Comunicação entre processos: *Pipes*

Sistemas de Computadores

2014/2015

Luís Nogueira / Nuno Pereira / Luís Miguel Pinho

Introdução

- Em UNIX há uma tentativa de uniformização da interface de comunicação entre processos com a interface do sistema de ficheiro

- Assim, para usar alguns dos mecanismos de comunicação entre processos é fundamental conhecer a interface com o sistema de ficheiros

I/O em Linux

- Em C é habitual usar-se a biblioteca *<stdio.h>* para efectuar operações de I/O
  - *printf()*, *scanf()*, *fopen()*, ...
- Fornece um elevado nível de abstracção
  - *Buffering*, conversões de I/O, formatação, ...
- No entanto, nem sempre estas funcionalidades são desejáveis/possíveis
  - Por ex., se se pretende controlar operações que são específicas de um dado dispositivo, controlar o tamanho do *buffer*, ou realizar I/O não bloqueante

I/O em Linux

- Os **descritores de ficheiro** fornecem um interface de baixo nível para operações de I/O
  - *open()*, *read()*, *write()*, *close()*, ...

- Em sistemas POSIX, um descritor de ficheiro é um inteiro que identifica um ficheiro aberto ou recurso do sistema
  - Um índice para uma tabela, denominada **tabela de descritores de ficheiro**
  - Esta tabela é apenas acedível pelo núcleo do SO e não pode ser directamente manipulada pelos processos

Tabela de descritores

- Cada processo **possui a sua tabela de descritores**
  - Descritores 0, 1 e 2 são pré-definidos e identificam o *stdin*, *stdout* e *stderr*, respectivamente

