



Universidade Federal de Lavras  
PPGCC  
PCC508 – Sistemas Operacionais

## **Tópico 6 Lista Avaliativa**

Douglas Aquino T. Mendes  
10 de janeiro de 2025

# Sumário

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Questões</b>	<b>2</b>
2.1	Questão 1 . . . . .	2
2.2	Questão 2 . . . . .	2
2.3	Questão 3 . . . . .	2
2.4	Questão 4 . . . . .	3
2.5	Questão 5 . . . . .	3
<b>3</b>	<b>Desenvolver um programa</b>	<b>3</b>
3.1	Enunciado 6 . . . . .	3
3.1.1	Código . . . . .	4
3.1.2	Testes e Resultados . . . . .	4
3.2	Enunciado 7 . . . . .	4
3.2.1	Código . . . . .	5
3.2.2	Testes e Resultados . . . . .	7

# 1 Introdução

Este documento tem como objetivo apresentar o desenvolvimento das atividades avaliativas para o tópico 6 da disciplina de Sistemas Operacionais, focando na implementação de códigos em linguagem C. Serão apresentadas as questões, a resolução, os códigos desenvolvidos, seguidos de uma explicação sobre sua lógica de funcionamento.

## 2 Questões

### 2.1 Questão 1

**Pergunta:** 1) Explique como nomes longos de arquivos podem ser manipulados em diretórios

**Resposta:** Uma opção é manter os nomes dos arquivos em um heap no final de cada diretório. Cada entrada do diretório contém então um ponteiro para o nome do arquivo no heap. Essa abordagem evita a fragmentação e garante que uma nova entrada de arquivo sempre caiba no diretório. No entanto, o heap precisa ser gerenciado e podem ocorrer falhas de página ao processar nomes de arquivos [1, p. 145].

### 2.2 Questão 2

**Pergunta:** 2) Como funciona o Journaling em um sistema de arquivos? Explique.

**Resposta:** O journaling em um sistema de arquivos é uma técnica que aumenta a “robustez” do sistema diante de falhas, como quedas de energia ou travamentos. A ideia central é registrar as operações que serão realizadas no sistema de arquivos em um diário (journal) antes de executá-las [1, p. 145].

- As operações de escrita são primeiramente registradas no diário.
- O diário é escrito no disco e verificado para garantir a integridade da escrita.
- Somente após a confirmação da escrita no diário, as operações de escrita no sistema de arquivos são executadas.
- Após a conclusão bem-sucedida das operações, a entrada correspondente no diário é apagada.
- Se ocorrer uma falha durante esse processo, o sistema de arquivos, ao ser reiniciado, consulta o diário para identificar as operações pendentes. As operações pendentes são então reexecutadas até que sejam concluídas com sucesso.
- Para garantir o funcionamento adequado do journaling, as operações registradas no diário devem ser idempotentes, o que significa que podem ser repetidas múltiplas vezes sem causar efeitos colaterais negativos.

### 2.3 Questão 3

**Pergunta:** 3) Explique como funciona a alocação de blocos de arquivos baseadas em Inodes.

**Resposta:** Funcionamento da alocação de blocos baseada em inodes:

- Criação do Inode: Quando um arquivo é criado, o sistema de arquivos aloca um inode (abreviação de “index node”) é uma estrutura de dados que contém informações sobre um arquivo) para ele e armazena os atributos do arquivo no inode.
- Alocação de Blocos: O inode contém uma lista de endereços de blocos de disco. Inicialmente, essa lista pode estar vazia. Conforme o arquivo cresce e os dados são gravados, o sistema de arquivos aloca blocos de disco para o arquivo e adiciona seus endereços à lista no inode.

- **Acesso aos Dados:** Quando um processo precisa acessar os dados do arquivo, o sistema de arquivos usa o nome do arquivo para localizar seu inode. Em seguida, o sistema de arquivos usa a lista de endereços de blocos no inode para acessar os blocos de disco que contêm os dados do arquivo.

