# Лабораторная работа №14

**Дисциплина: Операционные системы**

Аксенова Анастасия

1. [**Цель работы**](#_bookmark0) **5**
2. [**Задание**](#_bookmark1) **6**
3. [**Выполнение лабораторной работы**](#_bookmark2) **7**
4. [**Библиография**](#_bookmark13) **16**
5. [**Выводы**](#_bookmark14) **17**

[3.1 Создание файлов](#_bookmark3) . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

[3.2 Прграмма в файле common.h](#_bookmark4) . . . . . . . . . . . . . . . . . . . . 8 [3.3 Прграмма в файле server.c](#_bookmark5) . . . . . . . . . . . . . . . . . . . . . . 8 [3.4 Прграмма в файле server.c](#_bookmark6) . . . . . . . . . . . . . . . . . . . . . . 9 [3.5 Прграмма в файле client.c](#_bookmark7) . . . . . . . . . . . . . . . . . . . . . . 9 [3.6 Прграмма в файле client.c](#_bookmark8) 10

* 1. [Прграмма в Mikefile](#_bookmark9) 10
  2. [Команда make all](#_bookmark10) 11
  3. [Команда make all](#_bookmark11) 11
  4. [Проверка длительности работы сервера](#_bookmark12) 12

Цель данной лабораторной работы — Приобретение практических навыков рабо- ты с именованными каналами.

1. Сделать отчёт по лабораторной работе №15 в формате Markdown.
2. Приобрести практические навыки работы с именованными каналами.
3. Для начала я создала необходимые файлы с помощью команды «touch common.h server.c client.c Makefile» и открыла редактор emacsдля их редакти- рования (Рисунок [3.1](#_bookmark3) ). Вся необходимая информация про создания каталогов указана в следующем источнике: Программное обеспечение GNU/Linux. Лекция

10. Минимальный набор знаний (Г. Курячий, МГУ)

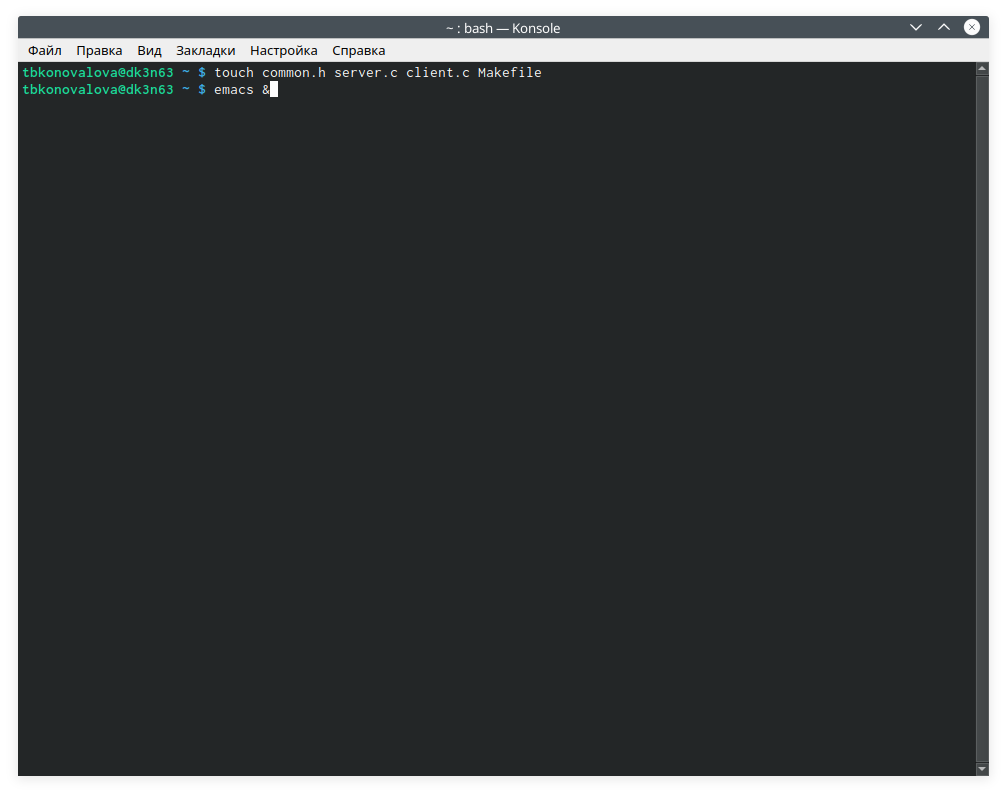


Figure 3.1: Создание файлов

* 1. Далее я изменила коды программ, представленных в тексте лабораторной работы. В файл common.h добавила стандартные заголовочные файлы unistd.h и time.h, необходимые для работы кодов других файлов. Common.h предназначен для заголовочных файлов, чтобы в остальных программах их не прописывать

