Comunicação Inter-Processo em Unix

- Pipes
- FIFO (Named Pipes)

1

Pipes

2

Comunicação entre pai e filho

Ao chamar um fork(), processos pai e filho deixam de compartilhar memória

Como fazer então para que o resultado de um processo possa ser recebido pelo outro?

Exemplos:

- Processo pai distribui dados para os processos filho, que devolvem resultados que são consolidados pelo pai.

3

Pipes como mecanismo de comunicação entre processos

- Na shell, em **`cmd1|cmd2`** estamos direcionando o STDOUT de cmd1 para o STDIN de cmd2
- Exemplos:

  > `ps aml | sort –r –k 7 | less` // lista os processos com maior uso de memória

  > `ps axl | grep zombie` // lista todos os processos zombie (*)
- Isso é um *pipe* (ou *pipeline*), ou seja, um canal, que é uma estrutura de dados do kernel
- Processos pai e filho podem se comunicar através de pipes. Usa-se a a função de sistema pipe()

Obs: (*) processo zombie já terminou a sua execução mais ainda tem sua entrada na tabela de processos, que é necessária para o pai ler o status de exit do filho.

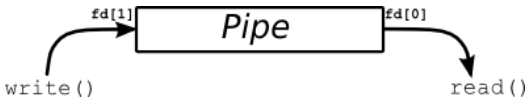
4

Função pipe()

- Cria um canal de comunicação entre processos
- Definido em <unistd.h>

  int pipe(int fd[2]);

- Cria dois canais de comunicação (visíveis pelo pai e filho)
  - fd[0] é aberto para leitura
  - fd[1] é aberto para escrita

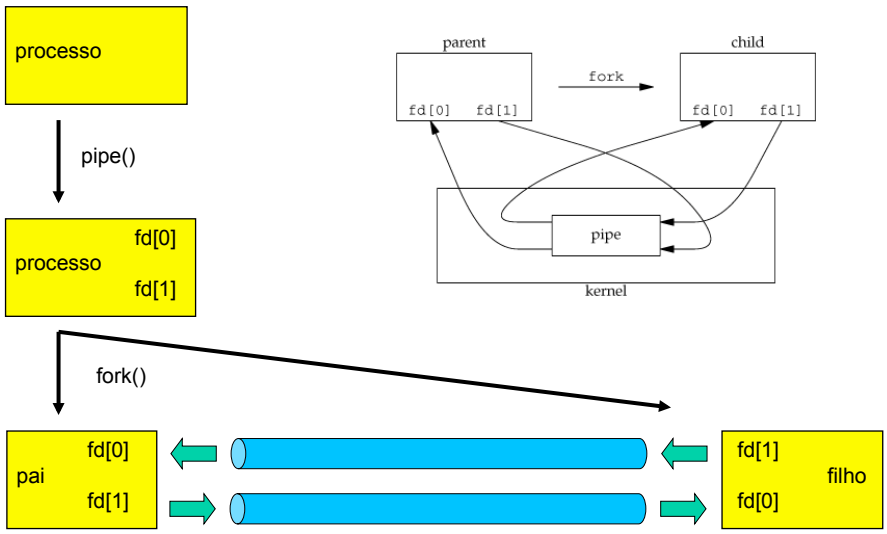


- Retorna:
  - 0 (zero) em caso de sucesso
  - -1 em caso de erro

Função pipe()

- Pipes abertos pela função ***pipe()*** devem ser fechados pela função ***close()***
- Dados **escritos** no descritor de arquivo ***fd[1]*** podem ser **lidos** do ***fd[0]***
- Dois processos podem se comunicar através de um pipe se eles lêem e escrevem em fd[0] e fd[1] respectivamente
- Os dados são **transmitidos** e recebidos através de
  - ***write()*** e ***read()***

