Влияние буферизации на скорость работы системы

Ядро Linux, работает с файловыми системами, оперируя блоками, как минимальной единицей чтения/записи. Обращения некратные размеру блока могут существенно замедлять работу программы.

Можно убедиться на примере утилиты `dd`, как размер блока данных может повлиять на скорость работы (В обоих слуаях считывается 2 мегабайта данных).

```
$ dd bs=1 count=2097151 if=/dev/zero of=test
$ dd bs=1024 count=2048 if=/dev/zero of=test
```

Считывание посимвольно

```
#include <stdio.h>

int main()
{
    FILE *file_ptr = fopen("test.txt", "r"); //Открытие файла на чтение
    if(file_ptr != NULL)
    {
        /* Если файл был успешно открыт. */
        int counter = 0; //Переменная – счётчик символов
        char symb;       //Переменная для хранения считанного символа

/* Циклический вызов функции fgetc(), пока не будет возвращено значение EOF*/
        while( (symb = fgetc(file_ptr)) != EOF)
        {
            printf("%c", symb);
            counter++;
        }

        printf("\nCounted %d symbols.\n", counter);

        fclose(file_ptr);     //Закрытие файла
    }
    else
        perror("fopen");      //Печать ошибки, если файл не был открыт
}
```

При проверки программы на текстовых файлах стоит обратить внимание, что часто текстовые редакторы автоматически дописывают символ перехода на новую строку в конец файла.

Проверить наличие символа перехода на новую строку можно с помощью утилиты xxd, позволяющей просматривать двоичное представление файла.

```
$ xxd /path/to/filename
```

Пример:

```
$ xxd test.txt
00000000: 3132 3334 3536 3738 3930 0a              1234567890.
```

Здесь `0a` является символом перехода на новую строку.

Отключить запись символа перехода на новую строку в редакторе Vim можно с помощью команд:

```
:set binary
:set noeol
:w
```

Считывание построчно

```
#include <stdio.h>

#define N 15

int main()
{
    FILE *test = fopen("test2.txt", "r"); //Открытие файла
    if(test != NULL)    //Проверка открыт ли файл правильно
    {
        char buf[N] = {0}; //Создаем буфер для чтения (строку)
        int counter = 0;   //Создаём счётчик строк

/* Аналогично посимвольному считыванию, циклично вызываем функцию fgets() */
/* Когда не остаётся данных для считывания fgets() вернёт NULL.           */
        while( fgets(buf, N, test) != NULL)
        {
            printf("%s", buf); // Печатаем строку
            counter++;
        }

        printf("Counted %d strings\n", counter);

        fclose(test);   //Закрываем файл
    }
    else
        perror("fopen");

    return 0;
}
```

Функция `fgets()` заканчивает свою работу, если закончился буфер для чтения (`buf` в нашем примере), если был прочитан первый символ новой строки или если был достигнут конец файла.

В качестве экспериментов для исследования можно создать текстовый файл с простым содержанием

```
0123456789
0123456789
0123456789
```

И регулулируя значение константы N (10, 11, 12) оценить возвращаемое программое значение количества строк в файле.

Ниже приведены листинги двух программ, демонстрирующих возможности работы с двоичными файлами. Первая программа, осуществляет запись двоичных данных в файл, а вторая считывание. Хранение данных осуществляется в следующем формате: вначале файла располагается количество объектов структуры `person` (определенной в файле `struct.h`), затем последовательно располагаются сами объекты структур.

Запись двоичных данных в файл

```
#include <stdio.h>
#include "struct.h"

#define PERSONS_COUNT 3 //Количество объектов для записи

int main()
{
    int size = PERSONS_COUNT;

/* Массив объектов структуры person для записи. */
    struct person persons[] = {
        { "Vasya", 23, 4.0 },
        { "Petya", 24, 4.1 },
        { "Kolya", 25, 4.2}
    };

    FILE *out = fopen("persons.dat", "w"); //Открытие файла для записи
    if(out)
    {

      /*    Запись количества объектов (хранится в переменной size) */
        int ret = fwrite(&size, sizeof(size), 1, out);
        if(ret > 0)
        {
            /* Если размер был успешно записан, записываем объекты структуры. */
            printf("Size wrote.\n");
            ret = fwrite(persons, sizeof( struct person), size, out);
            if( ret != size)
            {
              /* Если записалось меньше, чем требовалось, то произошла ошибка */
                perror("fwrite");
            }
            else
                /* Всё записалось успешно. */
                printf("Persons wrote.\n");
        }
        else
            perror("fwrite");

        fclose(out); //Закрываем файл.
    }

    return 0;
}
```

struct.h

```
#define NAME_LEN 20
struct person
{
    char name[NAME_LEN];
    int  age;
    double average_mark;
};
```

Считывание двоичых данных из файла

```
#include <stdio.h>
#include <stdlib.h>
#include "struct.h"

int main()
{
    FILE *in = fopen("persons.dat", "r"); //Открываем файл для чтения
    if(in)
    {
        int size = 0;   //Переменная для хранения количества объектов

        /* Считывание количества объектов person, хранящихся в файле. */
        int ret = fread(&size, sizeof(size), 1, in);
        if (ret == 1)
        {
            printf("Persons count successfully read.\n");

     /* Динамическое выделение памяти под хранение считываемых объектов. */
            struct person *persons = (struct person*)malloc(sizeof(struct
person) * size);

              /* Считывание объектов структуры person из файла. */
            ret = fread(persons, sizeof(struct person), size, in);
            if (ret == size)
            {
                printf("Persons readed:\n");

                      /* Вывод информации для проверки. */
                for(int i = 0; i < size; i++)
                {
                    printf("Name: %s\n",(*(persons + i)).name);
                    printf("Age:  %d\n",(*(persons + i)).age);
                    printf("Average mark: %f\n\n",(*(persons +
i)).average_mark);
                }

            }
            else
                perror("fread");

            free(persons); //Освобождение выделенной памяти
        }

        fclose(in);
    }
    return 0;
}
```

В качестве практики можно переработать программу записи двоичных данных, адаптировав её для работы с произвольным количеством объектов, а так же добавить считывание входнных данных для структур, как с клавиатуры так и из текстового файла.

В программу чтения можно так же добавить поведение при котором считанные данные записываются в файл.