каждый раз (Программа представлена на рис. [3.2](#_bookmark4) ). Вся необходимая информация про коды программ указана в следующем источнике: Программное обеспечение GNU/Linux. Лекция 12. Выбор дистрибутива (Г. Курячий, МГУ)

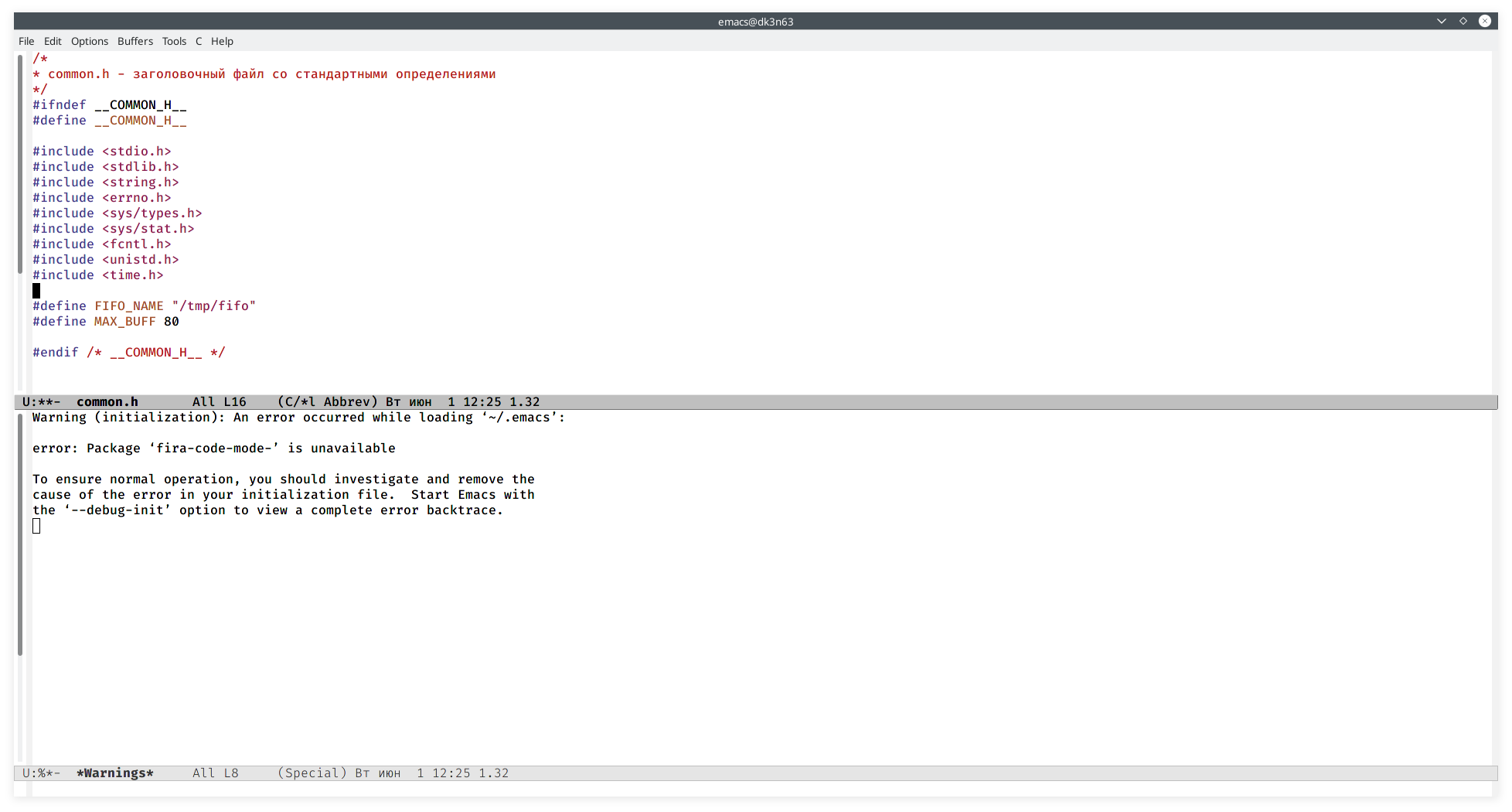


Figure 3.2: Прграмма в файле common.h

В файл server.c добавила 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) не должна превышать 30 секунд (алгоритм действий представлен на рис. [3.3](#_bookmark5) , [3.4](#_bookmark6) ).

