Documentação do Projeto t-rext2

O t-rext2 é uma implementação da primeira versão do sistemas de arquivos ext2. Ele é capaz de editar e criar arquivos e diretórios em disco.

Tutorial

O t-rext2 não está acoplado ao tipo de memória em que o sistema de arquivos é armazenado. O seu "disco" pode ser um disco rígido de fato, uma memória flash, um bloco de memória alocado na RAM, um arquivo dentro de um SO, ou qualquer outra coisa que suporte alguma forma de leitura e escrita de bytes. O disco deve ser previamente formatado para a versão 0 do `ext2` usando o comando `mkfs.ext2 -r 0 <nome do arquivo ou dispositivo>` (dependendo da distro pode ser necessário usar `sudo`).

Para inicializar o sistema de arquivos, é necessário criar uma struct `ext2_config_t`:

```
ext2_config_t cfg = {
    .read = my_read,
    .write = my_write,
    .context = my_context
}
```

`my_read` e `my_write` são funções que o programador deve definir. Elas devem ter as seguintes assinaturas:

```
int my_read(uint32_t start, uint32_t size, void* buffer, void* context);
int my_write(uint32_t start, uint32_t size, const void* buffer, void* context);
```

- `my_read` recebe um endereço físico de memória `start`, um tamanho `size` em bytes, um `buffer` de dados, e um contexto. Essa função deve ler `size` bytes, começando no endereço `start` do disco, e armazená-los em `buffer`. O contexto `context` será o pointeiro `my_context` passado na inicialização do objeto `cfg` acima (o programador pode colocar o que quiser dentro de `my_context`).
- `my_write` é análoga a `my_read` mas escreve o `buffer` na memória ao invés de ler.

Para indicar que um erro ocorreu, `my_read` ou `my_write` devem retornar um valor negativo.

Depois de criar a `cfg`, é preciso fazer o `mount` do sistema de arquivos:

```
ext2_t ext2;
ext2_mount(&ext2, &cfg);
```

Pronto! Agora você pode criar/editar arquivos/diretórios. Para criar um arquivo e editá-lo:

```
ext2_file_t file;
ext2_file_open(&ext2, "/my_file.txt", &file);

char* string = "Hello!";
ext2_file_write(&ext2, &file, strlen(string), string);
```

Todos os paths no `t-rext2` são absolutos e portanto começam com `/`.

A maior parte das funções retornam códigos de erro. Por exemplo:

```
ext2_error_t error;
error = ext2_file_write(&ext2, &file, 6, "Hello!");
if (error == EXT2_ERR_DISK_FULL)
    printf("Disco está cheio :( \n");
```

Observações

- Todos os nomes de arquivos/diretórios devem ter menos que `EXT2_MAX_FILE_NAME` = 255 caracteres.
- O `t-rext2` não é thread-safe. Se dois threads diferentes mexerem no mesmo disco, ele muito provavelmente será corrompido.

- Não existe uma função como `fclose` no `t-rext2`. Mesmo assim, tome cuidado para não ter dois `ext2_file_t` que se referem ao mesmo arquivo sendo utilizados ao mesmo tempo. Isso não corromperá o disco, mas pode ter resultados inesperados.

Referência

ext2_config_t

```
/**
 * configuration used to mount a filesystem
 */
typedef struct {
    // user defined read function. Negative return values will be passed back
    // to the caller
    int (*read)(uint32_t start, uint32_t size, void* buffer, void* context);

    // user defined write function. Negative return values will be passed back
    // to the caller
    int (*write)(uint32_t start, uint32_t size, const void* buffer, void* context);

    // this will be passed to the user defined read/write functions, you can
    // put whatever you want here
    void* context;
} ext2_config_t;
```

ext2_mount

```
/**
 * Mounts a filesystem based on a configuration
 *
 * @param ext2   the filesystem to mount
 * @param cfg    the configuration for the filesystem
 */
ext2_error_t ext2_mount(ext2_t* ext2, ext2_config_t* cfg);
```

ext2_file_open

```
/**
 * Opens a file, creating it if it doesn't exist
 *
 * @param ext2   pointer to the filesystem struct
 * @param path   the path to the file (always begins with a '/')
 * @param file   handle to the file
 */
ext2_error_t ext2_file_open(ext2_t* ext2, const char* path, ext2_file_t* file);
```

ext2_file_read

```
/**
 * Reads data from a file
 *
 * Read starts at the offset stored in the file handle and adds the amount of
 * data read to it
 *
 * @param ext2   pointer to the filesystem struct
```

```
 * @param file    file handle pointer
 * @param size    amount of bytes to read
 * @param buf     pointer to a buffer where the data will be stored
 */
ext2_error_t ext2_file_read(ext2_t* ext2, ext2_file_t* file, uint32_t size, void* buf);
```