## 2.4 Questão 4

**Pergunta:** 4) Explique como funciona uma tabela FAT (File Allocation Table).

**Resposta:** A FAT funciona como uma tabela na memória principal, onde cada entrada corresponde a um bloco no disco. Cada entrada na tabela contém um ponteiro para o próximo bloco do arquivo, ou um marcador especial para indicar o fim do arquivo ou um bloco defeituoso [1, p. 196, 222]. O funcionamento da FAT pode ser ilustrado da seguinte forma:

- **Criação de um Arquivo:** Quando um arquivo é criado, o sistema de arquivos aloca o primeiro bloco disponível no disco e registra o número desse bloco na entrada do diretório do arquivo. A entrada na FAT correspondente ao bloco alocado aponta para um marcador especial, indicando o fim do arquivo.
- **Escrita de Dados:** Quando dados são escritos no arquivo, o sistema de arquivos aloca outro bloco livre e atualiza a entrada na FAT do bloco anterior para apontar para o novo bloco alocado. A entrada na FAT do novo bloco, por sua vez, aponta para o marcador de fim de arquivo.
- **Leitura de Dados:** Para ler os dados do arquivo, o sistema de arquivos começa na entrada do diretório do arquivo, que contém o número do primeiro bloco. O sistema consulta a FAT para obter o número do próximo bloco e continua seguindo a cadeia de ponteiros até o marcador de fim de arquivo.

## 2.5 Questão 5

**Pergunta:** 5) Quando uma aplicação necessita de um controle mais aprimorado sobre as permissões associadas a um determinado arquivo, as Access Control Lists (ACL) podem ser utilizadas para isso. Descrever uma visão geral de como as ACLs funcionam.

**Resposta:**

- Uma ACL é composta por uma lista de ACEs. Cada ACE especifica um usuário ou grupo e as permissões concedidas ou negadas a eles para o arquivo em questão.
- As permissões podem incluir leitura, escrita, execução e outras ações específicas, como "anexar" para arquivos ou "listar conteúdo" para diretórios.
- As ACLs podem ser configuradas para herdar permissões de diretórios pai, simplificando a administração de permissões em estruturas hierárquicas de arquivos.

**Exemplo:** Imagine um arquivo confidencial de um projeto. Com as ACLs, é possível conceder permissões de leitura e escrita aos membros da equipe do projeto. Conceder permissão de leitura apenas para o gerente do projeto. Negar explicitamente o acesso a todos os outros usuários, independentemente de suas permissões de grupo.

# 3 Desenvolver um programa

## 3.1 Enunciado 6

**Enunciado:** A chamada `statvfs(...)` é utilizada para obter-se informações sobre um sistema de arquivos montado. Faça um programa que receba como parâmetro o caminho de um sistema de arquivos e apresente as informações obtidas através desta chamada.