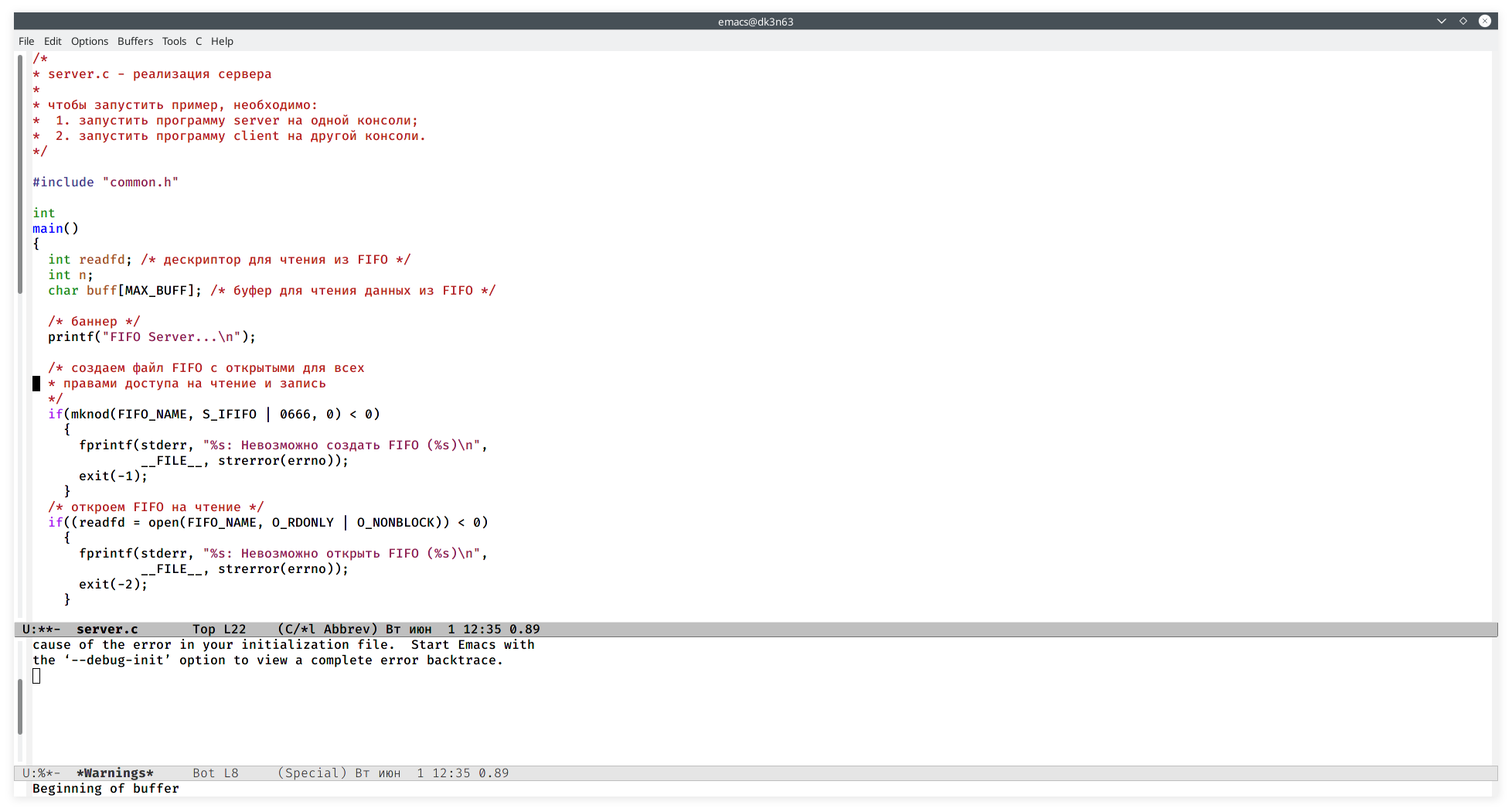


Figure 3.3: Прграмма в файле server.c

Figure 3.4: Прграмма в файле server.c

В файл client.c добавила цикл, который отвечает за количество сообщений о текущем времени (4 сообщения), которое получается в результате выполнения команд на Рисунке 7 (/*текущее время*/) и команду sleep(5) для приостановки работы клиента на 5 секунд (алгоритм действий представлен на рис. [3.5](#_bookmark7) , [3.6](#_bookmark8) ).

