

## Canos (Pipes)

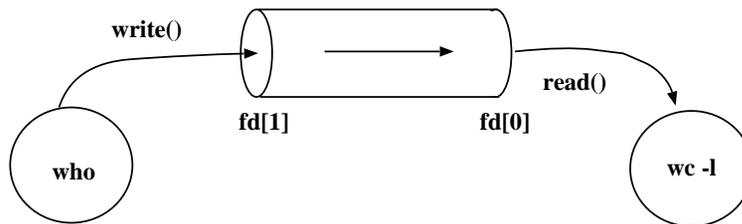
---

As pipes são um mecanismo de comunicação unidirecional entre dois ou mais processos.

Um exemplo do uso de pipes na shell é:

```
% who | wc -l
```

O programa `who` gera uma linha por utilizador e envia-a ao programa `wc` que vai contar o número de linhas recebidas.



## Canos (Pipes)

---

→ Criação de pipes:

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

instancia dois descritores de um ficheiro:

- `fd[0]` - descritor associado à extremidade de leitura.
- `fd[1]` - descritor associado à extremidade de escrita.

Retorna 0 se for bem sucedida e -1 caso ocorram erros.

→ Se um processo tentar ler de uma pipe vazia, mas cuja extremidade de escrita está aberta, o processo adormece até que exista input disponível.

## Exemplo do uso de pipes

---

pipes: geralmente usadas para comunicação entre processos

```
#define READ 0
#define WRITE 1
char *msg= "Ola papa!";
main() {
    int fd[2], msgSize=strlen(msg)+1;
    char buf[100];

    pipe(fd);
    if (fork()==0) {
        close(fd[READ]);
        write(fd[WRITE],msg,msgSize);
        close(fd[WRITE]);
    }
    else {
        close(fd[WRITE]);
        msgSize= read(fd[READ],buf,100);
        printf("Nbytes:%d, Msg:%s\n",msgSize,buf);
        close(fd[READ]);
    }
    exit(0);
}
```

## Exemplo do uso de pipes

---

→ Comunicação bi-direcional?  $\Rightarrow$  2 pipes.

## Somar os elementos de uma matriz: N processos e 1 pipe

---

Passos:

- Processo-pai:

- cria matriz e inicializa
- cria N processos, (processos-filhos herdam a matriz inicializada do pai)
- espera que os filhos devolvam o resultado
- calcula a soma total
- termina

- Processos-filhos:

- calculam soma de sua linha da matriz
- envia resultado para o pai
- termina

## Exemplo utilizando N processos e 1 pipe

---

```
#include <stdio.h>
#include <stdlib.h>
#define N      3
#define READ  0
#define WRITE 1

main() {
    int a[N][N]={{1,1,1},{2,2,2},{3,3,3}};
    int i, somaLin, somaGlo=0, fd[2];

    if (pipe(fd) == -1)
        msg_erro("pipe() falhou!");
    for (i=0; i<N; i++)
        if(fork() == 0) {                /* filho */
            somaLin= soma_vector(a[i]);
            close(fd[READ]);
            write(fd[WRITE], &somaLin, sizeof(int));
            exit(0);
        }
    close(fd[WRITE]);                    /* pai */
    for (i=0; i<N; i++) {
        read(fd[READ], &somaLin, sizeof(int));
        somaGlo += somaLin;
    }
    close(fd[READ]);
    printf("soma da matriz: %d\n", somaGlo); }
}
```

## Exemplo utilizando N processos e 1 pipe

---

```
// Definição da função auxiliar soma_vector
int soma_vector(int v[]) {
    int i, soma=0;

    for (i=0; i<N; i++) soma += v[i];
    return soma;
}
```

## Duplicação de descritores

---

Existem situações em que se torna necessário duplicar descritores de ficheiros, nomeadamente quando o ficheiro representa uma *pipe* e quisermos construir uma *pipeline* entre processos.

→ criação de uma cópia de um descritor:

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

- `dup`: sistema escolhe o menor descritor disponível e coloca-o a apontar para o mesmo ficheiro que `oldfd`.
- `dup2`: utilizador escolhe o descritor em `newfd`. Sistema fecha `newfd` se estiver ativo e coloca-o a apontar para o mesmo ficheiro que `oldfd`.

## Duplicação de descritores

---

Os descritores original e cópia partilham o mesmo apontador para o ficheiro assim como o modo de acesso.

As funções retornam o novo descritor ou -1 se ocorrer erro.

Necessário: `# include <unistd.h>`

Exemplo: redirecionar o standard-output de um processo ao standard-input do outro (e.g. pipes da shell).

## Exemplo com `dup2()`

---

Programa que executa 2 programas dados, ligando o `stdout` do primeiro ao `stdin` do segundo (e.g. `ligador who wc`).

