**Лекция 5**

**lseek()** – сискол для того, чтобы читать не с начала файла, а с какого-то заданного места. Можно также поставить указатель за концом файла. Тогда между концом файла и местом, куда мы записали данные, окажется дыра из нулевых байт. Например, если записать в файл 10 байт “Hello world” со сдвигом в 50 байт, то в записанном файле первые 50 байт будут дырой, остальные 10 будут информативными (то есть по сути файл реально весит 10). Если же этот файл скопировать командой cp, то в новом файле уже будет не дыра, а реально в начале будет 50 штук нулевых байт.

**Директории**

Хотим реализовать команду ls.

**opendir()** – сискол для открытия директории

*DIR\* opendir(const char\* path)*

**readdir()** – прочитать директорию (дай мне следующий элемент директории пж). Когда все обошли, возвращается null.

*struct dirent\* readdir(DIR\* dir)*

**closedir()** – закрыть директорию

Проход по директориям (команда ls)

#include <iostream>  
#include <dirent.h>  
  
  
int main(int argc, char\* argv[]) {  
 const char\* path = (argc > 1) ? argv[1] : ".";  
 DIR\* dir = opendir(path);  
 if (dir == NULL) {  
 perror("Error opening directory");  
 return 1;  
 }  
 struct dirent\* entry;  
 while ((entry = readdir(dir)) != NULL) {  
 std::cout << entry->d\_name << '\n';  
 }  
 closedir(dir);  
}

*getdents, getdents 64* – get directory entries

Получить информацию о файле можно с помощью сискола

**stat** – get file status

*int stat(const char\* path, struct stat\* statbuf)* – передаем имя файла и указатель, куда положить информацию об этом файле.

**lstat** – если файл был символической ссылкой, то выдаст информацию не о файле, на который она ссылается, а о самой ссылке.

Пример использования stat

#include <iostream>  
#include <sys/stat.h>  
  
int main() {  
 const char\* filename = "output.txt";  
 struct stat file\_stat;  
 if (stat(filename, &file\_stat) == -1) {  
 perror("Error using stat");  
 return 1;  
 }  
 std::cout << filename << '\n';  
 std::cout << file\_stat.st\_size << '\n';  
 std::cout << file\_stat.st\_mode << '\n';  
 std::cout << file\_stat.st\_uid << '\n';  
 std::cout << file\_stat.st\_gid << '\n';  
}

**tee** – команда, которая позволяет продублировать вывод программы в файл, и вывод идет одновременно как в поток, так и в файл.

*./a.out | tee result.txt*

**Cискол dup**

**dup** – duplicate a file descriptor.

*int dup(int oldfd)* – берет старый файловый дескриптор и возвращает новый - копию, который указывает на то же, что и старый, при этом для нового дескриптора использует наименьший незанятый файловый дескриптор.

То есть с помощью этой команды можно подменить поток ввода на файл.

*Int dup2(int oldfd, int newfd)* – по сути присваивание файловых дескрипторов. Вместо использования наименьшего незанятого дескриптора используется newfd.

#include <iostream>  
#include <fcntl.h>  
  
int main() {  
 int fd = open("path", O\_WRONLY|O\_CREAT);  
 dup2(fd, 1);  
 std::cout << "blablabla";  
 std::cout.flush();  
 close(fd);  
}

**Перемещение и удаление файлов**

**ln** – создание ссылки на файл.

*ln -s result.txt a.txt* – теперь a.txt – ссылка на result.txt. Этот файл весит очень мало и хранит лишь имя файла, на который ссылается. В информации о файле добавляется буква l.

Это символическая ссылка (-s – symbolic).

По аналогии с плюсами a.txt – будто бы указатель. Если удалить result.txt, a.txt станет битой ссылкой. Если после удаления снова создать result.txt, ссылка опять станет валидной. Если переместить result.txt, ссылка испортится. Если же саму ссылку переместить в другую директорию, то она тоже испортится. Ссылка хранит относительный путь.

