Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра ЭВМ

Дисциплина: Операционные системы и системное программирование

ОТЧЁТ
к лабораторной работе №8
на тему
Сокеты. Взаимодействие процессов.

Выполнил студент гр.230501 Лазовский И.А.

Проверил старший преподаватель кафедры ЭВМ
Поденок Л.П.

Минск 2024

1 УСЛОВИЕ ЛАБОРАТОРНОЙ РАБОТЫ.

Задача – разработка многопоточного сервера и клиента, работающих по простому протоколу.

Изучаемые системные вызовы: `socket(), bind(), listen(), conect(), accept()` и прочих, связанных с адресацией в домене `AF_INET`.

Протокол должен содержать следующие запросы:

`ECHO` – эхо-запрос, возвращающий без изменений полученное от клиента;

`QUIT` – запрос на завершение сеанса;

`INFO` – запрос на получения общей информации о сервере;

`CD` – изменить текущий каталог на сервере;

`LIST` – вернуть список файловых объектов из текущего каталога.

Протокол может содержать дополнительные запросы по выбору студента, не выходящие за пределы корневого каталога сервера и не изменяющих файловую систему в его дереве.

Запросы клиенту отправляются на `stdin`.

Ответы сервера и ошибки протокола выводятся на `stdout`.

Ошибки системы выводятся на `stderr`.

Подсказка клиента для ввода запросов – символ '>'.

Сервер принимает номер порта, на котором будет слушать и выводит протокол работы в `stdout`.

```
$ myserver port_no
Готов.
```

Формат протокола произвольный, каждое событие занимает одну строку, первое поле – дата и время в формате `YYYY.MM.DD-hh:mm:ss.sss`).

Клиент помимо интерактивных запросов принимает запросы из файла. Файл с запросами указывается с использованием префикса '@':

```
$ myclient server.domen
Вас приветсвует учебный сервер 'myserver'
> @file
> ECHO какой-то_текст
какой-то_текст
> LIST
dir1
dir2
file
> CD dir1
dir1> QUIT
BYE
$
```

`ECHO` – эхо-запрос, возвращающий без изменений полученное от клиента.

```
> ECHO "произвольный текст"
произвольный текст
>
```

QUIT – запрос на завершение сеанса

```
> QUIT
BYE
$
```

INFO – запрос на получения общей информации о сервере.

Сервер отправляет текстовый файл с соответствующей информацией.

```
> INFO
Вас приветсвует учебный сервер 'myserver'
>
```

Этот же файл сервер отправляет клиенту при установлении сеанса.

LIST – вернуть список файловых объектов из текущего каталога.

Текущий каталог – каталог в дереве каталогов сервера. Корневой каталог сервера устанавливается из командной строки при старте сервера.

```
> LIST
dir1/
dir2/
file1
file2 --> dir2/file2
file3 -->> dir1/file
>
```

Каталоги выводятся с суффиксом '/' после имени, файлы – как есть, симлинки на регулярные файлы разрешаются через '-->', симлинки на симлинки разрешаются через '-->>'. Корневой каталог сервера пр выводе указывается префиксом '/' перед именем.

CD – изменить текущий каталог на сервере

Выход за пределы дерева корневого каталога сервера запрещается, команда безмолвно игнорируется

```
> CD dir2
dir2> LIST
file2
dir2> CD ../dir1
dir1> LIST
file --> /file1
dir1> CD ..
> CD ..
>
```

Соединения функционируют независимо, т.е. текущий каталог у каждого соединения свой.

Примечания:

Раскрашивать вывод не нужно.

Для разработки и отладки лабораторной следует использовать редактор или IDE, поддерживающие несколько запущенных экземпляров, каж-

дый со своей конфигурацией, и поддерживающие отладку прямо в окне с кодом, например, slickedit (лучший выбор).

2 ОПИСАНИЕ АЛГОРИТМОВ И РЕШЕНИЙ.

2.1. Файл client_main.cpp

В начале программы включается заголовочный файл “Client.h”, который, вероятно, содержит объявление класса Client. Этот класс реализует функциональность клиента в сетевом взаимодействии.

Затем объявляется глобальная переменная client типа Client*. Она будет использоваться для управления клиентом во всей программе.

