**Отчётпо лабораторной работе №15**

### Дисциплина: Операционные системы

## Джеффри Родригес Сантос

**Содержание**

1. [Цель работы 5](#_TOC_250003)
2. [Задание 6](#_TOC_250002)
3. [Выполнение лабораторной работы 7](#_TOC_250001)
4. [Контрольные вопросы 13](#_TOC_250000)
5. [Выводы](#_bookmark0) 17

**Список таблиц**

# Цель работы

Приобретение практических навыков работы с именованными каналами.

# Задание

Изучите приведённые в тексте программы server.c и client.c. Взяв данные примеры за образец, напишите аналогичные программы, внеся следующие изменения:

* 1. Работает не 1 клиент, а несколько (например, два).
  2. Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Используйте функцию sleep() для приостановки работы клиента.
  3. Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). Используйте функцию clock() для определения времени работы сервера. Что будет в случае, если сервер завершит работу, не закрыв канал?

1. **Выполнение лабораторной работы**
2. Для начала я создал необходимые файлы с помощью команды «touch common.h server.c client.c Makefile» (рис. -fig. 3.1) и открыл редактор emacs для их редактирования.



Рис. 3.1: Создал необходимые файлы

1. Далее я изменил коды программ, представленных в тексте лабораторной работы.

В файл common.h добавил стандартные заголовочные файлы unistd.h и time.h, необходимые для работы кодов других файлов. Common.h предна- значен для заголовочных файлов, чтобы в остальных программах их не прописывать каждый раз (рис. -fig. 3.2).

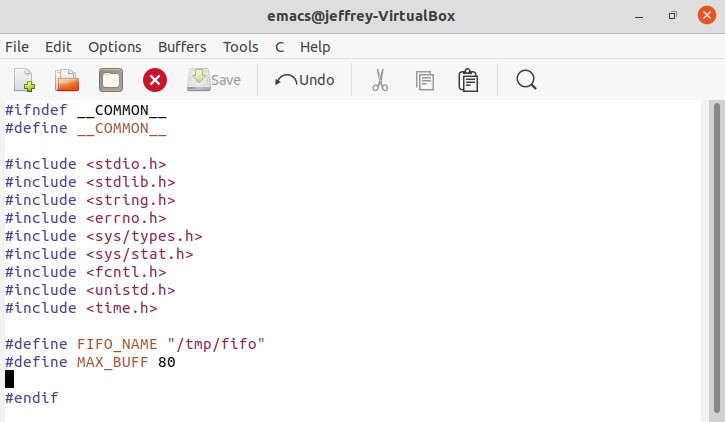
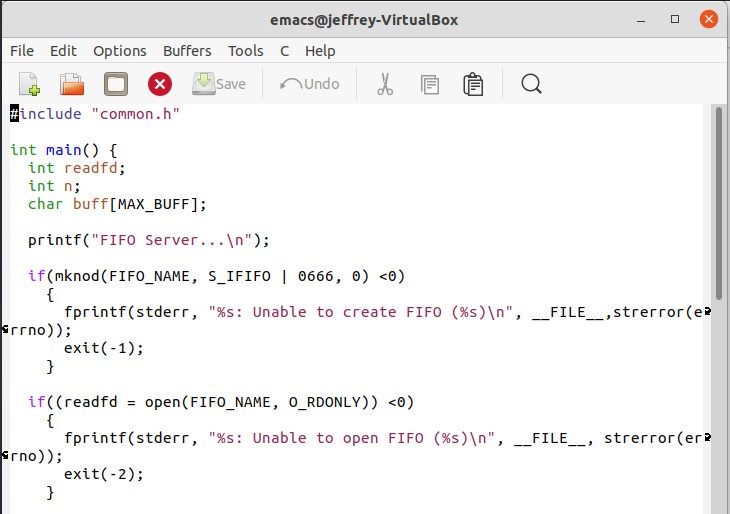


Рис. 3.2: Изменил код common.h

В файл server.c добавил цикл while для контроля за временем работы сервера. Разница между текущим временем time(NULL) и временем начала работы clock\_t start=time(NULL) (инициализация до цикла) недолжна превышать 30 секунд (рис.

-fig. 3.3) (рис. -fig. 3.4).



