СибГУТИ
Кафедра ВС
Программирование на языке высокого уровня (ПЯВУ), семестр 2
2010-2011 учебный год
Поляков А.Ю., Молдованова О.В.

Файловый ввод/вывод.

Понятие файла

Файл (англ. file — папка, скоросшиватель) - сущность, позволяющая получить доступ к какому-либо ресурсу вычислительной системы и обладающая рядом признаков:

- фиксированное имя (последовательность символов, число или что-то иное, однозначно характеризующее файл);
- определённое логическое представление (формат) и соответствующие ему операции чтения/записи. Формат может быть любым — от последовательности бит до базы данных с произвольной организацией или любым промежуточным вариантом. В первом случае операции чтения/записи для потока и/или массива (последовательные или с доступом по индексу). Во втором — команды системы управления базами данных (СУБД). Также существует большое количество промежуточных вариантов - всевозможных форматов файлов, для которых определяются свои операции чтения и разбора.

В информатике используется следующее определение: файл — это упорядоченная совокупность данных,хранимая на диске и занимающая именованную область внешней памяти. Величина файла характеризуется объемом содержащимся в нем информации. Для того чтобы систематизировать порядок хранения файлов на дисках их объединяют в каталоги.

В отличие от переменной, файл (в частности, его имя) имеет смысл вне конкретной программы. Работа с файлами (по крайней мере, в «простейшем» представлении) реализуется средствами операционной системы.

Ресурсами, доступными через файлы, в принципе, может быть что угодно, представимое в цифровом виде. Чаще всего в их перечень входят:

- области данных (необязательно на диске);
- устройства (как физические, так и виртуальные);
- потоки данных (в частности, вход или выход процесса);
- сетевые ресурсы;
- объекты операционной системы.

Файлы и файловая система

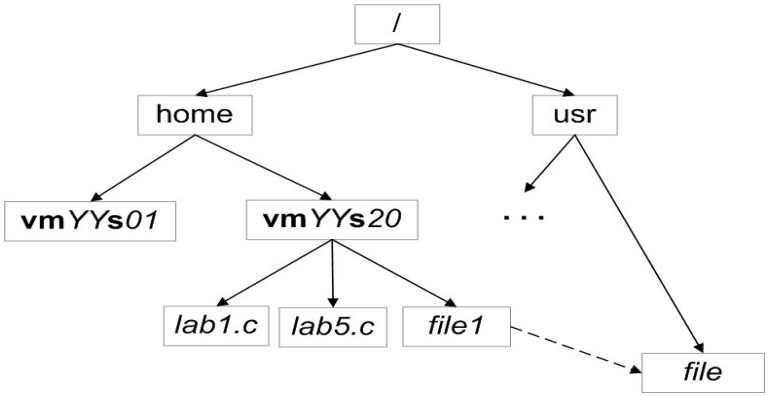

Рис. 1. Пример иерархии файловой системы

По мере развития вычислительной техники файлов в системах становилось всё больше. Для удобства работы с ними, их, как и другие данные, стали организовывать в структуры (тогда же появились символьные имена). Наибольшее распространение получила древовидная организация с возможностью монтирования и вставки дополнительных связей (то есть ссылок — file1 на рис. 1). В связи с этим имя файла приобрело характер пути к нему: перечисление узлов дерева *файловой системы* (ФС), которые нужно пройти, чтобы до файла, например /home/vmYYs20/lab1.c (рис. 1).
ОС предоставляет приложениям набор функций и структур для работы с файлами. Возможности ОС накладывают дополнительные ограничения на ограничения ФС. В зависимости от ФС, файл может обладать различным набором свойств. Рассмотрим некоторые из них.

Имя файла
В большинстве ФС имя файла используется для указания к какому именно файлу производится обращение. В различных ФС ограничения на имя файла сильно различаются:

- в FAT16 и FAT12 размер имени файла ограничен 8 символами (3 символа расширения).
- в VFAT, ext2/ext3 ограничение 255 байт.
- в FAT32 ограниченно 255 символами
- в NTFS имя ограничено 255 символами Unicode

Помимо ограничений файловой системы, интерфейсы операционной системы дополнительно ограничивают набор символов, который допустим при работе с файлами.

- в ОС Microsoft Windows в имени файла разрешены заглавные и строчные буквы, цифры, некоторые знаки препинания, пробел. Запрещены символы > < | ? * / \ : ".
- в ОС GNU/Linux (с учётом возможности маскировки) разрешены все символы, кроме / и нулевого байта.

