**УТОЧНЕНИЕ ЗАДАНИЯ ПО ТП**

Вот, собственно, текст задания:

МОДЕЛЬ ФАЙЛОВОЙ СИСТЕМЫ ОС РЕАЛЬНОГО ВРЕМЕНИ

Описываемая ниже структура размещения файлов и управляющей ин­формации на носителе ( диске, гибком диске и т.д.) называется фай­ловой структурой.

Файлом является совокупность связанных между собой логических записей. Файлы на носителе располагаются в логических блоках, кото­рые имеют фиксированный размер 512 байт. Блоки файла получают пос­ледовательные номера, начиная с 0. Файл размещается в смежных логи­ческих блоках, т.е. является непрерывным.

Информация на носителе распределяется следующим образом:

|  |  |
| --- | --- |
| № блока | Содержание |
| 0  1  2-5  6  6+N+2 | Программа первичный загрузчик  Блок системной информации  Вторичный загрузчик  Начало каталога файлов  Файлы и свободное место |

Блоки 0,1,2,3,4,5 содержат загрузчик или не используются, если носитель не содержит ОС ( отсутствуют файлы монитора и драйверов ). Блок 1 содержит системную информацию, включающую идентификатор то­ма, имя владельца, таблицу замещения дефектных блоков на резервные и т.д.

Содержимое системного блока:

Байты

0 - 129 - таблица расположения дефектных блоков,

132 - 169 - область сохранения, содержит информацию

для восстановления каталога при ошибоч­ной инициализации,

468 - 469 - номер блока начала каталога,

470 - 471 - версия системы

472 - 483 - имя тома,

484 - 495 - имя владельца,

496 - 507 - название системы.

Остальные байты зарезервированы для использования системными программами.

Информация о каждом имеющемся на носителе файле содержится в таблице, которая называется каталогом. Каталог файлов всегда начи­нается с блока 6 и состоит из сегментов, число которых не может превышать 31. Сегмент занимает два смежных блока (1024 байта). Все остальные блоки носителя (после каталога) используются для хранения файлов. Каждый сегмент каталога имеет заголовок, состоящии из 5 слов:

слово 1 - число сегментов, отведенных под каталог,

слово 2 - номер следующего сегмента. Сегменты каталога образуют

список. В последнем используемом сегменте это слово со­держит 0,

слово 3 - счетчик занятых сегментов. При использовании нового сег­мента данное слово корректируется

слово 4 - число дополнительных байтов в каждой записи о файле в

сегменте (не используется),

слово 5 - номер блока на носителе, с которого размещаются файлы,

описанные в данном сегменте.

Запись о файле состоит из 8 слов и дополнительных байтов ( если указано в четвертом слове заголовка сегмента ):

слово 1 - тип записи,

слово 2-4 - имя файла

слово 5-6 - тип файла

слово 7 - число блоков, занимаемое файлом,

слово 8 - дата создания файла в формате:

разряды 11-15 - номер месяца (1-12),

разряды 06-010 - день (1-31),

разряды 00-05 - год минус 1980. Дополнительные слова не используются

**//Примечание: изначально в задании для даты были указаны другие наборы разрядов, однако в них не помещался год, и поэтому пришлось их изменить**

Тип записи может принимать следующие значения:

1- запись о временном файле,

2 - запись о свободной области на носителе. Размер свободной области в блоках содержится в 5-ом слове, остальные слова не используются,

3 - запись о постоянном файле,

4 - запись о постоянном файле, защищенном от операции удаления,

5 - признак конца записей в данном сегменте.

