

Файлы

- Классификация файловых систем
- Именованение файлов. Каталоги. Пути
- Мягкие и жесткие ссылки
- Реализация файловых систем
- Контроль доступа к файлам
- Работа с файлами функциями CRT
- Работа с файлами функциями WinAPI

Цель файловой системы

- Целью файловой системы является **абстрагировать** пользовательскую программу от сложных операций с контроллерами устройств хранения данных, заменив их на простую логическую модель файлов
- **Файл** — это именованная единица внешней памяти определенного пользователем размера, над которой могут быть выполнены операции чтения и записи данных.

Типы файлов

- Обычные пользовательские файлы
 - текстовые (ASCII);
 - двоичные.
- Каталоги — системные файлы, необходимые для организации файлов.
- Системные специальные файлы (представляют внешние устройства и объекты ОС)
 - символьные (последовательные устройства);
 - блочные.

Классификация файловых систем

- По методу хранения принадлежащих файлу блоков:
 - системы с непрерывными файлами;
 - системы со связным списком блоков файла;
 - системы с таблицей расположения файлов;
 - системы с i-узлами.
- По способам восстановления при ошибках:
 - только частичное восстановление специальными программами;
 - системы с поддержкой журналирования (транзакций)

Именование файлов

- Файлы имеют строковые **имена**, как правило ограниченной размерности.
- **Каталоги** (папки) — это специальные файлы, позволяющие группировать имеющиеся файлы, образуя дерево так, что каждый каталог (узел дерева) содержит относительно небольшое количество других файлов и каталогов.
- **Путем** называется совокупность имен каталогов и файла (разделенных символом-разделителем), **однозначно** идентифицирующая конкретный файл.

Пути

- В качестве разделителя в ОС Unix используется дробная черта /, в ОС Windows — обратная дробная черта \
- В путях могут встречаться особые имена:
 - . (точка) — указывает на текущий каталог;
 - . . (две точки) — указывает на родительский каталог.

Виды путей

- **Абсолютный** путь — содержит путь к файлу от вершины дерева каталогов (корневой папки)
 - в ОС Unix абсолютные пути начинаются с разделителя: `/usr/local/apache`
 - в ОС Windows абсолютные пути начинаются с имени диска и двоеточия: `C:\Windows\explorer.exe`
- **Относительный** путь — содержит путь к файлу относительно текущего каталога, он начинается с имени файла или каталога: `../usr/bin/perl`

Ссылки

- Иногда требуется сослаться на один и тот же файл из разных каталогов. Для этого применяются ссылки:
 - **жесткая** ссылка — наличие нескольких ссылок на одни и те же данные о файле в разных каталогах; файл содержит счетчик и удаляется только при удалении всех ссылок;
 - **мягкая** (символическая) ссылка — текстовый файл специального типа, содержащий путь к другому файлу, который интерпретируется системой как вызов указанного файла; удаление ссылки не оказывает влияние на файл.

Сравнение типов ссылок

- Жесткие
 - все ссылки равноправны и неотличимы для пользователя;
 - владельцем файла с точки зрения прав доступа считается его автор.
- Мягкие
 - ссылки отличимы от исходного файла;
 - удаление исходного файла делает ссылки ошибочными;
 - ссылка может быть перенаправлена на другой файл;
 - мягкие ссылки значительно усложняют алгоритмы поиска файлов.

Факторы производительности файловой системы

- Наиболее длительной операцией является поиск на диске нужного сектора (т.е. поворот диска на нужный угол и, в меньшей степени, перемещение головки на нужный цилиндр)
- Чтение последовательно расположенных секторов значительно быстрее, чем чтение нескольких разбросанных по диску секторов.

Учет блоков диска, принадлежащих файлу

- Непрерывные файлы — хранится начало и длина каждого файла
 - легко и производительно, но размер файла должен быть известен заранее.
 - применяются в файловых системах CD и DVD
- Связные списки: каждый блок файла хранит указатель на следующий
 - файлы могут иметь переменный размер, однако доступ к концу файла очень медленный, т.к. требует чтения всех блоков.

