Sistemas Operativos

2025– 2026

Comunicação inter-processo em Unix com Named pipes (FIFOs)

1

Tópicos

Pipes (*named pipes* / FIFOs)

Exemplo Cliente-Servidor

Bibliografia específica:

- *Beginning Linux Programming*; Mathew & Stones
  Caps 10,11,12

2

Mecanismos de comunicação Unix – FIFOs / Named Pipes

Ficheiros em ambiente Unix
Brevíssima introdução

» Operações básicas
» Flags e modo de abertura
» Controlo de permissões por default

open

```
int open(const char *pathname, int flags);
```

Flags (as habituais)

Tipo de operação
- O_RDONLY → Só leitura
- O_WRONLY → Só escrita
- O_RDWR → Leitura e escrita

Acesso ao ficheiro
- O_CREAT → Cria se não existir
- O_EXCL → Falha se já existir

Comportamento no exec
- O_CLOEXEC → Fecha em **exec***xxx*()

open

```
int open(const char *pathname, int flags);
```

Flags (as habituais)

Acesso ao ficheiro

- O_CREAT → Cria se não existir

– Cria o ficheiro se não existir

Quando há lugar a criação de ficheiro

Quais as permissões (*modo*) do ficheiro?

– O ficheiro é criado com as permissões especificadas no terceiro parâmetro da função (-> próximo slide)

– Se esse parâmetro não for especificado, o código da função vai buscar na mesma à pilha o valor que corresponderia a esse parâmetro, resultando num *modo* de ficheiro imprevisível e provavelmente errado

open

```
int open(const char *pathname, int flags, mode_t mode);
```

Modo → Permissões – flags conjugáveis. Valor em octal e significado

– S_IRWXU (00**700**) → user pode: read, write, execute
– S_IRUSR (00**400**) → user pode: read
– S_IWUSR (00**200**) → user pode: write
– S_IXUSR (00**100**) → user pode: execute

– S_IRWXG (00**070**) → group pode: read, write, execute
– S_IRGRP (00**040**) → group pode: read
– S_IWGRP (00**020**) → group pode: write
– S_IXGRP (00**010**) → group pode: execute

– S_IRWXO (00**007**) → others podem: read, write, execute
– S_IROTH (00**004**) → others podem: read permission
– S_IWOTH (00**002**) → others podem: write permission
– S_IXOTH (00**001**) → others podem: execute permission

(Nota: User = dono)

read / write

```
ssize_t read(int fd, void *buf, size_t count);

ssize_t write(int fd, const void *buf, size_t count);
```

Transferem uma sequência de bytes de forma sequencia de ficheiro para memória (read) ou de memória para ficheiro (write)

É preciso indicar

- Qual o ficheiro (handle para o descritor)
- Onde estão/vão estar os dados
- Quantos bytes a ler/escrevr
- Retorna -1 (erro) ou o numero de **bytes** lidos/escritos e é importante confarmer se os bytes lidos/escritos são o valor que era esperado

umask / fmask / dmask

Identificam os bits que podem ser definidos nas permissões de novos ficheiros e directorias criados por um processo

- Na shell – umask é um commando
- Em programação C – umask é uma função
- Em /fstab – Define *defaults* para o dispositivo. Também disponíveis: fmask (apenas ficheiros) dmask (apenas directorias)

umask → ficheiros e directorias

fmask → ficheiros (específico para /etc/fstab)

dmask → directorias (específico para /etc/fstab)

fork() → Filho herda umask do pai
exec*xx*() → Não afecta esta definição (pertence ao processo)

umask / fmask / dmask

Identificam os bits que podem ser definidos nas permissões de novos ficheiros e directorias criados por um processo

Octal digit in `umask` command	Permissions the mask will prohibit from being set during file creation
0	any permission may be set (read, write, execute)
1	setting of execute permission is prohibited (read and write)
2	setting of write permission is prohibited (read and execute)
3	setting of write and execute permission is prohibited (read only)
4	setting of read permission is prohibited (write and execute)
5	setting of read and execute permission is prohibited (write only)
6	setting of read and write permission is prohibited (execute only)
7	all permissions are prohibited from being set (no permissions)

Mecanismos de comunicação Unix – FIFOs / Named Pipes

FIFOs / Named pipes

Mecanismo semelhante aos pipes anónimos mas com capacidade de identificação própria (visível a processos independentes)

Utilizam-se de forma semelhante a ficheiros

Obs.

