Отчет по лабораторной работе 14

Приобретение практических навыков с именнованными каналами

Желнов Алексей Николаевич

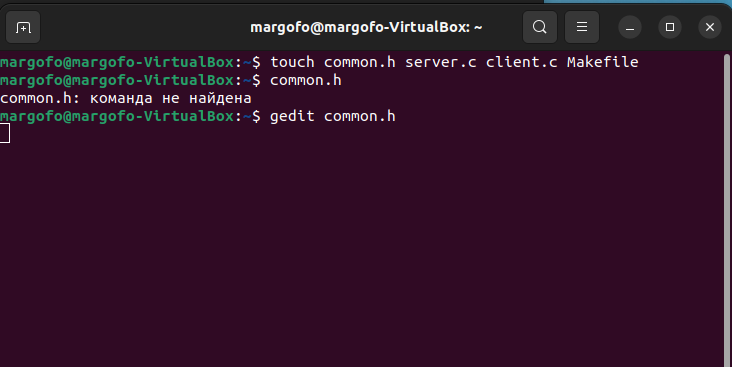
Содержание

# Цель работы

* Приобретение практических навыков работы с именноваными каналами.

# Ход работы

* Для начала я создал необходимые файлы с помощю команды “touch common.h server.c client.c Makefile” и открыл редактор emacs для их редактирования.

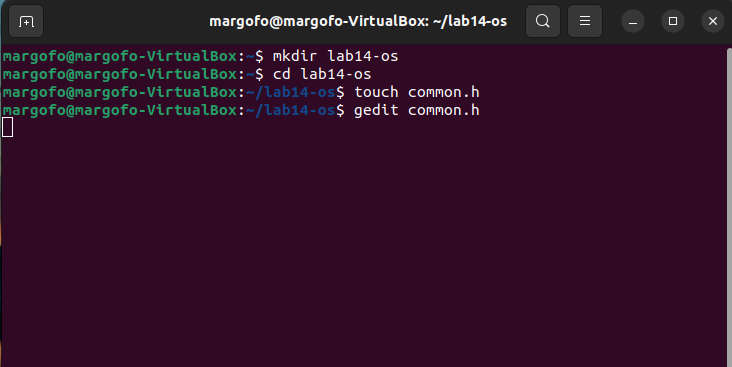


1

2

2

* Далее я изменил коды программ, представленных в тексте лабораторной работы. В файл 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.



3

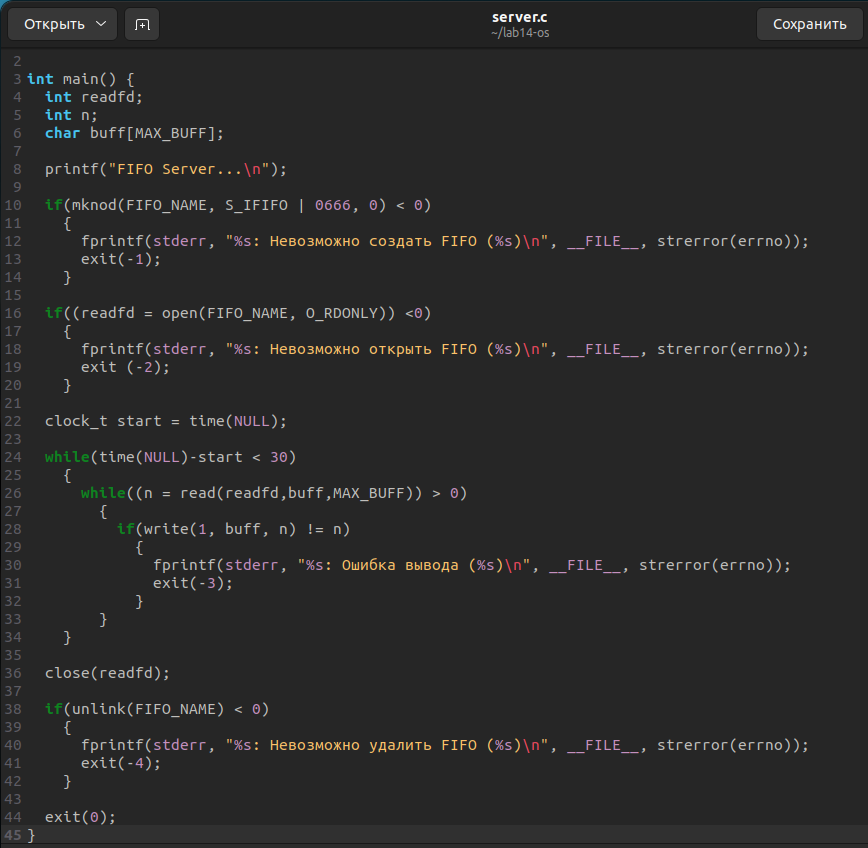


4

* В файл Server.c добавил цикл while для контролья за временем работы сервера. Разница между текущим временем time (NULL) и временем начала работы clok\_t start=time(NULL) не должна превышать 30 секунд

5

5



6

* В файл client.c добавил цикл, который отвечает за количество сообщений о текущем времени, которое получается в результате выполнения команд и команду для остановки работы клиента на 5 секунд

