Лабораторная работа №15 по предмету

Операционные системы

Группа НПМбв-01-19

Бондаренко Артем Федорович

Содержание

|  |  |
| --- | --- |
| [Цель работы](#page5) | 5 |
| [Задание](#page6) | 6 |
| [Выполнение лабораторной работы](#page7) | 7 |
| [Выводы](#page14) | 14 |
| [Ответы на контрольные вопросы](#page15) | 15 |

2

Список иллюстраций

1 [Создание директории](#page7) . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

* [Создание файлов командой touch](#page7) . . . . . . . . . . . . . . . . . . . . . 7

|  |  |  |
| --- | --- | --- |
| [3](#page8) | [Код файла common.h](#page8) . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 8 |
| [4](#page8) | [Код файла client.c](#page8) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 8 |
| [5](#page9) | [Код файла server.c](#page9) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . | 9 |

* [Работа исполняемого файла Server в терминале](#page10) . . . . . . . . . . . . . 10

[7 Работа исполняемого файла Client в терминале](#page10) . . . . . . . . . . . . . 10

8 [Изменение файла common.h](#page11) . . . . . . . . . . . . . . . . . . . . . . . . 11

* [Изменение файла client.c](#page11) . . . . . . . . . . . . . . . . . . . . . . . . . . 11

[10 Изменение файла server.c](#page12) . . . . . . . . . . . . . . . . . . . . . . . . . . 12 [11 Демонстрация работы получившегося файла server в терминале](#page13) . . . 13 [12 Демонстрация работы получившегося файла client в терминале](#page13) . . . . 13

3

Список таблиц

4

Цель работы

Приобретение практических навыков работы с именованными каналами.

5

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?

6

Выполнение лабораторной работы

Изучил приведённые в тексте программы server.c, client.c. и сommon.h после чего создал специальную директорию под тестирование этих файлов и назвал её server. (Ссылка: Рис.1)



Рис. 1: Создание директории

Создал аналогичные программам в лабораторной работе файлы посредством терминала с помощью команды touch. (Ссылка: Рис.2)



Рис. 2: Создание файлов командой touch

После чего в каждый из этих файлов скопировал программный код из ла-бораторной работы, внося небольшие корректировки. (Ссылка: Рис.3)(Ссылка: Рис.4)(Ссылка: Рис.5)

7

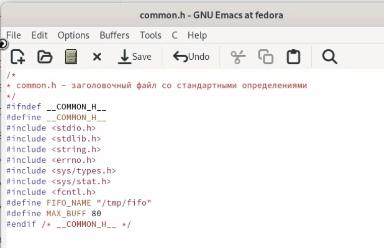


Рис. 3: Код файла common.h

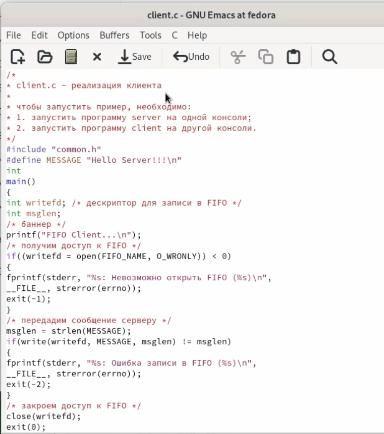


Рис. 4: Код файла client.c

8

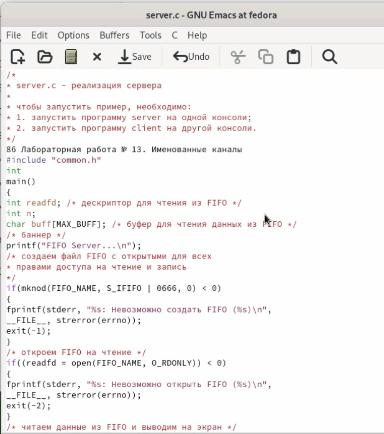


Рис. 5: Код файла server.c

После чего скомпилировал эти файлы в два исполняемых server и client. Про-верил их на работоспособность, открыв каждый в отдельном терминале. (Ссылка: Рис.6)(Ссылка: Рис.7)

