Sistemas de Arquivos

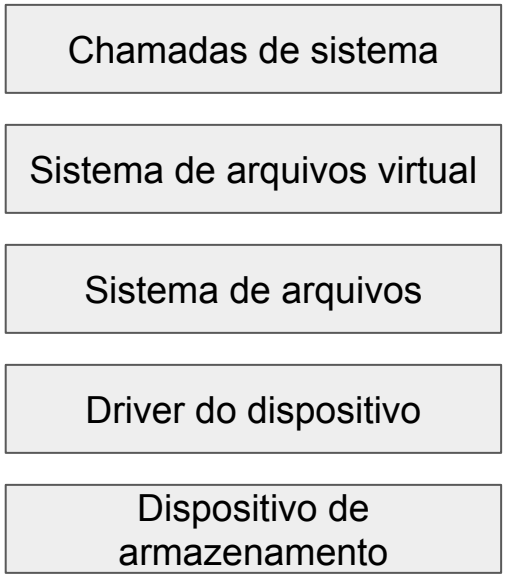
Wedson Almeida Filho

Dispositivos de armazenamento

- Grande diversidade
  - Discos rígidos rotacionais
  - SSDs (*solid state drives*)
  - RAID (*redundant array of independent disks*)
  - NBD (network block device)
  - Virtualizados

- A interface com o sistema operacional é comum

Pilha de armazenamento



Pontos em comum

- Armazenamento de alta capacidade
  - Persistente
- Mais lentos que memória volátil
- Granularidade de setor/bloco
  - Mínimo de 512 bytes
  - Potência de 2 normalmente: 1024, 2048, 4096, etc.
- Interface comum
  - Propriedades: tamanho e quantidade de blocos
  - Escrever blocos contíguos
  - Ler blocos contíguos
  - Descartar blocos contíguos

Abstração de armazenamento

- Dado um computador/celular com um dispositivo de bloco
- Como expor esse dispositivo?
  - Múltiplos processos
  - Múltiplos usuários
- Como evitar que defeitos em um programa afetem outros?

Sistemas de arquivos

- Introdução do conceito de **arquivo**
  - Uma sequência de bytes com uma **identidade**
  - Salvos em um dispositivo de armazenamento

- Nos primeiros sistemas de arquivos
  - Nomes tinham limites pequenos (e.g., 8 caracteres para o nome e 3 para a extensão)
  - Havia um número limitado de arquivos

Interface POSIX

```c
// Abertura, criação e fechamento de arquivos.
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
int close(int fd);

// Leitura, escrita e posicionamento de arquivos.
ssize_t write(int fd, const void *buf, size_t count);
ssize_t read(int fd, void *buf, size_t count);
off_t lseek(int fd, off_t offset, int whence);

// Remoção e truncamento de arquivos.
int ftruncate(int fd, off_t length);
int unlink(const char *pathname);
```

Prática: um sistema de arquivos de somente leitura

- Armazenamento em um dispositivo de blocos
  - Tamanho do bloco pode ser fixo (e.g., 512 bytes)
- Arquivos serão somente lidos
- Busca por arquivos eficiente

- Qual seria um formato adequado?

Projeto

- Cabeçalho (primeiro bloco):
  - Assinatura (MIFS, 0x5346494d)
  - Tamanho do bloco (4 bytes)
  - Número de arquivos (8 bytes)
- Lista de arquivos (segundo bloco em diante)
  - Cada registro de arquivo tem tamanho fixo (32 bytes)
    - Nome com tamanho máximo (20 bytes)
      - Terminado com zero se mais curto que 20 bytes
    - Bloco inicial (4 bytes)
    - Tamanho do arquivo (8 bytes)
  - Lista é ordenada
- Conteúdo dos arquivos (blocos seguintes à lista de arquivos)
  - Alinhados em blocos

Solução: estruturas

```
struct super {
   uint32_t assinatura;
   uint32_t tamanho_bloco;
   uint64_t nr_arquivos;
} __attribute__((packed));

struct arquivo {
   char nome[20];
   uint32_t bloco_inicial;
   uint64_t tamanho;
} __attribute__((packed));
```

Prática: escrever um leitor desse sistema de arquivos

- Duas funcionalidades
- Enumerar arquivos
  - `./leitor <imagem.img>`
- Extrair arquivo
  - `./leitor <imagem.img> <arquivo-para-extr`

