Отчёт к лабораторной работе №15

Именованные каналы.

Кекишева Анастасия Дмитриевна

Содержание

# Цель работы

Приобретение практических навыков работы с именованными каналами.

# Задание

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

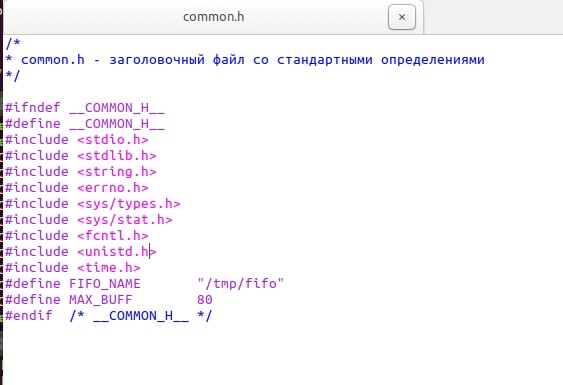
* Работает не 1 клиент, а несколько (например, два).
* Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Используйте функцию sleep() для приостановки работы клиента.
* Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). Используйте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

# Выполнение лабораторной работы

Перед выполнением лабораторной работы я хорошо ознакомилась с теоритическим материалом для её выполнения [Ссылка 1](https://esystem.rudn.ru/mod/resource/view.php?id=719031)

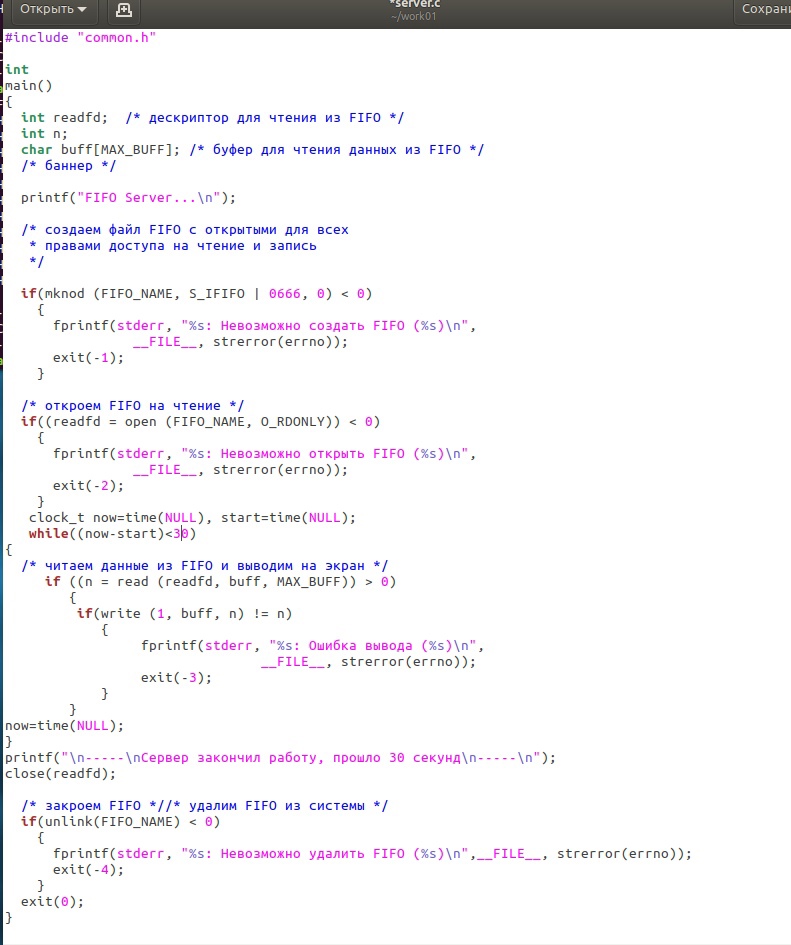
Во-первых, я изучила приведённые в тексте программы server.c и client.c. Взяв данные примеры за образец, написала аналогичные программы, внеся следующие изменения:

* Работает не 1 клиент, а несколько (например, два).
* Клиенты передают текущее время с некоторой периодичностью, я взяла что первый работает 3 сек, а второй 5, для приостановки работы клиента я использовала функцию sleep().