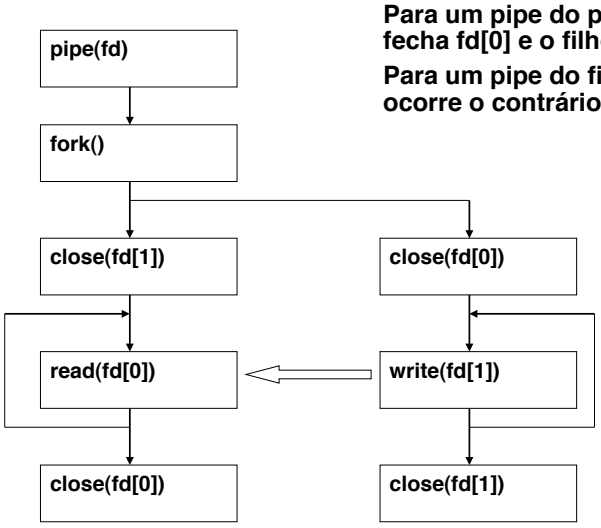
Função pipe()



Esquema de comunicação via pipe

Para um pipe do pai para o filho, o pai fecha fd[0] e o filho fecha o fd[1].

Para um pipe do filho para o pai, ocorre o contrário.



Processo pai

Processo filho

Criando os canais com pipe()

```
int fd[2];  // descritor dos pipes

if (pipe(fd) < 0)
{
   puts ("Erro ao abrir os pipes");
   exit (-1);
}
```

- Em caso de sucesso, a chamada à pipe() retorna 0 e fd[0] conterá o descritor de leitura e fd[1] o de escrita
- Em caso de falha, a função retorna -1

Função write()

Utilizada para escrever dados em um arquivo ou qualquer outro objeto identificado por um descritor de arquivo (file descriptor)

- Definido em <unistd.h>

  ssize_t write(int fildes, const void *buf, size_t nbyte);

- Onde
  - fildes : é o descritor do arquivo (ou do pipe)
  - buf : endereço da área de memória onde estão os dados que serão escritos
  - nbytes : número de bytes que serão escritos
- Valor retornado:
  - Em caso de sucesso, a função retorna a quantidade de dados escritas
  - Em caso de falha, o valor retornado difere da quantidade de bytes enviados

Função read()

- Lê dados de um arquivo ou de qualquer outro objeto identificado por um descritor de arquivo
- Definido em <unistd.h>

```
ssize_t  read(int fildes,void *buf, size_t nbyte);
```

- Onde:
  - fildes : descritor do arquivo
  - buf : endereço de memória onde os dados serão armazenados depois de lidos
  - nbyte : quantidade máxima de bytes que podem ser transferidos
- Retorna:
  - Quantidade de dados lidos

Exemplo do uso de pipe

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

int main (int argc, char *argv[])
{
    int nDadosTx, nDadosRx; // quantidade de dados transmitidos/recebidos
    int fd[2];              // descritor dos pipes
    const char textoTX[] = "uma mensagem";
    char textoRX[sizeof textoTX];

    if (pipe(fd) < 0)
    {
                puts ("Erro ao abrir os pipes");
                exit (-1);
    }

    nDadosTx = write(fd[1], textoTX, strlen(textoTX)+1);
    printf("%d dados escritos\n", nDadosTx);

    nDadosRx = read(fd[0], textoRX, sizeof textoRX);
    printf("%d dados lidos: %s\n", nDadosRx, textoRX);

    close(fd[0]); close(fd[1]);

    return 0;
}
```

```
pipe$ make teste
gcc -g -Wall -o teste teste.c
pipe$ ./teste
13 dados escritos
13 dados lidos: uma mensagem
pipe$
```

Exemplo: pai escreve para o filho

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
void main ()
{
   int fd[2];
   pipe(fd);
   if (fork() == 0)
   { /* filho */
              close(fd[1]); /* fd[1] desnecessario */
              read(fd[0], ...); /* lê do pai */
              ...
   }
   else
   {
              close(fd[0]); /* fd[1] desnecessario */
              write(fd[1], ...); /* escreve para o filho */
              ...
   }
}
```

