

EA872 Laboratório de Programação de Software Básico

Atividade 5



Vinícius Esperança Mantovani

RA 247395

Entrega (limite): 30/08/2023, às em sala de aula

Exercício a)

a1) Para iniciar a análise do programa “a.c”, compilei, executei e documentei o programa e sua saída conforme o que se apresenta na figura abaixo (Figura 1):

```
ra247395@le27-9:~/Downloads/lab5_arquivos_de_apoio$ ./xyz
Foram escritos 29 bytes.
Erro na 3a. operacao = 9
Foram lidos 29 bytes.
Erro na 6a. operacao = 9
```

Figura 1: Saída do terminal na execução de a.c compilado

O programa criaria um arquivo chamado “teste.a” e, em seguida, cria um file descriptor correspondente a este arquivo, sendo que, para ambas as criações, caso ocorra um erro na criação do arquivo ou na escrita da string descrita no código no “fd1”, são emitidas as respectivas mensagens de erro. Em seguida, é expresso o número de bytes escritos em “fd1”. Seguindo, há a tentativa de leitura do “fd1” para o buffer “buf”, na qual ocorre um erro, por conta de “fd1” não ser um file descriptor disponível para leitura, como sabemos, pois foi registrado em “fd1” uma sentença. Assim, ocorre o lançamento da mensagem “Erro na 3a. operacao = 9”, onde 9 é o valor do erro ocorrido na função “read”. Em sequência, “fd2” passa a assumir o valor de retorno da função “open”, que, neste caso, retorna um file descriptor para o arquivo passado “teste.a” apenas para leitura. Então, esse file descriptor “fd2” é usado para a leitura do arquivo “teste.a” para o buffer “buf”. Logo, é impressa a quantidade de bytes lidos, que, mais uma vez, é de 29 bytes. Por fim, o último erro apresentado ocorre por conta da tentativa de escrita em teste.a usando “fd2”, que não permite escrita.

A diferença entre open e creat está no fato de que open retorna um file descriptor geral, seguindo as opções especificadas pelo código, enquanto que creat já é mais restrito, uma vez que é equivalente a open com opções de write only, que só permite escrita a partir do

file descriptor, de criação do arquivo caso ele não exista e , de truncamento para 0 do tamanho de um arquivo no caso de ele ser um arquivo simples.

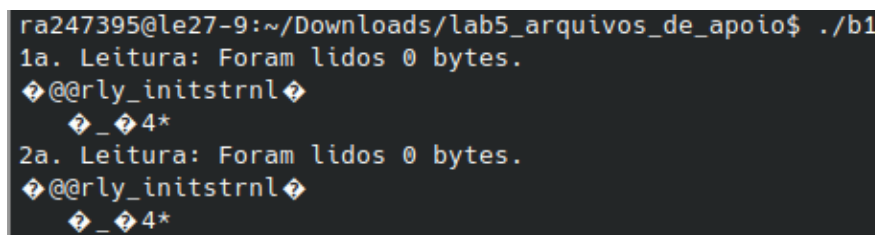
a2) Sim, basta que se usem a opção `O_WRONLY | O_CREAT | O_TRUNC`).

a3) Pode-se obter acesso diferenciado a um mesmo arquivo por meio do modo como se define o file descriptor, de maneira que, ao usarmos determinadas opções, por exemplo, em open, ou usarmos creat, podemos obter file descriptors que permitem apenas leitura ou apenas escrita do seu arquivo correspondente.

a4) Conforme citado em a.1, os erros são causados pelas tentativas de: ler um arquivo por meio de um file descriptor que não é de fato um file descriptor, é apenas uma sequência de bytes e; escrever por meio de um file descriptor que só permite leitura.

Exercício b)

b1) Obtemos como saída do terminal para b1, o que se segue na Figura 2:



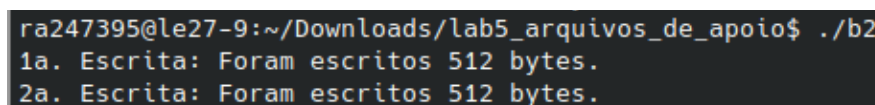
```
ra247395@le27-9:~/Downloads/lab5_arquivos_de_apoio$ ./b1
1a. Leitura: Foram lidos 0 bytes.
@@rly_initstrnl
_4*
2a. Leitura: Foram lidos 0 bytes.
@@rly_initstrnl
_4*
```

Figura 2: saída do terminal para b1

Além disso, temos como saída, ainda, um arquivo chamado teste.b, por conta do trecho de código: `fd = open("teste.b", O_EXCL|O_CREAT|O_RDONLY, 0600)`, no qual se cria esse arquivo e se define um file descriptor armazenado em “fd” para tal arquivo.