Функция signalHandler(int signum) является обработчиком сигналов. Она вызывается, когда программа получает сигнал (в данном случае - SIGINT, который обычно посылается при нажатии Ctrl+C). В этой функции вызывается деструктор для client и затем программа завершается с кодом, равным полученному сигналу.

Главная функция main(int argc, char *argv[]) начинается с проверки количества аргументов командной строки. Если их не три (имя программы, IP-адрес сервера и порт), то выводится сообщение об ошибке и программа завершается с кодом 1.

Если аргументы переданы корректно, то они используются для создания нового объекта Client, который сохраняется в глобальной переменной client.

Затем устанавливается обработчик сигнала SIGINT на signalHandler.

Наконец, вызывается метод start() для client, который, вероятно, начинает процесс взаимодействия с сервером.

После завершения работы метода start() программа завершается с кодом 0, что обычно означает успешное выполнение.

2.2. Класс Client

clientDescriptor: Это целочисленное значение, которое используется как дескриптор сокета для клиента.

serverAddress: Это структура sockaddr_in, которая содержит информацию об адресе сервера, включая IP-адрес и порт.

Client(const string &serverIP, int serverPort): Это конструктор класса, который принимает IP-адрес сервера и порт в качестве аргумен-

тов. Он инициализирует `serverAddress` и устанавливает соединение с сервером.

`start()`: Этот метод начинает основной цикл взаимодействия с сервером.

`getCommandFromUser()`: Этот метод считывает команду от пользователя.

`sendCommandToServer(const string &command)`: Этот метод отправляет команду на сервер.

`getResponseFromServer()`: Этот метод получает ответ от сервера.

`~Client()`: Это деструктор класса, который освобождает ресурсы перед выходом из программы.

2.3. Файл server_main.cpp

В начале программы проверяется количество аргументов командной строки. Если аргументы не переданы, сервер запускается на порту 8080. Если передан один аргумент, он интерпретируется как номер порта, на котором должен запуститься сервер.

Затем создается объект `Server`, который, вероятно, открывает сокет и начинает слушать на указанном порту.

Далее устанавливаются опции сокета с помощью функции `setsockopt`. В частности, устанавливается опция `SO_REUSEADDR`, которая позволяет повторно использовать локальные адреса.

Затем создается отдельный поток, который слушает ввод пользователя. Если пользователь вводит “`quit`”, сервер останавливается. Если вводится любая другая команда, выводится сообщение об ошибке.

После этого сервер начинает свою работу с помощью метода `start()`. Этот метод включает основной цикл обработки соединений.

Когда сервер останавливается, поток ввода присоединяется обратно к основному потоку, и программа выводит сообщение о том, что сервер остановлен.

2.4. Класс Server

`Server(int port)`: Конструктор класса. Инициализирует сервер на указанном порту.

`start()`: Запускает сервер. Сервер начинает прослушивание на указанном порту и обрабатывает входящие соединения.

`handleClient(int clientDescriptor)`: Обрабатывает входящее соединение от клиента. Этот метод запускается в отдельном потоке для каждого клиента.

`parseMessage(string &message, int clientDescriptor, bool *isRunning)`: Разбирает сообщение от клиента и выполняет соответствующие действия.

`getFileContents(const string &filePath)`: Возвращает содержимое файла по указанному пути.

`changeDirectory(const string &path, int clientDescriptor)`: Изменяет текущий рабочий каталог сервера.

`GetServerDescriptor() const`: Возвращает дескриптор сокета сервера.

`getCurrentTime() const`: Возвращает текущее время.

`stop()`: Останавливает сервер.

`listDirectory(const string &path, const string &prefix)`: Возвращает список файлов в указанном каталоге.

`writeToSocket(int clientDescriptor, const string &message)`: Отправляет сообщение клиенту.

`readFromSocket(int clientDescriptor)`: Читает сообщение от клиента.

`handleFileCommands(const string &command, int clientDescriptor, bool *isRunning)`: Обрабатывает команды, связанные с файлами.

`~Server()`: Деструктор класса. Закрывает все открытые соединения и освобождает ресурсы.

3. ФУНКЦИОНАЛЬНАЯ СТРУКТУРА ПРОЕКТА.