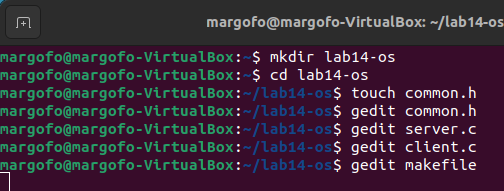
7

7

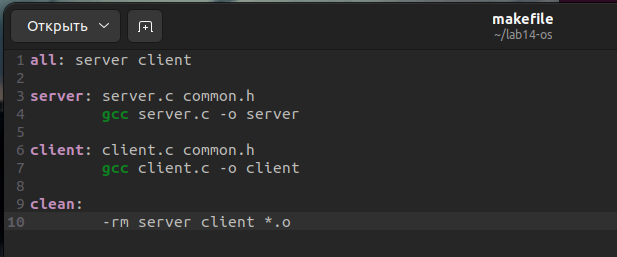


8

* Makefile не изменялся

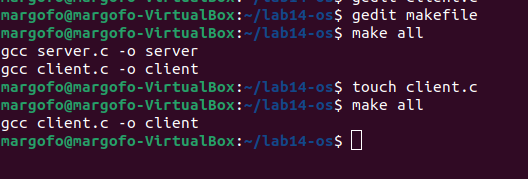


9

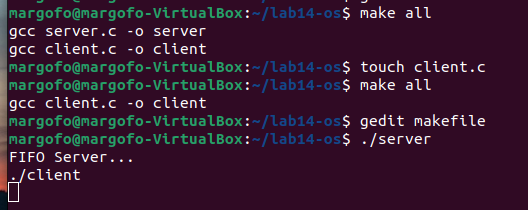


10

* После написания кодов я используя команду “make all”, скомпилировал необходимые файлы



11



12

# Вывод

* В ходе выполнения данной лабораторной работы я приобрел практические навыки работы с именованными наналами

# Контрольные вопросы

* Именнованные каналы отличаются от неименнованных наличием индефикатора канала, который представлен как специяальный файл. Поскольку файл находится на локальной файловой системе, данное IPC используется внутри одной системы
* Чтоб создать неименнованный канал из командной стоки нужно использовать символ | , служащий для обьединения двух и более процессов: процесс\_1 | процесс\_2 | процесс\_3…
* Чтобы создать именнованный канал из командной строки нужно использовать либо команду “mknod” ,либо команду “mkfifo”.
* Неименнованный канал является средством взаимодействия между связанными процессами - родителельским и дочерним. Родительский процесс создает канал при помощи системного вызова: “int pipe”; Массив из двух целых чисел является выходным параметром этого системного вызова. Если вызов выполняется нормально, то этот массив содержит два файловых дескриптора. fd[0] является дескриптором для чтения из канала, fd[1] - де-скриптором для записи в канал. Когда процесс пораждает другой процесс , дескрипторы родительского процесса наследуются дочерним процессом, таким образом, прокладывается трубопровод между двумя процессами. Естественнo, что один из процессов использует канал только для чтения, а другой только для записи. Поэтому если например через канал должны передаваться ланные из родительского процесса в дочерний, сразу после запуска дочерниго процесса закрывает дескриптор канала ждля чтени,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
* Файлы именнованных каналов создаются функцией mkfifo() или функцией mknod: “int mkfifo(const char pathname, mode\_t mode);” где первый параметр путь где будет распологаться FIFO, “mknod”(namfile, IFIFO | 0666. 0),где namefile - имя канала, 0666 - к каналу разрешён доступ на запись и на чтение любому запросившему процессу), “int mknod”(const char pathname, mode\_t mode, dev\_t dev); Функция mkfifo() создаёт 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, либо для чтения
* При чтении меньшего числа байтов, чем находится в канале или FIFO, возвращается требуемое число байтов, остаток сохраняется до последующих чтений. При чтении большего числа байтов, чем находится в канале или FIFO, возвращаетс ядоступное число байтов. Процесс, читающий из канала, должен соответствующим образом обработать ситуацию, когда прочитано меньше, чем заказано.
* Запись числа байтов, меньшего ёмкости 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 прцессом 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 умолчпнию - процесс завершается).
* Количесство процессов, которые могут параллельно присоединяться к любому концу канала, не ограниченою Однако если два и более процесса записывают в канал данные одновременно, каждый процесс за один раз может записать максимум PIPE BUF байтов данных. Предположим, процесс (назовём его А) пытается записать Х байтов данных в канал, в котором имеется место У байтов данных. Если Х больше, чем У, только первые У байтов данных записываются в канал, и процесс блокируется. Запускатеся другой процесс (например, В); в это время в канале появляется свободное пространство. Процесс В записывает данные в канал. Затем, когда выполнение процесса А возобновляется, он записывает оставшиеся Х-У байтов данных в канал. В результате данные в канал записываются поочерёдно двумя процессами. Аналогичным образом, если два тлт болнн процесса одновременно попытаютс япрочитать данные из канала, может случиться так, что каждый из них прочитает только часть необходимых данных.
* Функция write записывает байты cout 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 осуществления операций записи указатель на файл (если он есть) увеличивается на количество действительно записанных байтов. Возвращаемое значение должно быть положительным, но меньше числа count (например, когда размер для записи count байтов выходит зп пределы пространства на диске.) Возвращаемое значение -1 указывает на ошибку: errno устанавливается в одно из следующих значений: EACCES - файл открыт для чтения или закрыт для записи, EBADF - неверный handle-p файла, ENOSPC - на устройстве нет свободного места. Единица в вызове функции write в программе server.c означает идентификатор (дескриптор потока) стандартного потока вывода.
* 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 возникаюь 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 поймёт, как исправить ошибку, прочитав сообщение функции strerror. Возвращённый 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.