As saídas do terminal são justificadas pelo seguinte: A primeira linha é causada pelo print do número de bytes lidos no trecho: `i = read(fd, buf, sizeof(buf))`, que é 0, pois “fd” é file descriptor de um arquivo vazio; A segunda e a terceira linha são ocasionadas pela tentativa de imprimir um buffer cujo conteúdo seria o conteúdo de teste.b, no entanto, como teste.b não tinha valor especificado após sua criação, ocorre que o buffer contém o conjunto: `@@rly_initstrnl``n``_4*` de caracteres; Por fim, a quarta linha é causada pelo print da mensagem semelhante à da primeira linha, com procedimento igual ao da primeira, tentando ler um arquivo vazio por meio de um file descriptor e salvando o número de bytes lidos, que é de 0 bytes novamente pelo mesmo motivo. Finalmente, as duas últimas linhas têm o mesmo motivo de ser das linhas 2 e 3.

b2) Na figura 3 a seguir, está a saída de b.2 no terminal:



```
ra247395@le27-9:~/Downloads/lab5_arquivos_de_apoio$ ./b2
1a. Escrita: Foram escritos 512 bytes.
2a. Escrita: Foram escritos 512 bytes.
```

Figura 3: saída de b.2 no terminal

A primeira linha da saída existe por conta do trecho de código: `printf("1a. Escrita: Foram escritos %d bytes.\n", i);`, no qual é escrito o valor bytes escritos (armazenado em `i`), assim, como esse valor é pego no trecho `i = write(fd, buf, sizeof(buf))`, temos que ele é igual ao número de caracteres do `buf` que são escritos no arquivo `teste.b`, que é 512 por conta de `buf` ter esse tamanho e ter sido preenchido por `*`. A segunda linha da saída ocorre da mesma forma que a primeira.

b3) A saída segue na Figura 4 a seguir:

[illegible]

Figura 4: saída na chamada imediatamente subsequente dos dois comandos desejados

O que ocorre neste processo é que, após chamarmos a execução de b1.c, temos os dois programas rodando simultaneamente e, as saídas são justificadas do mesmo modo como foram nos itens anteriores desta questão, com exceção dos “*”, que foram, agora, impressos, pois o buffer é preenchido com tais caracteres durante a execução inicial de b2.

b4) Isso ocorre por conta do uso de file descriptors para acessar o mesmo arquivo de maneiras diferentes, seja para escrita ou para leitura, pelos dois programas. Desse modo, faz-se que os dois programas consigam seguir seus fluxos normalmente, mas que as alterações feitas por um deles no arquivo sejam sentidas e influenciem nas saídas do outro programa. Assim, programas com funções complementares podem trabalhar juntos para exercer a atividade esperada.

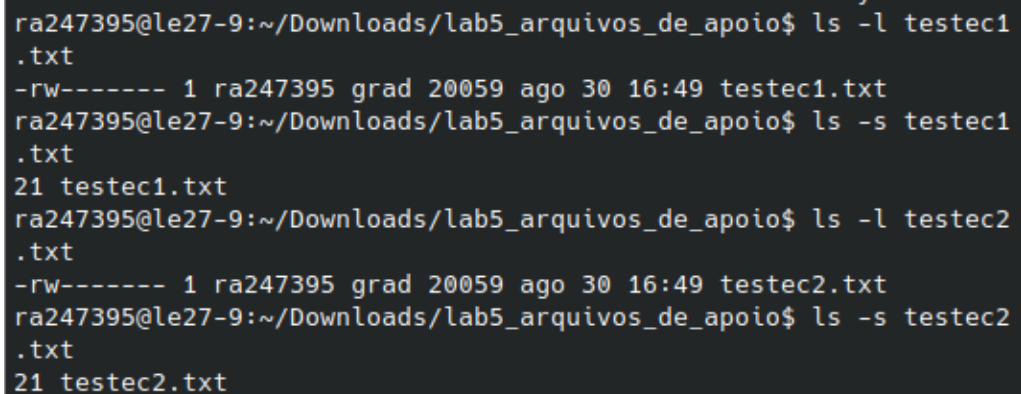
Exercício c)

c1) O programa c não tem saída no terminal, mas tem cria dois arquivos “testec1.txt” e “testec2.txt”.

O programa cria o arquivo “testec1.txt”, escreve nesse arquivo o seguinte: "Este e' o arquivo de teste c1: " e " fim do arquivo testec1.txt\n". Fecha o file descriptor criado para esse arquivo e,

cria um novo arquivo “testec2.txt” e seu file descriptor e escreve nesse arquivo: "Este e' o arquivo de teste c2: " e, 20000 caracteres “\$”. Por fim, escreve nesse arquivo, " fim do arquivo testec2.txt\n".

c2) A saída é apresentada na figura abaixo



```
ra247395@le27-9:~/Downloads/lab5_arquivos_de_apoio$ ls -l testec1
.txt
-rw----- 1 ra247395 grad 20059 ago 30 16:49 testec1.txt
ra247395@le27-9:~/Downloads/lab5_arquivos_de_apoio$ ls -s testec1
.txt
21 testec1.txt
ra247395@le27-9:~/Downloads/lab5_arquivos_de_apoio$ ls -l testec2
.txt
-rw----- 1 ra247395 grad 20059 ago 30 16:49 testec2.txt
ra247395@le27-9:~/Downloads/lab5_arquivos_de_apoio$ ls -s testec2
.txt
21 testec2.txt
```

Essa diferença para o mesmo comando ls ocorre, pois, ao usar -s, mostra-se o número de blocos de 1024 bytes do arquivo, enquanto que, ao usar o -l temos uma série de informações, mais o número de bytes do arquivo.

c3) Conforme se percebe na imagem, não existe diferença entre os números de blocos dos dois arquivos.