# Лабораторная работа №5

# ПОТОКОВАЯ ПЕРЕДАЧА ИНФОРМАЦИИ МЕЖДУ ПРОЦЕССОМ И ФАЙЛОМ

## **1. Цель работы**

Целью работы является изучение методов программирования по работе с файлами через системные вызовы и стандартную библиотеку ввода-вывода для языка С.

## **2. Задачи работы**

– Закрепление, углубление и расширение знаний студентов при использовании процессов и файлов в прикладных программах.

– Приобретение умений и навыков работы с системой программирования на языках С и С++ в операционной системе Linux.

– Выработка способности логического мышления, осмысления полученных результатов при применении набора функций для работы с файлами через системные вызовы и стандартную библиотеку ввода-вывода для языка C.

## **3. Теоретическая часть.**

Среди всех категорий средств коммуникации наиболее употребительными являются каналы связи, обеспечивающие достаточно безопасное и достаточно информативное взаимодействие процессов.

Существует две модели передачи данных по каналам связи – поток ввода-вывода и сообщения. Из них более простой является потоковая модель, в которой операции передачи/приема информации вообще не интересуются содержимым того, что передается или принимается. Вся информация в канале связи рассматривается как непрерывный поток байт, не обладающий никакой внутренней структурой.

Потоковая передача информации может осуществляться не только между процессами, но и между процессом и устройством ввода-вывода, например между процессом и диском, на котором данные представляются в виде файла.

Функции работы с файлами из стандартной библиотеки ввода-вывода, такие как *fopen()*, *fread()*, *fwrite()*, *fprintf()*, *fscanf()*, *fgets()* и т.д. входят как неотъемлемая часть в стандарт ANSI на язык C. Они позволяют программисту получать информацию из файла или записывать ее в файл при условии, что программист обладает определенными знаниями о содержимом передаваемых данных. Так, например, функция *fgets()* используется для ввода из файла последовательности символов, заканчивающейся символом «\n» – перевод каретки. Функция *fscanf()* производит ввод информации, соответствующей заданному формату, и т. д. С точки зрения потоковой модели операции, определяемые функциями стандартной библиотеки ввода-вывода, не являются потоковыми операциями, так как каждая из них требует наличия некоторой структуры передаваемых данных.

В операционной системе Linux эти функции представляют собой надстройку – сервисный интерфейс – над системными вызовами, осуществляющими прямые потоковые операции обмена информацией между процессом и файлом и не требующими никаких знаний о том, что она содержит.

**Файловый дескриптор**. Информация о файлах, используемых процессом, входит в состав его системного контекста и хранится в его блоке управления. В операционной системе Linux можно упрощенно полагать, что информация о файлах, с которыми процесс осуществляет операции потокового обмена, наряду с информацией о потоковых линиях связи, соединяющих процесс с другими процессами и устройствами ввода-вывода, хранится в некотором массиве, получившем название таблицы открытых файлов или таблицы файловых дескрипторов. Индекс элемента этого массива, соответствующий определенному потоку ввода-вывода, получил название файлового дескриптора для этого потока. Таким образом, файловый дескриптор представляет собой небольшое целое неотрицательное число, которое для текущего процесса в данный момент времени однозначно определяет некоторый действующий канал ввода-вывода. Некоторые файловые дескрипторы на этапе старта любой программы ассоциируются со стандартными потоками ввода-вывода. Так, например, файловый дескриптор 0 соответствует стандартному потоку ввода, файловый дескриптор 1 – стандартному потоку вывода, файловый дескриптор 2 – стандартному потоку для вывода ошибок. В нормальном интерактивном режиме работы стандартный поток ввода связывает процесс с клавиатурой, а стандартные потоки вывода и вывода ошибок – с текущим терминалом.

**Открытие файла. Системный вызов *open().*** Файловый дескриптор используется в качестве параметра, описывающего поток ввода-вывода, для системных вызовов, выполняющих операции над этим потоком. Поэтому прежде чем совершать операции чтения данных из файла и записи их в файл, необходимо поместить информацию о файле в таблицу открытых файлов и определить соответствующий файловый дескриптор. Для этого применяется процедура открытия файла, осуществляемая системным вызовом *open().*

Прототип системного вызова:

#include <fcntl.h>

int open(char \*path, int flags);

int open(char \*path, int flags, int mode);

Системный вызов *open()* предназначен для выполнения операции открытия файла и, в случае ее удачного осуществления, возвращает файловый дескриптор открытого файла (небольшое неотрицательное целое число, которое используется в дальнейшем для других операций с этим файлом). Параметр *path* является указателем на строку, содержащую полное или относительное имя файла. Параметр *flags* может принимать одно из следующих трех значений:

O\_RDONLY – если над файлом в дальнейшем будут совершаться только операции чтения;

O\_WRONLY – если над файлом в дальнейшем будут осуществляться только операции записи;

O\_RDWR – если над файлом будут осуществляться и операции чтения, и операции записи.