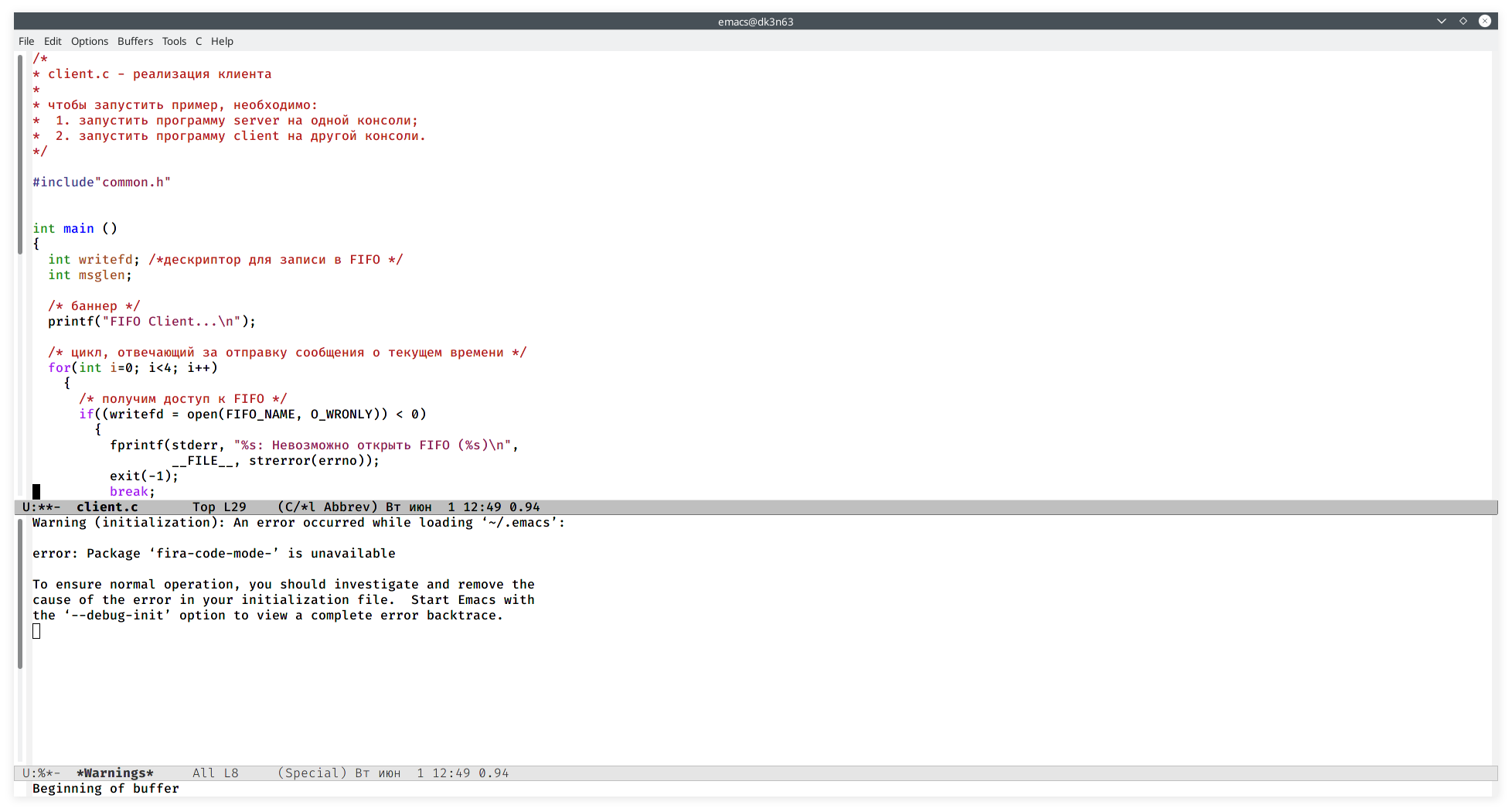


Figure 3.5: Прграмма в файле client.c

Figure 3.6: Прграмма в файле client.c

Makefile (файл для сборки) не изменяла (Рисунок [3.7](#_bookmark9) ).