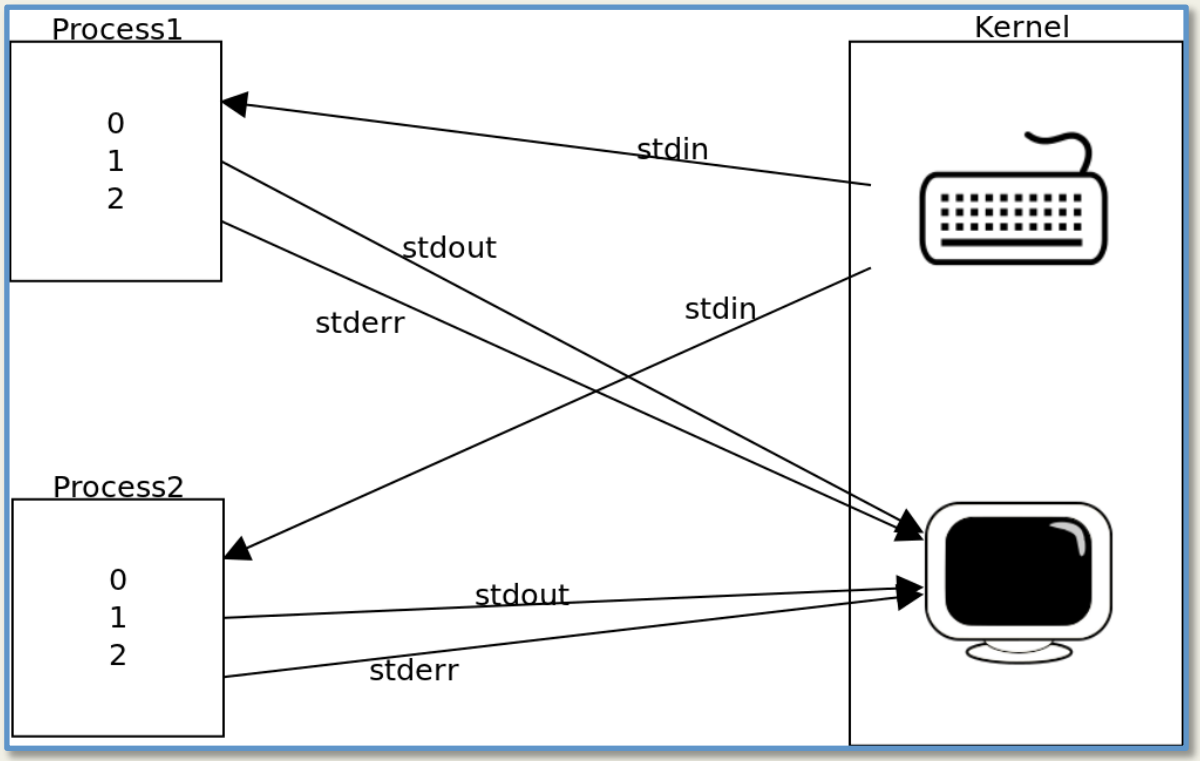
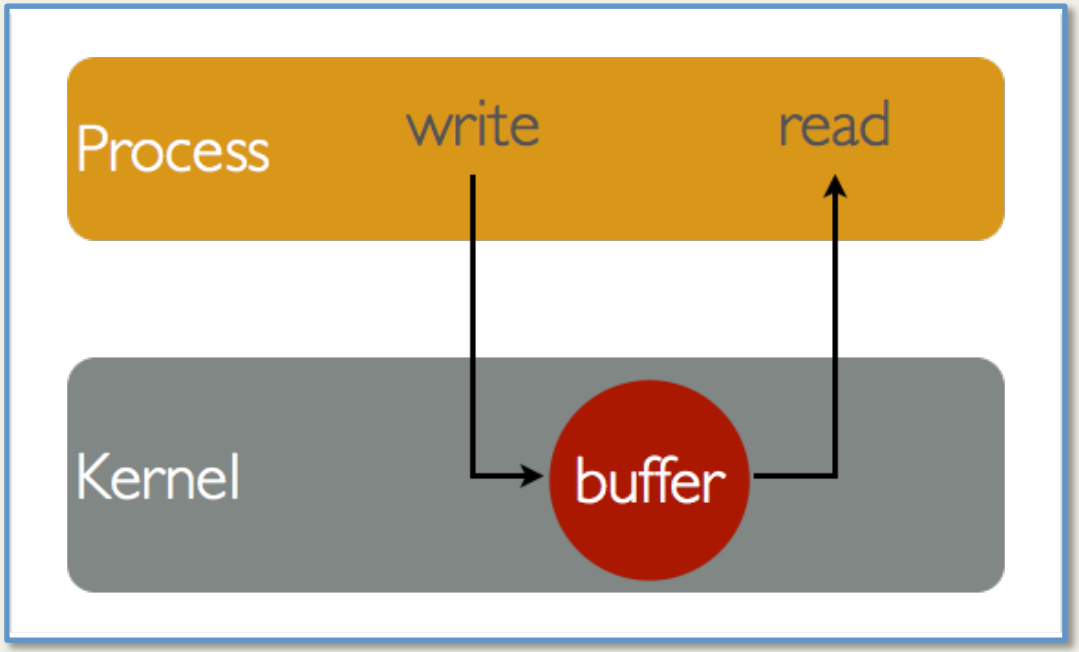


Tabela de descritores

- Um processo pode abrir o mesmo ficheiro/ recurso duas vezes, tendo como resultado dois descritores diferentes
- E se dois processos diferentes abrirem o mesmo ficheiro?
  - Não há qualquer garantia que nos dois processos o ficheiro seja associado ao mesmo descritor
- Os descritores de ficheiros são copiados num *fork* e (normalmente) são preservados num *exec*

Pipes

- Estrutura FIFO, orientada ao fluxo de bytes, acedidos pelos seus descritores de leitura e escrita



Pipes

- O fluxo de bytes escrito usando o descritor de escrita pode ser lido usando o descritor de leitura
  - **Leitura remove dados do *pipe***

- Como resultado, os ***pipes* são uni-direccionais**
  - Para o fluxo de dados nos dois sentidos têm obrigatoriamente de ser usados dois *pipes*, com sentidos opostos

Unnamed pipes

- Tornam-se um meio de comunicação entre processos explorando o facto dos descritores de um processo serem herdados pelos seus descendentes

