

Отчёт по лабораторной работе №14

Именованные каналы

НВЕ МАНГЕ ХОСЕ ХЕРСОН МИКО

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Выводы	11
6	Ответы на контрольные вопросы	12
	Список литературы	15

Список иллюстраций

4.1	Файл client.c	8
4.2	Файл client2.c	9
4.3	Файл server.c	9
4.4	Файл Makefile	10
4.5	Вызов файлов на исполнение, их компиляция	10

Список таблиц

1 Цель работы

Приобретение практических навыков работы с именованными каналами.

2 Задание

Изучите приведённые в тексте программы `server.c` и `client.c`. Взяв данные примеры за образец, напишите аналогичные программы, внося 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.

3 Теоретическое введение

В программировании именованный канал или именованный конвейер (англ. `named pipe`) — один из методов межпроцессного взаимодействия, расширение понятия конвейера в Unix и подобных ОС. Именованный канал позволяет различным процессам обмениваться данными, даже если программы, выполняющиеся в этих процессах, изначально не были написаны для взаимодействия с другими программами. Это понятие также существует и в Microsoft Windows, хотя там его семантика существенно отличается. Традиционный канал — «безымянный», потому что существует анонимно и только во время выполнения процесса. Именованный канал — существует в системе и после завершения процесса. Он должен быть «отсоединён» или удалён, когда уже не используется. Процессы обычно подсоединяются к каналу для осуществления взаимодействия между ними.

4 Выполнение лабораторной работы

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

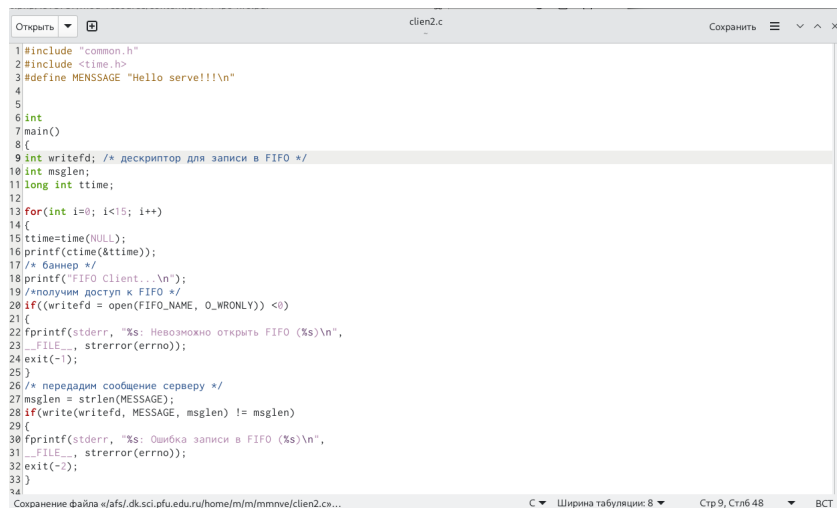
1. Работает не 1 клиент, а несколько (например, два) (рис. 4.1).



```
1#include "common.h"
2
3#define MESSAGE "Hello Server!!\n"
4
5int
6main()
7{
8    int writefd; /* дескриптор для записи в FIFO */
9    int msglen;
10
11    /* баннер */
12    printf("FIFO Client...\n");
13    /* получим доступ к FIFO */
14    if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
15    {
16        fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
17            __FILE__, strerror(errno));
18        exit(-1);
19    }
20
21    /* передадим сообщение серверу */
22    msglen = strlen(MESSAGE);
23    if(write(writefd, MESSAGE, msglen) != msglen)
24    {
25        fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
26            __FILE__, strerror(errno));
27        exit(-2);
28    }
29
30    /* закроем доступ к FIFO */
31    close(writefd);
32
33    exit(0);
34}
```

Рис. 4.1: Файл `client.c`

2. Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Используем функцию `sleep()` для приостановки работы клиента (рис. 4.2).



