Лабораторная работа №15

Именованные каналы

Ким Михаил Алексеевич

Содержание

# 1 Цель работы

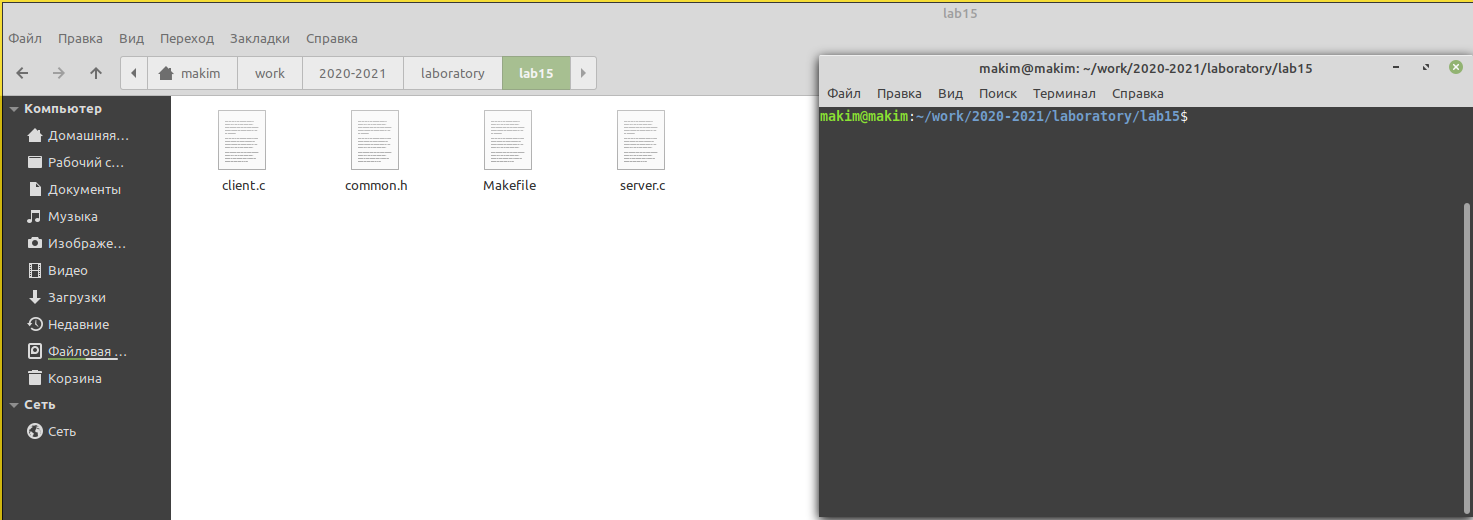
Приобретение практических навыков работы с именованными каналами.

# 2 Выполнение лабораторной работы.

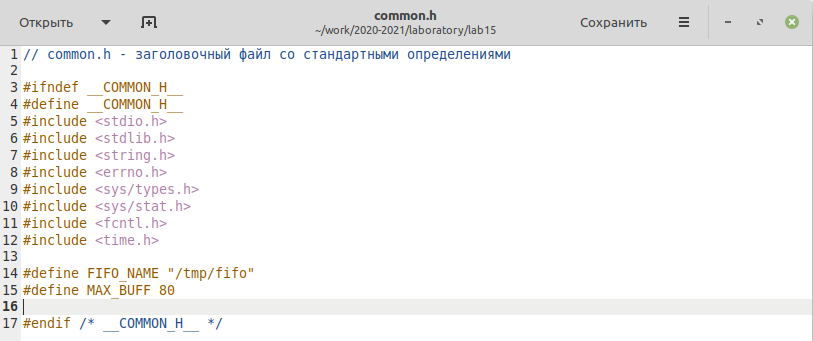
1. В домашнем каталоге создаём подкаталог ~/work. (рис. 2.1)

