Relatório 01

Vinícius de Oliveira Peixoto Rodrigues (245294)

Agosto de 2022

Questão 1

Item (a)

Foi gerado um arquivo `teste.a`, e o programa teve a seguinte saída:

```
nuke@laptop ~/faculdade/ea872/05-filesystem/class/lab5_arquivos_
de_apoio $ ./a
Foram escritos 29 bytes.
Erro na 3a. operacao = 9
Foram lidos 29 bytes.
Erro na 6a. operacao = 9
```

Item (b)

Sim; da man entry do `open`:

> creat()
> A call to creat() is equivalent to calling open()
> with flags equal to O_CREAT|O_WRONLY|O_TRUNC.

Item (c)

O argumento `mode_t mode` da syscall `open` serve para definir as permissões a serem dadas para um arquivo se ele for criado a partir da syscall, o que exige que as flags `O_CREAT` ou `O_TMPFILE` estejam setadas. Do manual:

> The mode argument specifies the file mode bits to be
> applied when a new file is created. If neither
> O_CREAT
> nor O_TMPFILE is specified in flags, then mode is
> ignored
> (and can thus be specified as 0, or simply omitted).
> The

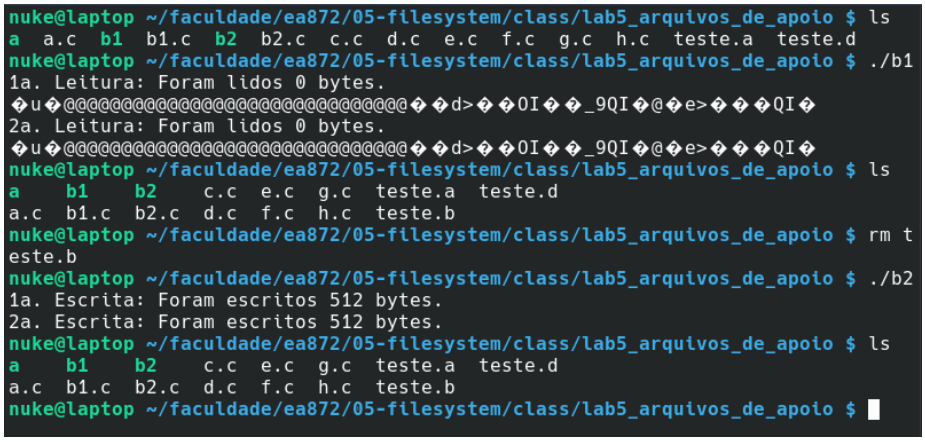
```
mode argument must be supplied if O_CREAT or
    O_TMPFILE is
specified in flags; if it is not supplied, some
    arbitrary
bytes from the stack will be applied as the file mode
    .
```

Item (d)

O erro na 3a operação se deve ao fato de que o `creat` é equivalente a um `open(..., O_CREAT | O_WRONLY | O_TRUNC)`, de modo que o descritor `fd1` está em modo write-only (e não pode ser lido). O erro na 6a operação é porque o descritor `fd2` foi aberto em modo read-only.

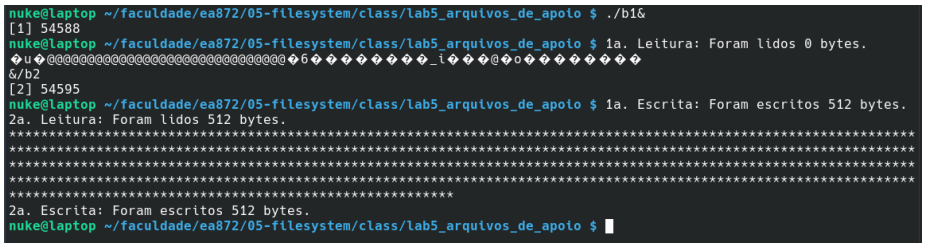
Questão 2

Itens (a) e (b)



Nos dois casos, foi criado um arquivo `teste.b` (visto que ele foi aberto com `O_CREAT`). No programa `b1`, ele lê lixo (visto que não havia nada escrito no programa ainda). No programa `b2` ele escreve asteriscos no programa.

Item (c)



O programa `b1` faz uma primeira leitura e lê lixo (não havia nada escrito no arquivo); em seguida, o programa `b2` escreve no arquivo `test.b`, e depois o programa `b1` lê de novo o arquivo, dessa vez lendo os asteriscos recém-escritos.

Item (d)

Isso é possível porque file descriptors são essencialmente só ponteiros para estruturas de dados do kernel; como nós temos apenas um writer e um reader, não há problemas em ter dois processos separados acessando um arquivo. Seria problemático se houvesse mais de um writer com file descriptor obtido por duas chamadas de `open` diferentes (porque aí o file offset/status dos arquivos não estaria sincronizados, de modo que os dois writers pisariam no pé um do outro).

Questão 3

Item (a)

O programa cria um arquivo `testec1.txt`, escreve a frase `Este e' o arquivo de teste c1:` ; em seguida, move o file offset (que indica o final do arquivo 20031 bytes para frente), e escreve uma mensagem no final (de modo que o arquivo final tem vários KB).

Em seguida, o programa cria um outro arquivo `testec2.txt`, escreve uma mensagem similar no começo, depois escreve em loop 20 mil caracteres `$` e uma mensagem final (de modo que ele também tem vários KB).

Item (b)

```
> ls -l testec*
-rw------- 1 nuke nuke 20059 Sep 21 16:39 testec1.txt
-rw------- 1 nuke nuke 20059 Sep 21 16:39 testec2.txt

> ls -s testec*
8 testec1.txt  20 testec2.txt
```

Item (c)

O sistema operacional tem a liberdade de escolher quantos blocos alocar para cada arquivo; como o arquivo `testec1.txt` tem um buraco enorme no meio dele (porque nós "aumentamos"o tamanho dele artificialmente usando um `lseek`), o S.O. só aloca 8 blocos (o tamanho de blocos reportado pelo `GNU ls` é 1024 bytes, de modo que o tamanho físico alocado é 1024 * 8 = 8 KB). Para o outro arquivo, que é preenchido completamente, o S.O. aloca 20 * 1024 = 20 KB físicos (onde cabe o arquivo inteiro de 20K caracteres).