9

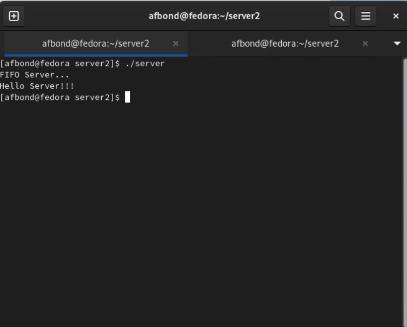


Рис. 6: Работа исполняемого файла Server в терминале

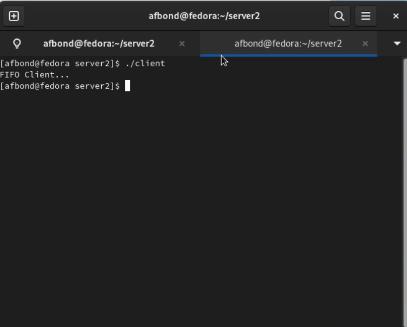


Рис. 7: Работа исполняемого файла Client в терминале

Далее в каждый из файлов внес соответствующие изменения (Ссылка:

10

Рис.8)(Ссылка: Рис.9)(Ссылка: Рис.10)

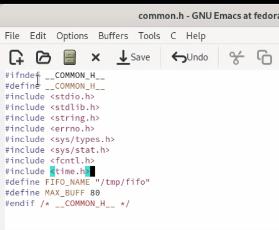


Рис. 8: Изменение файла common.h

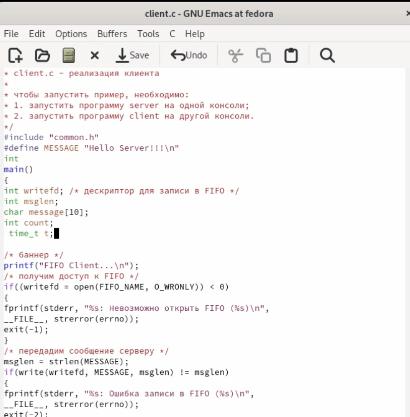


Рис. 9: Изменение файла client.c

11

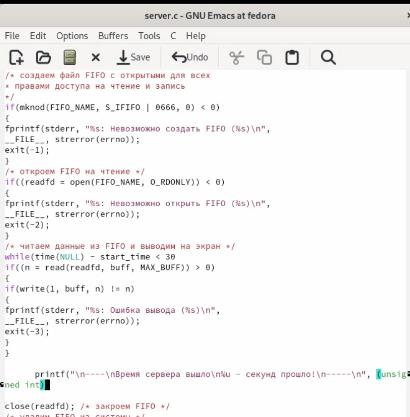


Рис. 10: Изменение файла server.c

По итогу этих изменений получилось следующее:

Работает не 1 клиент, а несколько.

Также теперь каждый последующий клиент присоединяется с определенным интервалом.

Теперь сервер работает не бесконечно, а прекращает работу через 30 секунд и выводит соответствующее сообщение. (Ссылка: Рис.11)(Ссылка: Рис.12)

12



Рис. 11: Демонстрация работы получившегося файла server в терминале

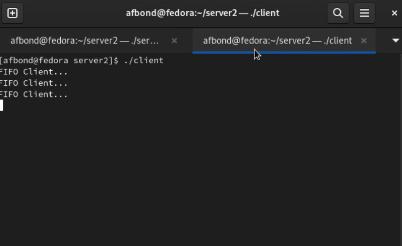


Рис. 12: Демонстрация работы получившегося файла client в терминале

Если сервер завершит работу, не закрыв канал FIFO, то клиент может зависнуть

* ожидании ответа от сервера. Это происходит потому, что канал FIFO остается открытым для чтения в клиенте, и процесс не может получить конец файла (EOF), что означает, что он продолжает ожидать ответа от сервера. Кроме того, если сервер не закроет канал FIFO перед завершением, канал будет оставаться в системе, занимая ресурсы, и его необходимо будет удалить вручную.

13

Выводы

Таким образом, в ходе работы с именованными каналами были приобретены прак-тические навыки создания и использования именованных каналов для организации межпроцессного взаимодействия.

14

Ответы на контрольные вопросы

1. В чем ключевое отличие именованных каналов от неименованных?

Именованные и неименованные каналы в UNIX/Linux используются для обмена данными между процессами.

Ключевое отличие между ними заключается в том, что именованные каналы имеют имена в файловой системе, и к ним можно обратиться из других процессов, как к обычному файлу. В то время как неименованные каналы не имеют имен в файловой системе, и могут использоваться только между процессами, которые были созданы из одного родительского процесса.

Кроме того, именованные каналы могут использоваться для обмена данными между процессами, которые не связаны друг с другом и не являются дочерними процессами родительского процесса. В то время как неименованные каналы могут использоваться только для обмена данными между процессами, которые были созданы из одного родительского процесса.

2. Возможно ли создание неименованного канала из командной строки?