Расширение имени файла
Для определения типа (формата) файла используется *расширение файла*. Это один из распространённых способов, с помощью которых пользователь или программное обеспечение компьютера может определить тип данных, хранящихся в файле. Расширение обычно отделяется от основной части имени файла точкой. Иногда могут использоваться несколько расширений, следующих друг за другом, например, «.tar.gz». *Операционная система* (ОС) или менеджер файлов могут устанавливать соответствия между расширениями файлов и приложениями. Когда пользователь открывает файл с зарегистрированным расширением, автоматически запускается соответствующая этому расширению программа. Некоторые расширения показывают, что файл сам является программой.

Время
Для файла могут быть определены следующие временные метки:

- Время создания
- Время модификации
- Время последнего доступа

Владелец, группа и права доступа к файлу
В некоторых файловых системах предусмотрено указание на владельца файла, и группу владельца и ограничение доступа пользователей к содержимому файла. В UNIX-подобных операционных системах для файлов обычно выделяют три типа прав:

- право на запись
- право на чтение
- право на выполнение

Каждое право задаётся раздельно для владельца, для группы и для всех остальных.

Операции с файлом

Условно можно выделить два типа операций с файлом — связанные с его открытием, и выполняющиеся без его открытия. Операции первого типа обычно служат для чтения/записи информации или подготовки к записи/чтению. Операции второго типа выполняются с файлом как с «объектом» файловой системы, в котором файл является неделимой единицей. Обычно выделяют дополнительные сущности, связанные с работой с файлом:

- *хэндлер файла*, или *дескриптор* (описатель). При открытии файла (в случае, если это возможно), операционная система возвращает число (или указатель на структуру), с помощью которого выполняются все остальные файловые операции. По их завершению файл закрывается, а хэндлер теряет смысл. В качестве аналога *хэндлера* может рассматриваться адрес дома или название организации.
- *файловый указатель* - число, являющееся смещением относительно нулевого байта в файле. Обычно по этому адресу осуществляется чтение/запись, в случае, если вызов операции чтения/записи не предусматривает указание адреса. При выполнении операций чтения/записи файловый указатель смещается на число прочитанных (записанных) байт. Последовательный вызов операций чтения таким образом позволяет прочитать весь файл не заботясь о его размере.
- *файловый буфер,* ОС (и/или библиотека языка программирования) осуществляет кэширование файловых операций в специальном буфере (участке памяти). При закрытии файла буфер сбрасывается.
- *режим доступа,* в зависимости от потребностей программы, файл может быть открыт на чтение и/или запись. Кроме того, некоторые операционные системы (и/или библиотеки) предусматривают режим работы с текстовыми файлами. Режим обычно указывается при открытии файла.
- *режим общего доступа,* В случае многозадачной операционной системы возможна ситуация, когда несколько программ одновременно хотят открыть файл на запись и/или чтение. Для регуляции этого существуют режимы общего доступа, указывающие на возможность осуществления совместного доступа к файлу (например, файл в который производится запись может быть открыт для чтения другими программами — это стандартный режим работы log-файлов). При совместном доступе для получения эксклюзивного права работы с файлом используются блокировки. В ОС GNU\Linux для реализации блокировок можно использовать функции fcntl или flock.

Операции, связанные с открытием файла

- *открытие файла,* обычно в качестве параметров передается имя файла, режим доступа и режим совместного доступа, а в качестве значения выступает файловый *хэндлер*, кроме того обычно имеется возможность в случае открытия на запись указать на то, должен ли размер файла изменяться на нулевой.
- *закрытие файла*, в качестве аргумента выступает значение, полученное при открытии файла. При закрытии все файловые буферы сбрасываются.
- *запись,* в файл помещаются данные.
- *чтение*, данные из файла помещаются в область памяти.
- *перемещение указателя* - указатель перемещается на указанное число байт вперёд/назад или перемещается по указанному смещению относительно начала/конца. Не все файлы позволяют выполнение этой операции (например, файл на ленточном накопителе может не «уметь» перематываться назад).
- *сброс буферов* - содержимое файловых буферов с не записанной в файл информацией записывается. Используется обычно для указания на завершение записи логического блока (для сохранения данных в файле на случай сбоя).
- *Получение текущего значения файлового указателя*.

