Лабораторная работа №14

Дисциплина: Операционные системы

Кабанова Варвара Дмитриевна

Содержание

List of Figures

List of Tables

1 Цель работы

Приобретение практических навыков работы с именованными каналами.

2 Выполнение лабораторной работы

Для начала я создала необходимые файлы с помощью команды «touch common.h server.c client.c Makefile» и открыла редактор emacsдля их редактирования (рис.1). Вся необходимая информация про создания каталогов указана в следующем источнике: Программное обеспечение GNU/Linux. Лекция 10. Минимальный набор знаний (Г. Курячий, МГУ)

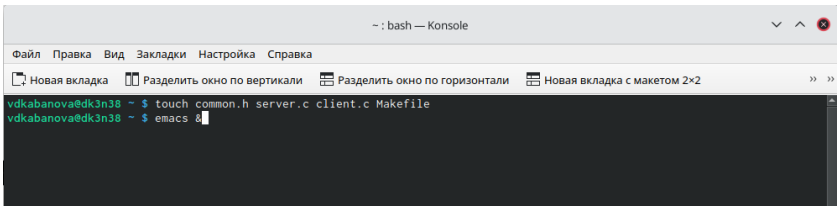

Figure 2.1: Создание файлов

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 (рис.2). Вся необходимая информация про коды программ указана в следующем источнике: Программное обеспечение GNU/Linux. Лекция 12. Выбор дистрибутива (Г. Курячий, МГУ)

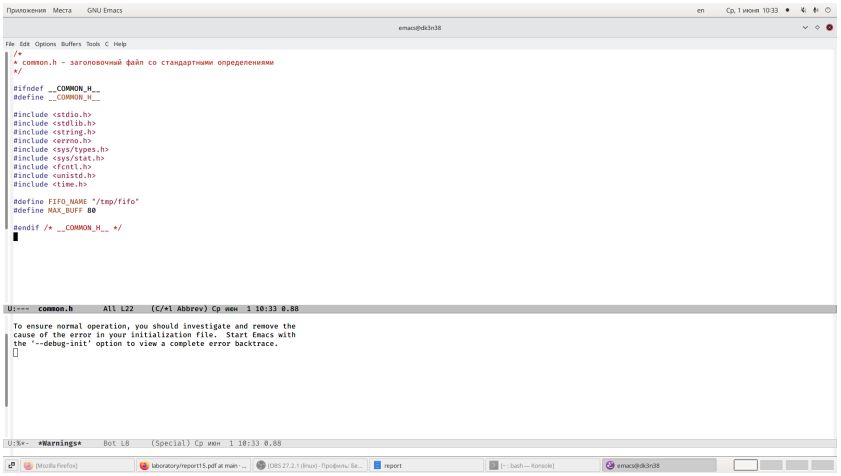

Figure 2.2: Прграмма в файле common.h

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 (рис.3-4).