* mkdir work

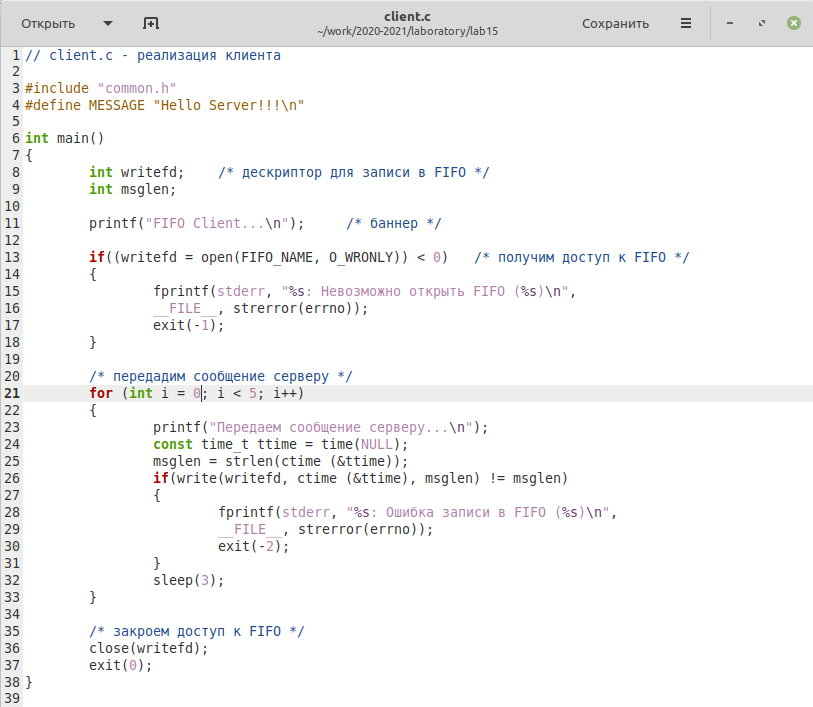
1. Создаём в нём файлы: common.h, server.c, client.c, Makefile. (рис. 2.1)

* cd work  
  touch common.h server.c client.c Makefile
* 
* Подготовка рабочей среды

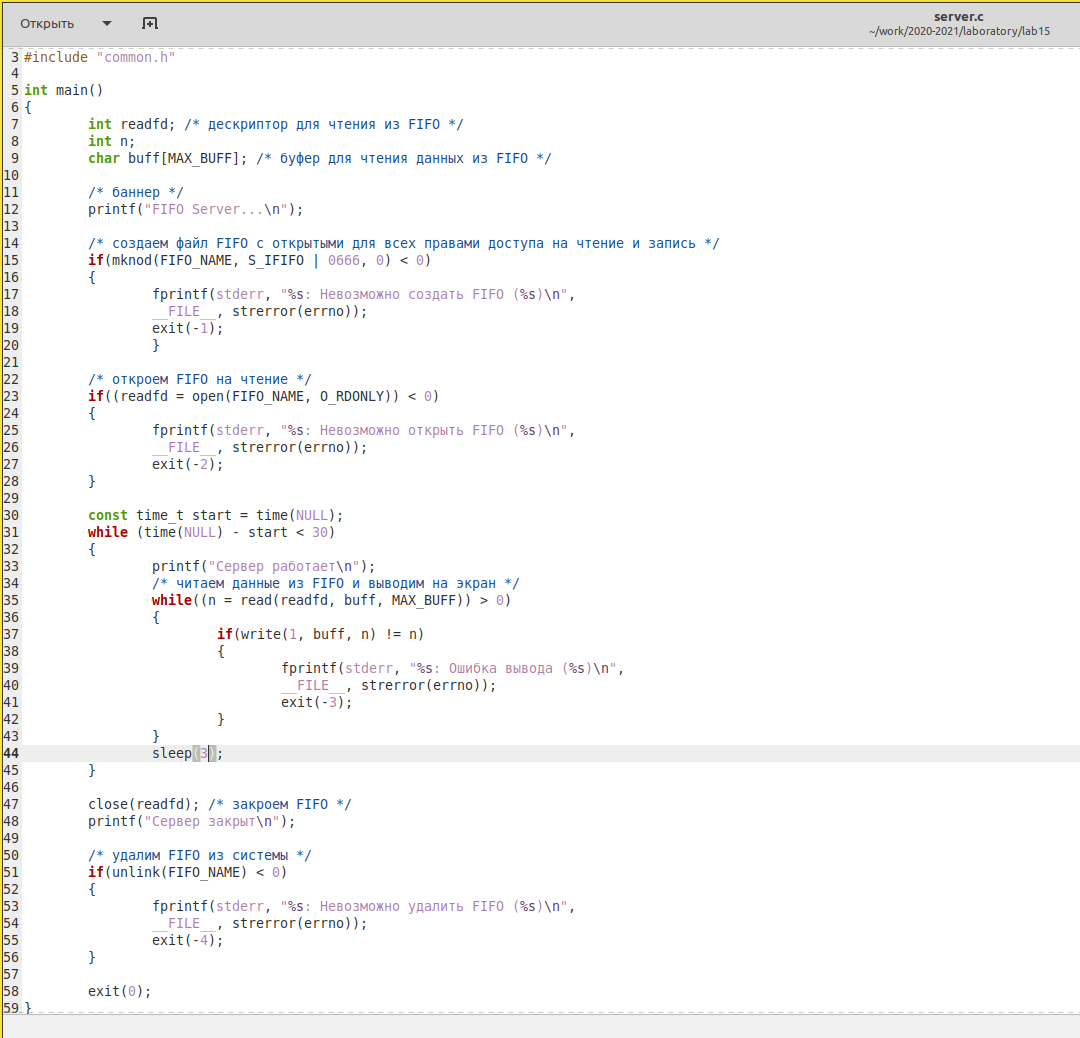
1. Нам необходимо написать программу на основе кода из примера. Программа должна работать с несколькими клиентами. Клиенты передают текущее время с некоторой периодичностью (например, раз в пять секунд). Сервер работает не бесконечно, а прекращает работу через некоторое время (например, 30 сек). Напишем заголовочный файл. Кроме библиотек из примера, добавляем библиотеку для работы со временем. (рис. 2.2)

