

Redirecionamento de Streams

Padrão em Sistemas POSIX

Pedro Bruel

pedro.bruel@gmail.com

18 de Junho de 2021

Introdução

Conteúdo e Objetivos de Aprendizagem

Tópicos

- **Streams**, ou **fluxos** padrão em Processos POSIX, chamadas de sistema **dup** e **dup2**

Habilidades

- Escrever scripts em **bash** e programas em **C** que **redirecionem** os **streams** de **Entrada/Saída** de programas

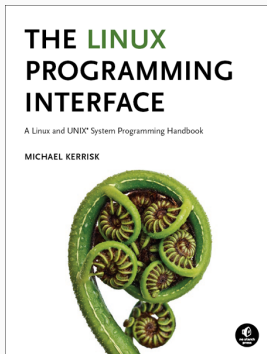
Conhecimentos

- Compreender o redirecionamento de streams de Entrada/Saída como a **duplicação de descritores de arquivo** em processos POSIX



Recursos Extras

The Linux API



- <https://man7.org/tlpi>
- Capítulos 4, 5, e 44

Site com Slides e Exemplos

PPD

Aulas e exemplos de código sobre ferramentas de programação paralela e distribuída, e tópicos correlatos

[View the Project on GitHub](#)
phrb/PPD

Redirecionamento de Streams Padrão em Sistemas POSIX

License [CC BY 4.0](#)

Conteúdo

Slides

Exemplos de Código

- Exemplos em *bash*
- Exemplo com *pipe* em C

Recursos

- [The Linux Programming Interface](#) (Michael Kerrisk, 2010)

- <https://phrb.github.io/PPD>

Registrando Mensagens e Monitorando Sistemas

Problemas

- Em sistemas POSIX, quais são as formas de **registrar as mensagens impressas** por um programa durante sua execução?
- No Linux, quais as formas de **monitorar o uso de recursos** como CPU, memória, entrada/saída, e rede?



Registrando Mensagens e Monitorando Sistemas

Registro de Mensagens

-

Monitoramento de Recursos

-

Redirecionamento de Entrada/Saída em *bash*

Streams Padrão em Processos POSIX

- Streams, ou fluxos, são canais de comunicação indexados por descritores de arquivo
- Todo processo POSIX abre os seguintes streams por padrão:

Descritor	Propósito	Nome POSIX	Stream <i>stdio</i>
0	Entrada padrão	STDIN_FILENO	<i>stdin</i>
1	Saída padrão	STDOUT_FILENO	<i>stdout</i>
2	Erro padrão	STDERR_FILENO	<i>stderr</i>

Usando Arquivos de *Log*: Redirecionamento de Saída e Erros

```
date > existe.txt && cat existe.txt
```

```
Wed Jun 16 05:38:58 PM -03 2021
```

Usando Arquivos de *Log*: Redirecionamento de Saída e Erros

```
date > existe.txt && cat existe.txt
```

```
Wed Jun 16 05:38:58 PM -03 2021
```

- Redirecionando **erro** e **saída** para arquivos diferentes:

```
cat existe.txt não-existe.txt 1> saída.log 2> erros.log
```

Usando Arquivos de *Log*: Redirecionamento de Saída e Erros

```
date > existe.txt && cat existe.txt
```

```
Wed Jun 16 05:38:58 PM -03 2021
```

- Redirecionando **erro** e **saída** para arquivos diferentes:

```
cat existe.txt não-existe.txt 1> saída.log 2> erros.log
```

- Verificando os **logs**:

```
cat saída.log
```

```
Wed Jun 16 05:38:58 PM -03 2021
```

Usando Arquivos de Log: Redirecionamento de Saída e Erros

```
date > existe.txt && cat existe.txt
```

```
Wed Jun 16 05:38:58 PM -03 2021
```

- Redirecionando **erro** e **saída** para arquivos diferentes:

```
cat existe.txt não-existe.txt 1> saída.log 2> erros.log
```

- Verificando os **logs**:

```
cat saída.log
```

```
Wed Jun 16 05:38:58 PM -03 2021
```

```
cat erros.log
```

```
cat: não-existe.txt: No such file or directory
```

Monitorando o Linux: Redirecionamento de Saída

Monitorando o Consumo de Memória

```
free -h > mem.log && cat mem.log
```

Monitorando o Linux: Redirecionamento de Saída

Monitorando o Consumo de Memória

```
free -h > mem.log && cat mem.log
```

	total	used	free	shared	buff/cache	available
Mem:	31Gi	7.2Gi	1.1Gi	1.4Gi	22Gi	22Gi
Swap:	0B	0B	0B			

Monitorando o Linux: Redirecionamento de Saída

Monitorando o Consumo de Memória

```
free -h > mem.log && cat mem.log
```

	total	used	free	shared	buff/cache	available
Mem:	31Gi	7.2Gi	1.1Gi	1.4Gi	22Gi	22Gi
Swap:	0B	0B	0B			

```
cat /proc/meminfo > meminfo.log
```