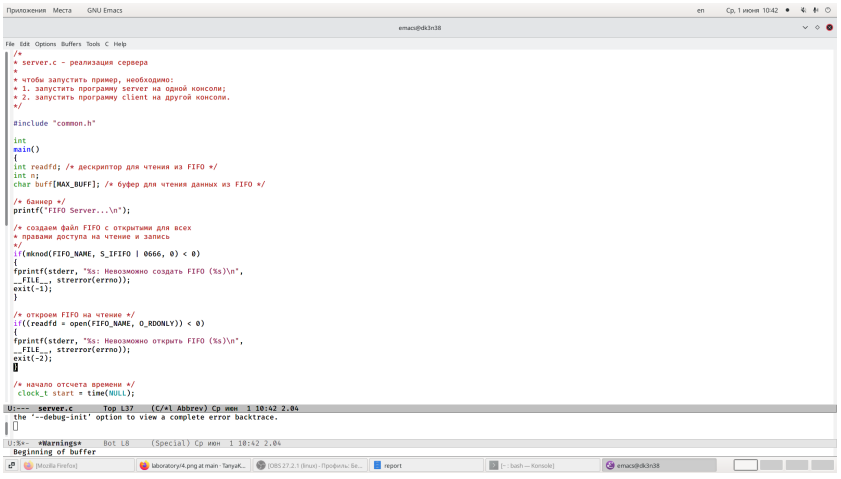

Figure 2.3: Прграмма в файле server.c

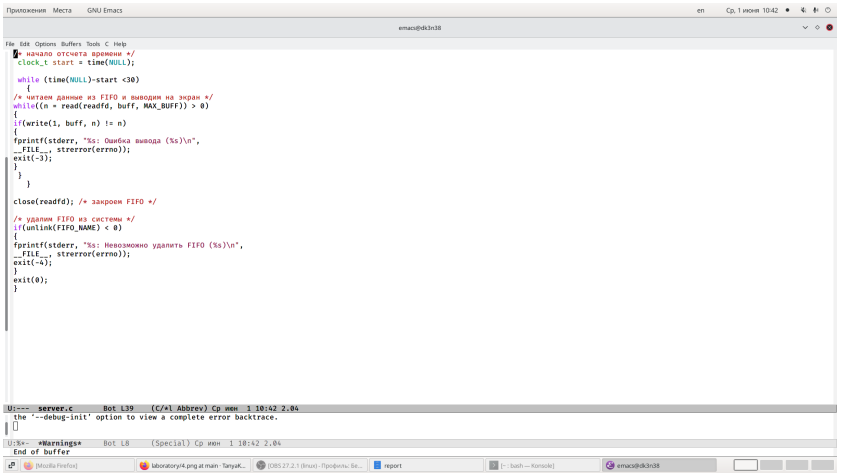

Figure 2.4: Прграмма в файле server.c

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 (рис.5-6).

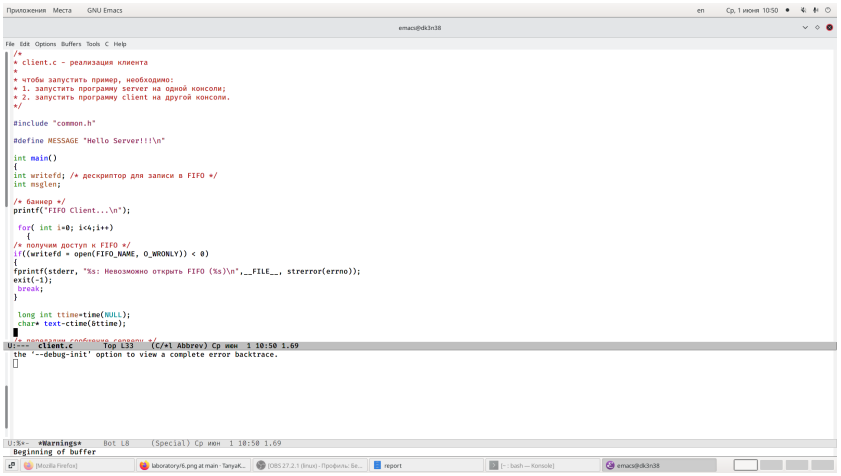

Figure 2.5: Прграмма в файле client.c

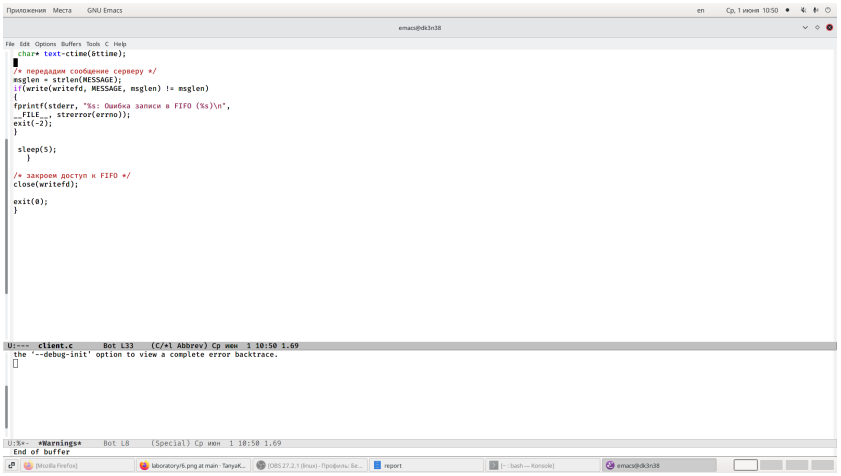

Figure 2.6: Прграмма в файле client.c

Makefile (файл для сборки) не изменяла (рис.7).

