**ОБЩАЯ ТЕОРИЯ**

**Мультиплексирование ввода-вывода** — это метод, позволяющий одному процессу или потоку одновременно отслеживать и обрабатывать события (например, поступление данных или возможность записи) на нескольких файловых дескрипторах (сокетах, файлах и т.д.). Вместо того чтобы создавать отдельный процесс или поток для каждого соединения, мультиплексирование позволяет эффективно управлять множеством соединений в рамках одного процесса, что снижает накладные расходы.

В контексте TCP эхо-сервера, мультиплексирование позволяет серверу одновременно обрабатывать несколько клиентских соединений, ожидая активности (например, поступление данных) на любом из сокетов.

**Как работает мультиплексирование?**

Мультиплексирование работает на основе системных вызовов, таких как select(2) или poll(2), которые предоставляют операционной системе список дескрипторов и типы событий (чтение, запись, ошибка), которые нужно отслеживать. Операционная система уведомляет процесс, когда на одном или нескольких дескрипторах происходит событие, после чего процесс может обработать эти события.

Создание набора дескрипторов: Программа формирует список файловых дескрипторов (например, сокетов), которые нужно отслеживать.

Указание событий: Программа указывает, какие события (чтение, запись, ошибка) интересуют для каждого дескриптора.

Ожидание событий: Системный вызов (select или poll) блокирует выполнение процесса, пока не произойдет событие на одном из дескрипторов или не истечет тайм-аут (если задан).

Обработка событий: После возврата системного вызова программа проверяет, какие дескрипторы готовы, и выполняет соответствующие действия (например, читает данные или отправляет ответ).

**Как мультиплексирование работает на уровне операционной системы?**

1. Файловые дескрипторы: Каждый дескриптор (например, сокет) связан с определенными структурами данных в ядре, такими как буферы ввода-вывода или флаги состояния.

2. Мониторинг событий: Системные вызовы, такие как select или poll, передают ядру список дескрипторов и типы событий (например, наличие данных для чтения). Ядро проверяет состояние каждого дескриптора (например, есть ли данные в буфере сокета).

3. Ожидание и уведомление: Если ни один из дескрипторов не готов, ядро переводит процесс в состояние ожидания (блокирует его). Как только событие происходит (например, данные поступают в сокет), ядро пробуждает процесс и возвращает информацию о готовых дескрипторах.

4. Эффективность: Ядро использует внутренние структуры данных (например, битовую маску для select или массив структур для poll) для минимизации накладных расходов на проверку состояния дескрипторов.

Операционная система использует механизмы вроде таймеров, очередей событий и прерываний от сетевых устройств, чтобы отслеживать изменения состояния дескрипторов. Это позволяет мультиплексированию быть эффективным даже при большом количестве соединений.

**Что такое select и poll?**

select(2) и poll(2) — это системные вызовы в POSIX-системах (например, Linux, Unix), используемые для мультиплексирования ввода-вывода. Они позволяют процессу отслеживать события на нескольких файловых дескрипторах.

**select(2)**

Описание: select отслеживает состояние дескрипторов, используя три набора (fd\_set) для чтения, записи и ошибок. Каждый набор — это битовая маска, где каждый бит соответствует дескриптору.

Прототип:

int select(int nfds, fd\_set \*readfds, fd\_set \*writefds, fd\_set \*exceptfds, struct timeval \*timeout);nfds: Максимальный дескриптор + 1.

readfds, writefds, exceptfds: Наборы дескрипторов для чтения, записи и ошибок.

timeout: Время ожидания (или NULL для бесконечного ожидания).

Как работает:

Программа заполняет наборы дескрипторов, которые нужно отслеживать.

select блокирует выполнение, пока не произойдет событие или не истечет тайм-аут.

После возврата select модифицирует наборы, оставляя только те дескрипторы, которые готовы.

**poll(2)**

Описание: poll использует массив структур struct pollfd, где каждая структура указывает дескриптор и типы событий для отслеживания.

Прототип:

int poll(struct pollfd \*fds, nfds\_t nfds, int timeout);fds: Массив структур pollfd, где каждая содержит дескриптор (fd), интересующие события (events) и возвращаемые события (revents).

nfds: Количество дескрипторов в массиве.

timeout: Время ожидания в миллисекундах (-1 для бесконечного ожидания).

Как работает:

Программа создает массив pollfd, указывая дескрипторы и события (например, POLLIN для чтения, POLLOUT для записи).

poll блокирует выполнение, пока не произойдет событие или не истечет тайм-аут.

После возврата poll заполняет поле revents в каждой структуре pollfd, указывая, какие события произошли.

**Сходства и различия между select и poll**

1. Сходства

Цель: Оба системных вызова используются для мультиплексирования ввода-вывода, позволяя отслеживать события на нескольких дескрипторах.

Блокировка: Оба блокируют процесс, пока не произойдет событие или не истечет тайм-аут.