- processo-pai executa primeiro programa (e.g., `who`)
- processo-filho executa o segundo programa (e.g., `wc`)
- pipe para comunicação entre processo-pai e processo-filho
- mas também precisamos indicar aos novos programas que esta comunicação existe

## Exemplo com dup2(): criar proc e pipe

---

```
#define Read 0
#define Write 1
main(int argc, char *argv[])
{
    int fd[2]; // declaração da pipe

    pipe(fd); // criação da pipe
    if (fork() == 0) { /* filho */ // criação do processo

    }
    else { /* pai */

    }
}
```

## Exemplo com dup2()

---

```
#define Read 0
#define Write 1
main(int argc, char *argv[])
{
    int fd[2];

    pipe(fd);
    if (fork() == 0) { /* filho */

        // filho executa segundo programa
        execlp(argv[2], argv[2], NULL);

    }
    else { /* pai */

        // pai executa primeiro programa
        execlp(argv[1], argv[1], NULL);

    }
}
```

## Exemplo com dup2(): pipes?

---

```
#define Read 0
#define Write 1
main(int argc, char *argv[])
{
    int fd[2];

    pipe(fd);
    if (fork() == 0) { /* filho */
        close(fd[Write]); // fechar extremidade
                          // não utilizada

        execlp(argv[2], argv[2], NULL);
    }
    else { /* pai */
        close(fd[Read]); // fechar extremidade
                          // não utilizada

        execlp(argv[1], argv[1], NULL);
    }
}
```

## Exemplo com dup2(): conecta pai com filho

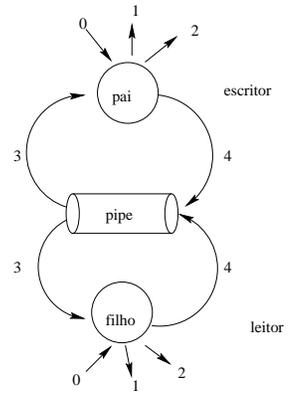
---

```
#define Read 0
#define Write 1
main(int argc, char *argv[])
{
    int fd[2];

    pipe(fd);
    if (fork() == 0) { /* filho */
        close(fd[Write]);
        dup2(fd[Read], 0); // redireciona input
        close(fd[Read]);
        execlp(argv[2], argv[2], NULL);
        perror("ligação não sucedida");
    }
    else { /* pai */
        close(fd[Read]);
        dup2(fd[Write], 1); // redireciona output
        close(fd[Write]);
        execlp(argv[1], argv[1], NULL);
        perror("ligação não sucedida");
    }
}
```

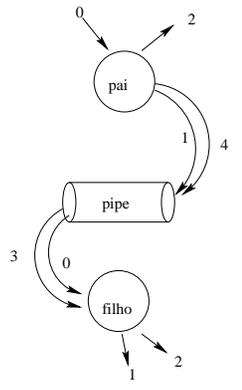
# O exemplo em detalhe (1)

A: logo apos o fork()



	pai	filho
0	stdin	stdin
1	stdout	stdout
2	stderr	stderr
3	pipe_read	pipe_read
4	pipe_write	pipe_write

B: apos os dois processos executarem dup2()

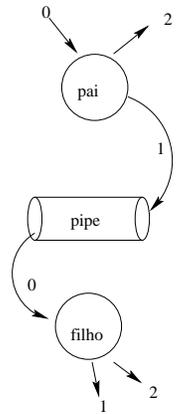


	pai	filho
0	stdin	pipe_read
1	pipe_write	stdout
2	stderr	stderr
3		pipe_read
4	pipe_write	

# O exemplo em detalhe (2)

A: logo apos o fork()

C: apos o close() e antes do execlp()



	pai	filho
0	stdin	pipe_read
1	pipe_write	stdout
2	stderr	stderr

- o processo-pai executa o comando em argv[1] e escreve o resultado na pipe.
- o processo-filho le da pipe e executa o comando argv[2]

Exercício: modifique o programa anterior de forma a criar comunicação bidirecional entre os dois processos.

## Mais um exemplo com pipes: paginação do output

---

Escreva um programa que crie um processo filho para paginar no ecrã as mensagens do processo pai. O processo pai deve enviar ao processo filho a seguinte sequência de linhas:

```
Linha 1
```

```
...
```

```
Linha 100
```

O processo filho deve executar o programa `more` para paginar no ecrã a sequência de linhas recebida do processo pai. Como o paginador `more` precisa ter acesso exclusivo ao terminal, o processo pai não deve terminar antes do processo filho.

