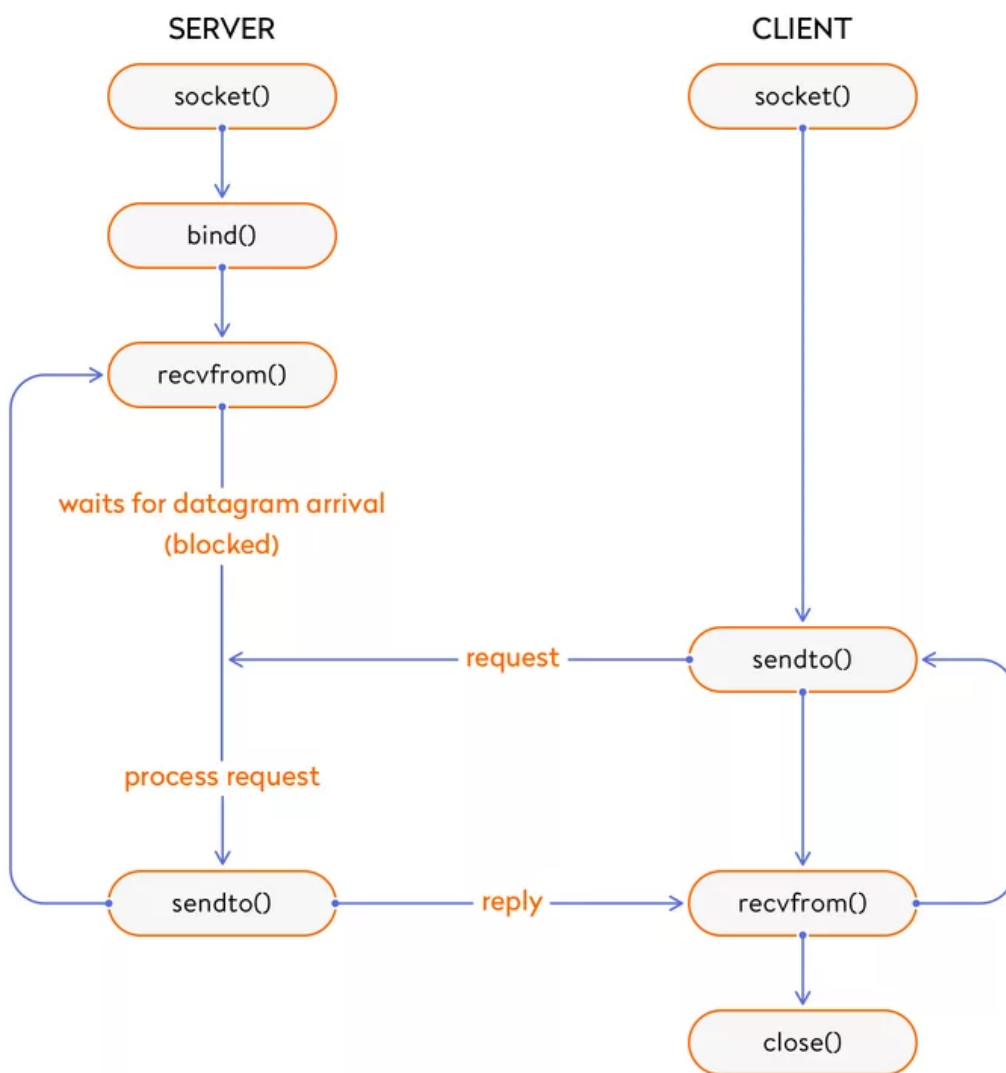


Рис.2 Общая схема взаимодействия клиентской и серверной частей приложения при использовании UDP (случай с блокирующими сокетами)

6. \*Проверить возможность взаимодействия клиентской и серверной частей программы через локальную сеть. Для этого необходимо два компьютера, подключенные к одной локальной сети. Чтобы настроить возможность взаимодействия двух компьютеров в локальной сети, необходимо (для случая, когда на обоих компьютерах используется VirtualBox, на котором запущена виртуальная машина Linux):
  - для каждого компьютера, участвующего во взаимодействии, изменить тип подключения виртуальной машины на «Сетевой мост» следующим образом: в окне Oracle VM VirtualBox запустить выбрать меню Устройства->Сеть-> Настроить сеть; на вкладке «Адаптер 1» выбрать в выпадающем меню «Тип подключения» значение «Сетевой мост»). Перезагрузить виртуальную машину. После этого ваша виртуальная машина с ОС Linux будет доступна из локальной сети по IP-адресу, полученному в автоматическом режиме;

- определить IP-адрес сетевого адаптера виртуальной машины, например, с помощью команды `ip address show`:

```
avs@vm1:~$ ip address show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:66:a1:5d brd ff:ff:ff:ff:ff:ff
    inet 10.41.0.193/27 brd 10.41.0.223 scope global dynamic noprefixroute enp0s3
        valid_lft 12449sec preferred_lft 12449sec
    inet6 fe80::a00:27ff:fe66:a15d/64 scope link
        valid_lft forever preferred_lft forever
```

В данном выводе на сетевом интерфейсе `enp0s3` (раздел 2 вывода команды `ip address show`) настроен IP-адрес 10.41.0.193 (`inet 10.41.0.193/27`).

- работая в парах, перенести клиентскую часть вашей программы с на компьютер партнера;
- подключить оба компьютера к одной локальной сети;
- запустить клиентскую часть программы на компьютере партнера, указав в качестве аргумента IP-адрес виртуальной машины, на котором запущена серверная часть вашей программы. Убедиться, что программа работает в соответствии с требованиями.