

Шаблон отчёта по лабораторной работе

1022204143

Надиа Эззакат

Содержание

| | | |
|----------|---------------------------------------|-----------|
| 1 | Цель работы | 5 |
| 2 | Задание | 6 |
| 3 | Выполнение лабораторной работы | 7 |
| 4 | Выводы | 14 |
| 5 | Контрольные вопросы | 15 |

List of Tables

List of Figures

| | | |
|-----|--------------------|----|
| 3.1 | server | 8 |
| 3.2 | client1 | 9 |
| 3.3 | client2 | 10 |
| 3.4 | common.h | 11 |
| 3.5 | Makefile | 12 |
| 3.6 | ВЫВОД | 13 |
| 3.7 | ВЫВОД | 13 |

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. Что будет в случае, если сервер завершит работу, не закрыв канал?

3 Выполнение лабораторной работы

Изучил приведённые в тексте программы `server.c` и `client.c`.

1. Работала не 1 клиент, а несколько (два).

```

1 #include "common.h"
2 int main()
3 {
4     int readfd;
5     int n;
6     char buff[MAX_BUFF];
7     printf("FIFO Server...\n");
8     if(mknod(FIFO_NAME, S_IFIFO | 0666, 0) < 0)
9     {
10         fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n", __FILE__,
11             strerror(errno));
12         exit(-1);
13     }
14     if((readfd = open(FIFO_NAME, O_RDONLY)) < 0)
15     {
16         fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__,
17             strerror(errno));
18         exit(-2);
19     }
20     clock_t now=time(NULL), start=time(NULL);
21     while(now-start<30){
22         while((n = read(readfd, buff, MAX_BUFF)) > 0)
23         {
24             if(write(1, buff, n) != n)
25             {
26                 fprintf(stderr, "%s: Ошибка вывода (%s)\n", __FILE__,
27                     strerror(errno));
28                 exit(-3);
29             }
30         }
31         close(readfd);
32         if(unlink(FIFO_NAME) < 0)
33         {
34             fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n",
35                 __FILE__, strerror(errno));
36             exit(-4);
37         }
38     }
39     exit(0);
40 }

```

Figure 3.1: server


```

1 #include "common.h"
2 #define MESSAGE "Hello Server!!!\n"
3
4 int main() {
5     int writefd;
6     int msglen;
7     printf("FIFO Client...\n");
8     if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
9     {
10         fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__,
strerror(errno));
11         exit(-1);
12     }
13     msglen = strlen(MESSAGE);
14     if(write(writefd, MESSAGE, msglen) != msglen)
15     {
16         fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n", __FILE__,
strerror(errno));
17         exit(-2);
18     }
19     close(writefd);
20     exit(0);
21 }