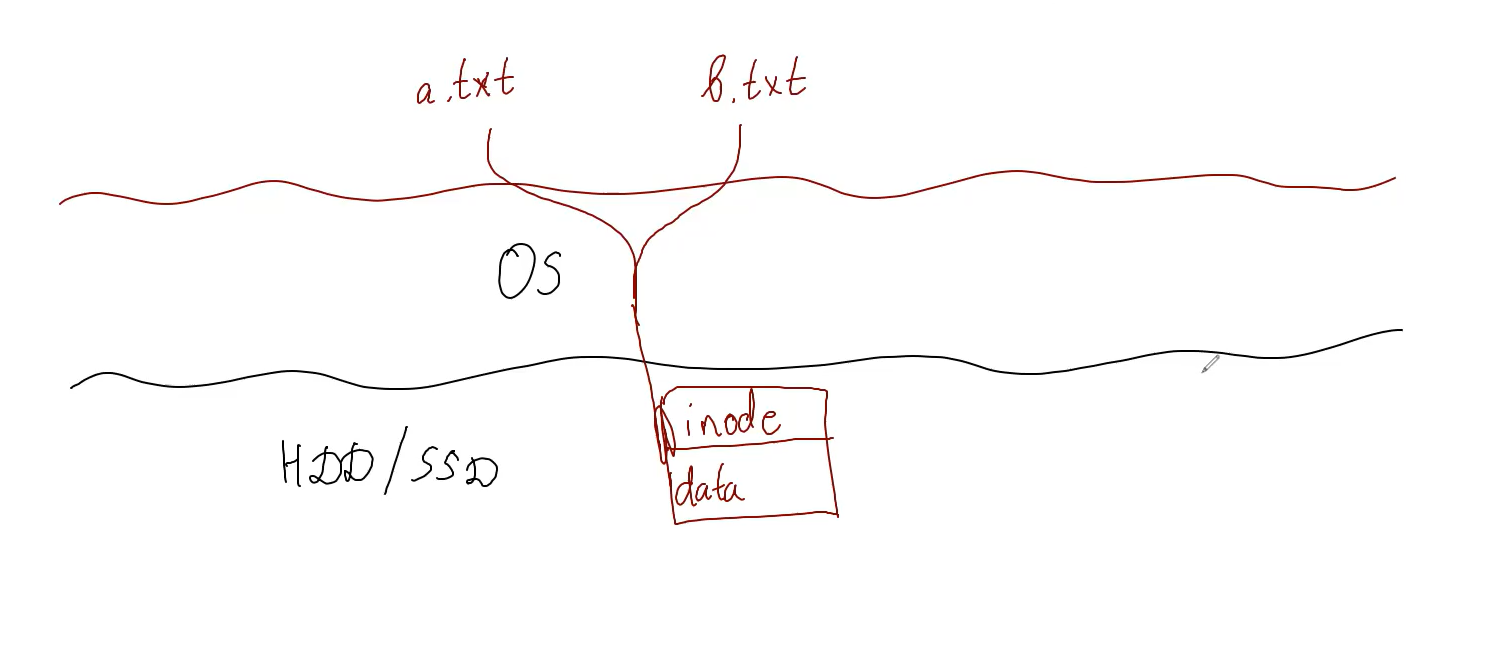
*ln result.txt a.txt* – создание hard link.

a.txt будет выглядеть как копия исходного файла, но на самом деле по сути это другое название исходного файла. Аналогия – ссылки в плюсах!

Также здесь работает аналогия с умными указателями. Все, что хранится на диске, имеет в файловой системе свои имена. ОС помнит про каждый файл, сколько файлов указывает на него, и это число hard links можно увидеть в ls -l.

Папка – тоже файл. У нее два имени – само имя и точка.

На жестком диске файлы проиндексированы id, которые называются **inode**. В нем хранится число, которое однозначно задает файл. При этом имен может быть несколько. Получается, что команда ln просто создает новое имя для заданной inode. Когда пишем команду rm, удаляется лишь одно из имен файла. А если ссылок(имен) не останется (и ни один процесс при удалении не держит файл открытым), тогда удалится сам физический файл. То есть hard links – что-то вроде shared\_ptr.



**ls -i** – выводит inode файлов

Как ln и rm реализованы с точки зрения сисколов?

Сискол **link** – создает новое имя файла

*int link(const char\* oldpath, const char\* newpath)*

Сискол **unlink** – забывает имя файла

*int unlink(const char\* path)*