ext2_file_seek

```
/**
 * Changes a file read/write offset
 *
 * All reads and writes happen relative to an offset that's stored in the
 * ext2_file_t struct. This function changes that offset.
 *
 * @param ext2    pointer to the filesystem struct
 * @param file    file handle pointer
 * @param offset  the new offset (should not be greater than size of file)
 */
ext2_error_t ext2_file_seek(ext2_t* ext2, ext2_file_t* file, uint32_t offset);
```

ext2_file_tell

```
/**
 * Returns the current read/write offset of the file
 *
 * @param ext2    pointer to the filesystem struct
 * @param file    file handle pointer
 */
uint32_t ext2_file_tell(ext2_t* ext2, const ext2_file_t* file);
```

ext2_file_write

```
/**
 * Writes data to a file
 *
 * Write starts at the offset stored in the file handle and adds the amount of
 * data read to it
 *
 * @param ext2    pointer to the filesystem struct
 * @param file    file handle pointer
 * @param size    amount of bytes to write
 * @param buf     pointer to the data that will be written
 */
ext2_error_t ext2_file_write(ext2_t* ext2, ext2_file_t* file, uint32_t size, const void* buf);
```

ext2_dir_record_t

```
/**
 * Contains information about a file/directory
 */
typedef struct {
    uint32_t inode; // inode number of the file/dir (not useful for now)
    char name[EXT2_MAX_FILE_NAME + 1]; // name of the file/dir
} ext2_dir_record_t;
```

ext2_dir_open

```
/**
 * Opens a directory
 *
 * @param ext2   pointer to the filesystem struct
 * @param path   directory path
 * @param dir    pointer to directory handle
 */
ext2_error_t ext2_dir_open(ext2_t* ext2, const char* path, ext2_dir_t* dir);
```

ext2_dir_read

```
/**
 * Reads an entry from a directory
 *
 * The next read to this directory will read the subsequent entry
 *
 * @param ext2    pointer to the filesystem struct
 * @param dir     pointer to directory handle
 * @param entry   pointer to the entry struct
 */
ext2_error_t ext2_dir_read(ext2_t* ext2, ext2_dir_t* dir, ext2_dir_record_t* entry);
```

ext2_dir_seek

```
/**
 * Changes the read offset of the directory
 *
 * Like files, directory also have a read offset. However, this cannot be set
 * to an arbitrary value since the data stored in a directory has a specific
 * structure that must be respected. Therefore, the offset value should always
 * a number that was previously returned by ext2_dir_tell for this directory.
 * This ensures all ext2_dir_read's happen in the proper boundaries.
 *
 * @param ext2     pointer to the filesystem struct
 * @param dir      pointer to directory handle
 * @param offset   new read offset. This should ALWAYS be a value that was previously
 *                 returned by ext2_dir_tell for this directory, otherwise it WILL
 *                 break something.
 */
ext2_error_t ext2_dir_seek(ext2_t* ext2, ext2_dir_t* dir, uint32_t offset);
```

ext2_dir_tell

```
/**
 * Returns the current read offset of the directory
 *
 * @param ext2   pointer to the filesystem struct
 * @param dir    pointer to directory handle
 */
uint32_t ext2_dir_tell(ext2_t* ext2, const ext2_dir_t* dir);
```

ext2_mkdir

```
/**
 * Creates a directory
 *
 * @param ext2   pointer to the filesystem struct
 * @param path   path of the directory
 */
ext2_error_t ext2_mkdir(ext2_t* ext2, const char* path);
```

ext2_error_t

```
// trext2-specific errors
typedef enum {
    EXT2_ERR_BIG_BLOCK = 1,       // filesystem block size is too big
    EXT2_ERR_INODE_NOT_FOUND,     // attempted to find inode with invalid number
    EXT2_ERR_BGD_NOT_FOUND,       // attempted to find group with invalid number
    EXT2_ERR_FILENAME_TOO_BIG,    // filename is greater than EXT2_MAX_FILE_NAME
    EXT2_ERR_DATA_OUT_OF_BOUNDS,  // reading/writing past the end of a file/dir
    EXT2_ERR_FILE_NOT_FOUND,      // file does not exist
    EXT2_ERR_BAD_PATH,            // invalid path (for example, using '\' instead of '/')
    EXT2_ERR_SEEK_OUT_OF_BOUNDS,  // attempted to seek past the end of a file/dir
    EXT2_ERR_DISK_FULL,           // disk is full. Cannot write more data.
    EXT2_ERR_NOT_A_FILE,          // called ext2_file_open on a path that does not point to a file
    EXT2_ERR_NOT_A_DIR,           // called ext2_dir_open on a path that does not point to a dir
    EXT2_ERR_INODES_DEPLETED,     // no more inodes left. Cannot create more files or directories.
} ext2_error_t;
```