

Laboratório #2: Fork

Departamento de Computação/UFC – Sistemas Operacionais
José Macedo



Operating System Concepts with Java – 7th Edition, Nov 15, 2006

2.1



Objetivo

- ❑ Entender a comunicação entre processos



COMUNICAÇÃO ENTRE PROCESSOS



Comunicação entre Processos

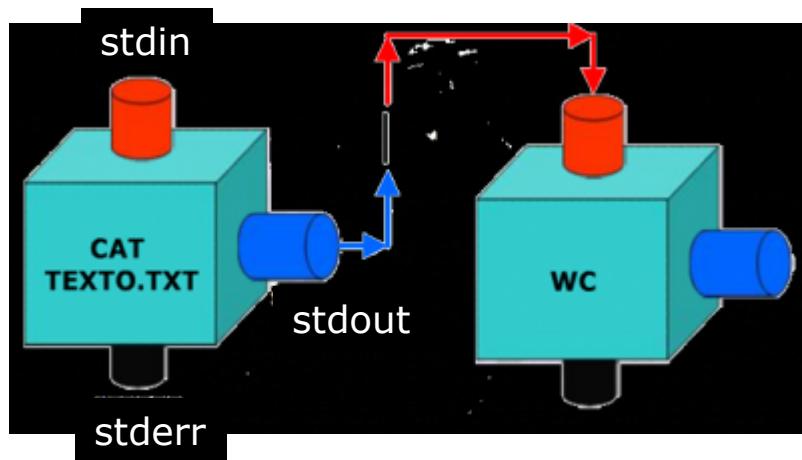
- IPC (Inter-Process Communication)
 - É o grupo de mecanismos ao qual permite que processos possam transferir informações entre si.
 - Se referem a processos em execução na mesma máquina
- Algumas das técnicas de comunicação entre processos no linux:
 - Por memória compartilhada (visto no laboratório de fork)
 - Pipes
 - Sinais
- Iremos estudar os Pipes



Pipe

- ❑ “Canalização”
- ❑ É o redirecionamento da saída padrão de um programa para a entrada padrão de outro.
- ❑ Pipes Unidirecionais
 - Em um UNIX shell, o símbolo do pipe é: | (a barra vertical)

```
$ cat texto.txt | wc
```



Pipe Unidirecional

- O próprio pipe provê a sincronização dos processos
 - Se um processo está tentando passar os dados, e o leitor ainda não está pronto, o pipe bloqueia o processo até que o mesmo esteja disponível.
- Vamos tentar!

```
$ man ls | head
```

```
$ man head
```

```
$ man ls
```



Arquivos para o Laboratório

- ❑ Baixar os arquivos em

https://github.com/josemacedo/sistemas-operacionais/tree/master/2_Lab_IPC

- ❑



Piping to “less”

- Programa Less
 - Leitura de arquivos
 - Não carrega o arquivo inteiro na memória
 - Leitura por páginas
 - Ideal para leitura de arquivos gigantes

```
$ less contatos.txt
```



Pipping to “less”

- Programa Less
 - Leitura de arquivos
 - Não carrega o arquivo inteiro na memória
 - Leitura por páginas
 - Ideal para leitura de arquivos gigantes

```
$ less contatos.txt
```

- Comando grep
 - Realiza buscas no conteúdo dos arquivos (ou input) procurando linhas que respeitem a expressão regular mencionada.

```
$ grep gmail contatos.txt
```

```
$ grep gmail contatos.txt | less
```



Prática usando pipe

- ❑ Qual o comando utilizamos para recuperar as 5 primeiras colunas das 10 primeiras linhas do arquivo cancer.data?
 - Dica:
 - ❑ \$ man head
 - ❑ \$ man cut

```
$ cut -d, -f 1-5 cancer.data | head -n 10
```



Prática usando pipe

- ❑ Qual o comando damos para recuperar as 5 primeiras colunas das 10 primeiras linhas do arquivo cancer.data?

- Dica:

- ❑ \$ man head
 - ❑ \$ man cut

```
$ cut -d, -f 1-5 cancer.data | head -n 10
```

- ❑ O que acontece?