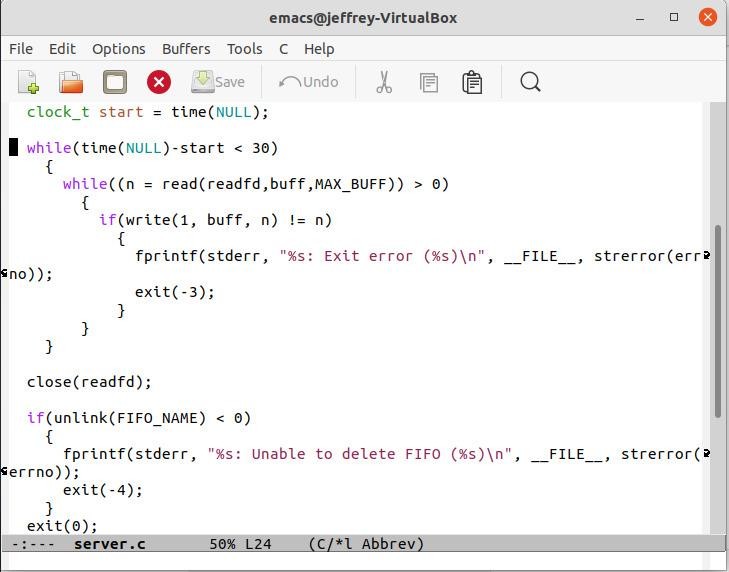
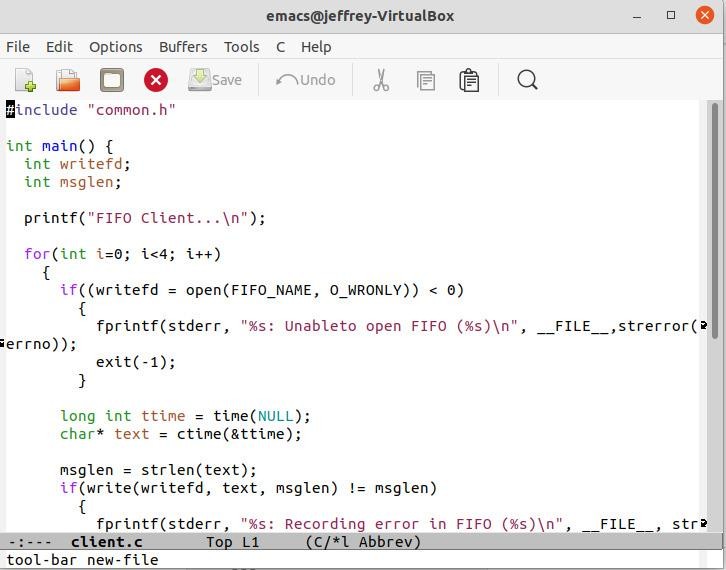
Рис. 3.3: Изменил кодserver.c

Рис. 3.4: Изменил кодserver.c

В файл client.c добавил цикл, который отвечает за количество сообщений о текущем времени (4 сообщения), которое получаетсяврезультате выполнения команд, икомандуsleep(5) дляприостановки работыклиента на 5 секунд (рис.- fig. 3.5).



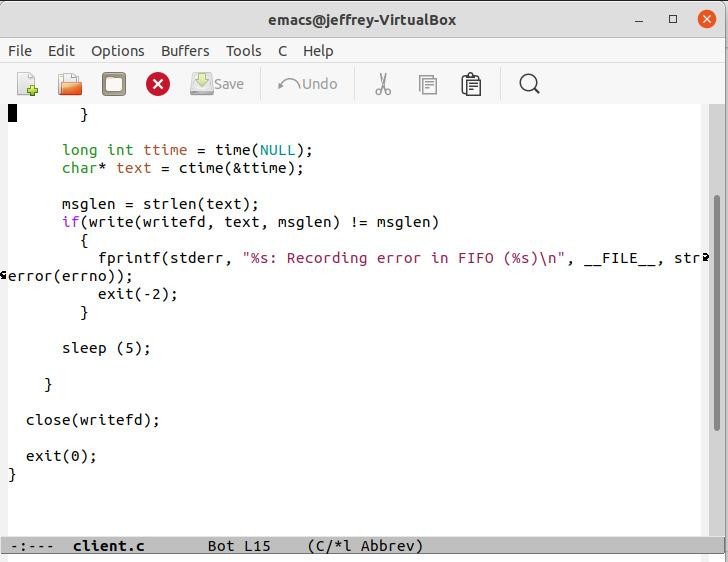


Рис. 3.5: Изменил код client.c Makefile (файл для сборки) не изменял (рис.fig. 3.6).