* // common.h - заголовочный файл со стандартными определениями\*  
    
  #ifndef \_\_COMMON\_H\_\_  
  #define \_\_COMMON\_H\_\_  
    
  #include <stdio.h>  
  #include <stdlib.h>  
  #include <string.h>  
  #include <errno.h>  
  #include <sys/types.h>  
  #include <sys/stat.h>  
  #include <fcntl.h>  
    
  // добавляем библиотеку для работы со временем  
  #include <time.h>   
    
  #define FIFO\_NAME "/tmp/fifo"  
  #define MAX\_BUFF 80  
    
  #endif   
  /\* \_\_COMMON\_H\_\_ \*/
* 
* Исходный код common.h

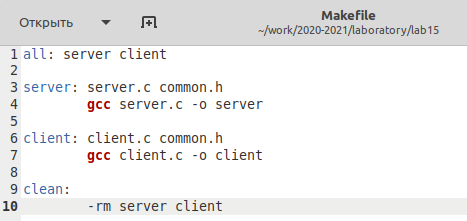
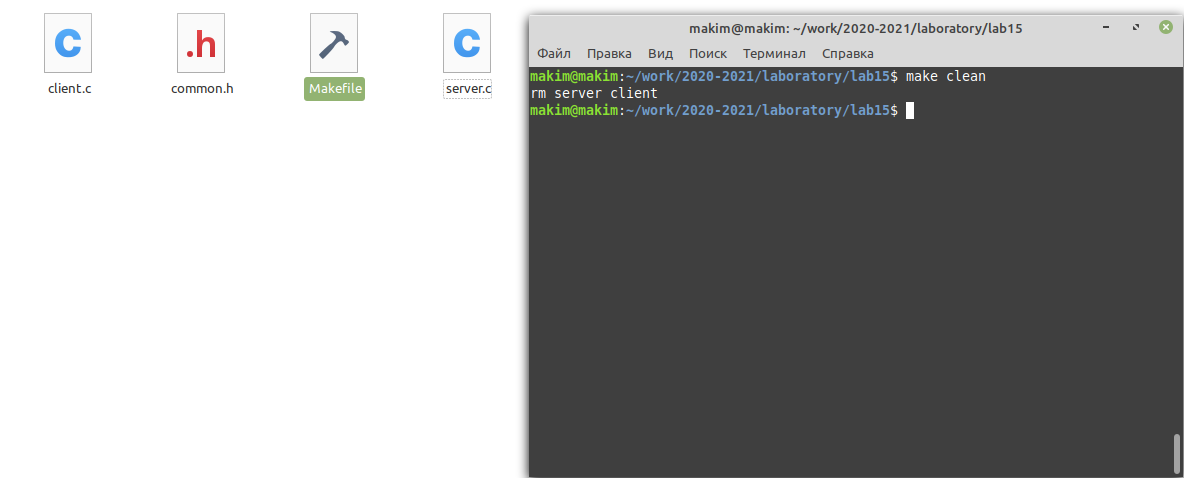
1. Пишем файл client.c. В коде зацикливаем запись в файл FIFO. В каждом цикле записываем в переменную время, переводим эту переменную в строку и отправляем на сервер. (рис. 2.3)

