Insper

22 - Entrada e Saída II

Sistemas Hardware-Software - 2019/1

Igor Montagner

Parte 1 - redirecionamento de arquivos

Neste exercício trabalharemos com descritores de arquivos juntamente com `fork` e `wait`. Nosso objetivo será redirecionar a saída de um processo filho para um arquivo temporário e depois reler este arquivo no processo pai, copiando todos os dados para o terminal. Vamos trabalhar com o arquivo `fork-io1.c`.

Exercício: Execute a `funcao_lenta` em um processo separado e, no pai, execute a `funcao_lenta_pai`. Ambas devem rodar paralelamente.

As mensagens devem estar sendo impressas todas bagunçadas no terminal. Vamos corrigir isto criando um arquivo temporário e redirecionando a saída padrão (`fd == 1`) do processo filho para este arquivo.

Exercício: Pesquise, no manual da chamada `open`, como criar um arquivo temporário. Escreva abaixo o comando `open`. Este arquivo estará no disco? Ele continua disponível após ser fechado? Qual o significado do argumento `pathname`?

Exercício: Para redirecionar a saída usaremos a chamada `dup2`. Desenhe abaixo a tabela de descritores de arquivos dos processos pai e filho após o fork. Em outra cor, desenhe como ela deverá ficar após o `dup2`.

Exercício: Modifique seu programa para usar `dup2` para redirecionar a saída do filho para um arquivo temporário. Este arquivo deverá ser legível também a partir do pai.

Exercício: Modifique seu programa para que o processo pai espere o fim do processo filho e mostre sua saída no terminal.

Dica: * A saída do filho está disponível em um arquivo temporário. * Onde estará o cursor de leitura/escrita do arquivo quando o processo filho terminar? * `man lseek`

Parte 2 - pipes

Um outro grande uso arquivos é para comunicação entre processos. No exemplo acima mandamos a saída do processo filho para um arquivo. Ou seja, trocamos o *file descriptor* que representada o terminal por um que representa um arquivo. Note que tanto o terminal quanto um arquivo no disco são tratados da mesma maneira.

Um *PIPE* é um par de arquivos, um somente leitura e outro somente escrita, tal que tudo o que for escrito no segundo estará disponível para leitura no primeiro. Logo, ele é canal unidirecional de comunicação entre processos e deve ser "configurado" antes antes do `fork` usado para criar os processos. Seu uso mais comum é enviar a saída de um programa diretamente para a entrada de outro. Ou seja, conectamos a saída padrão (`fd == 1`) de um programa na entrada padrão (`fd == 0`) de um segundo processo. Na linha de comando isso é feito com o caractere `|`.

No exemplo abaixo a saída do programa *ls* passada para o programa *wc*. Isto tem o efeito de contar quantos arquivos existem em um diretório.

```
ls | wc
```

Podemos fazer o mesmo em um programa usando a chamada `int pipe(int fds[2])`, que retorna dois descritores de arquivos. Tudo o que for escrito no segundo descritor (`fds[1]` - aberto em modo somente escrita) fica automaticamente disponível para leitura no primeiro (`fds[0]` - aberto em modo somente leitura). Ao usar uma ponta do pipe não se esqueça de dar `close` na outra ponta se ela não for usada por seu processo.

Vamos fazer um exemplo simples primeiro. Os *file descriptors* retornados pela chamada `pipe` funcionam igual arquivos (usando as chamadas `read` e `write`).

Exercício: Faça um programa que faz a chamada `pipe` e cria um processo filho. O processo filho deverá escrever um inteiro no pipe e o processo pai deverá receber este inteiro e dar print no terminal.

Exercício: Por que `pipe` tem que ser feito antes do `fork`?

Extra

Usando `dup2` e `pipe` em conjunto podemos redirecionar a entrada/saída padrão de processos de maneira a possibilitar comunicação sem que os processos sejam explicitamente preparados para isto.

Exercício: Explique por que não é possível redirecionar a saída do processo pai para a entrada do processo filho diretamente usando `dup2`.

Exercício: modifique o arquivo *pipe.c* para que

1. a saída do processo pai seja enviada para a entrada do processo filho usando um *pipe*
2. o processo filho execute um interpretador *python*