

Отчёт по лабораторной работе №14

Именованные каналы

Сильвен Макс Грегор Филс , НКАбд-03-22

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.

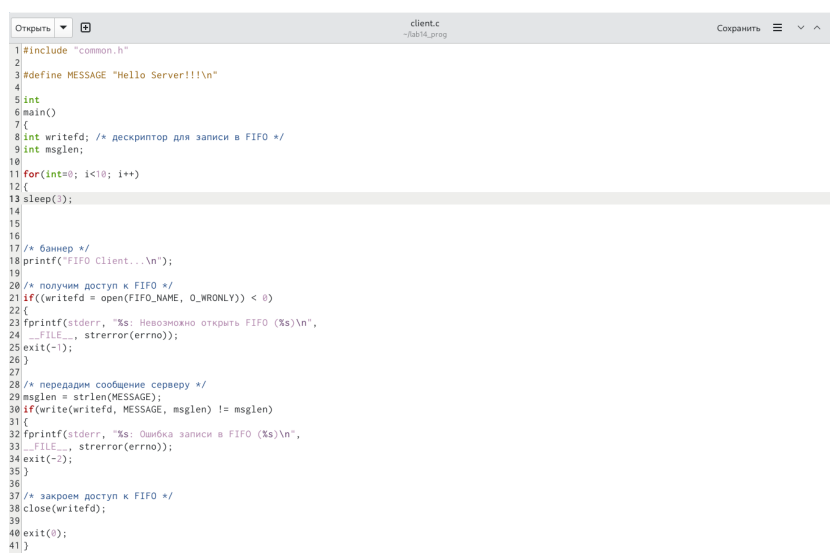
3 Теоретическое введение

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
5 int
6 main()
7 {
8     int writefd; /* дескриптор для записи в FIFO */
9     int msglen;
10
11     for(int=0; i<10; i++)
12     {
13         sleep(3);
14     }
15
16
17     /* баннер */
18     printf("FIFO Client...\n");
19
20     /* получим доступ к FIFO */
21     if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
22     {
23         printf(stderr, "%s: Невозможно открыть FIFO (%s)\n",
24             __FILE__, strerror(errno));
25         exit(-1);
26     }
27
28     /* передадим сообщение серверу */
29     msglen = strlen(MESSAGE);
30     if(write(writefd, MESSAGE, msglen) != msglen)
31     {
32         printf(stderr, "%s: Ошибка записи в FIFO (%s)\n",
33             __FILE__, strerror(errno));
34         exit(-2);
35     }
36
37     /* закроем доступ к FIFO */
38     close(writefd);
39
40     exit(0);
41 }
```

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).


```

Открыть client2.c
1 #include "common.h"
2 #include <time.h>
3 #define MESSAGE "Hello Server!!\n"
4
5 int
6 main()
7 {
8     int writefd; /* дескриптор для записи в FIFO */
9     int msglen;
10    long int ttime;
11
12    for(int i=0; i<15; i++)
13    {
14        time_t time(WALL);
15        printf("ctime(&ttime);\n");
16        /* баннер */
17        printf("FIFO Client...\n");
18
19        /* получим доступ к FIFO */
20        if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
21        {
22            printf(stderr, "Невозможно открыть FIFO (%s)\n",
23                _FILE_, strerror(errno));
24            exit(-1);
25        }
26
27        /* передадим сообщение серверу */
28        msglen = strlen(MESSAGE);
29        if(write(writefd, MESSAGE, msglen) != msglen)
30        {
31            printf(stderr, "Невозможно записать в FIFO (%s)\n",
32                _FILE_, strerror(errno));
33            exit(-2);
34        }
35        sleep(4);
36    }
37
38    /* закроем доступ к FIFO */
39    close(writefd);
40
41    exit(0);
42 }

```

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).

```

Открыть server.c
1 #include "common.h"
2
3 int
4 main()
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, "Невозможно создать FIFO (%s)\n",
19            _FILE_, strerror(errno));
20        exit(-1);
21    }
22
23    /* откроем FIFO на чтение */
24    if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
25    {
26        printf(stderr, "Невозможно открыть FIFO (%s)\n",
27            _FILE_, strerror(errno));
28        exit(-2);
29    }
30
31    /* читаем данные из FIFO и выводим на экран */
32    while((n = read(readfd, buff, MAX_BUFF)) > 0)
33    {
34        if(write(1, buff, n) != n)
35        {
36            printf(stderr, "Невозможно вывести (%s)\n",
37                _FILE_, strerror(errno));
38            exit(-3);
39        }
40        close(readfd); /* закроем FIFO */
41    }
42 }

```

Рис. 4.3: Файл server.c

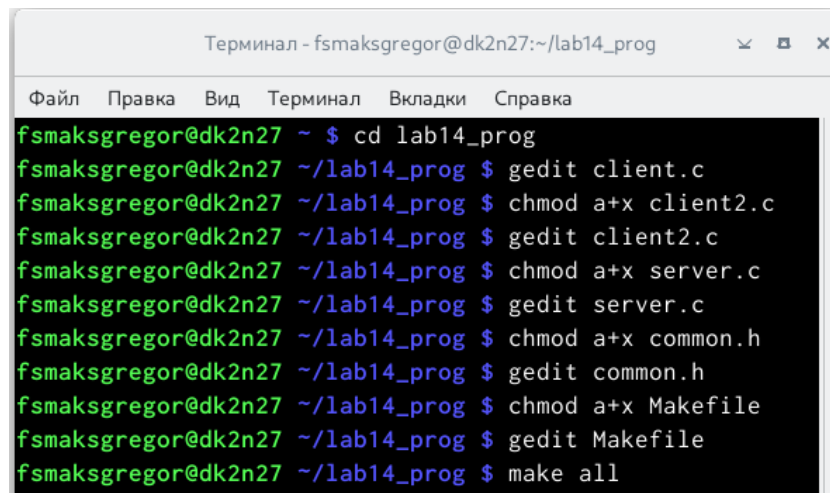
4. Пропишем Makefile (рис. 4.4).



```
1 all: server client client2
2
3 server: server.c common.h
4 gcc server.c -o server
5
6 client: client.c common.h
7 gcc client.c -o client
8
9 client2: client2.c common.h
10 gcc client2.c -o client2
11
12 clean:
13 -rm server client *.o
```

Рис. 4.4: Файл Makefile

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



```
Терминал - fsmaksgregor@dk2n27:~/lab14_prog
Файл  Правка  Вид  Терминал  Вкладки  Справка
fsmaksgregor@dk2n27 ~ $ cd lab14_prog
fsmaksgregor@dk2n27 ~/lab14_prog $ gedit client.c
fsmaksgregor@dk2n27 ~/lab14_prog $ chmod a+x client2.c
fsmaksgregor@dk2n27 ~/lab14_prog $ gedit client2.c
fsmaksgregor@dk2n27 ~/lab14_prog $ chmod a+x server.c
fsmaksgregor@dk2n27 ~/lab14_prog $ gedit server.c
fsmaksgregor@dk2n27 ~/lab14_prog $ chmod a+x common.h
fsmaksgregor@dk2n27 ~/lab14_prog $ gedit common.h
fsmaksgregor@dk2n27 ~/lab14_prog $ chmod a+x Makefile
fsmaksgregor@dk2n27 ~/lab14_prog $ gedit Makefile
fsmaksgregor@dk2n27 ~/lab14_prog $ make all
```

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