Estes slides contêm um exemplo. Há mais exemplos em documentos à parte dos slides

Os exemplos destinam-se a ser analisados com calma fora de aula. Recomenda-se a modificação do código e experimentação. Uma metodologia só de "olhar" não tem grandes resultados. Os exemplos dos slides não são necessariamente para uso directo no trabalho, mas podem ajudar.

Mecanismos de comunicação Unix - FIFO's

- Apesar de muito fáceis de utilizar, os pipes anónimos têm a desvantagem de não poderem ser utilizados por processos não relacionados entre si

  Razão:

  ➢ Os identificadores de acesso às extremidades dos pipes são meros *handles* (índices na tabela de ficheiros abertos)
  - Só fazem sentido no contexto do processo que os criou (ou de processos filhos dele)
  - Outros processos têm tabelas de ficheiros abertos totalmente independentes e o descritor de um pipe criado por um processo distinto (não relacionado) é totalmente inútil

  ➢ Para processos não relacionados pai/filho ou derivados do mesmo pai é necessário um mecanismo que tenha uma **identificação** que permita a qualquer processos de o referir

  → **named pipes** (slide seguinte)

Mecanismos de comunicação Unix - FIFO's

Named pipes (Pipes com nome) ou FIFO's (First in First Out)

- Mecanismo de comunicação de utilização análoga aos **pipes anónimos** (a interface tipo ficheiro é mantida) mas em que existe um nome que pode ser utilizado por processos não relacionados para obtenção de acesso ao FIFO

- A utilização de um FIFO é **análoga à utilização de um ficheiro**: qualquer processo o pode abrir/ler/escrever se souber o seu *pathname* (e tiver permissões)

- Há algumas diferenças na **semântica de bloqueio** entre ficheiros e FIFOS que são **importantes** (próximo slide)

Mecanismos de comunicação Unix - FIFO's

Named pipes (Pipes com nome) ou FIFO's (First in First Out)

Diferenças quando comparados com ficheiros regulares

- ***leitura de ficheiro vazio / fim de ficheiro*** *vs.* ***Leitura de FIFO vazio***

  Ficheiro: devolve logo — FIFO: bloqueia e aguara (que "outro" processo escreva)

- ***Escrita em dispositivo cheio*** *vs.* ***Escrita em FIFO vazio***

  Ficheiro: devolve logo (erro) — FIFO: bloqueia e aguarda (que "outro" processo leia)

- ***Abertura de ficheiro*** *vs.* ***Abertura de FIFO***

  Ficheiro: abre e devolve — FIFO: Normalmente bloqueia e aguarda que outro processo abra para a operação "inversa" (***R*** <-> ***W***)

Mecanismos de comunicação Unix - FIFO's

Diferenças Ficheiros regulares *vs.* FIFO

A lógica das diferenças na semântica de bloqueio ("de aguardar") é

Leitura

- *Um ficheiro regular não é um mecanismo de comunicação. Se já não tem mais dados, não faz sentido aguardar que haja e a leitura devolve logo*
- *Pelo contrário, num FIFO, como mecanismo de comunicação é natural que, se agora não tenham dados, é provável que venha a ter daqui a pouco (escritos pela outra parte) e o melhor é aguardar*

Escrita

- *A mesma ideia: se um FIFO está cheio, provavelmente daqui a pouco já não estará porque a outra parte entretanto leu e retirou a informação*

Abertura

- *A lógica aqui é "não vale a pena avançar já se "do outro lado" ainda não há ninguém para a operação inversa. Isto pode dar problema de deadlock*

Mecanismos de comunicação Unix - FIFO's

Diferenças ficheiros regulares *vs.* FIFO

- O uso descuidado das operações habituais de abertura/leitura/escrita pode causar problemas de espera mútua ("deadlock")
- Os processos ficam mutuamente à espera uns dos outros e nenhum avança
- É necessário ter cuidado com este aspeto
  - Pode facilmente tornar-se na parte mais complicada do uso de FIFOS mas é facilmente resolvido planeando com cuidado a sequência das operações
  - A semântica de bloqueio pode ser configurada e o uso dos FIFOS pode ser bloqueante = "síncrono" (descrito atrás) ou não-bloqueante (assíncrono)
    - A semântica síncrona é mais flexível mas mais complexa de usar

