Lista de Exercícios 2 - OA

29/03/2018

Otto Kristian von Sperling
12/0131510

Gabriel Guimarães A. de Castro
15/0126425

Universidade de Brasília - UnB
Departamento de Ciência da Computação
Curso: Ciência da Computação - Bacharelado
Disciplina: Organização de Arquivos
Professor: Oscar Gaidos

Item 1

Fator de Blocagem:

- Descubra qual o tamanho da cluster do seu HD

- Faça um programa que gere cinco arquivos contendo os mesmos dados, cada um com

um fator de blocagem correspondente aos valores abaixo:

512 bytes, 1/4, 1/2, 3/4 e uma cluster.

- O tamanho do registro e da cluster devem poder ser passados via linha de comando.

Tamanho de *Cluster* = 4096 bytes

```
C:\WINDOWS\system32>fsutil fsinfo ntfsinfo c:
NTFS Volume Serial Number :        0xb4b4142fb413f29a
NTFS Version   :                   3.1
LFS Version    :                   2.0
Number Sectors :                   0x000000000724cfff
Total Clusters :                   0x0000000000e499ff
Free Clusters  :                   0x000000000076c096
Total Reserved :                   0x0000000000002965
Bytes Per Sector  :                512
Bytes Per Physical Sector :        4096
Bytes Per Cluster :                4096
Bytes Per FileRecord Segment    :  1024
Clusters Per FileRecord Segment :  0
Mft Valid Data Length :            0x000000002f280000
Mft Start Lcn  :                   0x0000000000c0000
Mft2 Start Lcn :                   0x0000000000000002
Mft Zone Start :                   0x000000000046ae60
Mft Zone End   :                   0x000000000046e520
Max Device Trim Extent Count :     1
Max Device Trim Byte Count :       0x8000000
Max Volume Trim Extent Count :     1
Max Volume Trim Byte Count :       0x8000000
Resource Manager Identifier :      2684F439-06B4-11E8-AFC9-DFB59910FFF1

C:\WINDOWS\system32>
```

CÓDIGO - C++11

Obs1 compilar com flag -std=c++11

Obs2 compilado e testado com gcc version 5.4.1 20160904 (Ubuntu 5.4.1-2ubuntu1~16.04)

Includes e funções

```
    //LE2_OA
#include <iostream>  /* defines FILENAME_MAX */
#include <sys/stat.h>
#include <fstream>
#include <string>
#include <cstdio>
#include <vector>
#include <cstdint>
#include <unistd.h>
#define GetCurrentDir getcwd

using namespace std;

string get_pathInfo(struct stat &buffer);
int save_inChunks(string in_fileName, string out_fileName, int custom_blockSize, int word_size);
int generateCustomFiles(string in_fileName, int word_size, int cluster);
```

Criando os 5 arquivos

```
int generateCustomFiles(string in_fileName,int word_size, int cluster)
{
    vector<string> output_names(5, "out_randomTxt");
    vector<int> custom_sizes(5);

    output_names[0].append("512bytes.txt");
    output_names[1].append("_25percent.txt");
    output_names[2].append("_50percent.txt");
    output_names[3].append("_75percent.txt");
    output_names[4].append("_Cluster.txt");

    custom_sizes[0] = (512 / word_size);
    custom_sizes[1] = ((cluster / 4) / word_size);
    custom_sizes[2] = ((cluster) / word_size);
    custom_sizes[3] = (((3 * cluster) / 4) / word_size);
    custom_sizes[4] = ((cluster) / word_size);

    for(int i = 0; i < output_names.size(); ++i)
    {
        if(save_inChunks(in_fileName, output_names[i], custom_sizes[i], word_size) != 0)
            return (-1);
    }

    return 0;
}
```