* // client.c - реализация клиента  
    
  #include "common.h"  
  #define MESSAGE "Hello Server!!!\n"  
    
  int main()  
  {  
   int writefd; /\* дескриптор для записи в FIFO \*/  
   int msglen;  
    
   printf("FIFO Client...\n"); /\* баннер \*/  
    
   if((writefd = open(FIFO\_NAME, O\_WRONLY)) < 0) /\* получим доступ к FIFO \*/  
   {  
   fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",  
   \_\_FILE\_\_, strerror(errno));  
   exit(-1);  
   }  
    
   /\* передадим сообщение серверу \*/  
   // Цикл for для передачи некольких сообщений  
   for (int i = 0; i < 5; i++)  
   {  
   printf("Передаем сообщение серверу...\n");  
    
   // переменная для хранения текущего времени  
   const time\_t ttime = time(NULL);  
    
   // сохраняем в msglen длинну времени, приводя его к адекватному виду  
   msglen = strlen(ctime (&ttime));  
   if(write(writefd, ctime (&ttime), msglen) != msglen)  
   {  
   fprintf(stderr, "%s: Ошибка записи в FIFO (%s)\n",  
   \_\_FILE\_\_, strerror(errno));  
   exit(-2);  
   }  
   sleep(3);  
   }  
    
   /\* закроем доступ к FIFO \*/  
   close(writefd);  
   exit(0);  
  }
* 
* Исходный код client.c

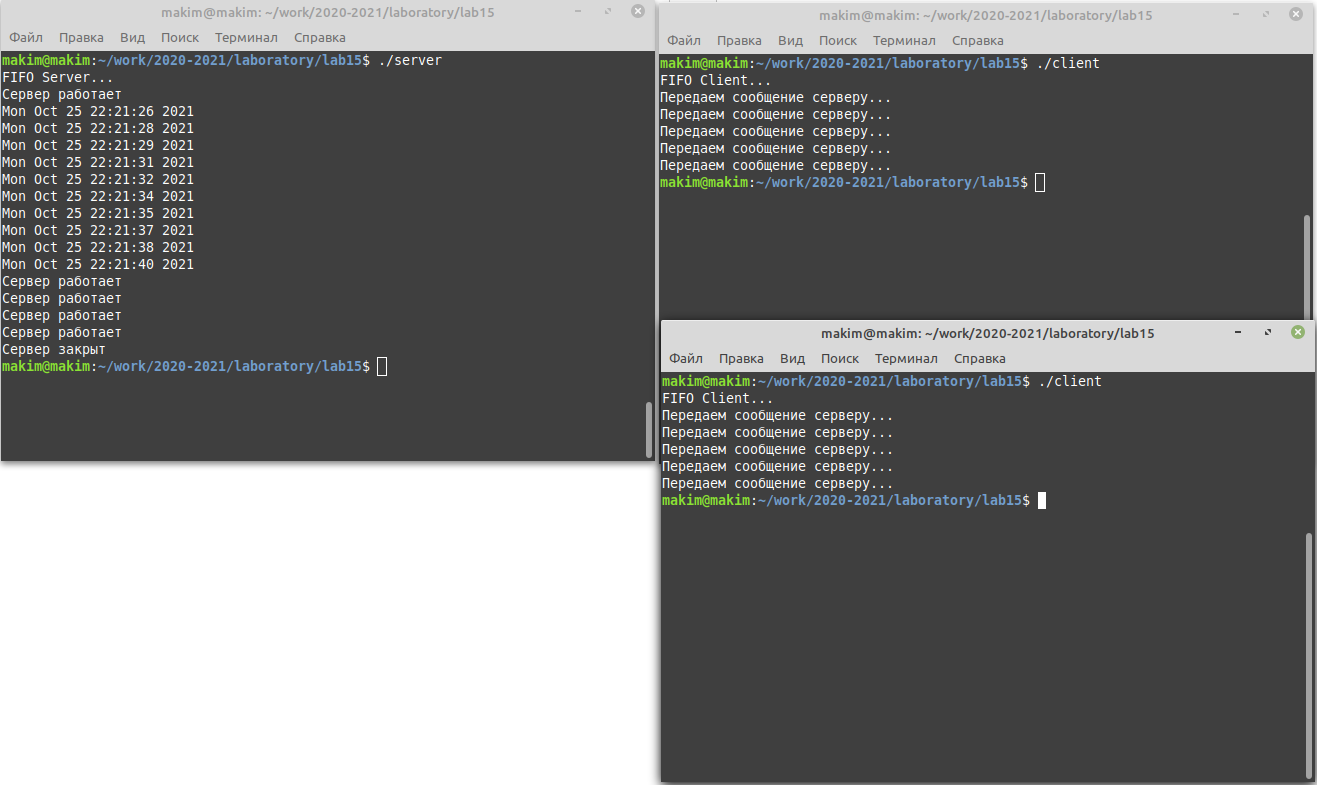
1. Реализуем файл server.c. Создаем переменную, которая хранит время начала запуска программы. Производим чтение файла FIFO, пока не истекли 30 секунд. (рис. 2.4)