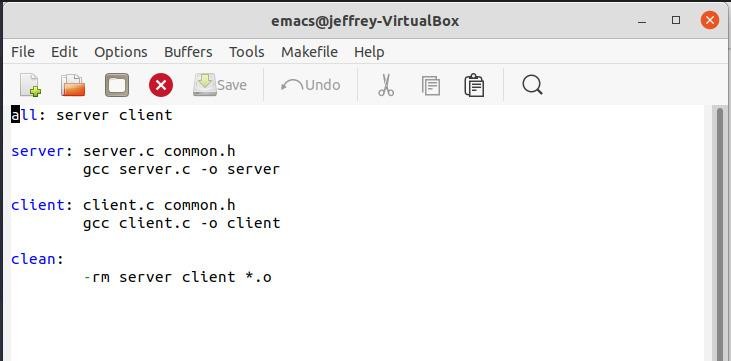


Рис. 3.6: Makefile не изменял

1. После написания кодов, я, используя команду «make all», скомпилировал необходимые файлы (рис. -fig. 3.7).

Рис. 3.7: Скомпилировал необходимые файлы Далее я проверил работу написанного кода.

Отрыл 3 консоли (терминала) и запустил: в первом терминале −«./server», в остальных двух – «./client». В результате каждый терминал-клиент вывел по 4 сообщения. Спустя 30 секунд работа сервера была прекращена (рис. -fig. 3.8). Программа работает корректно.

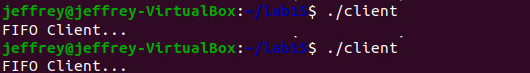
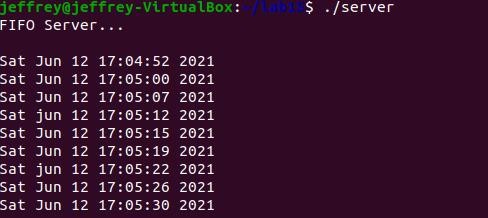


Рис. 3.8: Проверил работу написанного кода

Также я отдельно проверил длительность работы сервера, введя команду «.

/server» в одном терминале. Он завершил свою работу через 30 секунд.Если сервер завершит свою работу, не закрыв канал, то, когда мы будем запус- кать этот сервер снова, появится ошибка «Невозможно создать FIFO», так как у нас уже есть одинканал.

1. **Контрольные вопросы**
   1. Именованные каналы отличаются от неименованных наличием идентифи- катора канала, который представлен как специальный файл (соответственно имя именованного канала — это имя файла). Поскольку файл находится на локальной файловой системе, данное IPC используется внутри одной системы.
   2. Чтобы создать неименованный канал из командной строки нужно использо- вать символ |, служащий дляобъединения двух иболее процессов: процесс\_1

| процесс\_2 | процесс\_3…

* 1. Чтобы создать именованный канал из командной строки нужно использо- вать либо команду «mknod», либо команду «mkfifo ».
  2. Неименованный канал является средством взаимодействия между связанными процессами − родительским и дочерним. Родительский процесс со- здает канал при помощи системного вызова: «int pipe (int fd[2]);».

Массив из двух целых чисел является выходным параметром этого системного вызова.Если вызов выполнился нормально,тоэтотмассив содержит два файловых дескриптора. fd[0] являетсядескриптором для чтения из ка- нала, fd[1] − дескриптором для записи в канал. Когда процесс порождает другой процесс, дескрипторы родительского процесса наследуются дочерним процессом, и, таким образом, прокладывается трубопровод между двумя процессами. Естественно, чтоодинизпроцессовиспользуетканал толькодля чтения,адругой − толькодлязаписи.Поэтому,если,например, через канал должны передаваться данные из родительского процесса в

дочерний, родительский процесс сразу после запуска дочернего процесса закрывает дескриптор канала для чтения, а дочерний процесс закрывает дескриптор для записи. Если нужен двунаправленный обмен данными между процессами, то родительский процесс создает два канала, один из которых используется для передачи данных в одну сторону, а другой − в другую.