Операции, не связанные с открытием файла
Операции, не требующие открытия файла оперируют с его «внешними» признаками — размером, именем, положением в дереве каталогов. При таких операциях невозможно получить доступ к содержимому файла, файл является минимальной единицей деления информации. В зависимости от файловой системы, носителя информации, операционной системой часть операций может быть недоступна:

- Удаление файла
- Переименование файла
- Копирование файла
- Перенос файла на другую файловую систему/носитель информации
- Создание симлинка или хардлинка
- Получение или изменение атрибутов файла

Работа с файлами в языке СИ. Потоковый ввод/вывод.

В стандарте языка Си отсутствуют средства ввода-вывода. Все операции ввода-вывода реализуются с помощью функций, находящихся в библиотеках, поставляемых вместе с конкретной системой программирования.

Особенностью языка Си, который был впервые применен при разработке ОС UNIX, является отсутствие заранее спланированных структур файлов. Все файлы рассматриваются как неструктурированные последовательности байтов. Такой подход позволил распространить понятие файла и на различные устройства. В ОС UNIX конкретному устройству соответствует так называемый "специальный файл". Для обмена данными с файлами и устройствами используются одни и те же библиотечные функции Си.

Поток

По историческим причинам структура данных языка Си, описывающая поток данных называется FILE (а не STREAM (англ.) поток). Эта структура объявлена в файле stdio.h. FILE содержит внутреннюю информацию о состоянии соединения с ассоциированным файлом, включая информацию о позиции курсора, информацию о буферизации, индикатор конца файла и ошибки (они могут быть получены с помощью функций feof и ferror. Выделение памяти и управление данной структурой выполняется библиотечными функциями ввода/вывода . Программист не должен работать с FILE напрямую, для написания переносимого кода необходимо работать с ней ТОЛЬКО через соответствующие функции.

Алгоритм работы с файлом выглядит следующим образом:

1. открытие потока;
2. операции чтения/записи/смещения курсора;
3. закрытие потока.

Открытие потока

Открытие файла выполняется с помощью функции *fopen*, которая создает новый поток ассоциированный с открываемым файлом. Данное действие может привести к созданию нового файла.

```
FILE * fopen (const char *filename, const char *opentype)
```

Функция *fopen* открывает поток для ввода/вывода в/из файла *filename* и возвращает указатель на поток. Агрумент *opentype* — это строка, управляющая режимом открытия файла. Она должна начинаться со следующих символов:

opentype	Значение аргумента
"r"	Открытие существующего файла filename только для чтения
"w"	Открытие файла только для записи. Если файл уже существует его содержимое очищается. В противном случае будет создан новый файл с именем filename.
"a"	Открытие файла для добавления, т.е. только запись в конец файла. Если файл уже существует его содержимое не изменяется, а новые данные дописываются в конец. В противном случае создается новый файл.
"r+"	Открытие существующего файла как для записи так и для чтения. Содержимое файла не изменяется, позиция чтения/записи установлена в начало файла. При записи старое содержимое будет заменяться на новое.
"w+"	Открытие существующего файла как для записи так и для чтения. Если файл уже существует его содержимое очищается. В противном случае будет создан новый файл с именем filename.
"a+"	Открытие файла как для чтения так и для добавления. Если файл существует — его содержимое не изменяется, позиция чтения установлена на начало файла, добавление выполняется в конец файла./записи установлена в начало файла. Если файла не существует — он создается.

'+' запрашивает поток в котором разрешены операции как ввода так и вывода. Стандарт ANSI требует вызова функции fflush или вызова позиционирующей функции (например fseek) при переключении между чтением и записью и наоборот. В противном случае встроенные буферы могут быть очищены некорректно.

В библиотеке GNU также используется символ `x', означающий что open выполнится успешно ТОЛЬКО в том случае если файл не существовал на момент открытия.

Символ 'b' означает что открывается бинарный файл (не текстовый). В POSIX системах (включая GNU системы) – различий между бинарными и текстовыми файлами не делается.

Любые другие символы в аргументе *opentype* игнорируются.

В случае неуспеха fopen возвращает NULL.

Закрытие потока

Поток закрывается с помощью функции *fclose*, которая разрывает соединение между потоком и файлом. Буферизованный вывод записывается в файл, а любой буферизованный ввод отбрасывается. *fclose* возвращает 0 в случае успеха и EOF в случае ошибки.

```
int fclose (FILE *stream);
```

Очень важно проверять не произошла ли ошибка, поскольку это достаточно обычная ситуация. Например ситуация когда носитель переполнен.

В случае завершения функции *main* или вызова *exit* – все открытые потоки *автоматически* и *корректно* закрываются. В случае же аварийного завершения возможно некорректное закрытие и потеря данных в файле.