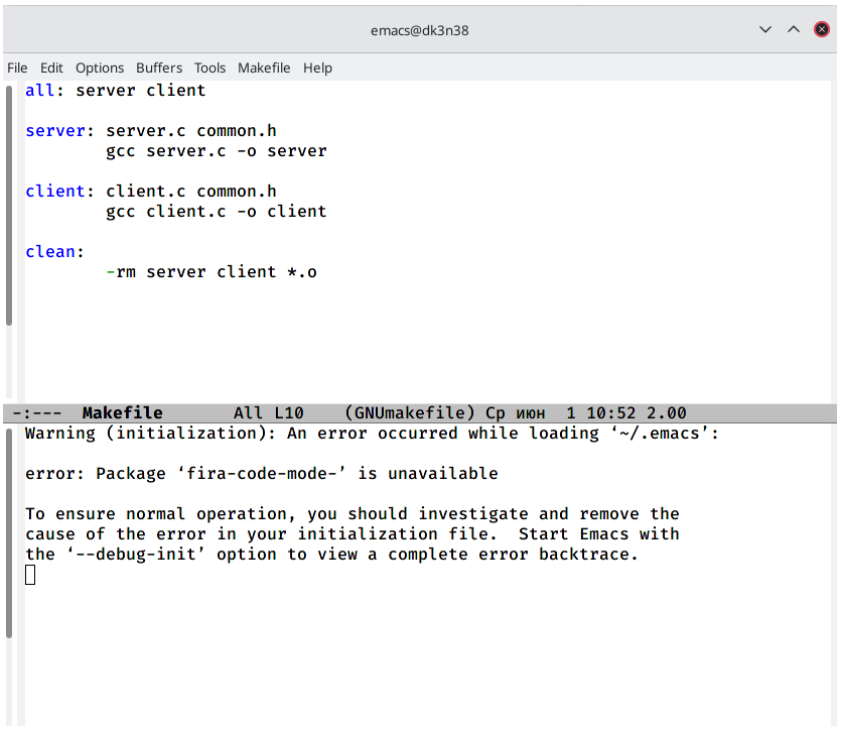

Figure 2.7: Прграмма в Mikefile

После написания кодов, я, используя команду «make all», скомпилировала необходимые файлы (рис.8). Подробную информацию по данной теме можно найти в следующих интернет-источниках: https://habr.com/ru/post/122108/ ,

https://www.opennet.ru/docs/RUS/linux_parallel/node17.html

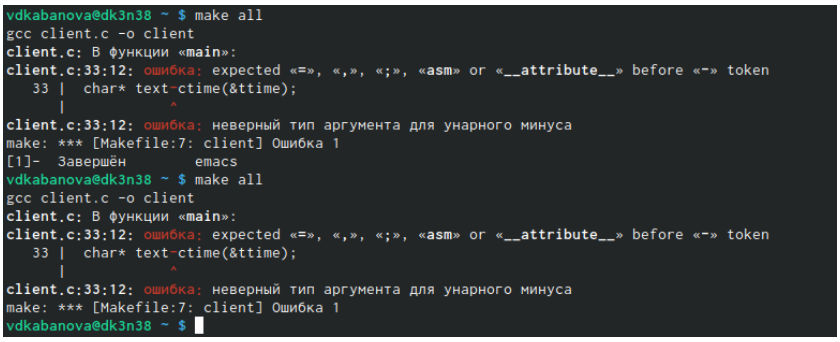

Figure 2.8: Команда make all

Вижу, что выскакивает ошибка в 33 строке файла client.c, проверяю его, понимаю, что допустила ошибку в использовании символа, меняю - на = (рис.9)

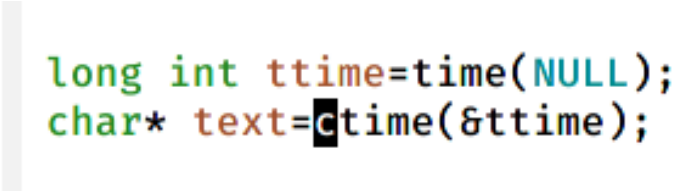

Figure 2.9: Исправление ошибок

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 (рис.10-11). Программа работает корректно.