* 1. Файлы именованных каналов создаются функцией mkfifo() или функцией mknod:
* «int mkfifo(const char pathname, mode\_t mode);», где первый параметр − путь,гдебудет располагаться FIFO(имяфайла, идентифицирующего канал), второй параметр определяет режим работы с FIFO (маска прав доступа к файлу),
* «mknod(namefile, IFIFO| 0666, 0)», гдеnamefile− имя канала, 0666 − кканалу разрешен доступ на запись и на чтение любому запросившему процессу),
* «int mknod(const char pathname, mode\_t mode, dev\_tdev);». Функция mkfifo() создаетканалифайл соответствующего типа. Еслиуказан- ный файл канала уже существует, mkfifo() возвращает -1. После создания файла канала процессы, участвующие в обмене данными, должны открыть этот файл либо для записи, любо для чтения.
  1. При чтении меньшего числа байтов, чем находится в канале или FIFO, возвращается требуемое число байтов, остаток сохраняется для последующих чтений.

При чтении большего числа байтов, чем находится в канале или FIFO, возвращается доступное число байтов. Процесс, читающий из канала, должен соответствующим образом обработать ситуацию, когда прочитано меньше, чем заказано.

* 1. Запись числа байтов, меньшего емкости канала или FIFO, гарантированно атомарно. Это означает, что в случае, когда несколько процессов одновре- менно записывают в канал, порции данных от этих процессов не переме- шиваются.

Призаписи большего числа байтов, чем это позволяет канал или FIFO, вызов write(2) блокируется до освобождения требуемого места. При этом атомар- ность операции не гарантируется. Если процесс пытается записать данные в канал, не открытый ни одним процессом на чтение, процессу генериру- ется сигнал SIGPIPE, а вызов write(2) возвращает 0 с установкой ошибки (errno=ERRPIPE) (если процесс не установил обработки сигнала SIGPIPE, производится обработка по умолчанию − процессзавершается).

* 1. Количество процессов, которые могут параллельно присоединяться к лю- бому концу канала, не ограничено. Однако если два или более процесса записывают в канал данные одновременно, каждый процесс за один раз можетзаписать максимум PIPE BUF байтов данных. Предположим, процесс (назовем его А) пытается записать X байтов данных в канал, в котором имеется место для Y байтов данных. Если X больше, чем Y, только первые Y байтов данных записываются в канал, и процесс блокируется. Запускается другой процесс (например. В); в это время в канале появляется свободное пространство (благодаря третьему процессу, считывающему данные из канала). Процесс В записывает данные в канал. Затем, когда выполнение процесса А возобновляется,он записывает оставшиеся X-Yбайтов данных в канал. Врезультате данныевканал записываются поочередно двумя процессами. Аналогичным образом, если два (или более) процесса одновременно попытаются прочитать данные из канала, может случиться так, что каждый из них прочитает только часть необходимых данных.
  2. Функция write записывает байты count из буфера buffer в файл, связанный с handle. Операции write начинаются с текущей позиции указателя на файл (указатель ассоциирован с заданным файлом). Если файл открыт для добавления, операции выполняются в конец файла. После осуществ- ления операций записи указатель на файл (если он есть) увеличивается на количество действительно записанных байтов. Функция write возвращает число действительно записанных байтов. Возвращаемое значениедолжно

быть положительным, но меньше числа count (например, когда размер для записи count байтов выходит за пределы пространства на диске). Возвра- щаемое значение -1 указывает на ошибку; errno устанавливается в одно из следующих значений:

EACCES − файл открыт для чтения или закрыт для записи, EBADF − неверный handle-рфайла,

ENOSPC − на устройстве нет свободного места.

Единица в вызове функции write в программе server.c означает идентифи- катор (дескриптор потока) стандартного потока вывода.

* 1. Прототип функции strerror:«char strerror(int errornum );». Функция strerror интерпретирует номер ошибки, передаваемый в функ- цию в качестве аргумента − errornum, в понятное для человека текстовое сообщение (строку).Откуда берутсяэти ошибки? Ошибки эти возникают при вызове функций стандартных Си-библиотек. То естьхорошимтоном программирования будет − использование этой функции в паре с другой, и если возникнет ошибка, то пользователь или программист поймет, как исправить ошибку, прочитав сообщение функции strerror.

Возвращенный указатель ссылается на статическую строку с ошибкой, которая не должна быть изменена программой.

Дальнейшие вызовы функ- ции strerror перезапишут содержание этой строки. Интерпретированные сообщения об ошибках могут различаться, это зависит от платформы и компилятора.

# Выводы

В ходе выполнения даннойлабораторной работы яприобрёл практические навыки работы с именованнымиканалами.