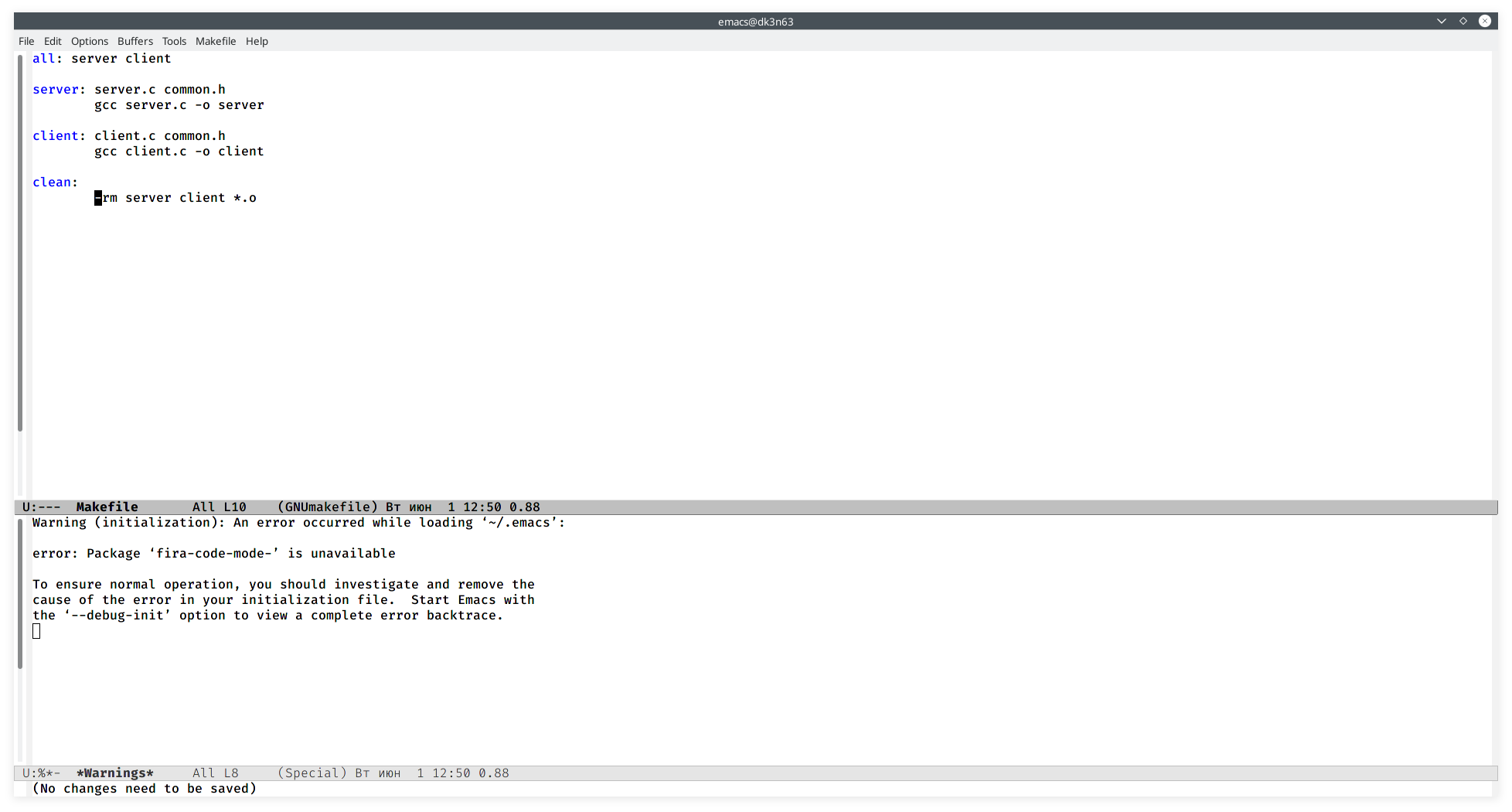


Figure 3.7: Прграмма в Mikefile

* 1. После написания кодов, я, используя команду «make all», скомпилировала необходимые файлы (Скриншот [3.8](#_bookmark10) ). Подробную информацию по данной теме можно найти в следующих интернет-источниках: https://habr.com/ru/post/122108/

, ht[tps://w](http://www.opennet.ru/docs/RUS/linux_parallel/node17.html)ww[.opennet.ru/docs/RUS/linux\_parallel/node1](http://www.opennet.ru/docs/RUS/linux_parallel/node17.html)7[.html](http://www.opennet.ru/docs/RUS/linux_parallel/node17.html)