### 3.1.1 Código

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/statvfs.h>
4
5 void print_filesystem_info(const char *path) {
6     struct statvfs fs_info;
7
8     if (statvfs(path, &fs_info) != 0) {
9         perror("Erro ao obter informations do sistema de arquivos");
10        exit(EXIT_FAILURE);
11    }
12
13    printf("Informations sobre o sistema de arquivos em '%s':\n", path);
14    printf("Tamanho de bloco: %lu bytes\n", fs_info.f_bsize);
15    printf("Tamanho de bloco preferido para I/O: %lu bytes\n", fs_info.f_frsize);
16    printf("Nuber total de blocos: %lu\n", fs_info.f_blocks);
17    printf("Nuber de blocos disponibles para user not root: %lu\n", fs_info.f_bavail);
18    printf("Nuber de blocos livres: %lu\n", fs_info.f_bfree);
19    printf("Nuber total de inodes: %lu\n", fs_info.f_files);
20    printf("Nuber de inodes livres: %lu\n", fs_info.f_ffree);
21    printf("Nuber de inodes available para user not root: %lu\n", fs_info.f_favail);
22    printf("ID do sistema de arquivos: %lu\n", fs_info.f_fsid);
23    printf("Flags do sistema de arquivos: %lu\n", fs_info.f_flag);
24    printf("Tamanho max de nome de arquivo: %lu\n", fs_info.f_namemax);
25 }
26
27 int main(int argc, char *argv[]) {
28     if (argc != 2) {
29         fprintf(stderr, "Uso: %s <caminho do sistema de arquivos>\n", argv[0]);
30         return EXIT_FAILURE;
31     }
32
33     print_filesystem_info(argv[1]);
34
35     return EXIT_SUCCESS;
36 }
```

### 3.1.2 Testes e Resultados

Como resultado da execução do código exibido na subceção 3.1.1, obtivemos a saída ilustrada na figura 1.

## 3.2 Enunciado 7

**Enunciado:** Criar dois programas. O primeiro cria um arquivo binário, com 30 ints, realizando uma contagem (1,2,3,4...30). Isso significa que os 30 ints devem ser armazenados em sequência no arquivo. O segundo programa deve utilizar a função `readv` para fazer a leitura simultânea em múltiplos buffers de 8 números do arquivo gerado pelo primeiro programa. Essa leitura deve ser feita somente com uma instrução `readv`. Os números a serem lidos devem ser a partir do décimo armazenado (décimo no buffer 1, décimo primeiro no

```

a/Codes/output$ ./atv6 /
Informações sobre o sistema de arquivos em '/':
Tamanho de bloco: 4096 bytes
Tamanho de bloco preferido para I/O: 4096 bytes
Número total de blocos: 12774522
Número de blocos disponíveis para usuário não root: 1735095
Número de blocos livres: 2392119
Número total de inodes: 3268608
Número de inodes livres: 2485959
Número de inodes disponíveis para usuário não root: 2485959
ID do sistema de arquivos: 3877234821187264138
Flags do sistema de arquivos: 4096
Tamanho máximo de nome de arquivo: 255

```

Figura 1: Resultado da execução do programa

buffer 2, e assim por diante). Imprimir os números na tela. Não esqueça que cada int tem o seu tamanho em bytes fixo.

### 3.2.1 Código

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define NUM_INTS 30
5 #define FILENAME "data.bin"
6
7 int main() {
8     FILE *file = fopen(FILENAME, "wb");
9     if (file == NULL) {
10         perror("Erro ao criar o arquivo");
11         return EXIT_FAILURE;
12     }
13
14     for (int i = 1; i <= NUM_INTS; i++) {
15         fwrite(&i, sizeof(int), 1, file);
16     }
17
18     fclose(file);
19     printf("Arquivo .bin '%s' criado com %d inteiros.\n", FILENAME, NUM_INTS);
20     return EXIT_SUCCESS;
21 }

```

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <fcntl.h>
4 #include <sys/uio.h>
5 #include <unistd.h>
6
7 #define FILENAME "data.bin"
8 #define START_INDEX 10
9 #define BUFFER_SIZE 8

```

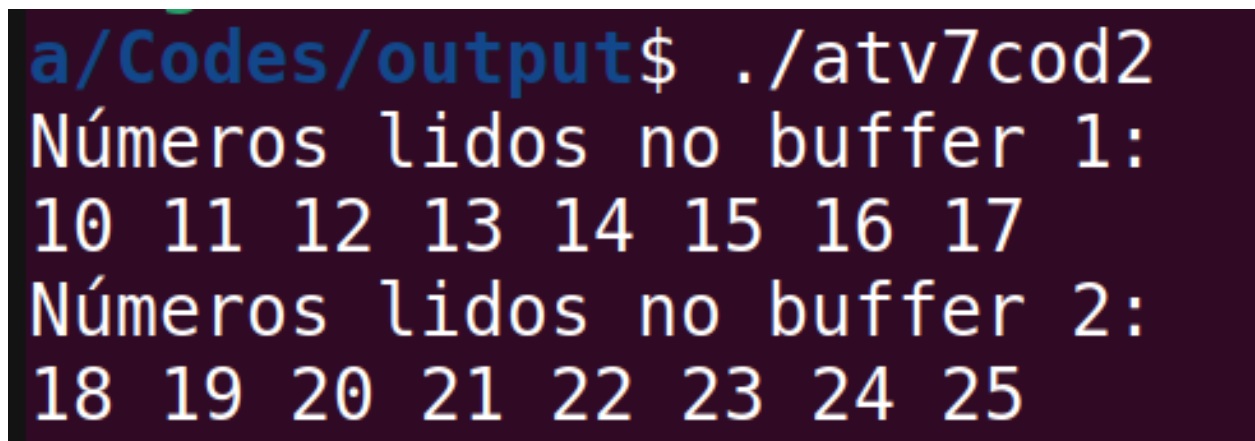
```

