Trabalho 2: Sistemas Operacionais

Mini Sistema Com Gerenciamento de Memória e Sistema de Arquivos Próprios.

Guilherme Sales Fernandes
Departmento de Computação
Universidade Federal do Ceará

February 27, 2025

Overview

Visão Geral

- Sistema de arquivos virtual baseado em um disco virtual armazenado como um arquivo (virtual_disk.img).
- Criação, leitura, ordenação, concatenação e exclusão de arquivos.
- Os arquivos armazenam números inteiros de 32 bits.
- Reutilização de espaço de arquivos apagados.
- Uso de Huge Pages (2MB) para otimizar a ordenação.
- Swap interno no disco virtual.

Estrutura do Sistema de Arquivos

Uma lista de até 1000 arquivos (FileEntry), onde cada entrada armazena: (Nome do arquivo; Quantidade de inteiros armazenados; Localização no disco; Indica se o arquivo ainda está ativo). O FileSystem: (Um contador de arquivos e um ponteiro para o espaço livre)

```
1  typedef struct {
2      char name[MAX_FILENAME_LEN];
3      uint32_t size;
4      uint32_t offset;
5      int valid;
6  } FileEntry;
7
8  typedef struct {
9      FileEntry files[MAX_FILES];
10     uint32_t file_count;
11     uint32_t free_offset;
12 } FileSystem;
```

Inicialização do Disco

Verifica se o arquivo existe. Se não existir, cria um novo com tamanho fixo de 1GB.

```
void init_fs() {
    FILE *disk = fopen(DISK_FILE, "rb");
    if (!disk) {
        disk = fopen(DISK_FILE, "wb");
        if (!disk || ftruncate(fileno(disk), DISK_SIZE) == -1) {
            perror("Erro ao definir tamanho do disco virtual");
            if (disk) fclose(disk);
            return;
        }
        fclose(disk);
        memset(&fs, 0, sizeof(fs));
    } else {
        fread(&fs, sizeof(fs), 1, disk);
        fclose(disk);
    }
}
```

Criar Arquivo

Evita criar arquivos duplicados. Reutiliza espaço de arquivos deletados se possível, se não, aloca um novo espaço e grava os números gerados.

```
void create_file(const char *name, uint32_t count) {
    for (uint32_t i = 0; i < fs.file_count; i++) {
        if (!fs.files[i].valid && fs.files[i].size >= count) {
            strncpy(fs.files[i].name, name, MAX_FILENAME_LEN); //reuse
            fs.files[i].valid = 1;
            fs.files[i].size = count;
            save_fs();
            return;
        }
    }

}
```

Criar Arquivo

Pt.2: Gerar números aleatórios e atualizar estado de `FileEntry`

```
    FILE *disk = fopen(DISK_FILE, "rb+");
    fseek(disk, fs.free_offset + sizeof(fs), SEEK_SET);
    for (uint32_t i = 0; i < count; i++) {
        uint32_t num = rand();
        fwrite(&num, sizeof(num), 1, disk);
    }

    FileEntry *file = &fs.files[fs.file_count++];
    strncpy(file->name, name, MAX_FILENAME_LEN);
    file->size = count;
    file->offset = fs.free_offset;
    file->valid = 1;
    fs.free_offset += count * sizeof(uint32_t);

    fclose(disk);
    save_fs();
```

Visão Geral

Ordenação por blocos utilizando o *QuickSort* - `qsort()` da biblioteca `stdlib.h`.

- Aloca uma Huge Page (2MB) na memória para processar os dados em partes.
- Lê o arquivo em blocos, ordena cada bloco e o armazena temporariamente em um espaço de *swap* dentro do próprio disco.
- Escreve os dados ordenados de volta ao local original do arquivo.
- Libera a memória e fecha os arquivos.

Alocar HUGEPAGE

```
1     uint32_t *buffer = mmap(NULL, HUGE_PAGE_SIZE, PROT_READ | PROT_WRITE
          ,
2                             MAP_PRIVATE | MAP_ANONYMOUS | MAP_HUGETLB,
                                 -1, 0);
3     if (buffer == MAP_FAILED) {
4         perror("Erro␣ao␣alocar␣Huge␣Page");
5         close(fd);
6         return;
7     }
```

Definição do Swap

```
1     uint32_t swap_offset = fs.free_offset;
2     uint32_t num_pages = (fs.files[i].size * sizeof(uint32_t) +
          HUGE_PAGE_SIZE - 1) / HUGE_PAGE_SIZE;
3
4     FILE *disk = fdopen(fd, "rb+");
```

Ordenar Blocos

```
1     for (uint32_t page = 0; page < num_pages; page++) {
2         fseek(disk, fs.files[i].offset + sizeof(fs) + page *
              HUGE_PAGE_SIZE, SEEK_SET);
3         size_t num_elements = fread(buffer, sizeof(uint32_t), PAGE_SIZE,
               disk);
4         qsort(buffer, num_elements, sizeof(uint32_t), compare);
```