- Exigem, portanto, **uma relação hierárquica entre os processos comunicantes**

Função *pipe*

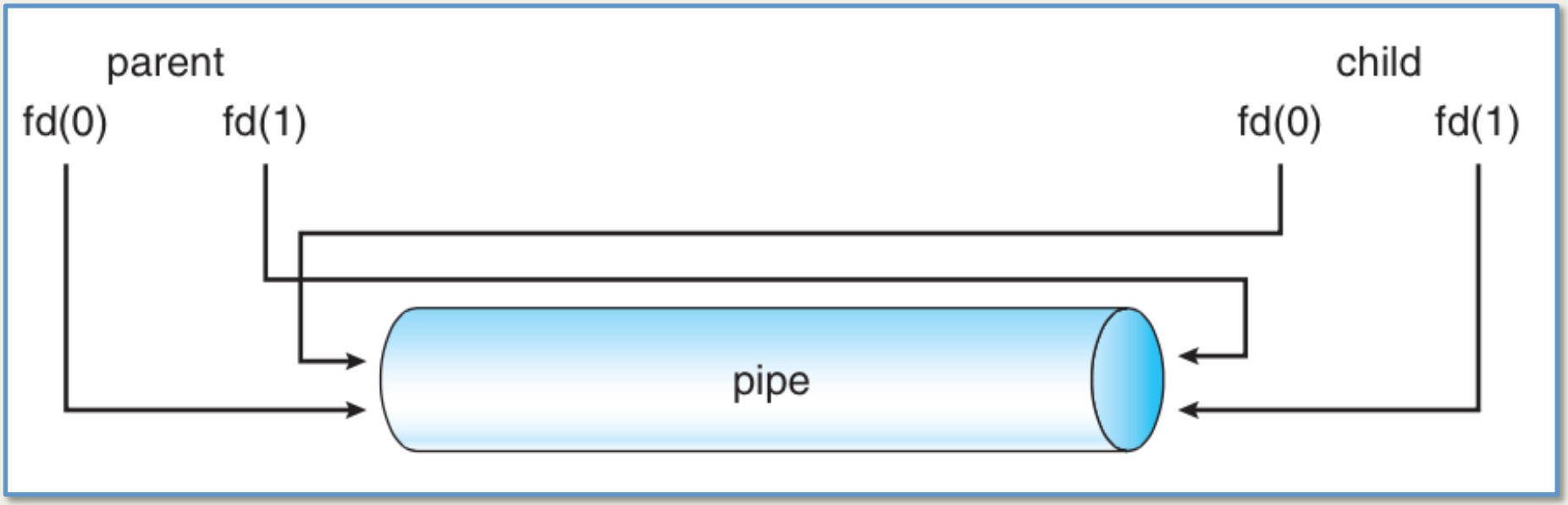
```
#include <unistd.h>

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

- Cria um *pipe*, instanciando dois descritores em fd[]
  - fd[0] é o descritor de leitura
  - fd[1] é o descritor de escrita

- Retorna
  - 0 em caso de sucesso; -1 em caso de erro

Partilha de descritores

- Pai tem de criar o *pipe* antes do *fork()*
  - Caso contrário, descritores não são partilhados



Função *read*