**//Примечание: изначально для типов записей в задании были указаны специальные коды(**001000 и т.д.**), но они не все помещались в переменную типа «слово», поэтому их было решено заменить на простые номера 1,2…**

**ПОЯСНЕНИЕ ЗАДАНИЯ**

Нам предлагается разработать модель файловой системы. Вся эта система должна быть где-то записана, мы её будем хранить в файле «filesystem.txt». Вся память в системе разбивается на блоки по 512 байт, поэтому удобно будет ввести тип данных «блок» - массив из 512 чаров:

typedef char block[512];

и читать\записывать информацию в файл такими блоками. Пример того, как это можно сделать:

#include <fstream>

#include <iostream>

using namespace std;

typedef char block[512];//Объявляем тип данных блок

void main()

{

block buf;

for(int i=0;i<512;i++)//Инициализируем буферный блок нулями

buf[i]='0';

//Создание файла и запись в него 10 блоков:

ofstream out;

out.open("filesystem.txt",ios::out|ios::binary);//Открываем файл на двоичный ввод

for(int i=0;i<10;i++)

{

out.write((char\*)&buf,sizeof block);//Пишем в файл

}

out.close();

//Произвольный доступ к файлу, забьем 5-й блок буквами x:

fstream inout;

inout.open("filesystem.txt",ios::in|ios::out|ios::binary);//Открываем файл для двоичного ввода\вывода

for(int i=0;i<512;i++)//Инициализируем буферный блок буквами x

buf[i]='x';

if(inout.is\_open())

{

inout.seekp(4\*sizeof block);//Переводим файловый указатель на нужную позицию

inout.write((char\*)&buf,sizeof block);//Записываем блок

}

else

{

cout<<"blyaaaa";

}

//Прочитаем первые элементы из блоков:

for(int i=0;i<10;i++)

{ inout.seekg(i\*sizeof(block));//Устанавливаем файловый указатель на нужный элемент

inout.read((char\*)&buf,sizeof block);//Читаем блок

cout<<buf[0];//Выводим первый элемент

}

inout.close();

cin.get();

}

**//Примечание: вообще весь процесс ввода/вывода в файл прекрасно описан в книжке С.Праты «Язык программирования C++» в 16 гл.**

Таким образом, мы можем создать файл нужного размера, прочитать из него блок с нужным номером, отредактировать в нем нужные байты и записать на место.

Согласно заданию, наш файл«filesystem.txt» имеет такое содержимое:

* Блоки 0 и 2-5 нам не нужны
* В блоке 1 в определенных байтах записана системная информация, в частности, имя владельца, метка тома и номер блока начала каталога
* С 6-го блока записывается каталог- таблица с информацией о расположении файлов
* Дальше лежат сами файлы

Главная часть - каталог(таблица расположения файлов). Он состоит из сегментов- участков по 2 блока. Общее количество сегментов в каталоге устанавливается при форматировании диска (при создании файла «filesystem.txt»), но не превышает 31. У каждого сегмента есть заголовок из 5 слов (**слово- это 2 байта**):

* 1-е слово- количество сегментов в каталоге
* 2-е слово- номер следующего сегмента
* 3-е слово- количество задействованных сегментов(если в системе мало файлов, то информация о них занимает не все сегменты)
* 4-е не используется
* 5-е номер блока, с которого начинаются файлы, описанные в данном сегменте

Дальше в сегменте следуют записи о файлах, они состоят из 8-ми слов, в которых записаны имя, тип, размер файла, дата создания и тип записи о файле. **Свободные участки памяти в таблице отображаются также как и файлы, но со специальным типом записи.**

Как происходит работа с таблицей расположения файлов:

* Первоначально, после того как файловая система отформатирована , в 1-м сегменте содержатся 2 записи- запись о всем свободном пространстве на диске(о нераспределенной памяти) и запись-признак конца записей в сегменте. В остальных сегментах содержится 1 запись-признак конца записей.
* При добавлении файла в систему в таблицу перед записью о нераспределенной памяти добавляется запись об этом файле, размер нераспределенной памяти уменьшается на размер этого файла. Запись о нераспределенной памяти и запись о конце записей сдвигаются вниз в таблице. По мере добавления файлов запись о нераспределенной памяти доходит до конца 1-го сегмента, затем переходит в следующий сегмент и т. д.
* При удалении файла тип записи о нем в таблице изменяется на «запись о свободной области». Позднее на это место в таблице может быть записан другой файл, размер которого меньше размеров этой области. В таком случае записываемый файл займет всю эту область(даже если ему не требуется вся область- иначе придется сдвигать все записи в таблице )
* Постепенно, по мере удаления файлов из системы, в таблице образуется много небольших свободных мест, которые затрудняют рациональное использование памяти. Чтобы прекратить это безобразие проводится дефрагментация- все свободные области сдвигаются в старшие адреса памяти, т.е. в конец таблицы, где образуется одна запись о нераспределенной памяти. Эта процедура довольно трудоемкая- приходится много раз сдвигать записи в таблице и переносить файлы на диске на новые места