- \$ cut -d, -f 1-5 cancer.data → extrai as colunas de 1 a 5
 - \$ head -n 10 → imprime as 10 primeiras linhas



Prática 2 usando pipe

- Vamos repetir o exemplo anterior, mas removendo todas as linhas iniciadas com “#”
 - Dica
 - \$ man grep

```
$ grep -v '#' cancer.data | cut -d, -f 1-5 | head -n 10
```



Prática 2 usando pipe

- Vamos repetir o exemplo anterior, mas removendo todas as linhas iniciando com “#”

- Dica

- \$ man grep

```
$ grep -v '#' cancer.data | cut -d, -f 1-5 | head -n 10
```

- O que acontece?

- \$ grep -v '#' cancer.data → extrai todas as linhas que não contém “#”
 - \$ cut -d, -f 1-5 → extrai as colunas de 1 a 5
 - \$ head -n 10 → imprime as 10 primeiras linhas



Pipes Nomeados (FIFO)

- Um pipe nomeado é uma extensão do pipe convencional.
- A diferença é que
 - um pipe “normal” termina juntamente com o fim da execução do programa
 - e um pipe nomeado deve ser criado e destruído explicitamente.
- Para criar um pipe nomeado podemos usar o comando *mkfifo*.
- Pode-se usar pipes nomeados em programas para prover meios de comunicação inter processos.



Pipes Nomeados (FIFO)

- ❑ Dois processos podem utilizar este pipe através do seu nome.
- ❑ Exemplo:
 - Abrir dois terminais

- ❑ Terminal 1

```
$ mkfifo pipe_teste
```

```
$ cat < pipe_teste
```

Direciona o conteúdo de um arquivo para a entrada de um comando.

- ❑ O pipe fica esperando até obter algum dado



Pipes Nomeados (FIFO)

- ❑ Dois processos podem utilizar este pipe através do seu nome.
- ❑ Exemplo:
 - Abrir dois terminais

- Terminal 1

```
$ mkfifo pipe_teste
```

```
$ cat < pipe_teste
```

Direciona o conteúdo de um arquivo para a entrada de um comando.

Direciona a saída de um comando para dentro de um arquivo, sobrepondo o seu conteúdo, caso o arquivo especificado não exista, ele o criará.

- O pipe fica esperando até obter algum dado

- Terminal 2

```
$ ls > pipe_teste
```

- a saída do comando ls será redirecionada para o pipe nomeado “pipe_teste”



Pipes Nomeados (FIFO)

□ Exemplo 2:

- Criar um pipe e instruir o programa gzip para comprimir aquilo que é enviado (*canalizado - piped*) para ele.

```
$ mkfifo meu_pipe
```

```
$ gzip -9 -c < meu_pipe > saida.gz &
```

Direciona o conteúdo de um arquivo para a entrada de um comando.

Direciona a saída de um comando para dentro de um arquivo, sobrescrevendo o seu conteúdo, caso o arquivo especificado não exista, ele o criará.



Pipes Nomeados (FIFO)

□ Exemplo 2:

- Criar um pipe e instruir o programa gzip para comprimir aquilo que é enviado (*canalizado - piped*) para ele.

```
$ mkfifo meu_pipe
```

```
$ gzip -9 -c < meu_pipe > saida.gz &
```

- O processo ficará esperando até obter algum dado
- Em um shell de processo separado, independentemente, poderiam ser enviados os dados a serem comprimidos:

```
$ cat contatos.txt > meu_pipe
```

- O pipe nomeado pode ser deletado como qualquer arquivo.



PIPES EM POSIX



Pipes em POSIX

- Sua programação é:
 - A função pipe() solicita a criação do pipe;
 - Cria um canal uni-direcional que pode ser usado na comunicação entre processos
 - A chamada de sistema write() é utilizada para escrever no pipe
 - A chamada de sistema read() é utilizada para ler do pipe
- A comunicação só pode ser feita entre processos parentes
 - Ex: processo criado com a chamada fork();
- A chamada de sistema pipe(), cria 2 descritores de arquivos
 - Mas não são arquivos!!
 - Então o que é?
 - É um índice inteiro que indica quais são os canais de entrada e saída



Descritores de arquivo

- Qualquer processo, ao ser iniciada sua execução, já possui 3 descritores pré-definidos: um para a saída padrão, um para a entrada padrão e outro para a saída padrão de erro:

Valor Inteiro	Nome
0	Entrada padrão (<i>stdin</i>)
1	Saída padrão (<i>stdout</i>)
2	Saída padrão de erro (<i>stderr</i>)



Descritores de arquivo

- Qualquer processo, ao ser iniciada sua execução, já possui 3 descritores pré-definidos: um para a saída padrão, um para a entrada padrão e outro para a saída padrão de erro:

Valor Inteiro	Nome
0	Entrada padrão (<i>stdin</i>)
1	Saída padrão (<i>stdout</i>)
2	Saída padrão de erro (<i>stderr</i>)

- Ao usarmos a chamada `pipe()`, cria-se 2 descritores
 - ocupam o descritor 3 e 4.
 - O primeiro descritor do pipe serve para ler e o segundo para escrever
 - Depois de criado, pode-se usar as chamadas de sistema `write()` para escrever e `read()` para ler.