```
#include <unistd.h>

size_t read(int fd, void *buffer,
   size_t n_bytes)
```

- Tenta ler *n_bytes*, a partir do descritor *fd*, colocando-os em *buffer*, retornado:
  - nº de bytes efectivamente lidos
  - 0 se não há mais bytes para ler
  - -1 em caso de erro

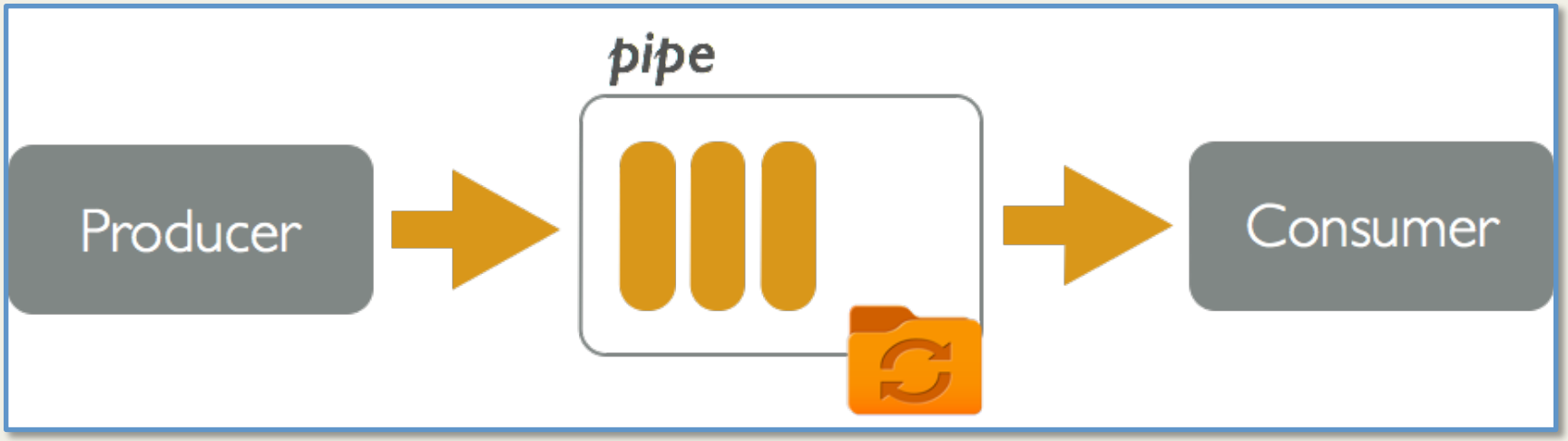
Função *write*

```
#include <unistd.h>

size_t write(int fd, void *buffer,
   size_t n_bytes)
```

- Tenta escrever *n_bytes* contidos em *buffer* no descritor *fd*, retornado:
  - o nº de bytes efectivamente escritos
  - -1 em caso de erro

Leituras e escritas

- Sincronização de escritas/leituras é garantida
  - *write* bloqueia quando o *pipe* está cheio
  - *read* bloqueia enquanto não existirem dados no *pipe*



Função *close*

```
#include <unistd.h>

int close(int fd)
```

- Fecha o ficheiro/recurso associado ao descritor indicado em *fd*
  - Remove o descritor da tabela de descritores do processo, podendo ser reutilizado
- Retorna
  - 0 em caso de sucesso; -1 em caso de erro

Fecho dos descritores

- Os descritores **não usados pelo processo devem ser fechados**
  - Boa prática de programação, vital para que o comportamento seguinte se verifique
- O fecho dos descritores usados num *pipe* indica o **fim comunicação por parte desse processo**
  - Um *write* para um *pipe* com todas as extremidades de leitura fechadas falha e o processo recebe um sinal SIGPIPE
  - Um *read* de um *pipe* com todas as extremidades de escrita fechada retorna 0 (*end-of-file*)

Exemplo

```
#define BUFFER_SIZE 80

int main(void){
   char read_msg[BUFFER_SIZE];
   char write_msg[BUFFER_SIZE] = ”Funciona!";
   int fd[2];
   pid_t pid;

   /* cria o pipe */
   if(pipe(fd) == -1){
      perror("Pipe failed”);
      return 1;
   }
```

Exemplo

```
   pid = fork();

   if(pid > 0){
      /* fecha a extremidade não usada */
      close(fd[0]);

      /* escreve no pipe */
      write(fd[1],write_msg,strlen(write_msg)
+1);

      /* fecha a extremidade de escrita */
      close(fd[1]);
   }
```

Exemplo

```
else{
   /* fecha a extremidade não usada */
   close(fd[1]);

   /* lê dados do pipe */
   read(fd[0], read_msg, BUFFER_SIZE);

   printf("Filho leu: %s", read_msg);

   /* fecha a extremidade de leitura */
   close(fd[0]);
}
```

Exercício 1

- Implemente um programa que cria um novo processo.
  - O processo pai envia 2 inteiros introduzidos pelo utilizador ao processo pai usando um *pipe*
  - O filho lê esses dois inteiros do *pipe*, calcula a sua soma e termina usando o valor da soma como valor de saída
  - O pai espera que o filho termine e imprime o resultado

Exercício 2

- Altere o exercício anterior para suportar um número indeterminado de inteiros que são enviados do pai para o filho usando o *pipe*
  - Assuma que o envio termina quando o utilizador introduzir o valor 0