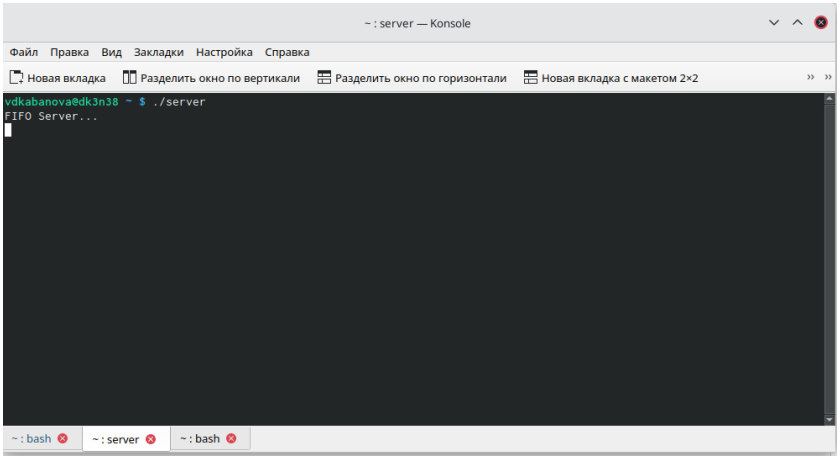

Figure 2.10: Проверка длительности работы сервера

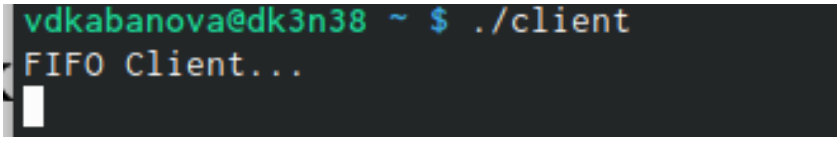

Figure 2.11: Проверка длительности работы сервера

Также я отдельно проверила длительность работы сервера, введя команду «./server»в одном терминале. Он завершил свою работу через 30 секунд (алгоритм действий представлен на рис. -fig. 2.10). 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.

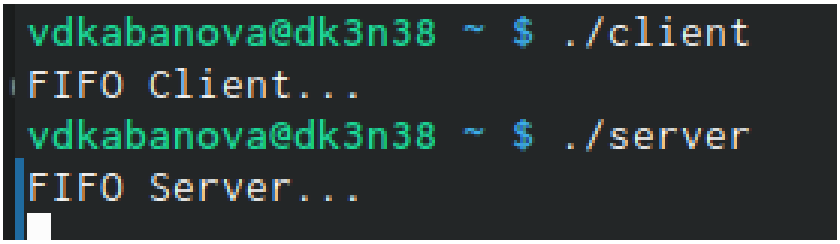

Figure 2.12: Проверка длительности работы сервера

3 Контрольные вопросы:

1. Именованные каналы отличаются от неименованных наличием идентификатора 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 использовать символ |, служащий для объединения двух и более процессов: процесс_1 |процесс_2 | процесс_3…

3. Чтобы создать именованный канал из командной строкинужно использовать либо команду «mknod», либо команду «mkfifo».

4. Неименованный канал является средством взаимодействия между связанными 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 выполнился 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 использует 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 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.

5. Файлы именованных каналов создаются функцией mkfifo() или функцией mknod:

- «intmkfifo(constchar*pathname, mode_tmode);», где первый параметр – путь, где будет располагаться FIFO (имя файла, идентифицирующего канал), второй параметр определяет режим работы с FIFO (маска прав доступа к файлу),
- «mknod (namefile, IFIFO | 0666, 0)», где namefile –имя канала, 0666 –к каналу разрешен доступ на запись и на чтение любому запросившему процессу),
- «int mknod(const char *pathname, mode_t mode, dev_t dev);».Функциямkfifo() 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.

6. 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.

7. Запись числа байтов, меньшего емкости канала или FIFO, гарантированно атомарно. Это означает, что в случае, когда несколько процессов одновременнозаписывают 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).

8. Количество процессов, которые могут параллельно присоединяться к любому концу канала, не ограничено. Однако если два или более процесса записывают в канал данные одновременно, каждый процесс за один раз может записать максимум PIPE BUF байтов данных. Предположим, процесс (назовем его A) 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. B); в это время в канале появляется свободное пространство (благодаря третьему процессу, считывающему данные из канала). Процесс B записывает данные в канал. Затем, когда выполнение процесса A 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.

9. Функция write записывает байты count из буфера buffer вфайл, 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 осуществления операций записи указатель нафайл(если он есть) увеличивается на количество действительно записанных байтов.Функция write возвращает число действительно записанныхбайтов. Возвращаемое значение должно быть

положительным, но меньше числа count (например, когда размер для записи count байтоввыходит за пределы 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 программе server.сознаетидентификатор (дескриптор потока) стандартного потока вывода.

10. 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 берутся эти ошибки? Ошибки эти возникают при вызове функций стандартных Си-библиотек. То есть хорошим тоном 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 функции strerror перезапишут содержание этой строки. Интерпретированные сообщения об ошибках могут различаться, это зависит от платформы и компилятора.

4 Выводы

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