Insper

15 - Chamadas de Sistemas: Arquivos

Sistemas Hardware-Software - 2020/1

Igor Montagner

Parte 1 - leitura e escrita de arquivos

Nesta seção trabalharemos com as chamadas `open`, `close`, `read` e `write`.

Exercício 1: Leia o código do arquivo `exemplo_io1.c`. O quê ele faz?

Vamos agora usar o comando `man` para ver a documentação das chamadas de sistema do Linux. Digite no terminal `man read`. A tela mostrada contém a documentação da função `read`, incluindo sua assinatura e quais cabeçalhos devem ser incluídos para que ela possa ser usada.

Exercício 2: Usando como apoio a documentação aberta pelo comando `man`, como saber que um arquivo foi lido até o fim? **Dica**: a seção `RETURN VALUE` pode te ajudar.

Exercício 3: Baseado em sua resposta acima, modifique `exemplo_io1.c` para que ele mostre o arquivo inteiro. Para ficar melhor de visualizar o resultado, faça o `printf` interno mostrar somente o caractere lido.

Exercício 4: Vamos agora trabalhar com a chamada `write`. Um exemplo de seu uso está mostrado no arquivo `exemplo_io2.c`. Você deverá criar um programa `copy_file` que lê dois nomes de arquivos usando `scanf` e copia o conteúdo do primeiro para o segundo. Ou seja, você deverá abrir ambos arquivos (cada um terá seu próprio *file descriptor*), ler do primeiro para um buffer e escrever este buffer no segundo.

Extra

Estes exercícios não terão correção durante a aula e devem ser checados no atendimento. Faça-os se você já acabou os anteriores e a correção ainda não começou.

Exercício 5: Até agora trabalhamos com um buffer com somente um caractere. Isso facilita a programação, mas deixa nosso código muito lento. Modifique o `copy_file` para usar um buffer de 100 caracteres. Meça o tempo de execução para a cópia de um arquivo grande (~100Mb) usando o comando `time`. Não se esqueça de verificar que os resultados ficaram iguais com o comando `diff`.

> **!** Não sabe como usar `time` ou `diff`? Use o comando `man` visto acima. Não sabe usar `man`? Use o comando `man` para aprender a usá-lo com `man man`.

Exercício 6: Use o manual para entender o significado dos tempos mostrados pelo comando `time`. Escreva abaixo seu entendimento.

Exercício 7: Você consegue explicar a diferença de desempenho entre as duas versões? Lembre-se da aula de hierarquia de memória e da velocidade de acesso aos diferentes tipos de memória.

Parte 2 - permissões e posse de arquivos

No `exemplo_io2.c` passamos algumas opções extras para poder criar o arquivo:

```
int fd1 = open(arq1, O_WRONLY | O_CREAT, 0700);
```

A flag `O_CREAT` é usada para indicar que o arquivo deve ser criado caso ele não existe. O número `0700` representa os bits de acesso visto na expositiva. Cada dígito contém 3 dígitos que representam as seguintes permissões

- 4 - permissão de leitura
- 2 - permissão de escrita
- 1 - permissão de execução

O primeiro dígito contém as permissões do usuário dono do arquivo. O segundo dígito contém as permissões do grupo dono do arquivo. Usuários que pertencem a este grupo possuem estas permissões. O terceiro dígito lista as permissões para todos os outros usuários.

Exercício 1: Use `ls -l` na pasta do arquivo criado por `copy_file` (ou por `exemplo_io2`). Onde é possível obter as informações de permissões do arquivo? Qual o usuário e grupo donos do arquivo? As permissões passadas para o `open` foram corretamente colocadas no arquivo?

Exercício 2: Quais permissões são garantidas pela máscara `640`? É uma boa ideia usá-la?

Exercício 3: Qual máscara usaria se quisesse que um arquivo possa ser modificado somente por seu dono, mas possa ser executado por qualquer usuário do sistema (incluindo o dono do arquivo)? Justifique.

Exercício 4: Rode o comando `copy_file` usando `sudo`. Use `ls -l` para listar as informações do arquivo e verifique seu dono e as permissões. Use os comandos `chown` para mudar o dono do arquivo para seu usuário e `chmod` para deixar suas permissões como leitura e escrita para você e somente leitura para o restante.

Extra

Estes exercícios trabalham com o conceito de posse de arquivos e de sobrescrita de arquivos já existentes.

Exercício 5: Tente usar `copy_file` usando como fonte algum arquivo que você não possui acesso de leitura (você pode criá-lo e depois usar `chmod` para editar os acessos). O quê ocorre? Você consegue explicar este comportamento?

Exercício 6: Conserte o erro ocorrido acima checando a saída de `open`. Consulte o manual caso necessário. Não se esqueça de fazê-lo para o arquivo fonte e destino.

Exercício 7: Vamos realizar um experimento neste exercício:

1. Crie um arquivo com 100 caracteres a e chame-o de `100a`;
2. Crie um arquivo com 3 caracteres b e chame-o de `3b`;
3. Use `copy_file` para copiar `3b` em cima de `100a`;
4. Mostre o conteúdo do arquivo sobrescrito `100a`.

Você consegue explicar o que ocorreu? Se sim, busque no manual uma flag que, ao ser passada para o `open`, evita que isto ocorra.

Exercício 8: O `copy_file` sobrescreve arquivos sem dó. Use o manual para encontrar a flag que faz `open` falhar caso o arquivo de destino já exista e conserte seu programa para perguntar se o usuário deseja sobrescrever o arquivo. Note que isto conflita com seu exercício 6. Entenda como usar `errno` para que você consiga diferenciar os dois tipos de erros.