10 #define NUMBUFFERS 2
11
12 int main() {
13     int fd = open(FILENAME, O_RDONLY);
14     if (fd < 0) {
15         perror("Erro ao abrir o arquivo");
16         return EXIT_FAILURE;
17     }
18
19     // Buffers para armazenar os nùbers lidos
20     int buffer1[BUFFER_SIZE] = {0};
21     int buffer2[BUFFER_SIZE] = {0};
22
23     // Configurando os vectores para leitura
24     struct iovec iov[NUMBUFFERS];
25     iov[0].iov_base = buffer1;
26     iov[0].iov_len = sizeof(buffer1);
27     iov[1].iov_base = buffer2;
28     iov[1].iov_len = sizeof(buffer2);
29
30     // Definir a position inicial para leitura (10 nùber)
31     off_t offset = (STARTINDEX - 1) * sizeof(int);
32     if (lseek(fd, offset, SEEK_SET) == -1) {
33         perror("Erro ao posicionar no arquivo");
34         close(fd);
35         return EXIT_FAILURE;
36     }
37
38     // Leitura com readv
39     ssize_t bytes_read = readv(fd, iov, NUMBUFFERS);
40     if (bytes_read < 0) {
41         perror("Erro na leitura com readv");
42         close(fd);
43         return EXIT_FAILURE;
44     }
45
46     // Exibir os nùbers lidos
47     printf("Nùbers lidos no buffer 1:\n");
48     for (int i = 0; i < BUFFER_SIZE; i++) {
49         printf("%d ", buffer1[i]);
50     }
51     printf("\n");
52
53     printf("Nùbers lidos no buffer 2:\n");
54     for (int i = 0; i < BUFFER_SIZE; i++) {
55         printf("%d ", buffer2[i]);
56     }
57     printf("\n");
58
59     close(fd);
60     return EXIT_SUCCESS;
61 }

```

### 3.2.2 Testes e Resultados

Como resultado da execução do código exibido na subseção 3.2.1, obtivemos a saída ilustrada na figura 2. Fica a observação de que não tenho certeza se entendi corretamente o que deveria ser implementado.

A terminal window with a dark purple background and light blue text. The prompt is 'a/Codes/output\$'. The command executed is './atv7cod2'. The output consists of two lines of text: 'Números lidos no buffer 1:' followed by the numbers 10, 11, 12, 13, 14, 15, 16, and 17 on the next line; and 'Números lidos no buffer 2:' followed by the numbers 18, 19, 20, 21, 22, 23, 24, and 25 on the next line.

```
a/Codes/output$ ./atv7cod2
Números lidos no buffer 1:
10 11 12 13 14 15 16 17
Números lidos no buffer 2:
18 19 20 21 22 23 24 25
```

Figura 2: Resultado da execução do programa

## Referências

- [1] A. S. Tanenbaum e H. Bos, *SISTEMAS OPERACIONAIS MODERNOS*, 4<sup>a</sup> ed. Boston: Pearson, 2021.