Funções

- ❑ \$ man pipe
- ❑ int pipe (int fda[])
 - retorna 0 se ocorrer com sucesso
 - Retorna -1 se ocorrer erro
 - fda: vetor de duas posições:
 - ❑ fda[0]: leitura
 - ❑ fda[1]: escrita
- ❑ Após criar o pipe, qualquer filho gerado herdará os endereços de escrita e leitura deste pipe e portanto poderão ler e escrever no mesmo pipe.
- ❑ Para evitar conflito de dados dentro do pipe, devemos definir quem irá escrever e quem irá ler.



Funções

- Leitura do pipe:
 - `read (fda[0], &buffer, 3*sizeof(char));`
 - `fda[0]`: Endereço de leitura do pipe
 - `&buffer`: Endereço de um buffer de escrita para receber os dados lidos do pipe
 - `3*sizeof(char)`: Quantidade de bytes que se deseja ler
- Escrita no pipe:
 - `write (fda[1], &buffer, 3);`
 - `fda[1]`: Endereço de escrita do pipe
 - `&buffer`: Local onde está a informação que se deseja enviar
 - `3`: Quantidade de bytes que se deseja enviar



exemplo.c

```
#include <stdio.h>
#include <unistd.h>

int main () {
    int fda[2];
    int retorno;
    int envio=10;
    pipe( fda );
    pid_t pid = fork();

    /* Processo dividido
    Em 2 pelo fork() */

    /* Será usado para guardar o
    número descriptor dos pipes */

    /* Cria o pipe (dois descritores) e armazena seus valores no
    vetor fda. Agora: fda[0] para ler, fda[1] para escrever. */

    if ( pid == 0 ) { /* ações do filho */
        close ( fda[0] ); /* fecha o descritor de leitura não utilizado */
        write ( fda[1], &envio, 4 );
    }else { /* ações do pai */
        close ( fda[1] ); /* fecha o descritor de escrita não utilizado */
        read ( fda[0], &retorno, 4 );
        printf ("PAI: O retorno do filho é %d", retorno);
    }
}
```



Prática

- Simular o que o shell faz ao executar um ls | wc



pipe.c

/* Processo dividido
Em 2 pelo fork() */

```
main()
{
    int fda[2];
    /* Será usado para gu
    número descriptor dos p
    */
    /* Cria o pipe
    Testa se
    */
    fprintf(stderr, "%d %d\n", fda[0], fda[1]);
    switch (fork()) {
        case -1:
            erro("fork");
            break;
        case 0: /* ações do filho */
            close(1);
            dup(fda[1]);
            close(fda[1]);
            close(fda[0]);
            execvp("ls", "ls", 0);
            erro("Voltou do ls");
            break;
        default: /* ações do pai */
            close(0);
            dup(fda[0]);
            close(fda[0]);
            close(fda[1]);
            execvp("wc", "wc", 0);
            erro("Voltou do wc");
            break;
    }
}
```

Qual a implicação disso?

/* Será usado para gu
número descriptor dos p

/* Cria o pipe
Testa se

/* Imp
0, 1 e 2 já estão ocupados na tabela de descritores . */

Inteiro	Nome
0	Entrada padrão (stdin)
1	Cópia de fda[1]
2	Saída padrão de erro (stderr)
3	
4	

Depois de modificadas as
tabelas, o que acontecerá
de agora em diante?

Inteiro	Nome
0	Cópia de fda[0]
1	Saída padrão (stdout)
2	Saída padrão de erro (stderr)
3	
4	



Desafio

- Implemente uma nova versão do programa criado no laboratório 1, relacionado com cálculo dos termos da série de Fibonacci, usando o mecanismo de Pipes;
- Quem implementar o programa corretamente e demonstrá-lo até o fim da aula, ganhará 1.0 ponto na primeira prova;
- Às 12:00h iniciarei a avaliação;



*** FIM ***