## Mais um exemplo com pipes: paginação do output

---

```
void paginacao() {
    pid_t new_proc;
    int fd[2];

    pipe(fd);
    if (is_parent(fork())) {

        wait(NULL);
    } else { // filho

        execlp("more", "more", NULL);
        printf("FILHO: falha no execlp!\n");
    }
    return;
}
```

## Mais um exemplo com pipes: paginação do output

---

```
void paginacao() {
    pid_t new_proc;
    int fd[2];

    pipe(fd);
    if (is_parent(fork())) {
        char buf[100];
        int n;
        close(fd[READ]);

        wait(NULL);
    } else { // filho
        close(fd[WRITE]);

        execlp("more", "more", NULL);
        printf("FILHO: falha no execlp!\n");
    }
    return;
}
```

## Mais um exemplo com pipes: paginação do output

---

```
void paginacao() {
    pid_t new_proc;
    int fd[2];

    pipe(fd);
    if (is_parent(fork())) {
        char buf[100];
        int n;
        close(fd[READ]);
        for (n=1; n<=100; n++) {
            sprintf(buf, "Linha %d\n", n);
            write(fd[WRITE], buf, strlen(buf));
        }
        close(fd[WRITE]);
        wait(NULL);
    } else { // filho
        close(fd[WRITE]);
        dup2(fd[READ], STDIN_FILENO);
        close(fd[READ]);
        execlp("more", "more", NULL);
        printf("FILHO: falha no execlp!\n");
    }
    return;
}
```

## Exemplo: pipe de processos (exame Julho 2002)

---

Escreva um programa que tire partido do uso de *pipes* para implementar comunicação entre processos. O programa deverá implementar um cenário em que o processo principal cria NP processos filho ( $NP > 0$ ) para escreverem alternadamente linhas de *output* de modo a obterem uma sequência semelhante à do exemplo que se segue:

```
Filho 1: 1
Filho 2: 2
Filho 3: 3
Filho 1: 4
Filho 2: 5
Filho 3: 6
Filho 1: 7
Filho 2: 8
Filho 3: 9
Filho 1: 10
```

O primeiro filho a ser criado inicia a sequência a que se seguem os restantes filhos pela ordem que foram criados. O processo decorre repetidamente até que um dos filhos atinga um determinado valor VAL ( $VAL > 0$ ). Atingido esse valor (10 no exemplo), todos os processos filhos devem terminar e o processo pai só deve terminar assim que todos os filhos terminem. Assuma que NP e VAL são constantes do sistema.

## Exemplo: pipe de processos (cont.)

---

```
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>
#define READ 0
#define WRITE 1
#define NP 3
#define VAL 10

main() {
    pid_t proc[NP];
    int fd[NP][2], np, val, i;

    for (np=0; np<NP; np++) // criação de NP pipes
        pipe(fd[np]);
    for (np=0; np<NP; np++) { // criação de NP processos
        if ((proc[np] = fork()) == 0) { // processos filhos
            // processos filhos fecham extremidades adequadas das pipes
            // primeiro processo filho começa tudo e envia primeiro valor
            // todos os processos filhos ficam lendo e escrevendo nas
            // extremidades adequadas até que um deles receba o valor 10
            // filhos terminaram, fecham suas extremidades e saem do programa
        }
    }

    // pai fecha todas as extremidades
    for (np=0; np<NP; np++) {
        close(fd[np][WRITE]);
        close(fd[np][READ]);
    }
    // pai espera pelos filhos
    for (np=0; np<NP; np++)
        waitpid(proc[np],0,0);
    exit(0);
} // termina o programa
```

## Exemplo: pipe de processos (cont.)

---

```
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>
#define READ 0
#define WRITE 1
#define NP 3
#define VAL 10

main() {
    pid_t proc[NP];
    int fd[NP][2], np, val, i;

    for (np=0; np<NP; np++)
        pipe(fd[np]);

    for (np=0; np<NP; np++) {
        if ((proc[np] = fork()) == 0) {
            // processos filhos fecham extremidades adequadas das pipes
            for (i=0; i<NP; i++) {
                if (i != np) close(fd[i][WRITE]);
                if (i != (np-1+NP)%NP) close(fd[i][READ]);
            }
            // primeiro processo filho começa tudo e envia primeiro valor
            if (np == 0) {
```

```

    val = 1;
    printf("Filho 1: 1\n");
    write(fd[np][WRITE], &val, sizeof(int));
}
// todos os processos filhos ficam lendo e escrevendo nas
// extremidades adequadas até que um deles receba o valor 10
while(read(fd[(np-1+NP)%NP][READ], &val, sizeof(int))) {
    if (val == VAL) break;
    val++;
    printf("Filho %d: %d\n", np + 1, val);
    write(fd[np][WRITE], &val, sizeof(int));
}
// filhos terminaram, fecham suas extremidades e saem do programa
close(fd[np][WRITE]);
close(fd[(np-1+NP)%NP][READ]);
exit(0);
}
}
// pai fecha todas as extremidades
for (np=0; np<NP; np++) {
    close(fd[np][WRITE]);
    close(fd[np][READ]);
}
// pai espera pelos filhos
for (np=0; np<NP; np++)
    waitpid(proc[np],0,0);
exit(0);
} // termina o programa

```