- Exemplos de imagens:
  - https://drive.google.com/drive/folders/1W2DpfboiBxQo2EZFm_c3JugPu84SnRH-

Solução: mapear a imagem inteira

```
int fd = open(image_path, O_RDONLY);
if (fd < 0) {
        perror("open");
        exit(1);
}

struct stat st;
if (fstat(fd, &st) < 0) {
        perror("fstat");
        exit(1);
}

size_t image_size = st.st_size;
void *map = mmap(NULL, image_size, PROT_READ, MAP_PRIVATE, fd, 0);
if (map == MAP_FAILED) {
        perror("mmap");
        exit(1);
}
```

Solução: verificar super-bloco

```
struct super *sb = (struct super *)map;
if (sb->assinatura != 0x5346494D) {
        fprintf(stderr, "Invalid MIFS signature\n");
        exit(1);
}
```

Solução: enumerar arquivos

```
uint32_t block_size = sb->tamanho_bloco;
struct arquivo *table = (struct arquivo *)((uint8_t *)map + block_size);

if (argc == 2) {
	for (uint64_t i = 0; i < sb->nr_arquivos; i++) {
		printf("%.*s\n", 20, table[i].nome);
	}
	exit(0);
}
```

Solução: encontrar arquivo

```c
int compare_nome(const void *key, const void *elem) {
    const char *name = key;
    const struct arquivo *arq = elem;
    if (strlen(name) > 20) {
        return 1;
    }
    return strncmp(name, arq->nome, sizeof(arq->nome));
}

    struct arquivo *found = bsearch(
        target_name, table, sb->nr_arquivos, sizeof(struct arquivo), compare_nome);
    if (!found) {
        fprintf(stderr, "File '%s' not found in image.\n", target_name);
        exit(1);
    }

    size_t offset = (size_t)found->bloco_inicial * block_size;
    if (offset + found->tamanho < tamanho || offset + found->tamanho > image_size) {
        fprintf(stderr, "Invalid file data range\n");
        exit(1);
    }
```

Solução: escrever arquivo

```c
int write_all(int fd, const void *buf, size_t count)
{
   const char *p = buf;
   while (count > 0) {
       ssize_t written = write(fd, p, count);
       if (written < 0) {
           return -1;
       }
       p += written;
       count -= written;
   }
   return 0;
}

   int out_fd = open(target_name, O_WRONLY | O_CREAT | O_TRUNC, 0644);
   if (out_fd < 0) {
       perror("open output");
       exit(1);
   }

   if (write_all(out_fd, (uint8_t *)map + offset, found->tamanho) < 0) {
       perror("write");
       exit(1);
   }
```

Prática: montar um sistema de arquivos com FUSE

- Instalar o fuse; no Ubuntu (e.g., Google cloud shell): `sudo apt install` **`libfuse3-dev fuse3`**
- Incluir:

```
#define FUSE_USE_VERSION 31
#include <fuse.h>
```

- Chamar: `fuse_main(argc, argv, NULL, NULL);`
- Compilar: `gcc -Wall fuse-01.c -o fuse-01 `pkg-config fuse3 --cflags --libs``

- Rodar o programa e verificar se o sistema de arquivo está montado

```
$ mount | grep fuse

/home/wedsonaf/fs/fuse-01 on /home/wedsonaf/fs/tmp type fuse.fuse-01
(rw,nosuid,nodev,relatime,user_id=1000,group_id=1000)
```

Solução

```c
#define FUSE_USE_VERSION 31
#include <fuse.h>

int main(int argc, char **argv)
{
	return fuse_main(argc, argv, NULL, NULL);
}
```

Prática: retornar atributos

- Implementar o método getattr
- Usar o código da próximas páginas como exemplo
  - Também disponível neste link: https://pastebin.com/raw/iaVLEjqp
- Montar sistema de arquivos
- Ler atributos do arquivo `teste.txt`

```
$ ./fuse-01 out1024.img ./tmp
$ stat tmp/teste.txt
  File: tmp/teste.txt
  Size: 10              Blocks: 0          IO Block: 4096   regular file
Device: 0,98    Inode: 2           Links: 1
Access: (0444/-r--r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 1970-01-01 00:00:00.000000000 +0000
Modify: 1970-01-01 00:00:00.000000000 +0000
Change: 1970-01-01 00:00:00.000000000 +0000
 Birth: -
```