Бинарные файлы

Файлы могут быть разделены на текстовые и бинарные. Текстовые содержат только символы, распознаваемые текстовыми редакторами. Двоичный (бинарный) файл — в широком смысле: последовательность произвольных байтов. В узком смысле слова двоичные файлы противопоставляются текстовым файлам. При этом с точки зрения технической реализации на уровне аппаратуры, текстовые файлы являются частным случаем двоичных файлов, и, таким образом, в широком значении слова под определение «двоичный файл»

подходит любой файл.

Для работы с бинарными файлами могут быть использованы различные функции. Рассмотрим наиболее распространенные: *fread* и *fwrite*. Эти функции позволяют читать и записывать блоки данных любого типа. Прототипы этих функций следующие:

```
size_t fread (void * buffer, size_t size, size_t count, FILE * fp);
size_t fwrite (const void * buffer, size_t size, size_t count, FILE * fp);
```

Буфер (*buffer*)– указатель на область памяти, в которую будут прочитаны данные из файла. Счетчик — *count* — определяет, сколько считывается и записывается элементов данных, причем длина каждого элемента в байтах равна *size*. Функция fread возвращает количество прочитанных элементов, если достигнут конец файла или произошла ошибка, то возвращаемое значение может быть меньше, чем счетчик. Функция *fwrite* возвращает количество записанных элементов. Если ошибка не произошла, то возвращаемый результат будет равен значению счетчика. Одним из самых полезных применений функций fread() и fwrite() является чтение и запись данных пользовательских типов, особенно структур. Например:

```
. . . . . .
FILE * fp;
char filename[] = "some_file";
struct some_struct buff[64]; // массив структур из 64-х элементов
fp =fopen(filename, "wb")
if( fp == NULL ){
 printf("Error opening file %s\n",filename);
}else{
 fwrite(buff, sizeof(struct some_struct), 64, fp);
 fclose(fp);
}
. . . . . .
fp =fopen(filename, "rb")
if( fp == NULL ){
 printf("Error opening file %s\n",filename);
}else{
 fread(buff, sizeof(struct some_struct), 64, fp);
 fclose(fp);
}
```

Позиционирование в потоке

Как уже было сказано, поток характеризуется позицией чтения и(или) записи. В ОС GNU\Linux это просто счетчик байтов относительно начала файла.

При дисковом вводе/выводе позиция может меняться произвольно. Некоторые другие типы файлов позволяют то же. Файлы, поддерживающие смену позиции – файлы с произвольным доступом. Следующие функции, позволяющие осуществлять произвольный доступ:

```
long int ftell (FILE *stream);
```

Функция *ftell* возвр текущую позицию в файле. Она может завершиться с ошибкой, если файл не поддерживает позиционирование или если позиция не может быть представлена как long int. В случае ошибки возвр -1.

```
int fseek (FILE *stream, long int offset, int whence);
```

Функция *fseek* используется для смены позиции чтения/записи в потоке. Значение *whence* должно быть одной из констант SEEK_SET, SEEK_CUR или SEEK_END, которые соответствуют первому, текущему или последнему байту файла. *fseek* устанавливает позицию

чтения/записи смещаясь на *offset* байт от позиции, указанной параметром *whence*.

ПРИМЕР С БИНАРНЫМ ФАЙЛОМ. Связанный с предыдущим примером.

Стандартные потоки

Когда управление передается функции main имеется 3 готовых к использованию потока. Они называются "стандартными" каналами ввода/вывода:

```
FILE * stdin;
FILE * stdout;
FILE * stderr;
```

Соответственно стандартный входной поток, выходной поток и поток ошибок (который может быть использовать для вывода диагностических сообщений).
В GNU C это обычные переменные, поэтому с ними можно работать как с любыми другими переменными, объявленными программистом. Поэтому возможна такая операция:

```
fclose (stdout);
stdout = fopen ("standard-output-file", "w");
```

Но в других ОС stdin, stdout, and stderr – это макросы и доступ к ним не может быть выполнен вышеописанным образом.