Coletando dados do HD onde os arquivos serão salvos

```
string get_pathInfo(struct stat &buffer)
{
    //char* cCurrentPath = param;
    char cCurrentPath[FILENAME_MAX];

    if(!GetCurrentDir(cCurrentPath, sizeof(cCurrentPath)))
        return "ERROR\n";   // if fail

    cCurrentPath[sizeof(cCurrentPath) - 1] = '\0'; // mark the end of the "string"
    stat((const char*)cCurrentPath, &buffer);

    return cCurrentPath;    //if success
}
```

Salvando os dados do arquivo fonte de acordo com as customizações de registro e cluster

```
int save_inChunks(string in_fileName, string out_fileName, int custom_blockSize, int word_size)
{
    ifstream fin;
    ofstream fout;
    char word[word_size];   //register size
    vector<string> buffer(custom_blockSize); //cluster size
    vector<string>::iterator it = buffer.begin();

    fin.open(in_fileName, ifstream::in | ifstream::binary); //opens and checks files
    if(!fin.is_open())
        return (-1);
    fout.open(out_fileName, ofstream::out | ofstream::binary);
    if(!fout.is_open())
        return (-1);

    filebuf *inbuf = fin.rdbuf();   //pointer to the file buffer
    int i = 0;
    while(true)
    {
        word[i] = inbuf->sbumpc();  // reads first element from file
        while((word[i] != EOF) && (i < word_size))
        {
            i++;
            word[i] = inbuf->sbumpc();  // reads next until EOF or the register is full
        }

        if(word[i] == EOF)
        {
            (*it) = word;
            for(it = buffer.begin(); it != buffer.end(); ++it)  // loads the register to the buffer (== cluster size)
                fout << (*it);
            fin.close();
            fout.close();
            return 0;
        }

        (*it) = word;
        i = 0;
        it++;
        if(it == buffer.end())
        {
            for(it = buffer.begin(); it != buffer.end(); ++it)  // loads the buffer to the end file
                fout << (*it);
            fout << "\n";
            it = buffer.begin();
        }
    }
}
```

Main()

```
int main(int argc, char** argv)
{
    int wordsize = 32; //default
    int clusterSize;
    string appPath;
    struct stat pathInfo;
    string in_file = "randomTxt.txt";

    cout << "ALUNOS\n\tOtto K. von Sperling - 12/0131510\n\tGabriel G. de Castro - 15/0126425\n";

    if(argc == 3) // if arguments passed in the commandline, use them
    {
        wordsize = atoi(argv[1]);
        clusterSize = atoi(argv[2]);
    }
    else if(argc == 1) // if no arguments passed in the commandline, get current directory info
    {
        appPath = get_pathInfo(pathInfo);   // gets directory info
        if(appPath == "ERROR")
            return (-1);
        clusterSize = pathInfo.st_blksize;  // atributes cluster size from current directory
    }
    else    // parameter input controll
    {
        cout << "Wrong number of parameters. Either none or two -> <register_size> <cluster_size>" << endl;
        cout << "\tNo parameters implies default 32bits for register and cluster size from PATH:" << endl;
        cout << "\t\t" << appPath << endl;
        return (-1);
    }

    if(generateCustomFiles(in_file, wordsize, clusterSize) != 0) // makes .txt files and checks for error
        return (-1);

    cout << "\tArquivos criados com sucesso. Por favor, verique o seu diretorio." << endl;
    return 0;
}
```

EXECUÇÃO

```
ottok92@Otto-Avell:~/Documents/UnB/2018-1/OA/LE2$ ./a.out
ALUNOS
        Otto K. von Sperling - 12/0131510
        Gabriel G. de Castro - 15/0126425
        Arquivos criados com sucesso. Por favor, verique o seu diretorio.
ottok92@Otto-Avell:~/Documents/UnB/2018-1/OA/LE2$
```

Item 2

Em uma fita de 2400 pés, densidade 30000 bpi, gap de tamanho 0,3 polegadas, ache o espaço necessário para armazenar 10000 registros de tamanho de 150 bytes.

8 bits são 1 byte, então cada registro de 150 byte será 150 * 8 = 1200 bits.