* // server.c  
    
  #include "common.h"  
    
  int main()  
  {  
   int readfd; /\* дескриптор для чтения из FIFO \*/  
   int n;  
   char buff[MAX\_BUFF]; /\* буфер для чтения данных из FIFO \*/  
    
   /\* баннер \*/  
   printf("FIFO Server...\n");  
    
   /\* создаем файл FIFO с открытыми для всех правами доступа на чтение и запись \*/  
   if(mknod(FIFO\_NAME, S\_IFIFO | 0666, 0) < 0)  
   {  
   fprintf(stderr, "%s: Невозможно создать FIFO (%s)\n",  
   \_\_FILE\_\_, strerror(errno));  
   exit(-1);  
   }  
    
   /\* откроем FIFO на чтение \*/  
   if((readfd = open(FIFO\_NAME, O\_RDONLY)) < 0)  
   {  
   fprintf(stderr, "%s: Невозможно открыть FIFO (%s)\n",  
   \_\_FILE\_\_, strerror(errno));  
   exit(-2);  
   }  
    
   // создаём константу, где будет храниться время начала работы программы  
   const time\_t start = time(NULL);  
    
   // пока не прошло 30 секунд, сервер будет "принимать сообщения"  
   while (time(NULL) - start < 30)  
   {  
   printf("Сервер работает\n");  
   /\* читаем данные из FIFO и выводим на экран \*/  
   while((n = read(readfd, buff, MAX\_BUFF)) > 0)  
   {  
   if(write(1, buff, n) != n)  
   {  
   fprintf(stderr, "%s: Ошибка вывода (%s)\n",  
   \_\_FILE\_\_, strerror(errno));  
   exit(-3);  
   }  
   }  
   sleep(3);  
   }  
    
   close(readfd); /\* закроем FIFO \*/  
   printf("Сервер закрыт\n");  
    
   /\* удалим FIFO из системы \*/  
   if(unlink(FIFO\_NAME) < 0)  
   {  
   fprintf(stderr, "%s: Невозможно удалить FIFO (%s)\n",  
   \_\_FILE\_\_, strerror(errno));  
   exit(-4);  
   }  
    
   exit(0);  
  }
* 
* Исходный код server.c

1. Создаём Makefile и компилируем программу. (рис. 2.5, 2.6)

* all: server client  
    
  server: server.c common.h  
   gcc server.c -o server  
    
  client: client.c common.h  
   gcc client.c -o client  
    
  clean:  
   -rm server client
* 
* Makefile
* make
* 
* Использование Makefile

1. Наблюдаем процесс работы. (рис. 2.7)

* 
* Результат (1)

# 3 Выводы

Мы приобрели практические навыки работы с именованными каналами. Реализовали программу для обмена сообщениями.

# 4 Термины

* Сообщение - последовательность байтов, передаваемая от одного процесса другому.
* Named pipes - механизм именованных каналов для передачи данных между неродственными процессами.
* FIFO - принцип передачи данных: First In First Out (первым записан — первым прочитан).
* GCC (GNU Compiler Collection) - этото набор компиляторов для разного рода языков программирования (С, C++, Java, Фортран и др.).
* Утилита make позволяет автоматизировать процесс преобразования файлов программы из одной формы в другую, отслеживает взаимосвязи между файлами.
* Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера.
* Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным.
* Каталог, он же директория, (от англисйкого Directory) – это объект в ФС (файловой системе), необходимый для того, чтобы упросить работу с файлами.
* Домашний каталог - каталог, предназначенный для хранения собственных данных пользователя Linux. Как правило, является текущим непосредственно после регистрации пользователя в системе.
* Команда - записанный по специальным правилам текст (возможно с аргументами), представляющий собой указание на выполнение какой-либо функций (или действий) в операционной системе.