Типы событий: Поддерживают похожие события (чтение, запись, ошибка).

Портативность: Оба являются частью стандарта POSIX и поддерживаются в большинстве Unix-подобных систем.

Использование в сетевом программировании: Подходят для серверов, таких как ваш TCP эхо-сервер, для обработки множества клиентов.

2. Различия



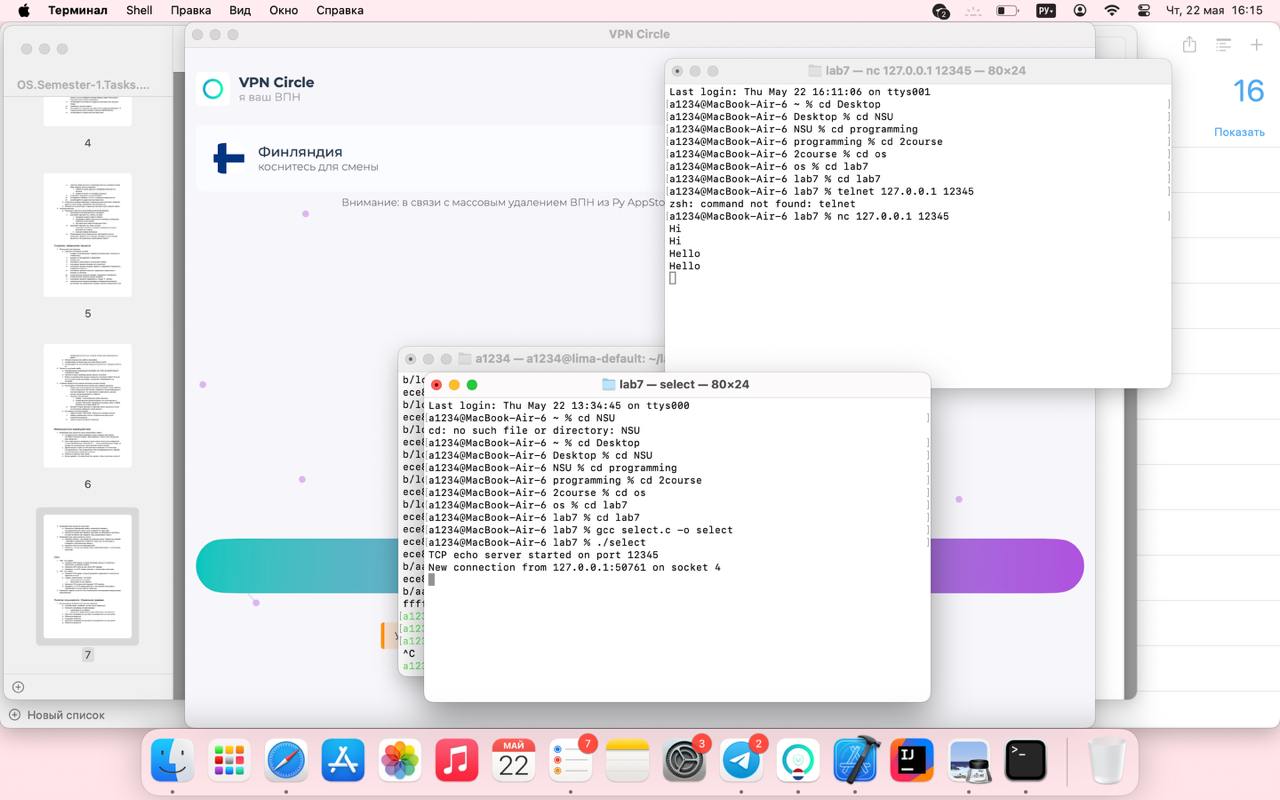
setsockopt(listen\_sock, SOL\_SOCKET, SO\_REUSEADDR, &opt, sizeof(opt));SOL\_SOCKET: Уровень протокола, на котором применяется опция. SOL\_SOCKET означает, что мы задаем опцию на уровне сокета (а не, например, на уровне протокола TCP или IP).

SO\_REUSEADDR: Название опции, которую мы хотим установить. Эта опция позволяет повторно использовать адрес и порт сокета, даже если они недавно использовались.

