Relatório da Atividade Prática 1: Chamadas de Sistema e Checagem de Erros

Laila Pereira Mota[1], **Matheus Hofstede**[1], **Rafael Correa Nagy**[1]

[1]Instituto de Matemática e Estatística – Departamento de Ciência da Computação
Universidade Federal da Bahia (UFBA)
Salvador – BA – Brasil

1. Considerações gerais

Todos os alunos já são familiarizados com lógica de programação, com a linguagem C e com a utilização do git, então não foi necessário revisar nenhum conteúdo e utilizamos o git para trabalharmos em conjunto.
Foi necessário um estudo sobre as chamadas de sistema e os erros provenientes de tais chamadas:

- Para entender as chamadas, utilizamos a documentação dos manuais do linux (sugeridos pela atividade) que foi suficiente para utilizar os métodos com seus parâmetros e structs corretamente, mas, quando necessário, usamos o stackoverflow para resolver erros.
- Para lidar com os erros, utilizamos o header do sistema errno.h.

```
nagy@nagy-Escavador:~$ cat /usr/include/asm-generic/errno.h
/* SPDX-License-Identifier: GPL-2.0 WITH Linux-syscall-note */
#ifndef _ASM_GENERIC_ERRNO_H
#define _ASM_GENERIC_ERRNO_H

#include <asm-generic/errno-base.h>

#define EDEADLK         35      /* Resource deadlock would occur */
#define ENAMETOOLONG    36      /* File name too long */
#define ENOLCK          37      /* No record locks available */

/*
 * This error code is special: arch syscall entry code will return
 * -ENOSYS if users try to call a syscall that doesn't exist.  To keep
 * failures of syscalls that really do exist distinguishable from
 * failures due to attempts to use a nonexistent syscall, syscall
 * implementations should refrain from returning -ENOSYS.
 */
#define ENOSYS          38      /* Invalid system call number */

#define ENOTEMPTY       39      /* Directory not empty */
#define ELOOP           40      /* Too many symbolic links encountered */
#define EWOULDBLOCK     EAGAIN  /* Operation would block */
#define ENOMSG          42      /* No message of desired type */
#define EIDRM           43      /* Identifier removed */
#define ECHRNG          44      /* Channel number out of range */
#define EL2NSYNC        45      /* Level 2 not synchronized */
#define EL3HLT          46      /* Level 3 halted */
#define EL3RST          47      /* Level 3 reset */
#define ELNRNG          48      /* Link number out of range */
#define EUNATCH         49      /* Protocol driver not attached */
#define ENOCSI          50      /* No CSI structure available */
#define EL2HLT          51      /* Level 2 halted */
#define EBADE           52      /* Invalid exchange */
#define EBADR           53      /* Invalid request descriptor */
#define EXFULL          54      /* Exchange full */
#define ENOANO          55      /* No anode */
#define EBADRQC         56      /* Invalid request code */
#define EBADSLT         57      /* Invalid slot */

#define EDEADLOCK       EDEADLK

#define EBFONT          59      /* Bad font file format */
#define ENOSTR          60      /* Device not a stream */
#define ENODATA         61      /* No data available */
#define ETIME           62      /* Timer expired */
#define ENOSR           63      /* Out of streams resources */
#define ENONET          64      /* Machine is not on the network */
#define ENOPKG          65      /* Package not installed */
```

O repositório está disponível em https://github.com/hofstede-matheus/so-ufba-lab-01, onde é possível encontrar as orientações e comandos de execução, descritas no ”README.md”, a partir do makefile, com as seguintes informações:

> Clone o projeto, e no diretório raiz, execute em uma linha de comando:
> make filecopy
> make treecopy
> Os arquivos gerados estão na pasta build

2. Exercício 1: Cópia de Arquivo Simples

Para esse exercício, adicionamos uma verificação inicial dos parâmetros fornecidos e implementamos a cópia do arquivo seguindo a lógica do pseudocódigo fornecido no exercício.
Os erros encontrados foram:

1. Número inválido de argumentos
2. Arquivo não existe
3. Erro ao ler arquivo
4. Erro ao escrever arquivo

Para que as mensagens de erros fossem retornadas de maneira amigável, criamos uma biblioteca auxiliar que chamamos de “errors”. Ela contém funções que irão sobrescrever as funções de erros padrão do C, essas funções consistem apenas em uma impressão de mensagens referentes ao erro, e depois, retorna a falha para o sistema. Além disso, também existe um header para essa biblioteca.

3. Exercício 2: Cópia de Arquivo e Diretório

Para esse exercício, reaproveitamos o código do exercício anterior, para a verificação dos argumentos, as mensagens de erro e a cópia de arquivos.
Esse algoritmo precisa ser recursivo, entretanto sua complexidade não aumentou so-freu grande impacto. De forma simplificada, o algoritmo funciona da seguinte maneira, quando encontramos um arquivo copiamos ele para o destino, caso o arquivo em questão seja uma pasta pasta criamos uma pasta com mesmo nome no destino e chamamos a função recursivamente para seus filhos (os arquivos e diretórios contidos nela).
Para fazer esse algoritmo, precisamos utilizar uma função auxiliar de concatenação de string e uma função que verifica se o ponteiro que estamos lidando é um arquivo ou um diretório.
Os erros que encontramos foram os mesmos do exercício anterior, com a inclusão dos erros referentes a diretórios e não somente a arquivos.