Outras formas de comunicação entre processos

- Named pipes (FIFO) usa o Sistema de Arquivos (mas não cria um arquivo de fato)
- Usa-se `mkfifo()` ou `mknod()`

  Exemplo:

```
mkfifo my_pipe
gzip -9 –c < my_pipe > out.gz &
cat file > my_pipe
…
rm my_pipe
```

- Anonymous pipe = apenas em uma direcão

14

FIFO (ou named pipes)

- FIFO permite que dois processos quaisquer se comuniquem
- Um named pipe é um tipo especial de arquivo presente no sistema de arquivo

Criando FIFO's

- FIFO's são criadas pelo comando mkfifo

  $ mkfifo filename

- FIFO pela linha de comando:

```
$ mkfifo fpipe
$ ls -ls
total 0
0 prw-r--r-- 1 meslin meslin 0 2008-10-11 00:16 fpipe
$ ls ../IPC/reserva/
aviao    aviao.c~  aviao.h~  compra.c   makefile  vende.c
aviao.c  aviao.h   compra    compra.c~  vende     vende.c~
$ grep "\.c" < fpipe &
[1] 5852
$ ls -ls ../IPC/reserva/ > fpipe
 4 -rw-r--r-- 1 meslin meslin   918 2008-09-11 22:28 aviao.c
 4 -rw-r--r-- 1 meslin meslin   920 2008-09-11 22:27 aviao.c~
 4 -rw-r--r-- 1 meslin meslin   675 2008-09-11 22:29 compra.c
 4 -rw-r--r-- 1 meslin meslin   670 2008-09-11 22:05 compra.c~
 4 -rw-r--r-- 1 meslin meslin   666 2008-09-11 22:30 vende.c
 4 -rw-r--r-- 1 meslin meslin   662 2008-09-11 22:08 vende.c~
[1]+  Done                    grep "\.c" < fpipe
$
```

Criando FIFO's

- Em um programa C, podemos utilizar a seguinte chamada do sistema (system call)
  - Definido em: <sys/stat.h>

    int mkfifo(const char *filename, mode_t mode);

  - Onde:
    - filename : nome da FIFO a ser criada
    - mode : bits com as permissões de criação da mesma forma que para arquivos (user, group, other) x (read, write)
      - S_IRUSR S_IWUSR S_IRGRP S_IWGRP S_IROTH S_IWOTH ...
  - Retorna:
    - Em caso de sucesso, 0 (zero)
    - Em caso de erro, -1

Criando uma FIFO

```
#include <stdio.h>
#include <sys/stat.h>
int main (void)
{
   if (mkfifo("minhaFifo", S_IRUSR | S_IWUSR) == 0)
   {
               puts ("FIFO criada com sucesso");
               return 0;
   }
   puts ("Erro na criação da FIFO");
   return -1;
}
```

```
fifo$ ./mkfifo
FIFO criada com sucesso
fifo$ ls -ls minhaFifo
0 prw------- 1 meslin meslin 0 2008-10-11 13:42 minhaFifo
fifo$ ./mkfifo
Erro na criação da FIFO
fifo$
```

Utilizando a FIFO

```
int main (void)
{
   FILE *pArq;
   if ((pArq = fopen("minhaFifo", "w")) == NULL)
   {
               puts ("Erro ao abrir a FIFO para escrita"); return -1;
   }
   fputs ("Melancia sem caroço", pArq);
   fclose (pArq);
   return 0;
}
```

```
int main (void)
{
   FILE *pArq;
   char ch;
   if ((pArq = fopen("minhaFifo", "r")) == NULL)
   {
               puts ("Erro ao abrir a FIFO para escrita"); return -1;
   }
   while ((ch = fgetc(pArq)) != EOF) putchar (ch);
   fclose (pArq);
   return 0;
}
```

