

Duplicação e partilha de ficheiros

José Pedro Oliveira
(jpo@di.uminho.pt)

Grupo de Sistemas Distribuídos
Departamento de Informática
Escola de Engenharia
Universidade do Minho

Sistemas Operativos I
2006-2007

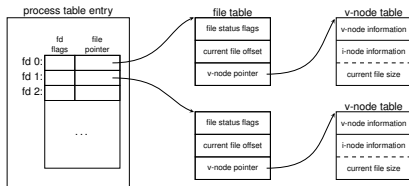


José Pedro Oliveira

Duplicação e partilha de ficheiros

Estruturas de dados do kernel

Estruturas de dados do kernel: ficheiros abertos



José Pedro Oliveira

Duplicação e partilha de ficheiros

Conteúdo

- 1 Estruturas de dados do kernel
- 2 Chamadas ao sistema
 - dup e dup2
- 3 Exemplos de redirecção
- 4 Exercícios
- 5 Referências

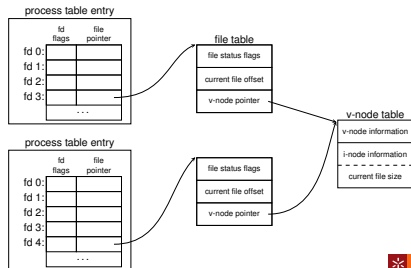


José Pedro Oliveira

Duplicação e partilha de ficheiros

Estruturas de dados do kernel

Dois processos independentes com o mesmo ficheiro aberto



José Pedro Oliveira

Duplicação e partilha de ficheiros

Conteúdo

- 1 Estruturas de dados do kernel
- 2 Chamadas ao sistema
 - dup e dup2
- 3 Exemplos de redirecção
- 4 Exercícios
- 5 Referências

Chamadas ao sistema **dup** e **dup2**

Notas

- **dup** e **dup2** criam uma cópia do descritor `oldfd`. Caso a invocação seja realizada com sucesso, ambos os descritores - `oldfd` e `newfd` - podem ser usados "interchangeably". Os descritores partilham locks, file position pointers e flags. Não partilham, no entanto, a flag `close-on-exec`.
- **dup** - o novo descritor é o menor não utilizado
- **dup2** - `newfd` passa a ser uma cópia de `oldfd`, sendo fechado se necessário

Nota

A invocação `dup2(fd, fd2)` é semelhante a `close(fd2); dup(fd)`. No entanto `dup2` é uma operação atómica.

Chamadas ao sistema **dup** e **dup2**

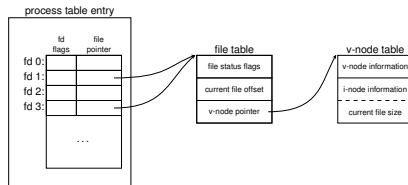
Synopsis

```
#include <unistd.h>

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

Valores de retorno

As chamadas ao sistema **dup** e **dup2** retornam o novo descritor se a operação foi concluída com sucesso ou -1 em caso de erro.

Estruturas de dados do Kernel depois de um **dup**

Efeito de fork, exec e _exit (exit) sobre pipes

Pipes e FIFOs

- fork** - o processo filho recebe uma cópia de todos os descritores abertos do processo pai
- exec** - todos os descritores abertos permanecem abertos a não ser que o bit `FD_CLOEXEC` esteja activo
- _exit** - todos os descritores abertos são fechados; a informação existente em pipes/FIFOs é removida no último close



Conteúdo

- 1 Estruturas de dados do kernel
- 2 Chamadas ao sistema
 - dup e dup2
- 3 Exemplos de redirecção
- 4 Exercícios
- 5 Referências



Redirecção do STDOUT

Exemplo 1 - close/creat

```

1 // includes: stdio.h, stdlib.h, unistd.h, fcntl.h
2
3 int main(void)
4 {
5     int fd;
6
7     close(STDOUT_FILENO);
8     fd = creat("/tmp/lis.txt", 0644);
9
10    execlp("ls", "ls", "-l", NULL);
11
12    perror("execl");
13    exit(EXIT_FAILURE);
14 }
```



Redirecção do STDOUT

Exemplo 2 - dup2

```

1 // includes: stdio.h, stdlib.h, unistd.h, fcntl.h
2
3 int main(void)
4 {
5     int fd;
6
7     fd = creat("/tmp/lis.txt", 0644);
8     dup2(fd, STDOUT_FILENO);
9     close(fd);
10
11    execlp("ls", "ls", "-l", NULL);
12
13    perror("execlp");
14    exit(EXIT_FAILURE);
15 }
```



Redirecção do STDOUT de um processo filho

Exemplo 1 - close/creat

```

1  p = fork();
2  if (p == -1) {
3      perror("fork"); exit(1);
4  } else if (p == 0) {
5      close(STDOUT_FILENO);
6      fd = creat("/tmp/output.txt", 0644);
7      execl("./filho", "filho", NULL);
8      perror("execl");
9      exit(2);
10 } else {
11     wait(NULL);
12 }

```



Conteúdo

- 1 Estruturas de dados do kernel
- 2 Chamadas ao sistema
 - dup e dup2
- 3 Exemplos de redirecção
- 4 Exercícios
- 5 Referências



Redirecção do STDOUT de um processo filho

Exemplo 2 - dup2

```

1  p = fork();
2  if (p == -1) {
3      perror("fork"); exit(1);
4  } else if (p == 0) {
5      fd = creat("/tmp/output.txt", 0644);
6      dup2(fd, STDOUT_FILENO);
7      close(fd);
8      execl("./filho", "filho", NULL);
9      perror("execl");
10     exit(2);
11 } else {
12     wait(NULL);
13 }

```



Interpretadores de comandos e redirecções

```

1  #include <unistd.h>
2
3  int main(void) {
4      write(STDOUT_FILENO, "OUT\n", 4);
5      write(STDERR_FILENO, "ERR\n", 4);
6      return 0;
7  }

```

Compile o programa anterior e execute os seguintes comandos:

- ./a.out > 1.txt
- ./a.out 2> 2.txt
- ./a.out > 3.txt 2>&1
- ./a.out 2>&1 > 4.txt



Exercícios

Implemente programas em C que permitam duplicar as seguintes tarefas:

```
• cat /etc/fstab > /tmp/out.txt
• cat /etc/fstab > /tmp/out.txt 2> /tmp/err.txt
• cat < /etc/fstab > /tmp/out.txt
• cat < /etc/fstab 2> /tmp/err.txt
```



Referências

Bibliografia

- **Advanced Programming in the UNIX Environment, 2nd ed.**
W. Richard Steven, Stephen A. Rago
<http://www.apuebook.com/>
 - Capítulo 3 - File I/O
 - Capítulo 4 - Files and Directories
 - Capítulo 5 - Standard I/O Library
- **Advanced UNIX Programming, 2nd ed.**
Marc J. Rochkind
<http://www.basepath.com/aup/>
- **Linux Programming by Example: The Fundamentals**
Arnold Robbins
<http://authors.phptr.com/robbins/>



Conteúdo

- 1 Estruturas de dados do kernel
- 2 Chamadas ao sistema
 - dup e dup2
- 3 Exemplos de redirecção
- 4 Exercícios
- 5 Referências