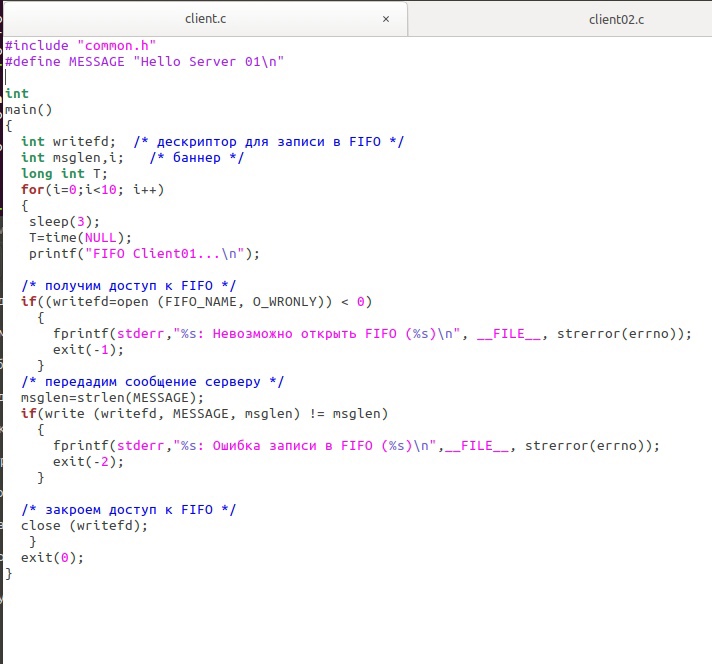
Файл common.h

В файл common.h я добавила две библиотеки time.h и unistd.h (рис. 3.1). Первую для времени, и вторую для функций read, wright, close и т.д.



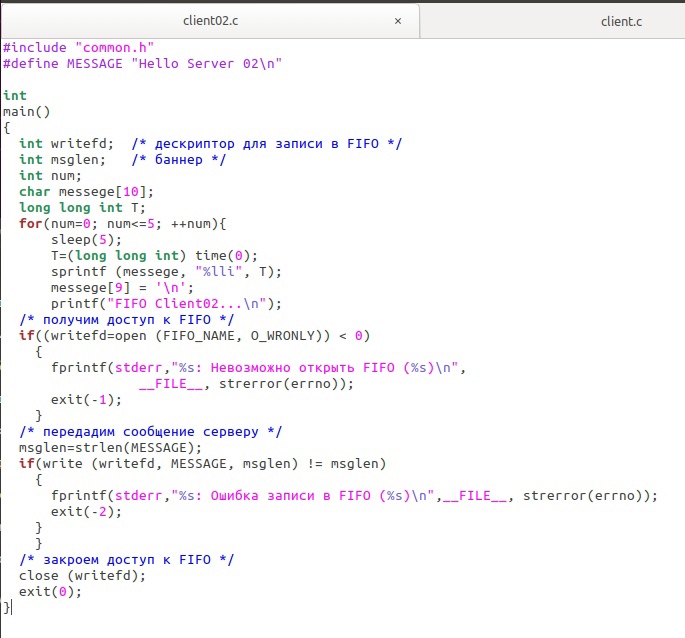
Файл server.c: время работы 30 сек

В файле server.c добавила функцию clock, присваиваю переменным start и now значение time(NULL). Также, создала цикл while установила время работы 30 сек, в конце цикла мы присваиваем значение time(NULL) переменной now. Вывожу на экран сообщение о том, что сервер закончил свою работу и время его работы(рис. 3.2).



Файл client.c

В файле client.c (рис 3.3) я ввела дополнительную переменную T для времени, а также сделала цикл for, в котором я определяю сколько раз в первый клиент будет обработан сервером, а также фунцию sleep(3), где задаю время ожидания 3 сек, и вывод на экран, что это клиент первый. Также передаю сообщение серверу, что это первый клиент (#define MESSAGE “Hello Server 01”).

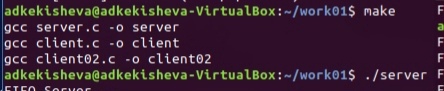


Файл client02.c

В файле client02.c (рис 3.4) я также ввела дополнительную переменную T для времени, массив messege[10], также сделала цикл for, в котором я определяю сколько раз в клиент будет обработан сервером, а также фунцию sleep(5), где задаю время ожидания 5 сек, и вывод на экран, что это второй клиент. Функцией sprintf() произвожу вывод в массив. Также передаю сообщение серверу, что это второй клиент (#define MESSAGE “Hello Server 02”).

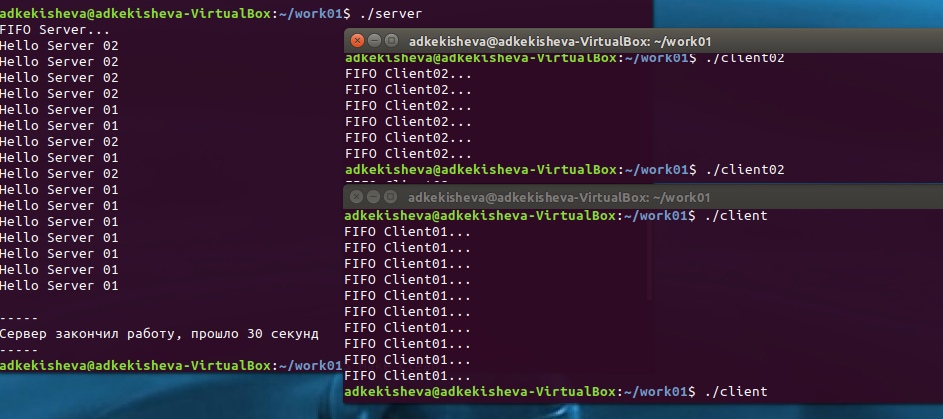


Makefile



Компиляция файлов

Добавила в Makefile команду для компиляции файла client02.c (рис. 3.5). И выполнила компиляцию файлов (рис. 3.6).