```

Figure 3.2: client1

```

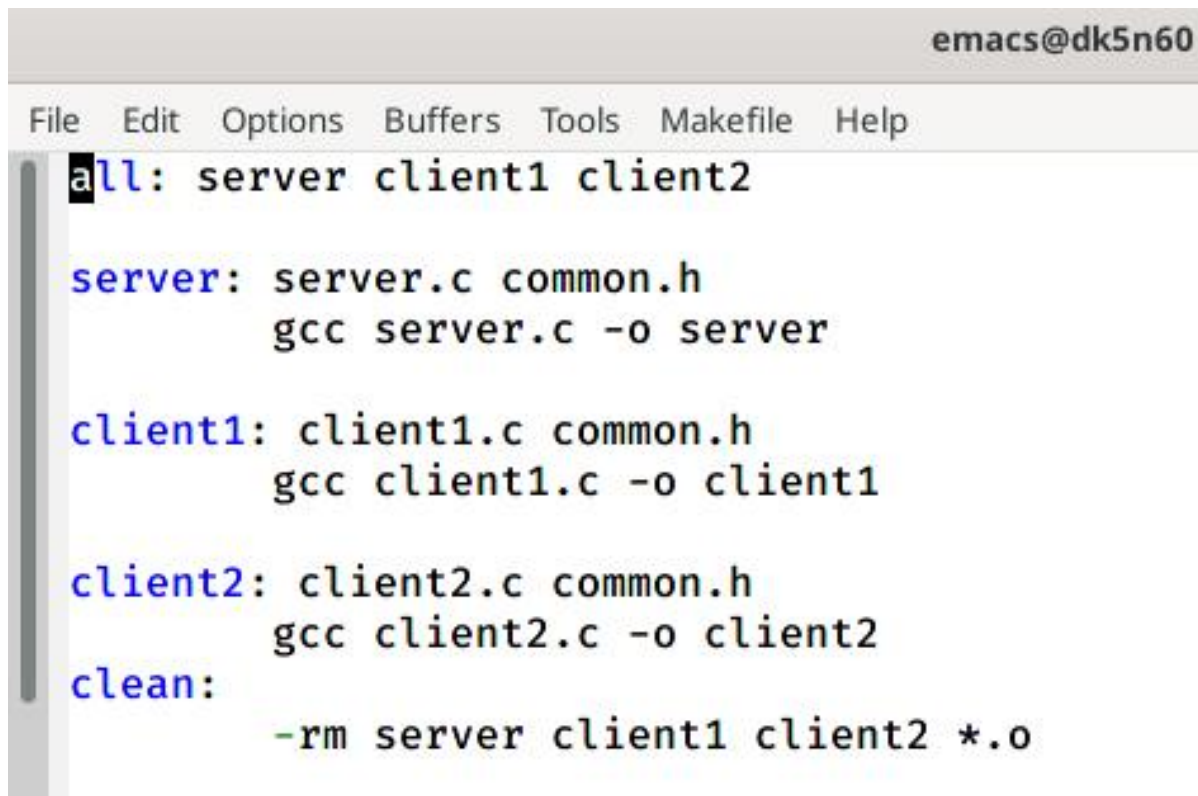
1 #include "common.h"
2 #define MESSAGE "Hello Server!!!\n"
3
4 int main() {
5     int writefd;
6     int msglen;
7     char message[10];
8     int count;
9     long long int T;
10    for(count=0; count<=5; ++count){
11        sleep(5);
12        T=(long long int) time(0);
13        sprintf(message, "%lli", T);
14        message[9] = '\n';
15        printf("FIFO Client...\n");
16        if((writefd = open(FIFO_NAME, O_WRONLY)) < 0)
17        {
18            fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n", __FILE__,
19                strerror(errno));
20            exit(-1);
21        }
22        msglen = strlen(MESSAGE);
23        if(write(writefd, MESSAGE, msglen) != msglen)
24        {
25            fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n", __FILE__,
26                strerror(errno));
27            exit(-2);
28        }
29        close(writefd);
30    }
31    exit(0);
32 }

```

Figure 3.3: client2

```
1 #ifndef FIFO_H
2 #define FIFO_H
3
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <string.h>
7 #include <errno.h>
8 #include <sys/types.h>
9 #include <sys/stat.h>
10 #include <fcntl.h>
11 #include <time.h>
12 #include <unistd.h>
13
14 #define FIFO_NAME      "/tmp/fifo"
15 #define MAX_BUFF      80
16
17 #endif /* __COMMON_H__ */
```

Figure 3.4: common.h



```
emacs@dk5n60
File Edit Options Buffers Tools Makefile Help
all: server client1 client2

server: server.c common.h
    gcc server.c -o server

client1: client1.c common.h
    gcc client1.c -o client1

client2: client2.c common.h
    gcc client2.c -o client2

clean:
    -rm server client1 client2 *.o
```

Figure 3.5: Makefile

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

```
nehzzakat@dk5n60 ~/lab15 $ ./server
FIFO Server...
Hello Server!!!
nehzzakat@dk5n60 ~/lab15 $ ./server
FIFO Server...
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
Hello Server!!!
nehzzakat@dk5n60 ~/lab15 $
```

Figure 3.6: вывод

```
nehzzakat@dk5n60 ~/lab15 $ ./client1
FIFO Client...
nehzzakat@dk5n60 ~/lab15 $ ./client2
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
FIFO Client...
nehzzakat@dk5n60 ~/lab15 $
```

Figure 3.7: вывод

3. В случае, если сервер завершит работу, не закрыв канал, файл FIFO не удалится, поэтому его в следующий раз создать будет нельзя и вылезет ошибка, следовательно, работать ничего не будет.

4 Выводы

В результате работы , я приобрел 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, гарантировано 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.