Да, создание неименованного канала (трубы) из командной строки возможно при помощи оператора | (вертикальная черта), который позволяет направлять вывод одной команды на вход другой команды. Например, команда ls -l | grep “file.txt” создаст неименованный канал между командами ls и grep, где вывод команды ls будет передан на вход команды grep, которая отфильтрует строки, содержащие “file.txt”. Однако при использовании таких каналов нет возможности управлять процессами, передающими данные через этот канал, в отличие от именованных

15

каналов, которые могут быть созданы в явном виде и управляемы через файловую систему.

3. Возможно ли создание именованного канала из командной строки?

Да, создание именованного канала (также известного как FIFO) возможно из командной строки с помощью утилиты mkfifo. Синтаксис команды выглядит следу-ющим образом:

mkfifo имя\_канала

Здесь имя\_канала - это имя, которое будет присвоено каналу. После создания именованного канала, его можно использовать для передачи данных между процес-сами.

4. Опишите функцию языка С, создающую неименованный канал.

Функция языка C, используемая для создания неименованного канала - это pipe().

Она определена в заголовочном файле <unistd.h> и имеет следующее объявление:

int pipe(int pipefd[2]);

Аргумент pipefd - это массив целых чисел длиной 2, который будет использоваться для возврата файловых дескрипторов, связанных с каналом. Первый элемент массива (т.е. pipefd[0]) будет использоваться для чтения из канала, а второй элемент (т.е. pipefd[1]) - для записи в канал.

5. Опишите функцию языка С, создающую именованный канал.

Функция языка C для создания именованного канала называется mkfifo() и объ-явлена в заголовочном файле <sys/stat.h>.

Синтаксис функции:

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

16

Параметры функции:

pathname - указатель на строку, содержащую имя именованного канала.

mode - режим доступа, который будет присвоен именованному каналу.

Функция возвращает 0 в случае успеха и -1 в случае ошибки.

1. Что будет в случае прочтения из fifo меньшего числа байтов, чем находится в канале? Большего числа байтов?

При чтении из FIFO будет считано только то количество байтов, которое доступно

* данный момент. Если запрошенное количество байтов меньше, чем находится
* канале, то функция чтения блокируется до тех пор, пока не будет доступно запрошенное количество байтов.

Если запрошенное количество байтов больше, чем находится в канале, то функция чтения блокируется до тех пор, пока не будет доступно запрошенное количество байтов. Если другой процесс записывает данные в FIFO, то функция чтения будет блокироваться, пока не будет доступно запрошенное количество байтов. Если процесс записи закрывает FIFO, то функция чтения возвращает 0, что сигнализирует о том, что канал закрыт

1. Аналогично, что будет в случае записи в fifo меньшего числа байтов, чем позволяет буфер? Большего числа байтов?

Если в FIFO записывается меньшее количество байт, чем позволяет буфер, то они будут записаны в FIFO, а оставшееся место в буфере останется свободным.

Если в FIFO записывается большее количество байт, чем позволяет буфер, то запись будет произведена только до тех пор, пока весь буфер не будет заполнен. После этого процесс, который пишет в FIFO, будет заблокирован до тех пор, пока другой процесс не прочитает данные из FIFO и не освободит место в буфере.

8. Могут ли два и более процессов читать или записывать в канал?

17

Да, два и более процессов могут читать или записывать в канал, если они име-ют соответствующие права доступа к каналу и следуют правильному протоколу чтения/записи в канал.

При чтении из канала каждый процесс получает свою копию данных, записанных

* канал. Таким образом, если два или более процесса читают из канала, они могут получить одни и те же данные в разных экземплярах.

При записи в канал каждый процесс может записывать данные в канал независимо от других процессов. Тем не менее, при записи в канал следует учитывать размер буфера канала, чтобы избежать блокировки процессов, ожидающих чтения данных из канала.

1. Опишите функцию write (тип возвращаемого значения, аргументы и логику работы). Что означает 1 (единица) в вызове этой функции в программе server.c (строка 42)?

Функция write используется для записи данных в файловый дескриптор. Она возвращает количество записанных байтов или -1 в случае ошибки.

Синтаксис функции:

ssize\_t write(int fd, const void \*buf, size\_t count);

где:

fd - файловый дескриптор, куда будут записываться данные;

buf - указатель на буфер с данными, которые нужно записать;

count - количество байтов для записи.

* программе server.c значение 1 (единица) в вызове функции write указывает на файловый дескриптор стандартного вывода (STDOUT\_FILENO), куда будут выводиться данные из канала. В данном случае, функция write используется для вывода прочитанных данных из FIFO на экран.

10. Опишите функцию strerror.

18

strerror() - это функция языка программирования С, которая преобразует код ошибки в строку сообщения об ошибке. Она возвращает указатель на строку, содер-жащую сообщение об ошибке, соответствующее переданному коду ошибки.

19