Текстовые файлы
Для работы с файлами существуют функции, аналогичные *printf* и *scanf*:

```
int fprintf (FILE *stream, const char *template, ...);
int fscanf (FILE *stream, const char *template, ...);
```

Форматная строка (*template*) позволяет задать входной/выходной формат аналогично функциям *printf/scanf*.
Следующий пример демонстрирует ввод/вывод в режиме *чтение+запись*.

```
#include <stdio.h>
int main()
{
    FILE *fp = fopen("testfile","a+");
    int i,j;
    if( fp == NULL ){
        printf("Cannot open file testfile\n");
    }
    for(i=0;i<10;i++){
        printf("Reading stage: ");
        for(j=0;j<10;j++){
            char ch;
            int ret = fscanf(fp,"%c",&ch);
            if( ret == EOF )
                printf("E");
            else
                printf("%c ",ch);
        }
        printf("\nWriting stage\n");
        for(j=0;j<10;j++){
            fprintf(fp,"a");
        }
    }
```

```
        fclose(fp);
    }
```

Расширенные возможности форматного ввода/вывода

Ввод осуществляется в соответствии с форматом, который описывается форматной строкой.

Спецификация преобразования НЕ должна содержать пробелов

Спецификация преобразования для форматной строки вывода имеет вид:

% flags [width [. precision]] [type] conversion

- *flags:* 0 или более флагов, которые изменяют модифицируют стандартное отображение спецификации преобразования.
- *width:* Необязательное поле – положительное десятичное целое, указывающее минимальный размер поля. Если указано просто десятичное число, а фактич. ширина меньше указ – заполняется пробелами, если указано десятичное с 0-м вначале – заполняется нулями.
- *precision:* (необязательное поле - точность), определяющее:
  1. минимальное число цифр, которые могут быть выведены для целых.
  2. Число цифр после запятой для вещественных
  3. максимальное число символов для спецификатора s.
- type (необязательное поле - модификатор), которое указывает тип данных соответствующего аргумента, если оно отличается от типа по умолчанию. (Например преобразование целого типа подразумевает тип int, но вы можете указать `h', `l', or `L' для обозначения других целых типов.) .
  'h' => преобразование применяется к типу short
  'l' => -//- к типу long
  'L' => к типу long double
- *conversion:* Символ-спецификатор, указывающий тип преобразования.

Спецификация преобразования для форматной строки ввода имеет вид:

% flags width type conversion

- *flags:* (необязательный параметр) **'*'** – значит игнорировать значение считанное для данного преобразования. Для спецификации со звездочкой аргумент пропускается. Счетчик успешного считывания также не инкрементируется.
- *width:* (необязательны параметр) положительное десятичное целое – определяет максимальное количество символов, которое может быть считано (фактически может быть меньше). В случае ввода строки – необходимо учитывать что признак окончания строки не учтен в ширине поля.
- *type:* Аналогично выводу.
- *conversion:* Аналогично выводу.

Таблица преобразователей (*conversion*)

%d	десятичное целое (int *)
%i	десятичное, восьмиричное или шестнадцатиричное целое (int *)
%x,%X	шестнадцатиричное (маленькими или большими)
%u	unsigned
%o	восьмиричное целое
%c	один символ
%s	строка, ограниченная справа и слева пробелами

%p	значение указателя
%f	печать числа с плавающей точкой в обычном формате
%e, %E	печать вещественного числа в экспоненциальной форме (%e – в нижнем регистре, E – в верхнем)
%g,%G	печать в наиболее удобном формате (обычном или эксп.)

Флаги (*flags*) преобразователей для целых (спецификаторы '%d', '%i', '%o', '%u', '%x', '%X'):

'-'	Выравнивание по левому краю (по умолчанию – по правому)
'+'	Для знаковых – печатает '+', если число положительное
' '	Для знаковых печатает пробел, если нет знака минус. Этот флаг перекрывается флагом '+'.
'#'	Для спецификатора `%o' – первая цифра '0'. Для `%x' or `%X' - префикс `0x' or `0X'. Не влияет на `%d', `%i', or `%u'. Данный флаг выводит данные в формате понятном функции strtoul и scanf с `%i' спецификатором.
'0'	дополнять нулями вместо пробелов. Перекрывается флагом `-' или точностью.

// НЕДОРАБОТАНО, не используется на экзамене 2010 г.

Буферизация потока

Символы, записываемые в поток обычно накапливаются и перезаются асинхронно в файлы блоками вместо моментальной передачи. Получение информации потоком осуществляется аналогично. Этот процесс называется буферизацией.

Для интерактивных приложений можно заметить что символы не сразу появляются на экране.

3 СТРАТЕГИИ:
1.Characters written to or read from an unbuffered stream are transmitted individually to or from the file as soon as possible.
2.Characters written to a line buffered stream are transmitted to the file in blocks when a newline character is encountered.
3.Characters written to or read from a fully buffered stream are transmitted to or from the file in blocks of arbitrary size.