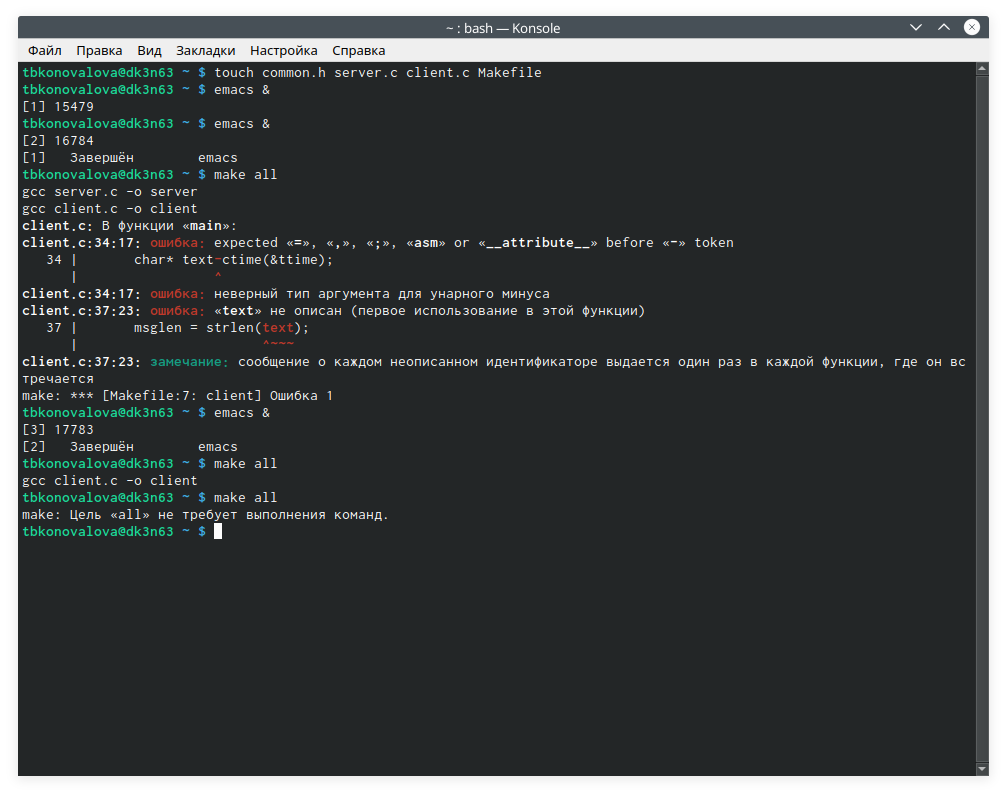


Figure 3.8: Команда make all

Далее я проверила работу написанного кода. Отрыла 3 консоли (терминала) и запустила: в первом терминале − «./server», в остальных двух – «./client». В результате каждый терминал-клиент вывел по 4 сообщения. Спустя 30 секунд работа сервера была прекращена (Рисунок [3.9](#_bookmark11) ). Программа работает корректно.