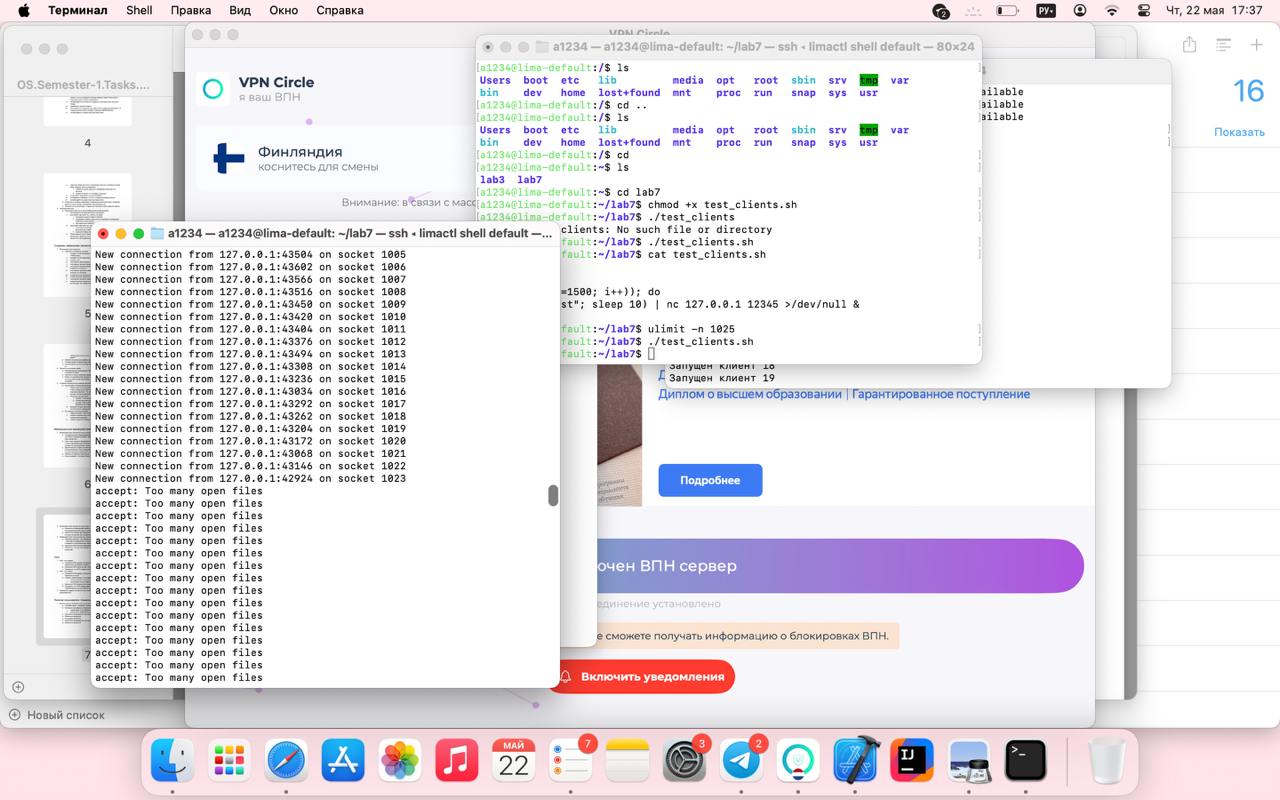
&opt: Указатель на значение опции. Здесь мы передаем адрес переменной opt, которая равна 1, чтобы включить SO\_REUSEADDR.

sizeof(opt): Размер данных, передаваемых в opt (в байтах). Поскольку opt — это int, обычно это 4 байта.

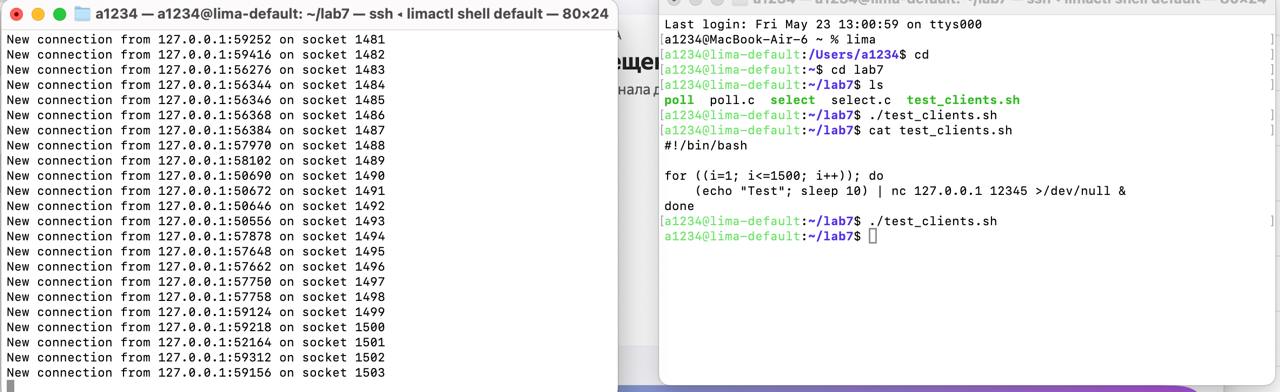
Проверка работы сервера и запуск клиента с помощью *nc <ip> <port>*



Проверка запуска 1500 клиентов с помощью скрипта test\_clients.sh сервера, работающего с **select**



Проверка запуска 1500 клиентов с помощью скрипта test\_clients.sh сервера, работающего с **poll**

****