Учет блоков диска, принадлежащих файлу

- таблица расположения файлов (file allocation table, FAT)
 - работает как предыдущая схема, но указатели хранятся не в самих блоках, а в отдельной таблице;
 - производительность растёт, но таблица занимает много места в памяти, особенно при больших дисках;
 - наиболее известные ее реализации — FAT16 и FAT32, используемые в MS Windows.

Учет блоков диска, принадлежащих файлу

- использование индексных узлов (i-узлов) — таблица блоков файла хранится отдельно для каждого файла
 - значительно уменьшается объем памяти, необходимый для обработки таблицы, т.к. в памяти необходимо хранить только узлы открытых файлов;
 - производительность немного хуже, чем у предыдущего метода;
 - используется в большинстве современных файловых систем как Unix, так и Windows (NTFS)

Учет свободных блоков диска

- Ведение списка номеров свободных блоков
 - преимущество: быстрый поиск свободного блока;
 - недостаток: в начале размер таблицы очень большой.
- Хранение занятости блоков в побитовой карте, т.е. каждый бит хранит 0 (свободен)/1 (занят) для соответствующего блока.
 - этот метод требует меньше дискового пространства, однако менее производителен, т.к. для поиска свободного блока надо проходить по списку.

Контроль доступа к файлам

- Частичная классификация пользователей (ОС Unix)
 - простота реализации, небольшое занимаемое место, однако возможности контроля весьма ограничены.
- Списки контроля доступа (access control list, ACL)
 - детальный контроль доступа, однако значительно более затратный в реализации;
 - применяется в NTFS и современных ФС Unix

Базовая модель Unix

- Все пользователи относительно файла делятся на три категории:
 - владелец;
 - группа владельца;
 - все остальные.
- Каждой категории задаются права на:
 - исполнение файла;
 - запись файла;
 - чтение файла.

Списки контроля доступа

- **Список контроля доступа** представляет собой связный список, перечисляющий пользователей и их права на доступ к файлу (каталогу).
- Поскольку ввод большого числа пользователей может быть утомительным, а также для простоты управления списком при добавлении новых пользователей (например если файл открыт для всех), вводится возможность указания списка в **негативной форме** (т.е. списка тех, кто доступа не имеет).

Работа с файлами

- При работе с файлами из программы учитываются следующие понятия:
 - **режим открытия** — определяет режим обработки данных (двоичный или текстовый), права доступа, а также способ записи (дописывать в конец файла или стереть существующее содержимое);
 - **указатель позиции** — указывает на место в файле, с которого будет производиться следующая операция чтения или записи;
 - **конец файла** (end of file, **eof**) — состояние, когда указатель позиции указывает за последним байтом файла.

Работа с файлами в CRT

- В CRT существуют два способа работы с файлами:
 - работа на низком уровне (функции типа `open`, `read`, `write`, `close`) — функции прямо соответствуют системным вызовам ОС;
 - потоковый ввод/вывод (функции типа `fopen`, `fread`, `fwrite`, `fclose`, `fprintf`, `fgets`, `fscanf`) — обеспечивает дополнительную буферизацию средствами CRT и форматированный ввод/вывод (в случае текстовых файлов).
- Эти два метода **несовместимы** в пределах открытого файла.

Файловые дескрипторы CRT

- Для управления доступа процесса к файлам на низком уровне в CRT используются **файловые дескрипторы** — целые числа, являющиеся номерами в таблице открытых файлов процесса.
- Дескриптор получается при открытии файла с помощью функции `open`, используется при вызовах функций `read` и `write`, закрывается при помощи функции `close`.
- Дескрипторы 0, 1 и 2 указывают на *стандартные* потоки ввода, вывода и ошибок соответственно.

Открытие файла с получением дескриптора

- `int` — дескриптор открытого файла, -1 при ошибке
- `_open (`
 - `const char *` - путь к файлу
 - `int` - флаги
 - `_O_CREAT` — создает файл если его нет
 - `_O_EXCL` — возвращает ошибку, если файл существует
 - `_O_APPEND` — режим записи в конец файла
 - `_O_TRUNC` — уничтожает содержимое файла если он есть
 - `_O_NOINHERIT` — делает дескриптор ненаследуемым
 - `_O_BINARY`, `_O_TEXT` — режим работы с файлом
 - `_O_RDONLY`, `_O_WRONLY`, `_O_RDWR` — режим доступа
 - `int` — режим доступа, если файл создается (в Windows задается константами `_S_IREAD` и/или `_S_IWRITE`)
-)

Задание

- Откройте файл `new.txt`, создав его если он отсутствует, в режиме записи в конец файла
 - файл открывается как текстовый с возможностью только записи;
 - если файла нет, то он создается с доступом как на чтение, так и на запись
 - дескриптор открытого файла сохраните в переменную `myfile`.