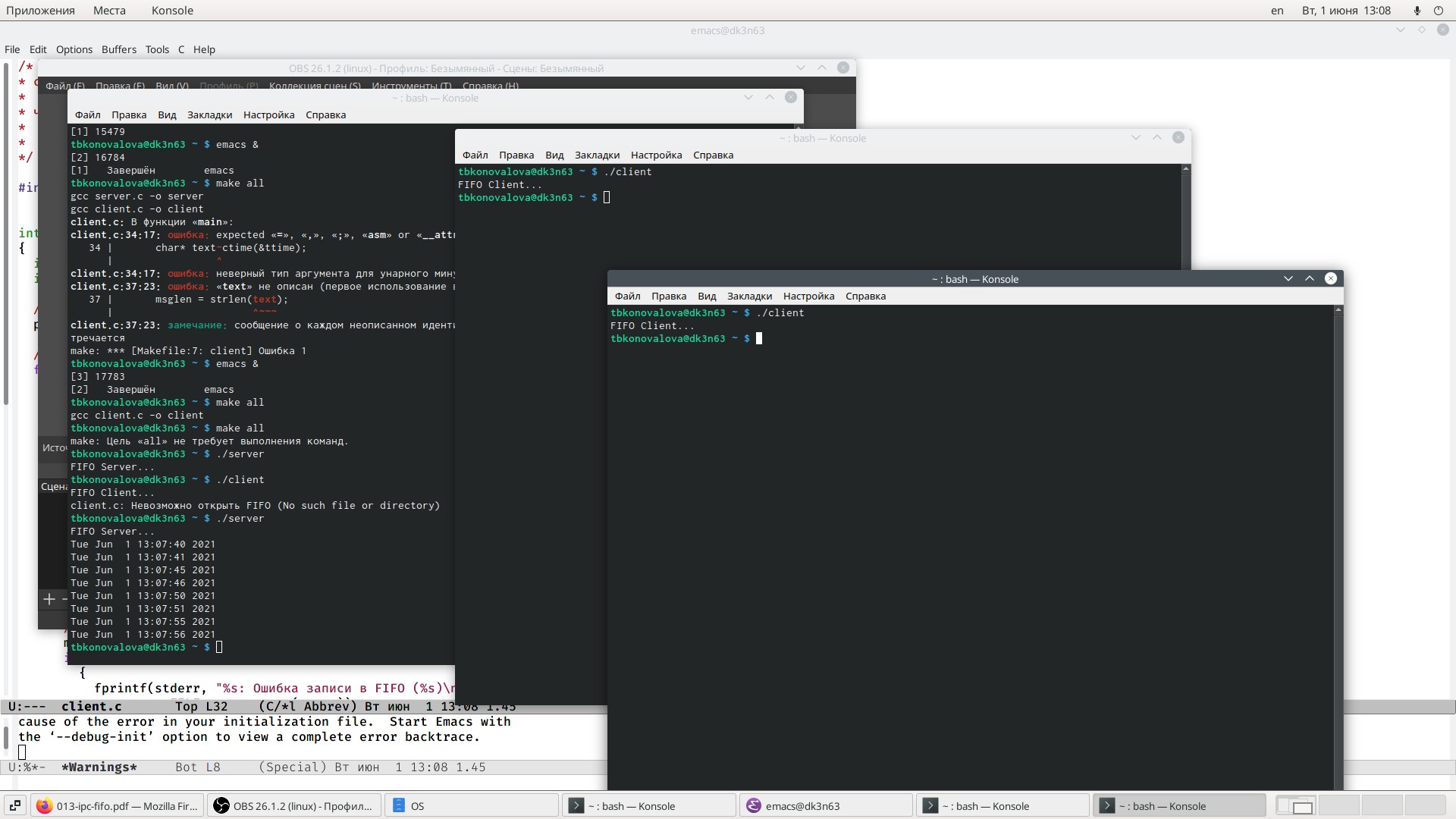


Figure 3.9: Команда make all

Также я отдельно проверила длительность работы сервера, введя команду

«./server»в одном терминале. Он завершил свою работу через 30 секунд (алго- ритм действий представлен на рис. [3.10](#_bookmark12) ). Если сервер завершит свою работу, не закрыв канал, то, когда мы будем запускать этот сервер снова, появится ошибка

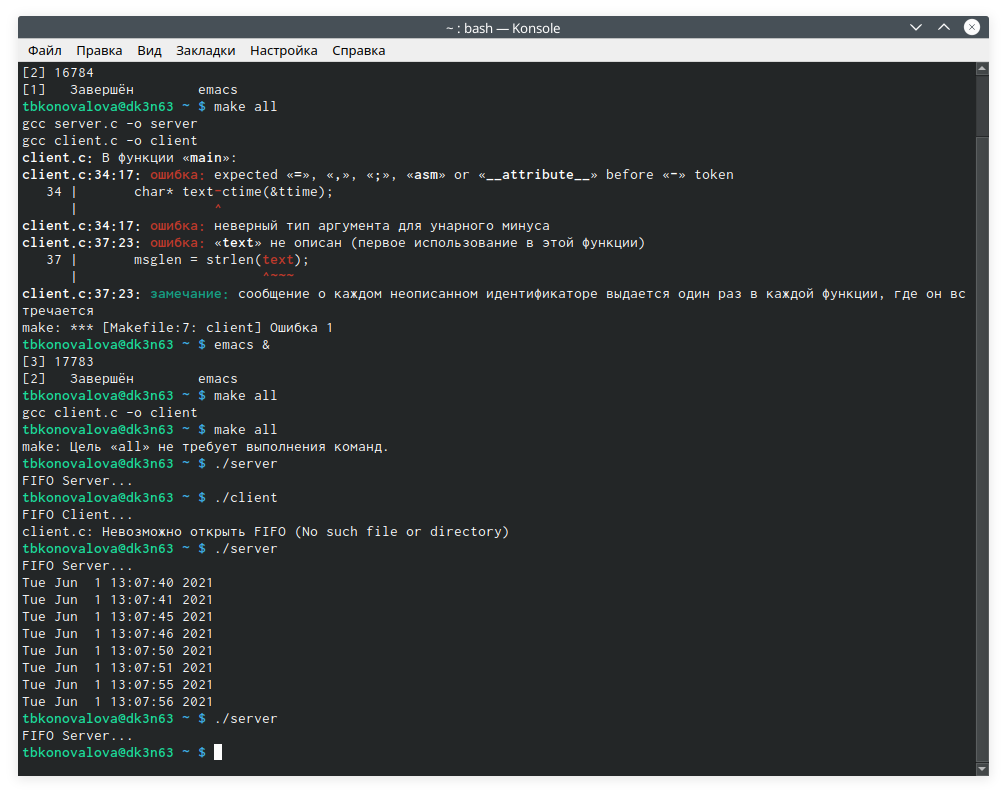
«Невозможно создать FIFO», так как у нас уже есть один канал.