Monitorando o Linux: Redirecionamento de Entrada

Monitorando o Clock da CPU

```
grep "cpu MHz" < /proc/cpuinfo > cpu_clock.log && cat cpu_clock.log
```


Monitorando o Linux: Redirecionamento de Entrada

Monitorando o Clock da CPU

```
grep "cpu MHz" < /proc/cpuinfo > cpu_clock.log && cat cpu_clock.log
```

```
cpu MHz      : 3000.000
cpu MHz      : 1746.784
cpu MHz      : 3000.000
cpu MHz      : 3000.000
cpu MHz      : 3000.000
cpu MHz      : 3000.000
cpu MHz      : 900.006
cpu MHz      : 900.001
```

Monitorando o Linux: Redirecionamento de Entrada

Monitorando o Clock da CPU

```
grep "cpu MHz" < /proc/cpuinfo > cpu_clock.log && cat cpu_clock.log
```

```
cpu MHz      : 3000.000
cpu MHz      : 1746.784
cpu MHz      : 3000.000
cpu MHz      : 3000.000
cpu MHz      : 3000.000
cpu MHz      : 3000.000
cpu MHz      : 900.006
cpu MHz      : 900.001
```

Mais Exemplos

- https://www.gnu.org/software/bash/manual/html_node/Redirections.html

Exercício: Script de Redirecionamento

Escreva um *script bash* que:

1. Receba 3 argumentos:

- \$1: Um programa e seus argumentos
- \$2: Um arquivo de entrada
- \$3: Um arquivo de saída

2. Execute o programa \$1 com stream de Entrada \$2, e redirecione Saída e Erro para \$3

Exercício: Script de Redirecionamento

Escreva um *script bash* que:

1. Receba 3 argumentos:

- \$1: Um programa e seus argumentos
- \$2: Um arquivo de entrada
- \$3: Um arquivo de saída

2. Execute o programa \$1 com stream de Entrada \$2, e redirecione Saída e Erro para \$3

```
#!/usr/bin/bash  
/usr/bin/bash -c "$1 < $2 &> $3"
```

Exercício: Script de Redirecionamento

```
#!/usr/bin/bash
```

```
/usr/bin/bash -c "$1 < $2 &> $3"
```

```
./src/bash_example/log.sh 'grep "cpu MHz"' /proc/cpuinfo out.log && cat out.log
```

```
cpu MHz      : 1303.327
cpu MHz      : 1682.174
cpu MHz      : 3000.000
cpu MHz      : 3000.000
cpu MHz      : 3000.000
cpu MHz      : 3000.000
cpu MHz      : 3000.000
cpu MHz      : 3000.000
cpu MHz      : 3000.000
```

Síntese: Redirecionamento de Streams em *bash*

Sintaxe	Efeito
<code>[n]> arquivo</code> <code>[n]>> arquivo</code>	Redireciona o descritor <code>n</code> para <code>arquivo</code> , sobrescrevendo (<code>></code>) ou adicionando (<code>>></code>)
<code>&> arquivo</code>	Redireciona Erro e Saída Padrão para <code>arquivo</code>
<code>> arquivo 2>&1</code>	Redireciona Saída Padrão (<code>fd=1</code>) para <code>arquivo</code> , e redireciona Erro Padrão (<code>fd=2</code>) para Saída Padrão
<code>< arquivo</code>	Redireciona Entrada Padrão para ler de <code>arquivo</code>

Conectando Streams de Entrada/Saída

Problemas

- Como **conectar os streams** de entrada/saída de dois processos através do bash?
- E através de um **programa em C**?



Conectando Streams de Entrada/Saída

Conectando Streams em *bash*

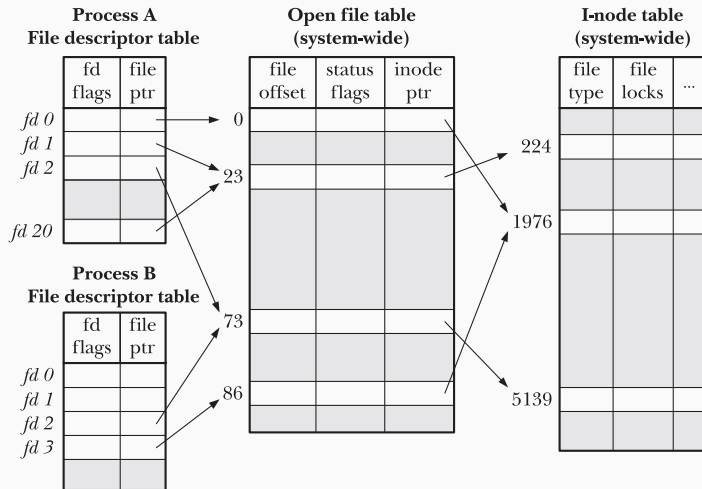
-

Conectando Streams em *C*

-

Duplicando Descritores de Arquivo com Chamadas de Sistema

Descritores de Arquivo em Processos POSIX



Algumas Chamadas POSIX

```
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int open(const char *pathname, int flags, ...);
int close(int fd);
ssize_t read(int fd, void *buffer, size_t count);
ssize_t write(int fd, void *buffer, size_t count);

pid_t fork(void);
int execlp(const char *filename, const char *arg, ...);
```