Função `getattr`

```c
#define FUSE_USE_VERSION 31
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <fuse.h>

static int mifs_getattr(const char *path, struct stat *stbuf,
                        struct fuse_file_info *fi)
{
        if (strcmp(path, "/teste.txt") != 0) {
                return -ENOENT;
        }

        *stbuf = (struct stat) {
                .st_mode = S_IFREG | 0444,
                .st_nlink = 1,
                .st_size = 10,
        };

        return 0;
}
```

Função `main`

```c
int main(int argc, char **argv)
{
	if (argc != 3) {
		fprintf(stderr, "Uso: %s <imagem> <diretorio>\n", argv[0]);
		exit(1);
	}

	static const struct fuse_operations mifs_ops = {
		.getattr = mifs_getattr,
	};
	struct fuse_args args = FUSE_ARGS_INIT(0, NULL);
	fuse_opt_add_arg(&args, argv[0]);
	fuse_opt_add_arg(&args, argv[2]);
	return fuse_main(args.argc, args.argv, &mifs_ops, NULL);
}
```

Prática: integrar com leitor do sistema de arquivos

- Juntar o exemplo seguinte com a primeira prática
- Objetivo: `stat` retornar informações sobre os arquivos na imagem
- Exemplo:

```
$ ./fuse-01 ./out1024.img ./tmp
$ stat tmp/entry.s
  File: tmp/entry.s
  Size: 235             Blocks: 0          IO Block: 4096   regular file
Device: 0,98    Inode: 2           Links: 1
Access: (0444/-r--r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 1970-01-01 00:00:00.000000000 +0000
Modify: 1970-01-01 00:00:00.000000000 +0000
Change: 1970-01-01 00:00:00.000000000 +0000
 Birth: -
$ stat tmp/Makefile
  File: tmp/Makefile
  Size: 264             Blocks: 0          IO Block: 4096   regular file
Device: 0,98    Inode: 3           Links: 1
Access: (0444/-r--r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 1970-01-01 00:00:00.000000000 +0000
Modify: 1970-01-01 00:00:00.000000000 +0000
Change: 1970-01-01 00:00:00.000000000 +0000
 Birth: -
```

  -

Prática: implementar leitura

- Implementar a função `read`:

```
int mifs_read(const char *path, char *buf, size_t size, off_t offset,
                      struct fuse_file_info *fi);
```

- Ler arquivos com cat, cp, etc.

Solução

```
static int mifs_read(const char *path, char *buf, size_t size, off_t offset,
                     struct fuse_file_info *fi)
{
        struct arquivo *found = bsearch(
                path+1, table, sb->nr_arquivos, sizeof(struct arquivo), compare_nome);
        if (!found) {
                return -ENOENT;
        }

        size_t len = found->tamanho;
        if (offset >= len || offset + size < size) {
                return 0;
        }

        if (offset + size > len)
                size = len - offset;
        size_t foff = (size_t)found->bloco_inicial * sb->tamanho_bloco;
        memcpy(buf, (uint8_t*)map + foff + offset, size);
        return size;
}
```

Prática: enumeração

- Implementar a função `readdir`:

```
int mifs_readdir(const char *path, void *buf, fuse_fill_dir_t filler,
         off_t offset, struct fuse_file_info *fi, enum fuse_readdir_flags);
```

- Adicionar atributos do diretório raiz a getattr:

```
if (strcmp(path, "/") == 0) {
        *stbuf = (struct stat) {
                .st_mode = S_IFDIR | 0755,
                .st_nlink = 2,
        };
        return 0;
}
```

- Listar arquivos com `ls`, `find`, etc.

Solução

```c
static int mifs_readdir(const char *path, void *buf, fuse_fill_dir_t filler,
        off_t offset, struct fuse_file_info *fi, enum fuse_readdir_flags)
{
        char nome[21];
        nome[20] = 0;

        if (strcmp(path, "/") != 0) {
                return -ENOENT;
        }

        filler(buf, ".", NULL, 0, 0);
        filler(buf, "..", NULL, 0, 0);

        for (uint64_t i = 0; i < sb->nr_arquivos; i++) {
                memcpy(nome, table[i].nome, sizeof(table[i].nome));
                filler(buf, nome, NULL, 0, 0);
        }

        return 0;
}
```