Каждое из этих значений может быть скомбинировано посредством операции «побитовое или ( | )» с одним или несколькими флагами:

O\_CREAT – если файла с указанным именем не существует, он должен быть создан;

O\_EXCL – применяется совместно с флагом O\_CREAT. При совместном их использовании и существовании файла с указанным именем, открытие файла не производится и констатируется ошибочная ситуация;

O\_NDELAY – запрещает перевод процесса в состояние ожидание при выполнении операции открытия и любых последующих операциях над этим файлом;

O\_APPEND – при открытии файла и перед выполнением каждой операции записи (если она, конечно, разрешена) указатель текущей позиции в файле устанавливается на конец файла;

O\_TRUNC – если файл существует, уменьшить его размер до 0, с сохранением существующих атрибутов файла, кроме, быть может, времен последнего доступа к файлу и его последней модификации.

Кроме того, в некоторых версиях операционной системы UNIX могут применяться дополнительные значения флагов:

O\_SYNC – любая операция записи в файл будет блокироваться (т. е. процесс будет переведен в состояние ожидание) до тех пор, пока записанная информация не будет физически помещена на соответствующий нижележащий уровень hardware;

O\_NOCTTY – если имя файла относится к терминальному устройству, оно не становится управляющим терминалом процесса, даже если до этого процесс не имел управляющего терминала.

Параметр *mode* устанавливает атрибуты прав доступа различных категорий пользователей к новому файлу при его создании. Он обязателен, если среди заданных флагов присутствует флаг O\_CREAT, и может быть опущен в противном случае. Этот параметр задается как сумма следующих восьмеричных значений:

0400 – разрешено чтение для пользователя, создавшего файл;

0200 – разрешена запись для пользователя, создавшего файл;

0100 – разрешено исполнение для пользователя, создавшего файл;

0040 – разрешено чтение для группы пользователя, создавшего файл;

0020 – разрешена запись для группы пользователя, создавшего файл;

0010 – разрешено исполнение для группы пользователя, создавшего файл;

0004 – разрешено чтение для всех остальных пользователей;

0002 – разрешена запись для всех остальных пользователей;

0001 – разрешено исполнение для всех остальных пользователей.

При создании файла реально устанавливаемые права доступа получаются из стандартной комбинации параметра *mode* и маски создания файлов текущего процесса *umask*, а именно – они равны *mode & ~umask*.

Системный вызов возвращает значение файлового дескриптора для открытого файла при нормальном завершении и значение -1 при возникновении ошибки.

Системный вызов *open()* использует набор флагов для того, чтобы специфицировать операции, которые предполагается применять к файлу в дальнейшем или которые должны быть выполнены непосредственно в момент открытия файла. Из всего возможного набора флагов на лабораторной работе понадобятся только флаги O\_RDONLY, O\_WRONLY, O\_RDWR, O\_CREAT и O\_EXCL. Первые три флага являются взаимоисключающими: хотя бы один из них должен быть применен и наличие одного из них не допускает наличия двух других. Эти флаги описывают набор операций, которые, при успешном открытии файла, будут разрешены над файлом в дальнейшем: только чтение, только запись, чтение и запись. У каждого файла существуют атрибуты прав доступа для различных категорий пользователей. Если файл с заданным именем существует на диске, и права доступа к нему не противоречат запрошенному набору операций, то операционная система сканирует таблицу открытых файлов от ее начала к концу в поисках первого свободного элемента, заполняет его и возвращает индекс этого элемента в качестве файлового дескриптора открытого файла. Если файла на диске нет, не хватает прав или отсутствует свободное место в таблице открытых файлов, то констатируется возникновение ошибки.

Если файл на диске отсутствует и его нужно создать, флаг для набора операций должен использоваться в комбинации с флагом O\_CREAT. Если файл существует, то все происходит по рассмотренному выше сценарию. Если файла нет, сначала выполняется создание файла с набором прав, указанным в параметрах системного вызова.

Чтобы создать файл в момент открытия, флаг для набора операций должен использоваться в комбинации с флагами O\_CREAT и O\_EXCL.

**Системные вызовы *read()*, *write()*, *close()*.** Для совершения потоковых операций чтения информации из файла и ее записи в файл применяются системные вызовы *read()* и *write()*.