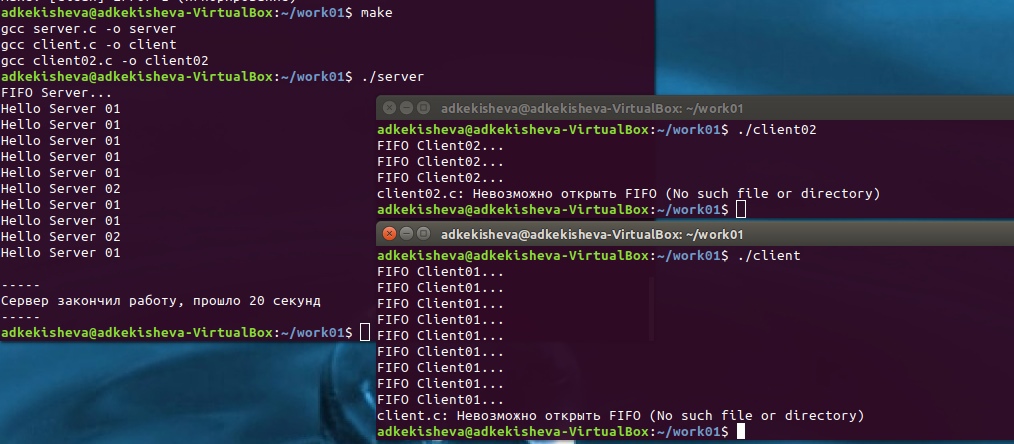


Результат работы программы 1

Далее запустила в одной консоли server, во второй client, в третьей client02. Сервер обработал клиентов необходимое количество раз и по происшествии 30 сек закончил работу (рис. 3.7).



Файл server.c: время работы 20 сек



Результат работы программы 2

Изменила время работы сервера на 20 сек (рис. 3.8). В это раз сервер не успел обработать всех. Если сервер завершит работу, не закрыв канал, для клиентов выйдет сообщение о том, что невозможно открыть FIFO (рис. 3.9).

# Вывод

Я приобрела практические навыки работы с именованными каналами, написала программы, с требующимися изменениями.

# Библиография

1. [Ссылка 1](https://esystem.rudn.ru/mod/resource/view.php?id=719031)

# 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. Создание неименованного канала из командной строки невозможно.
3. Создание именованного канала из командной строки возможно.
4. int read(int pipe\_fd, void \*area, int cnt);

int write(int pipe\_fd, void \*area, int cnt);

Первый аргумент этих вызовов - дескриптор канала, второй - указатель на область памяти, с которой происходит обмен, третий - количество байт. Оба вызова возвращают число переданных байт (или -1 - при ошибке).

1. int mkfifo (const char \*pathname, mode\_t mode) ;

mkfifo(FIFO\_NAME, 0600) ;

Первый параметр — имя файла, идентифицирующего канал, второй параметр маска прав доступа к файлу. Вызов функции mkfifo() создаёт файл канала (с именем, заданным макросом FIFO\_NAME).

1. При чтении меньшего числа байтов, чем находится в канале, возвращается требуемое число байтов, остаток сохраняется для последующих чтений. При чтении большего числа байтов, чем находится в канале или FIFO возвращается доступное число байтов.
2. 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. 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.
3. В общем случае возможна много направленная работа процессов с каналом, т.е. возможна ситуация, когда с одним и тем же каналом взаимодействуют два и более процесса, и каждый из взаимодействующих каналов пишет и читает информацию в канал. Но традиционной схемой организации работы с каналом является однонаправленная организация, когда канал связывает два, в большинстве случаев, или несколько взаимодействующих процесса, каждый из которых может либо читать, либо писать в канал.
4. Write - Функция записывает length байтов из буфера buffer в файл, определенный дескриптором файла fd. Эта операция чисто ‘двоичная’ и без буферизации. Реализуется как непосредственный вызов DOS. С помощью функции write мы посылаем сообщение клиенту или серверу.
5. Строковая функция strerror - функция языков C/C++, транслирующая код ошибки, который обычно хранится в глобальной переменной errno, в сообщение об ошибке, понятном человеку. Ошибки эти возникают при вызове функций стандартных Си-библиотек. 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.