

Лабораторная работа 15

Дисциплина: Операционные системы

Куликов Максим Игоревич

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	10
5	Контрольные вопросы	11

Список таблиц

Список иллюстраций

3.1	Создание файлов	7
3.2	Скрипты	8
3.3	Компиляция	8
3.4	Проверка работы программы	9
3.5	Проверка работы сервера	9

1 Цель работы

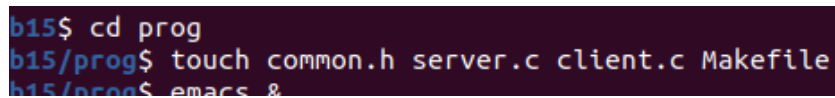
Приобретение практических навыков работы с именованными каналами.

2 Задание

1. Ознакомиться с теоретическим материалом.
2. Выполнить работу.

3 Выполнение лабораторной работы

1. Создаю в новом каталоге 4 скрипта (рис. -fig. 3.1)



```
b15$ cd prog
b15/prog$ touch common.h server.c client.c Makefile
b15/prog$ emacs &
```

Рис. 3.1: Создание файлов

2. Содержимое 4-х скриптов. В файл `common.h` добавил 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. В файл `server.c` добавил 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. В файл `client.c` добавил цикл, который отвечает за количество сообщений о текущем времени (4 сообщения). `Makefile` я оставил без изменения (рис. -fig. 3.2)

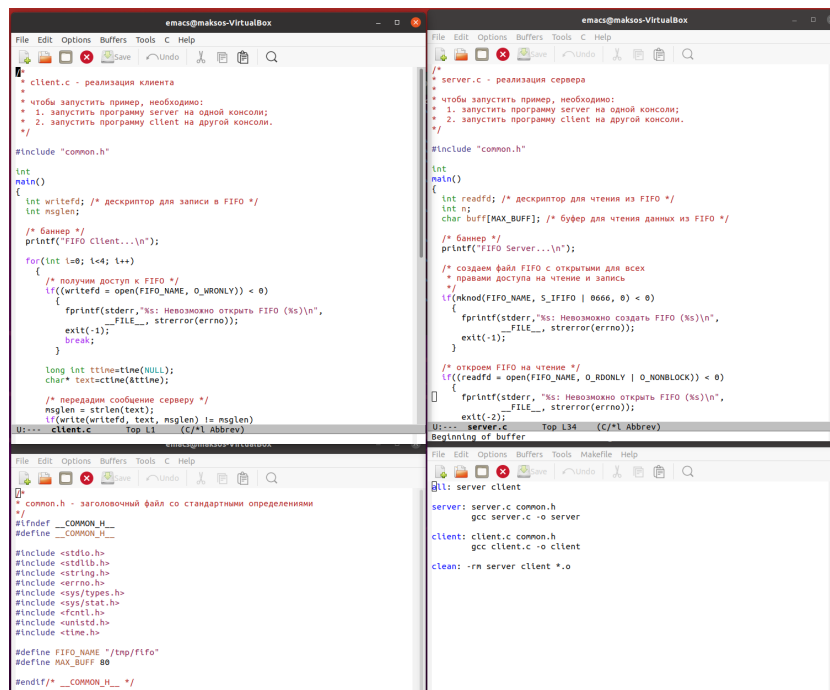


Рис. 3.2: Скрипты

3. Выполняю компиляцию файлов с помощью Makefile. (рис. -fig. 3.3)

```
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab15/prog$ make all
gcc server.c -o server
gcc client.c -o client
```

Рис. 3.3: Компиляция

4. Запускаю сервер в 1 терминале. Запускаю 2 клиента из других терминалов. Они работают верно (каждый клиент выводит в терминале, где запустили сервер, по 4 сообщения) (рис. -fig. 3.4)


```
maksos@maksos-VirtualBox: ~/laby/work/2020-2021/Операционные системы/laboratory/lab15/prog
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab15/prog$ ./server
FIFO Server...
Sat Jun 12 20:33:26 2021
Sat Jun 12 20:33:28 2021
Sat Jun 12 20:33:31 2021
Sat Jun 12 20:33:33 2021
Sat Jun 12 20:33:36 2021
Sat Jun 12 20:33:38 2021
Sat Jun 12 20:33:41 2021
Sat Jun 12 20:33:43 2021
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab15/prog$

maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab15/prog$ ./client
FIFO Client...
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab15/prog$

maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab15/prog$ ./client
FIFO Client...
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab15/prog$
```

Рис. 3.4: Проверка работы программы

5. Проверка работы сервера. Он должен закрываться автоматически через 30 секунд. Работает верно (рис. -fig. 3.5)

```
maksos@maksos-VirtualBox: ~/laby/work/2020-2021/Операционные системы/laboratory/lab15/prog
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab15/prog$ ./server
FIFO Server...
maksos@maksos-VirtualBox:~/laby/work/2020-2021/Операционные системы/laboratory/lab15/prog$
```

Рис. 3.5: Проверка работы сервера

4 Выводы

Приобрёл практические навыки работы с именованными каналами.

5 Контрольные вопросы

1. Именованные каналы отличаются от неименованных наличием идентификатора канала, который представлен как специальный файл (соответственно имя именованного канала — это имя файла).
2. Для создания неименованного канала используется системный вызов `pipe`. Массив из двух целых чисел является выходным параметром этого системного вызова.
3. Вы можете создавать именованные каналы из командной строки и внутри программы. С давних времен программой создания их в командной строке была команда: `mknod - $ mknod имя_файла`, однако команды `mknod` нет в списке команд X/Open, поэтому она включена не во все UNIX-подобные системы. Предпочтительнее применять в командной строке - `$ mkfifo имя_файла`.

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).

5. `int mkfifo (const char *pathname, mode_t mode);` 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`): `mkfifo(FIFO_NAME, 0600);`

6. 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.
7. 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.
8. В общем случае возможна многонаправленная 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.
9. `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.

10. 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.