Открытие файла с получением дескриптора

```
int myfile=_open("new.txt",  
    _O_CREAT|_O_APPEND|_O_TEXT|_O_WRONLY,  
    _S_IREAD|_S_IWRITE);
```

Права доступа к файлам в ОС Unix

- В ОС Unix права определяются тремя битами:
 - право на исполнение файла (1);
 - право на запись файла (2);
 - право на чтение файла (4).
- Эти биты формируют число, задаваясь для трех категорий пользователей
 - для владельца файла (биты 6-8);
 - для группы владельца (биты 3-5);
 - для всех остальных пользователей (биты 0-2).
- Для записи прав доступа удобно использовать восьмеричную константу, т. к. одна цифра в ней соответствует 3-м битам.

Задание

- Укажите число, описывающее следующие права доступа к создаваемому файлу в ОС Unix:
 - для владельца — все права;
 - для группы владельца — права на чтение и исполнение;
 - для остальных пользователей — права только на чтение.
- Укажите число, описывающее права доступа на чтение и запись к файлу для всех без исключения пользователей.

Права доступа к файлам в ОС Unix

- Детализированный доступ: 0754
- Доступ на чтение и запись для всех: 0666

Работа с файловым дескриптором

- `int` — количество прочитанных/записанных байт
- `read/write (`
 - `int` — дескриптор файла
 - `void *` - указатель на начало данных, для функции `write` может быть константным,
 - `unsigned int` — количество байт для чтения/записи`)`

Работа с файловым дескриптором

- `int` — 1 если достигнут конец файла, 0 — если нет, -1 при ошибке
- `eof (`
 - `int` — дескриптор файла
 -)
 - `int` — 0 при успехе, -1 при ошибке
- `close (`
 - `int` — дескриптор файла
 -)

Задание

- Запишите в файл, чей дескриптор сохранен в переменной `fdw` массив целых чисел `arr` из 10 элементов
- Считайте из файла, чей дескриптор сохранен в переменной `fdr`, структуру в переменную `point`.

Работа с файловым дескриптором

```
write(fdw, (void *)arr,  
      sizeof(int)*10);
```

```
read(fdr, (void *)&point,  
     sizeof(point));
```

Потоковый ввод/вывод в CRT

- Для хранения информации об открытом файле используются структура `FILE`
- Все функции потокового ввода/вывода получают указатель на эту структуру
- Ввод/вывод из текстовых файлов возможен двух типов (допускается их смешивание):
 - прямой: функциями `fread` и `fwrite`
 - форматный: функциями `fgets`, `fscanf`, `fputs`, `fprintf` и т.д.

Создание потока с открыванием файла

- `FILE *` - указатель на структуру потока, 0 при ошибке
- `fopen (`
 - `const char *` - имя файла,
 - `const char *` - режим
 - `r` — чтение,
 - `w` — запись, предыдущее содержимое файла стирается
 - `a` — дозапись в конец файла
 - `w+` - чтение и запись, предыдущее содержимое файла стирается
 - `b` или `t` — режим двоичного или текстового файла
-)

Задание

Объявите переменную `fstream` и откройте в нее файл `result.txt` в текстовом режиме для записи, уничтожив старое содержимое файла

Создание потока с открыванием файла

```
FILE * fstream = fopen("result.txt",  
    "wt");
```

Прямые чтение/запись в поток

- `size_t` — количество элементов, реально записанных или прочитанных
- `fread/fwrite (`
 - `void *` - указатель на начало данных для записи (буфера для чтения),
 - `size_t` — размер элемента,
 - `size_t` — количество элементов,
 - `FILE *` - поток, из которого производится чтение/запись
-)

Форматированный ввод/вывод

- Для каждой функции стандартного ввода/вывода (`printf`, `puts`, `putc` и т.д.) существуют их файловые аналоги, начинающиеся с буквы `f` (`fprintf`, `fputs` и т.д.)
- Для функций с переменным числом параметров (`fprintf`, `fscanf`) файл указывается первым параметром, с постоянным (`fputs`, `fgets`) — последним.
- В остальном параметры те же, кроме того что `fgets` дополнительно получает размер буфера.