После каталога следуют блоки, занятые файлами, причем файлы занимают непрерывные участки памяти. Записывать реальную информацию в них не требуется, но для реалистичности моделирования нужно пространство, занятое файлом, хотя бы забить единицами.

**ОПИСАНИЕ ПОДПРОГРАММ**

Содержание практически всех заданий сводится к работе с таблицей расположения файлов. Как уже было сказано, она состоит из сегментов. В файле «filesystem.txt» она хранится в виде блоков, поэтому для её обработки удобно создать специальный класс segment, что-то вроде этого:

class segment

{

unsigned short word1;//Переменные для слов заголовка сегмента

unsigned short word2;

unsigned short word3;

unsigned short word4;

unsigned short word5;

struct filerecord//Структура"запись о файле"

{

int rectype;//Тип записи

char filename[7];//Имя файла

char filetype[5];//Тип файла

unsigned short filesize;//Размер файла

unsigned short filedate;//Дата создания

};

filerecord filetable[63];//Сегмент содержит 63 записи о файлах

};

В классе можно реализовать специальные методы для записи в сегмент данных из 2-х блоков (инициализирующий конструктор), специальные методы для выполнения тех действий, которые должна выполнять подпрограмма и т.д.

Предполагается, что все процедуры будут писаться в заголовочных файлах .h, которые затем будут подключены к основному модулю программы.

1. **Создать файловую систему с заданными параметрами (форматировать диск)**

Процедура получает следующие аргументы:

Размер системы(в блоках)

Метка тома

Владелец

Процедура должна создать файл «filesystem.txt», записать в него укзанное количество пустых блоков, в системном блоке записать метку тома, владельца, версию системы(1.0?), название системы, номер блока начала каталога(6); разметить нужное количество сегментов каталога- записать в них содержимое заголовков, в первом сегменте добавить запись о всем свободном пространстве и признак конца записей, в остальных сегментах- признаки конца записей.

Размер сегмента-1024 байта, размер записи о файле- 16 байт, учитывая, что в сегменте есть заголовок, получим что сегмент описывает 62файла(1 запись-признак конца записей в сегменте). Файл занимает, как минимум, 1 блок, значит количество сегментов в каталоге- это (количество свободных блоков в системе)\63+1.

Процедура должна вернуть код завершения:

0-успешное завершение

1-не удалось записать файл «filesystem.txt»

1. **Вывести оглавление как оно есть**

Процедура не получает аргументов.

Процедура должна пройти по каталогу (по всем задействованным сегментам), просмотреть все записи до признака конца записей и вывести названия , типы и размер всех файлов.

Процедура должна вернуть код завершения:

0-успешное завершение

1-файл «filesystem.txt» не найден

1. **Вывести оглавление в алфавитном порядке**

То же, что и в предыдущей процедуре, только файлы сортируются по имени в алфавитном порядке.

1. **Вывести информацию о пустом месте на диске и его характеристиках**

То же самое, что и выше, только нужно найти в каталоге записи о пустых областях и вывести их общее количество и размеры каждой.