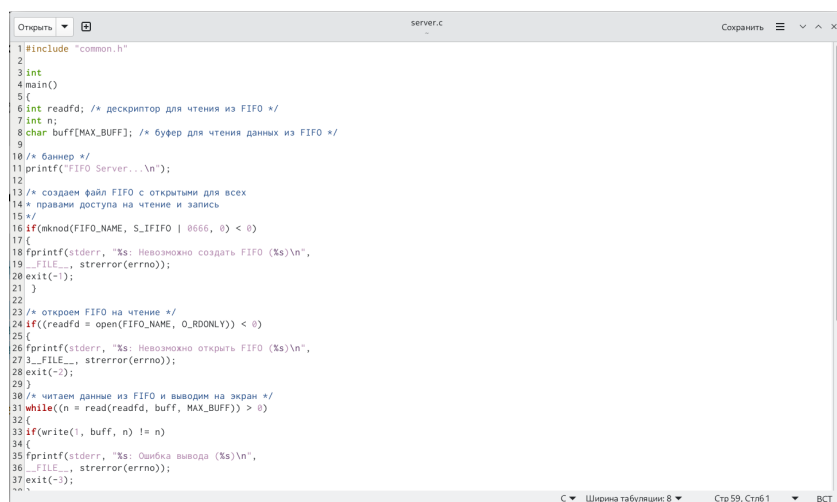
```

1#include "common.h"
2#include <time.h>
3#define MESSAGE "Hello server!!!\n"
4
5
6int
7main()
8{
9    int writefd; /* дескриптор для записи в FIFO */
10   int msglen;
11   long int ttime;
12   for(int i=0; i<10; i++)
13   {
14       ttime = time(NULL);
15       printf("time: %ld\n", ttime);
16       /* баннер */
17       printf("FIFO Client...\n");
18       /*получим доступ к FIFO */
19       if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
20       {
21           printf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
22                 _FILE_, strerror(errno));
23           exit(-1);
24       }
25       /* передаем сообщение серверу */
26       msglen = strlen(MESSAGE);
27       if(write(writefd, MESSAGE, msglen) != msglen)
28       {
29           printf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
30                 _FILE_, strerror(errno));
31           exit(-1);
32       }
33   }
34 }

```

Рис. 4.2: Файл client2.c

- Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). Используем функцию `clock()` для определения времени работы сервера (рис. 4.3).



```

1#include "common.h"
2
3int
4main()
5{
6    int readfd; /* дескриптор для чтения из FIFO */
7    int n;
8    char buff[MAX_BUFF]; /* буфер для чтения данных из FIFO */
9
10   /* баннер */
11   printf("FIFO Server...\n");
12
13   /* создаем файл FIFO с открытыми для всех
14   * правами доступа на чтение и запись
15   */
16   if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
17   {
18       printf(stderr, "%s: Невозможно создать FIFO (%s)\n",
19             _FILE_, strerror(errno));
20       exit(-1);
21   }
22   /* открываем FIFO на чтение */
23   if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
24   {
25       printf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
26             _FILE_, strerror(errno));
27       exit(-1);
28   }
29   /* читаем данные из FIFO и выводим на экран */
30   while((n = read(readfd, buff, MAX_BUFF)) > 0)
31   {
32       if(write(1, buff, n) != n)
33       {
34           printf(stderr, "%s: Ошибка вывода (%s)\n",
35                 _FILE_, strerror(errno));
36           exit(-1);
37       }
38   }
39 }

```

Рис. 4.3: Файл server.c

- Пропишем Makefile (рис. 4.4).

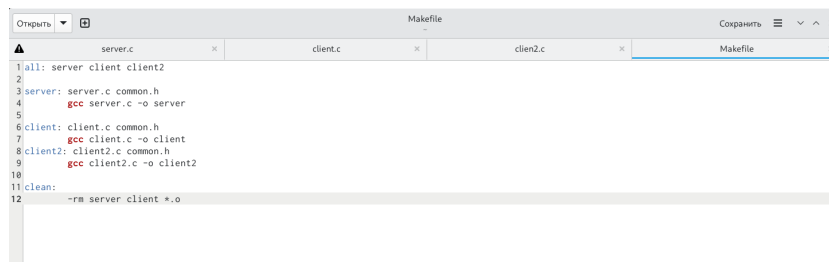


Рис. 4.4: Файл Makefile

5. Вызовем файды на исполнение (рис. 4.5).

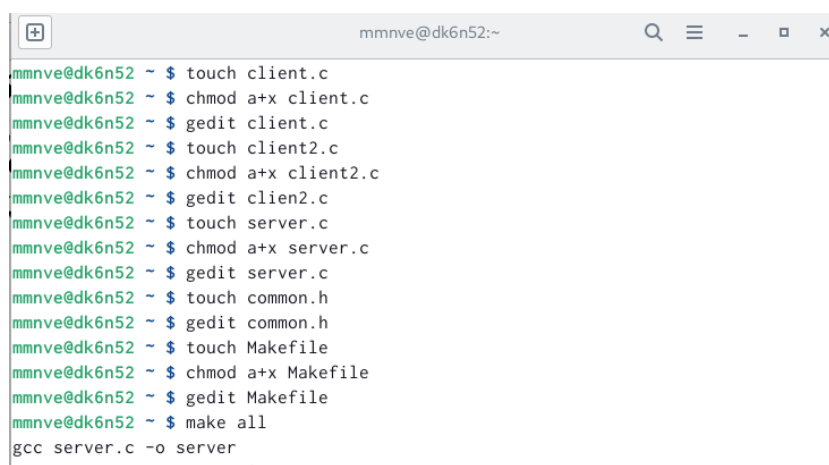


Рис. 4.5: Вызов файлов на исполнение, их компиляция

5 Выводы

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

6 Ответы на 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.

Список литературы