Figure 3.10: Проверка длительности работы сервера 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 строкинужно исполь- зовать символ |, служащий для объединения двух и более процессов: процесс\_1

|процесс\_2 | процесс\_3…

1. Чтобы создать именованный канал из командной строкинужно использо- вать либо команду «mknod», либо команду «mkfifo».
2. Неименованный канал является средством взаимодействия между связан- ными процессами −родительским и дочерним. Родительский процесс создает

канал при помощи системного вызова: «int pipe(int fd[2]);». Массив из двух целых чисел является выходным параметром этого системного вызова. Если вызов вы- полнился нормально, то этот массив содержит два файловых дескриптора. fd[0] является дескриптором для чтения из канала, 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, что один из процессов ис- пользует канал только для чтения, а другой −только для записи. Поэтому, если, например, через канал должны передаваться данные из родительского процесса в дочерний, родительский процесс сразу после запуска дочернего процесса закры- вает 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: 1. «intmkfifo(constchar*pathname, mode\_tmode);», где первый параметр − путь, где будет располагаться FIFO (имя файла, идентифицирующего канал), вто- рой параметр определяет режим работы с FIFO (маска прав доступа к файлу), 2.*

*«mknod (namefile, IFIFO | 0666, 0)», где namefile −имя канала, 0666 −к каналу разрешен доступ на запись и на чтение любому запросившему процессу), 3. «int mknod(const char* pathname, mode\_t mode, dev\_t dev);».Функцияmkfifo() создает канал и файл соответствующего типа. Если указанный 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 одновременно- записывают в канал, порции данных от этих процессов не перемешиваются.При записи большего числа байтов, чем это позволяет канал или FIFO, вызов write(2) блокируется до освобождения требуемого места. При этом атомарность операции не гарантируется. Если процесс пытается записать данные в канал, не открытый ни одним процессом на чтение, процессу генерируется сигнал SIGPIPE, а вызов write(2) возвращает 0 с установкой ошибки (errno=ERRPIPE) (если процесс не установил обработки сигнала SIGPIPE, производится обработка по умолчанию

−процесс завершается).

1. Количество процессов, которые могут параллельно присоединяться к любо- му концу канала, не ограничено. Однако если два или более процесса записывают в канал данные одновременно, каждый процесс за один раз может записать мак- симум 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, считы- вающему данные из канала). Процесс В записывает данные в канал. Затем, когда выполнение процесса А возобновляется, он записывает оставшиеся X-Y байтов данных в канал. В результате данные в канал 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 вфайл, связанный с handle. Операции write начинаются с текущей позиции указателя на файл (ука- затель ассоциирован с заданным файлом). Если файл открыт для добавления, операции выполняются в конец файла. После осуществления операций запи-

си указатель нафайл(если он есть) увеличивается на количество действительно записанных байтов.Функция write возвращает число действительно записан- ныхбайтов. Возвращаемое значение должно быть положительным, но меньше числа count (например, когда размер для записи count байтоввыходит за пре- делы пространства на диске). Возвращаемое значение-1 указывает на ошибку; errno устанавливается в одно из следующихзначений:EACCES − файл открыт для чтения или закрыт длязаписи,EBADF− неверный handle-р файла,ENOSPC − на устройстве нет свободного места.Единица в вызове функции writeв програм- ме server.cозначаетидентификатор (дескриптор потока) стандартного потока вывода.

1. Прототип функции strerror: «char strerror( int errornum );». Функция strerror интерпретирует номер ошибки, передаваемый в функцию в качестве аргумента

−errornum, в понятное для человека текстовое сообщение (строку). Откуда бе- рутся эти ошибки? Ошибки эти возникают при вызове функций стандартных Си-библиотек. То есть хорошим тоном программирования будет − использова- ние 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 функции strerror перезапишут содержание этой строки. Интерпретированные сообщения об ошибках могут различаться, это зависит от платформы и компилятора.

1. Программное обеспечение GNU/Linux. Лекция 10. Минимальный набор знаний (Г. Курячий, МГУ)
2. Программное обеспечение GNU/Linux. Лекция 12. Выбор дистрибутива (Г. Курячий, МГУ)
3. Электронный ресурс: https://habr.com/ru/post/122108/
4. Электронный ресурс: ht[tps://w](http://www.opennet.ru/docs/RUS/linux_parallel/node17.html)ww[.opennet.ru/docs/RUS/linux\_parallel/node1](http://www.opennet.ru/docs/RUS/linux_parallel/node17.html)7[.html](http://www.opennet.ru/docs/RUS/linux_parallel/node17.html)

В ходе выполнения данной лабораторной работы я приобрела практические навыки работы с именованными каналами.