Прототипы системных вызовов

#include <sys/types.h>

#include <unistd.h>

size\_t read(int fd, void \*addr, size\_t nbytes);

size\_t write(int fd, void \*addr, size\_t nbytes);

Системные вызовы *read* и *write* предназначены для осуществления потоковых операций ввода (чтения) и вывода (записи) информации над каналами связи, описываемыми файловыми дескрипторами, т.е. для файлов, *pipe*, *FIFO* и *socket*.

Параметр *fd* является файловым дескриптором созданного ранее потокового канала связи, через который будет отсылаться или получаться информация, т. е. значением, которое вернул один из системных вызовов *open()*, *pipe()* или *socket().*

Параметр *addr* представляет собой адрес области памяти, начиная с которого будет браться информация для передачи или размещаться принятая информация.

Параметр *nbytes* для системного вызова *write* определяет количество байт, которое должно быть передано, начиная с адреса памяти *addr*. Параметр *nbytes* для системного вызова *read* определяет количество байт, которое необходимо получить из канала связи и разместить в памяти, начиная с адреса *addr*.

В случае успешного завершения системный вызов возвращает количество реально посланных или принятых байт. Это значение (большее или равное 0) может не совпадать с заданным значением параметра *nbytes*, а быть меньше, чем оно, в силу отсутствия места на диске или в линии связи при передаче данных или отсутствия информации при ее приеме. При возникновении какой-либо ошибки возвращается отрицательное значение.

При работе с файлами информация записывается в файл или читается из файла, начиная с места, определяемого указателем текущей позиции в файле. Значение указателя увеличивается на количество реально прочитанных или записанных байт. При чтении информации из файла она не пропадает из него. Если системный вызов *read* возвращает значение 0, то это означает, что файл прочитан до конца.

После завершения потоковых операций процесс должен выполнить операцию закрытия потока ввода-вывода, во время которой произойдет окончательный сброс буферов на линии связи, освободятся выделенные ресурсы операционной системы, и элемент таблицы открытых файлов, соответствующий файловому дескриптору, будет отмечен как свободный. За эти действия отвечает системный вызов *close()*. При завершении работы процесса с помощью явного или неявного вызова функции *exit()* происходит автоматическое закрытие всех открытых потоков ввода-вывода.

**Системный вызов *close*.** Прототип системного вызова:

#include <unistd.h>

int close(int fd);

Системный вызов *close* предназначен для корректного завершения работы с файлами и другими объектами ввода-вывода, которые описываются в операционной системе через файловые дескрипторы: *pipe*, *FIFO*, *socket*.

Параметр *fd* является дескриптором соответствующего объекта, т. е. значением, которое вернул один из системных вызовов *open()*, *pipe()* или *socket()*.

Системный вызов возвращает значение 0 при нормальном завершении и значение -1 при возникновении ошибки.

В качестве иллюстрации рассмотрим следующую программу:

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main(){

int fd;

size\_t size;

char my\_string[] = "Hello, world!";

/\* Обнуляем маску создания файлов текущего процесса для того,

чтобы права доступа у создаваемого файла точно соответствовали

параметру вызова open() \*/

(void)umask(0);

/\* Попытаемся открыть файл с именем myfile в текущей директории

только для операций вывода. Если файла не существует, попробуем

его создать с правами доступа 0666, т. е. read-write для всех

категорий пользователей \*/

if((fd = open("myfile", O\_WRONLY | O\_CREAT, 0666)) < 0){

/\* Если файл открыть не удалось, печатаем об этом сообщение и

прекращаем работу \*/

printf("Can\'t open file\n");

exit(-1);

}

/\* Пробуем записать в файл 14 байт из нашего массива, т.е. всю

строку "Hello, world!" вместе с признаком конца строки \*/

size = write(fd, my\_string, 14);

if(size != 14){

/\* Если записалось меньшее количество байт, сообщаем об ошибке \*/

printf("Can\'t write all string\n");

exit(-1);

}

/\* Закрываем файл \*/

if(close(fd) < 0){

printf("Can\'t close file\n");

}

return 0;

}

Обратите внимание на использование системного вызова *umask()* с параметром 0 для того, чтобы права доступа к созданному файлу точно соответствовали указанным в системном вызове *open()*.

## **4. Задание на лабораторную работу**

1. С помощью компилятора С создать и выполнить программу, исходный текст которой приведен в примере 1.

2. Модифицировать эту программу для вывода на экран информации о результате создания файла.

3. Написать программу, объединяющую содержимое двух файлов в один.

4. Написать программу, заменяющую указанное слово в файле другим.

## **Контрольные вопросы**

1. Какие системные вызовы используются для записи информации в файл?
2. Как формируются права доступа к файлу, создаваемому пользовательской программой?
3. Для чего предназначен системный вызов *close*?
4. По каким признакам можно определить неудачную попытку открыть файл?
5. Чем отличаются системные вызовы *fread()* и *read()*?