O total de bits em 10.000 registros será de 10.000 * 1200 = 12.000.00 bits.

Com densidade de 30.000 bpi, temos 12.000.000 / 30.000 = 400 polegadas.

Considerando um fator de blocagem (número de registros por bloco de dados) sendo de 45 registros por bloco e sabendo que temos 10.000 registros, precisaremos de 10.000 / 45 = 222 blocos (arredondado).

Com isso, teremos 221 gaps entre os blocos (IBG) e por consequência um gap total de 221 * 0.3 polegadas de IBG = 66 polegadas.

No total, teremos 400 + 66 = 466 polegadas de fita == 38 pés e 10 polegadas serão necessários para armazenar os 10.000 registros.

Item 3

Segundo a visão do projetista de arquivo, determine:

a. O fator do bloco, com perdas mínimas, para armazenar registros de 128 bytes, em setores de 512 bytes, cujo bloco não pode ser superior ao cluster de 1536 bytes. Cada Página é igual a 4 blocos.
b. Faça a mesma coisa com registros de 125 bytes.
c. Qual é a fragmentação interna no caso 3a e no caso 3b?

a. Para um cluster de 1536 bytes e setores de 512 bytes, sabemos que cada cluster contém 3 setores. Com registros de 128 bytes, precisamos de 1536(cluster) / 128(registro) = fator 12, e tendo que cada página tem 4 blocos, derivamos 12 / 4 = 3 registros por bloco(fator do bloco)

b. Resumindo a conta como no passo anterior, temos: 1536 bytes(cluster) / 125 bytes(registro) = 12.288, portanto precisariamos de 4 registros por bloco(fator de bloco 4), o que acarretaria em 16(4 blocos por página com 4 registros) - 12.288(fator ideal de armazenamento) = 3.712 → 3.712 * 125 bytes(por registro) = 464 bytes de fragmentação interna por cluster.

c. Como evidenciado nos itens anteriores, a fragmentação interna em a) é nula e em b) é igual a 464 bytes por cluster.

Item 4

Qual a vantagem de um arquivo em disco armazenado

a. numa única extensão?

A vantagem de armazenar dados de forma contínua é na movimentação das cabeças de disco que é algo mecânico e que é mais rápido dessa forma porque não terá que se deslocar para nenhuma parte distante do disco.

b. várias extensões distintas?

A vantagem é que há um desperdício menor de espaço. Por exemplo, se há 4 partes do disco vazias de 4 Mb continuos, e se há um arquivo de 25 Mb para ser armazenado, pode ser colocado 16Mb nessas lacunas e os outros 9 Mb em outro lugar, enquanto se tivesse que procurar um lugar contínuo essas 4 partes seriam desperdiçadas a não ser que houvesse arquivos menores que 4 Mb que coubessem nelas.

Item 5

Quais são as vantagens e desvantagens da organização de trilhas em setores com

grande capacidade?

Uma das vantagens de organizar os discos em trilhas com grandes setores é que para a gravação de arquivos grandes o processo é mais rápido. Pois além das várias cabeças de gravação em trilhas do cilindro, tem-se o fato de que os cilindros disponíveis podem estar distantes uns dos outros, e quanto maior os setores serão menores a quantidade de cilindros necessários para fazer essa gravação. Ou seja, quanto maior for um setor maior será um cluster(um grupo de setores), e maior o espaço contínuo para armazenamento dentro de um disco, isso torna a leitura ou escrita mais rápida já que serão menores os movimentos mecânicos que é a parte lenta.

Agora será maior o desperdício de espaço, pois o cluster é a menor unidade a ser reconhecida pelo SO e não pode pertencer a mais de um arquivo. Por exemplo:

Setor = 512 bytes

Tamanho do cluster = 4 setores = 4 * 512 bytes = 2048 bytes

Se um arquivo tem 2049 bytes, 2048 serão escritos em um cluster e o 1 restante em outro cluster deixando 2047 bytes que não poderão ser utilizados.