Algumas Chamadas POSIX

```
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>

int open(const char *pathname, int flags, ...);
int close(int fd);
ssize_t read(int fd, void *buffer, size_t count);
ssize_t write(int fd, void *buffer, size_t count);

pid_t fork(void);
int execlp(const char *filename, const char *arg, ...);

int dup(int oldfd);
int dup2(int oldfd, int newfd);
int pipe(int filedess[2]);
```

Chamadas *dup* e *dup2*

```
int dup(int oldfd);
```

- **Duplica** o descritor de arquivo `oldfd`, usando o menor descritor disponível

```
int dup2(int oldfd, int newfd);
```

- **Duplica** `oldfd` usando `newfd`, fecha `newfd` se necessário

```
man dup | grep "NAME" -A 10
```

NAME

`dup`, `dup2` – duplicate an open file descriptor

SYNOPSIS

```
include <unistd.h>
int dup(int fildes);
int dup2(int fildes, int fildes2);
```

DESCRIPTION

The `dup()` function provides an alternative interface to the service provided [...]

Chamada *pipe*

```
int pipe(int fildes[2]);
```

- Abre um canal de comunicação **entre processos**, usando **descritores de arquivo**

```
man pipe | grep "NAME" -A 10
```

NAME

pipe – create an interprocess channel

SYNOPSIS

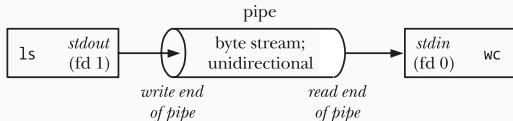
```
include <unistd.h>
int pipe(int fildes[2]);
```

DESCRIPTION

The pipe() function shall create a pipe and place two file descriptors, one each into the arguments fildes[0] and fildes[1], that refer to the open file descriptions [..]

Exercício: Redirecionamento em C

Usando um *pipe* para Conectar Filtros



The Linux Programming API, Michael Kerrisk, cap. 44, pág. 890

Exemplo em C

```
tree src/pipe_example
```

```
src/pipe_example
├── Makefile
├── pipe_example.md
├── pipe_example.org
└── pipe_ls_wc.c
```

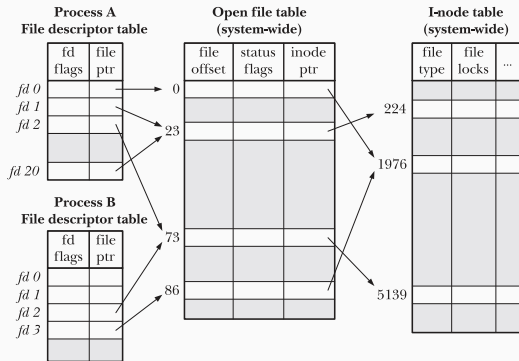
```
0 directories, 4 files
```

Código Fonte

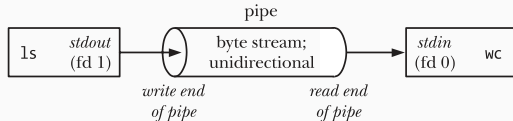
- The Linux Programming API, Michael Kerrisk, cap. 44, pág. 890
- https://man7.org/tlpi/code/online/dist/pipes/pipe_ls_wc.c.html

Síntese: Duplicando Descritores de Arquivo

Descritores de Arquivo



Pipes



Chamadas de Sistema

```
int dup(int oldfd);
```

- **Duplica** o descritor de arquivo `oldfd`, usando o menor descritor disponível

```
int dup2(int oldfd, int newfd);
```

- **Duplica** `oldfd` usando `newfd`, fecha `newfd` se necessário

```
int pipe(int filedes[2]);
```

- Abre um canal de comunicação **entre processos**, usando **descritores de arquivo**

Revisitando os Objetivos de Aprendizagem

Objetivos de Aprendizagem e Exercício

Tópicos

- **Streams**, ou **fluxos** padrão em Processos POSIX, chamadas de sistema **dup** e **dup2**

Objetivos

- Escrever scripts em **bash** e programas em **C** que **redirecionem** os **streams** de **Entrada/Saída** de programas
- Compreender o redirecionamento de streams de Entrada/Saída como a **duplicação de descritores de arquivo** em processos POSIX

Exercício

- Pesquise sobre a chamada de sistema **fcntl**, **escreva** as chamadas **dup** e **dup2** com ela, e as **substitua** no exemplo da conexão de filtros com **pipe**



Redirecionamento de Streams

Padrão em Sistemas POSIX

Pedro Bruel

pedro.bruel@gmail.com

18 de Junho de 2021