1. **Создать файл**

Процедура получает следующие параметры:

Имя файла

Тип файла

Размер

Процедура должна пройти по каталогу файлов и найти свободную область, в которую может поместиться файл. Если запись о такой области расположена в теле каталога(то есть свободная область осталась после удаления какого-либо файла), то область захватывается целиком и на место записи об этой свободной области вносится запись о файле. Если единственная область, в которую может поместиться файл- область нераспределенной памяти(последняя запись в каталоге, её признаки- номер текущего сегмента совпадает с содержимым счетчика задействованных сегментов(содержится в заголовке всех сегментов) и следующая запись-признак конца сегмента), так вот, если единственная область, в которую может поместиться файл- область нераспределенной памяти, то запись о нераспределенной памяти и следующий за ней признак конца записей сдвигаются на одну позицию вниз, перед записью о нераспределенной памяти добавляется запись о создаваемом файле, размер нераспределенной памяти уменьшается на размер файла. Если при сдвиге записей окажется, что сегмент заполнен записями(признак конца записей попал на 63-ю позицию), то нужно перейти на следующий сегмент, при этом в заголовках всех сегментов нужно увеличить счетчик задействованных сегментов. Может сложиться ситуация, что все имеющиеся сегменты полностью заполнены записями. В этом случае процедура должна вернуть код ошибки. Также может оказаться, что в системе недостаточно памяти для записи файла, при этом тоже должен возвращаться код ошибки.

Перед записью файла неплохо было бы пробежаться по всему каталогу и проверить, что файла с таким именем и типом не существует.

После записи файла нужно перейти в ту область памяти, куда записан файл и забить единичками байты, которые он занял.

Процедура возвращает следующие коды:

0-успешное завершение

1-файл «filesystem.txt» не найден

2-недостаточно места в системе

3-недостаточно места в каталоге

4-такой файл уже существует

1. **Удалить файл**

Процедура принимает следующие аргументы:

Имя файла

Тип файла

Процедура должна найти в каталоге запись о файле с заданными именем и типом и поменять тип записи на «запись о свободной области». Если файл не будет найден- вернуть код ошибки.

Процедура возвращает следующие коды:

0-успешное завершение

1-файл «filesystem.txt» не найден

2-файл с таким именем не найден

1. **Переименовать файл**

Процедура принимает следующие аргументы:

Старое имя файла

Старый тип файла

Новое имя файла

Новый тип файла

Аналогично предыдущему заданию, процедура ищет в каталоге запись о файле с заданными именем и типом и меняет их на новые.

Процедура возвращает следующие коды:

0-успешное завершение

1-файл «filesystem.txt» не найден

2-файл с таким именем не найден

1. **Добавить информацию в существующий файл**

Процедура принимает следующие аргументы:

Имя файла

Тип файла

Добавляемый размер

Процедура ищет в каталоге запись о файле с заданными именем и типом. Затем выполняется процедура «создание файла» для найденного файла с увеличенным на нужную величину размером, а тип старой записи меняется на «запись о свободной области». Задача может быть решена использованием готовой процедуры создания файла из задания №5.

Процедура возвращает следующие коды:

0-успешное завершение

1-файл «filesystem.txt» не найден

2-файл с таким именем не найден

3-недостаточно места в системе

4-недостаточно места в каталоге

1. **Изменение типа файла**

Процедура принимает следующие аргументы:

Имя файла

Старый тип файла

Новый тип файла

Процедура должна найти в каталоге запись о файле с заданными именем и типом и поменять тип на новый.

Процедура возвращает следующие коды:

0-успешное завершение

1-файл «filesystem.txt» не найден

2-файл с таким именем не найден

1. **Сжатие файловой системы**

Процедура не принимает параметров

Процедура должна пробежать по каталогу и найти записи о свободных областях, затем поочередно сдвинуть все эти записи к записи об области нераспределенной памяти(к концу каталога) и объединить с ней. При перемещении свободной области на диске должно производиться соответствующее перемещение содержимого памяти, т.е. данные на диске сдвигаются, заполняя свободную область.

Процедура возвращает следующие коды:

0-успешное завершение

1-файл «filesystem.txt» не найден

1. **Записать метку тома и владельца**

Процедура принимает следующие аргументы:

Метка тома

Владелец

Процедура должна перезаписать содержимое соответствующих байтов в блоке системной информации (в 1-м блоке).