Проект собирается с помощью makefile. Для запуска проекта нам требуется в терминале запустить программу server. После этого, если сервер успешно стартовал мы можем запускать клиента.

В проекте имеется каталог для сборки debug и release. Каталог git для системы контроля версий моего проекта. Директория src с исходным кодом. И makefile для компиляции и сборки моего проекта.

4. ПОРЯДОК СБОРКИ И ИСПОЛЬЗОВАНИЯ.

Для компиляции и сборки проекта используется makefile.

Порядок сборки:

1. Определение переменных:

CXX - компилятор (g++).

CXXFLAGS - флаги компиляции, включающие предупреждения, отключение определенных предупреждений и стандарт C++17.

BUILD_DIR и RELEASE_DIR - пути к каталогам для отладочной и релизной сборки соответственно.

SRC_DIR - каталог с исходным кодом.

2. Определение целей:

all - основная цель, которая компилирует сервер и клиент.

server и client - цели для создания исполняемых файлов сервера и клиента соответственно.

release - цель для создания релизной версии, которая добавляет флаг оптимизации и изменяет каталог сборки на релизный.

$(BUILD_DIR) - цель для создания каталога сборки, если он не существует.

clean - цель для удаления всех объектных файлов и исполняемых файлов в каталогах сборки.

3.1 Определение правил:

$(BUILD_DIR)/server_main.o, $(BUILD_DIR)/client_main.o, $(BUILD_DIR)/Server.o, $(BUILD_DIR)/Client.o - правила для компиляции каждого исходного файла в объектный файл. Зависимости указывают, что объектные файлы должны быть перекомпилированы, если соответствующий исходный файл или заголовочный файл изменяется.

4.1 Определение зависимостей:

server и client зависят от соответствующих объектных файлов. Если объектные файлы изменяются, исполняемые файлы будут пересобраны.

Объектные файлы зависят от соответствующих исходных файлов и заголовочных файлов. Если исходный файл или заголовочный файл изменяется, объектный файл будет перекомпилирован.

Каталог сборки создается перед компиляцией объектных файлов, если он не существует.

Порядок использования.

1. Компиляция:

Для отладочной сборки: make или make MODE=debug

Для релизной сборки: make MODE=release

2. Очистка:

`make clean` - удаляет все объектные файлы и исполняемый файл.

3. Запуск:

После успешной компиляции запустите исполняемый файл, например: `./build/debug/server` или `./build/debug/client <ip> <port>`.

5. МЕТОД ТЕСТИРОВАНИЯ И РЕЗУЛЬТАТ ТЕСТИРОВАНИЯ.

1. Запуск server:

```
ilua@fedora:~/Рабочий стол/lab08/build/debug# ./server
Welcome to the educational server 'myserver'. myserver
started on port 8080.
If you want to start the server on a different port,
run: ./server <port_number>
Server is ready.
2024-05-14 21:56:37 Client connected
2024-05-14 21:57:32 Client disconnected
quit
Server stopped.
```

2. Запуск client:

```
ilua@fedora:~/Рабочий стол/lab08/build/debug# ./client
127.0.0.1 8080
> LIST
commands.txt
testdir/
 111.txt
serverInfo.txt
server_main.o
Server.o
server
client_main.o
Client.o
client

> INFO
Welcome to our server!
Here are some commands you can use:
ECHO - write message to server
INFO - Get server information
CD <directory> - Change current directory
LIST - List files in the current directory
QUIT - Disconnect from the server

> CD ..
```

```
Directory changed
> LIST
debug/
 commands.txt
 testdir/
   111.txt
 serverInfo.txt
 server_main.o
 Server.o
 server
 client_main.o
 Client.o
 client

> CD ..
Directory changed
> LIST
Makefile
src/
 Client.h
 Client.cpp
 Server.cpp
 Server.h
 client_main.cpp
 server_main.cpp
 serverInfo.txt
build/
 debug/
   commands.txt
   testdir/
     111.txt
   serverInfo.txt
   server_main.o
   Server.o
   server
   client_main.o
   Client.o
   client
command.txt
ОСиСП_L8_Лазовский_И.А..odt
.~lock.ОСиСП_L8_Лазовский_И.А..odt#

> CD ..
Cannot change directory outside the root directory
> ECHO hello
hello
> QUIT
```

```
Server closed connection.
```