Задание

В текстовом файле хранится числовой параметр в виде строки типа `<имя>=<значение>`.

Введите из файла `fstream` такой параметр, сохранив имя в переменную `par_name` (строка), а значение — в переменную `par_value` (целое число).

Форматированный ввод/вывод

```
fscanf(fstream, "%[^]=%d" ,  
       par_name, &par_value);
```

Завершение работы с файлом через ПОТОК

- `int` — 0 если конец файла не достигнут
- `feof (`
 - `FILE *` - проверяемый поток`)`
 - `int` — 0 при успешном закрытии
- `fclose (`
 - `FILE *` - закрываемый поток`)`

Работа с файлами в WinAPI

- Дескриптор открытого файла имеет тип `HANDLE`
- Файл открывается (и, при необходимости, создается) функцией `CreateFile`
- Для чтения и записи существуют функции `ReadFile` и `WriteFile`
- Любой дескриптор типа `HANDLE` закрывается функцией `CloseHandle`
- Возможен асинхронный ввод/вывод

Открываем файл

- HANDLE — дескриптор открытого файла, INVALID_HANDLE_VALUE при ошибке
- CreateFile (
 - LPCTSTR — имя файла,
 - DWORD — желаемый доступ к файлу
 - GENERIC_READ и/или GENERIC_WRITE
 - DWORD — режим разделения: могут ли (и с какими правами) открывать этот файл другие процессы (одновременно с текущим)
 - 0 — нельзя, либо FILE_SHARE_READ и/или FILE_SHARE_WRITE

Открываем файл

- LPSECURITY_ATTRIBUTES — может быть NULL
- DWORD — режим создания: одно из следующих значений
 - CREATE_ALWAYS — создает новый файл, если был то содержимое стирается;
 - CREATE_NEW — только создать новый, ошибка если файл существует;
 - OPEN_ALWAYS — открывает файл сохраняя содержимое, создавая при необходимости,
 - OPEN_EXISTING — открыть файл, ошибка если его нет
 - TRUNCATE_EXISTING — открыть существующий файл, стерев его содержимое

Открываем файл

- `DWORD` — атрибуты файла, обычно `FILE_ATTRIBUTE_NORMAL`
- `HANDLE` — дескриптор файла-шаблона: если создается новый файл, то для него и этот дескриптор не `NULL`, то атрибуты и права доступа для нового файла берутся из файла-шаблона.

)

Задание

- Объявите переменную `myFile` и сохраните в нее дескриптор файла `MyFile.txt` при следующих условиях:
 - файл открывается для чтения и записи;
 - если его не было, то он создается;
 - если файл был, то если переменная `Clear` равна 1 то его содержимое стирается, иначе сохраняется
 - совместный доступ к файлу запрещен

Открываем файл

```
DWORD dwCreate=OPEN_ALWAYS;  
if (Clear) dwCreate=CREATE_ALWAYS;  
HANDLE myFile =  
    CreateFile("MyFile.txt",  
GENERIC_READ | GENERIC_WRITE,  
NULL, NULL, dwCreate,  
FILE_ATTRIBUTE_NORMAL, NULL);
```

Чтение/запись в файл

- BOOL - 0 при ошибке
- ReadFile/WriteFile (
 - HANDLE — дескриптор файла,
 - LPVOID — указатель на начало данных (буфера) для записи (чтения)
 - DWORD — количество байт для записи (чтения)
 - LPDWORD — указатель, по которому будет записано количество реально записанных (прочитанных) байт
 - LPOVERLAPPED — указатель на структуру для асинхронного ввода/вывода, обычно NULL

Закрываем дескриптор

- BOOL — 0 при ошибке
- CloseHandle (
 - HANDLE — закрываемый дескриптор)