Escrever Ordenação no Swap

```
1           fseek(disk, swap_offset + page * HUGE_PAGE_SIZE, SEEK_SET);
2           fwrite(buffer, sizeof(uint32_t), num_elements, disk);
3       }
```

Escrever Ordenação no Arquivo

```
1     for (uint32_t page = 0; page < num_pages; page++) {
2         fseek(disk, swap_offset + page * HUGE_PAGE_SIZE, SEEK_SET);
3         size_t num_elements = fread(buffer, sizeof(uint32_t), PAGE_SIZE,
              disk);
4
5         fseek(disk, fs.files[i].offset + sizeof(fs) + page *
              HUGE_PAGE_SIZE, SEEK_SET);
6         fwrite(buffer, sizeof(uint32_t), num_elements, disk);
7     }
```

Liberar Memória

```
    munmap(buffer, HUGE_PAGE_SIZE);
    fclose(disk);
    close(fd);

    clock_t end_time = clock();
    printf("Arquivo ordenado com sucesso em %.6f segundos.\n",
           (double)(end_time - start_time) / CLOCKS_PER_SEC);
}
```

Apagar Arquivo

Marcar um arquivo como inválido para reutilização futura.

```
1  void delete_file(const char *name) {
2      for (uint32_t i = 0; i < fs.file_count; i++) {
3          if (fs.files[i].valid && strcmp(fs.files[i].name, name) == 0) {
4              fs.files[i].valid = 0;
5              save_fs();
6              printf("Arquivo ’%s’ apagado.\n", name);
7              return;
8          }
9      }
10     printf("Arquivo ’%s’ nao encontrado.\n", name);
11 }
```

Concatenar Arquivos

Juntar dois arquivos no primeiro, liberando espaço do segundo.

- Posicionar no final de file1 antes de copiar file2
- Copiar conteúdo de file2 para o final de file1
- Atualizar o tamanho de file1 e marcar file2 como deletado

```
1  void delete_file(const char *name) {
2      for (uint32_t i = 0; i < fs.file_count; i++) {
3          if (fs.files[i].valid && strcmp(fs.files[i].name, name) == 0) {
4              fs.files[i].valid = 0;
5              save_fs();
6              printf("Arquivo ’%s’ apagado.\n", name);
7              return;
8          }
9      }
10     printf("Arquivo ’%s’ nao encontrado.\n", name);
11 }
```

Listar Arquivos

- Itera sobre a tabela de arquivos (fs.files) e verifica se o arquivo está ativo (`valid == 1`).
- Exibe o nome do arquivo e seu tamanho em bytes (`size * sizeof(uint32_t)`).
- Calcula o espaço total e o espaço livre no disco virtual.

```
1  void list_files() {
2      uint32_t total_size = 0;
3      printf("Arquivos no diret rio:\n");
4      for (uint32_t i = 0; i < fs.file_count; i++) {
5          if (fs.files[i].valid) {
6              printf("%s - %u bytes\n", fs.files[i].name, fs.files[i].size
                   * (uint32_t)sizeof(uint32_t));
7              total_size += fs.files[i].size * (uint32_t)sizeof(uint32_t);
8          }
9      }
10     printf("Espaco total: %lu bytes\n", (unsigned long)DISK_SIZE);
11     printf("Espaco disponivel: %lu bytes\n", (unsigned long)(DISK_SIZE -
           sizeof(fs) - fs.free_offset));
12 }
```

Ler Arquivos

- Procura pelo arquivo name na tabela de arquivos (`fs.files`) e verifica se ele está ativo (`valid == 1`).
- Abre o disco virtual (`virtual_disk.img`) para leitura. Posiciona o ponteiro no início da região solicitada (`fseek`).
- Lê e exibe os números inteiros do intervalo `[start, end]`.

Ler Arquivos

```
1  void read_file(const char *name, uint32_t start, uint32_t end) {
2      for (uint32_t i = 0; i < fs.file_count; i++) {
3          if (fs.files[i].valid && strcmp(fs.files[i].name, name) == 0) {
4              FILE *disk = fopen(DISK_FILE, "rb");
5              if (!disk) {
6                  perror("Erro ao abrir o disco virtual");
7                  return;}
8              fseek(disk, fs.files[i].offset + sizeof(fs) + start * sizeof
                   (uint32_t), SEEK_SET);
9              for (uint32_t j = start; j <= end && j < fs.files[i].size; j
                   ++) {
10                 uint32_t num;
11                 if (fread(&num, sizeof(num), 1, disk) != 1) {
12                     perror("Erro ao ler do arquivo");
13                     break;}
14                 printf("%u ", num);}
15             printf("\n");
16             fclose(disk);
17             return;}}
18     printf("Arquivo '%s' nao encontrado.\n", name);}
```

Fim