FIFO: comunicação unidirecional

- Um programa não deve abrir uma FIFO para leitura e escrita.
- Se houver necessidade de comunicação bidirecional, devemos utilizar duas FIFO's, uma para cada direção

Abertura de FIFO pode ser blockin ou non-blocking

- Determinado pela flag O_NONBLOCK

open(const char *path, O_RDONLY);
– Neste caso, a chamada a open( ) ficará bloqueada até que um processo abra a mesma FIFO para escrita.

open(const char *path, O_RDONLY | O_NONBLOCK);
– A chamada a open( ) retornará imediatamente, mesmo que a FIFO não tenha sido aberta para escrita por outro processo

open(const char *path, O_WRONLY);
– Neste caso, a chamada a open( ) irá bloquear o processo até que a mesma FIFO seja aberta para leitura

open(const char *path, O_WRONLY | O_NONBLOCK);
– A chamada retorna imediatamente, mas se a FIFO não estiver aberta para leitura por algum processo, open( ) retorna -1 indicando erro

Abrindo FIFO para leitura sem bloqueio

```
#include <…>
#define OPENMODE (O_RDONLY | O_NONBLOCK)
#define FIFO "minhaFifo"
int main (void)
{
    int fpFIFO;
    char ch;
    if (access(FIFO, F_OK) == -1)
    {
                if (mkfifo (FIFO, S_IRUSR | S_IWUSR) != 0)
                {
                        fprintf (stderr, "Erro ao criar FIFO %s\n", FIFO);
                        return -1;
                }
    }
    puts ("Abrindo FIFO");
    if ((fpFIFO = open (FIFO, OPENMODE)) < 0)
    {
                fprintf (stderr, "Erro ao abrir a FIFO %s\n", FIFO);
                return -2;
    }
    puts ("Começando a ler...");
    while (read (fpFIFO, &ch, sizeof(ch)) > 0)
                putchar (ch);
    puts ("Fim da leitura");
    close (fpFIFO);
    return 0;
}
```

```
fifo$ ./leFifoNonBlocking
Abrindo FIFO
Começando a ler...
Fim da leitura
fifo$
```

Abrindo FIFO para escrita sem bloqueio

```
#include <…>
#define OPENMODE (O_WRONLY | O_NONBLOCK)
#define FIFO "minhaFifo"
int main (void)
{
    int fpFIFO;
    char mensagem[] = "Melancia sem caroço";
    if (access(FIFO, F_OK) == -1)
    {
                if (mkfifo (FIFO, S_IRUSR | S_IWUSR) != 0)
                {
                        fprintf (stderr, "Erro ao criar FIFO %s\n", FIFO);
                        return -1;
                }
    }
    puts ("Abrindo FIFO");
    if ((fpFIFO = open (FIFO, OPENMODE)) < 0)
    {
                fprintf (stderr, "Erro ao abrir a FIFO %s\n", FIFO);
                return -2;
    }
    puts ("Começando a escrever...");
    write(fpFIFO, mensagem, strlen(mensagem));
    puts ("Fim da escrita");
    close (fpFIFO);
    return 0;
}
```

```
fifo$ ./escreveFifoNonBlocking
Abrindo FIFO
Erro ao abrir a FIFO minhaFifo
fifo$
```

O mesmo programa, com bloqueio

```
fifo$ ./leFifoNonBlocking &
Abrindo FIFO
[1] 6045
fifo$ ./escreveFifoNonBlocking
Abrindo FIFO
Começando a escrever...
Fim da escrita
Começando a ler...
Melância sem caroçoFim da leitura
[1]+  Done                    ./leFifoNonBlocking
fifo$
fifo$ ./escreveFifoNonBlocking &
Abrindo FIFO
[1] 6047
fifo$ ./leFifoNonBlocking
Abrindo FIFO
Começando a ler...
Começando a escrever...
Fim da escrita
Melância sem caroçoFim da leitura
[1]+  Done                    ./escreveFifoNonBlocking
fifo$
```