Mecanismos de comunicação Unix: FIFO's

FIFO's (*named pipes*):

– O facto de terem um nome que pode ser conhecido por outros processos (não relacionadas via **`fork()`**) permite a sua aplicação numa gama mais vasta de situações

Utilização de FIFO's

– São utilizadas as funções sistema habituais para ficheiros.

```
open
close
read
write
...
```

– A única diferença que poderá existir relativamente a ficheiros verdadeiros está relacionada com questões de sincronização

Mecanismos de comunicação Unix: FIFO's

Funções Unix/C necessárias (1)

- Os FIFO's são manipulados através dos mesmos mecanismos que os ficheiros normais

```
open(const char * filename, int flags)
```

Abre um FIFO/ficheiro já existente

```
write(int fd, const void * buffer, size_t size)
```

Escreve num FIFO/ficheiro previamente aberto

```
read(int, void * buf, size_t size)
```

Lê de um FIFO previamente aberto

Mecanismos de comunicação Unix: FIFO's

Funções Unix / C necessárias (2)

```
mkfifo(const char * filename, int flags)
```

Cria um FIFO

```
unlink(const char * filename)
```

Remove um FIFO/ficheiro

```
fcntl(int fd, int command, long arg)
```

Manipula as propriedades do FIFO/ficheiro

Mecanismos de comunicação Unix: FIFO's

Semântica de sincronização na utilização de FIFO's

- Existem duas categorias de uso: "bloqueante" e "não-bloqueante"

  A semântica síncrona (com bloqueio) é a mais fácil de utilizar)

  Normalmente as funções de abertura, leitura e escrita bloqueiam caso a informação não possa ser lida/escrita imediatamente, ou , no caso da abertura, se não existir ainda nenhum processo com o mesmo FIFO aberto para a operação inversa (leitura - escrita)

- Depende da forma como FIFO é aberto
  - Os FIFO's são abertos ou para escrita, ou para leitura
    - O mesmo processo pode ler e escrever no mesmo FIFO se o abrir duas ou mais vezes
  - Na função **`open()`** pode-se especificar se se deseja a semântica bloqueante (situação por omissão) ou a não bloqueante (O_NONBLOCK)
  - Existem 4 combinações possíveis (e 4 comportamentos diferentes)

FIFO's – Semântica de bloqueio de `open`, `read` e `write`

1 – Abertura do FIFO para leitura

1-a) Comportamento síncrono (com bloqueio)

```
open(<filename>, O_RDONLY)
```

- A chamada bloqueia o processo até que algum processo abra o mesmo FIFO para escrita
- A chamada **`read()`** com o FIFO vazio bloqueia o processo

1-b) Comportamento assíncrono (sem bloqueio)

```
open(<filename>, O_RDONLY | O_NONBLOCK)
```

- A chamada retorna logo mesmo que nenhum processo tenha o FIFO aberto para escrita
- A chamada a **`read()`** com o FIFO vazio retorna 0 (bytes lidos)

FIFO's – Semântica de bloqueio de `open`, `read` e `write`

2 – Abertura do FIFO para escrita

2-a) Comportamento síncrono (com bloqueio)

`open(<filename>, O_WRONLY)`

- A chamada bloqueia o processo até que algum processo abra o mesmo FIFO para leitura
- A chamada **`write()`** com o FIFO cheio bloqueia até poder escrever todos os dados

2-b) Comportamento assíncrono (sem bloqueio)

`open(<filename>, O_WRONLY | O_NONBLOCK)`

- A chamada retorna imediatamente mesmo que nenhum processo tenha o FIFO aberto para leitura
- A chamada a **`write()`**, quando os dados não cabem:
  - Bytes a escrever ≤ PIPE_BUF → falha
  - Bytes a escrever > PIPE_BUF → escreve o que ainda couber e retorna

FIFO's – Exemplo de aplicação a um caso concreto

Exemplo de aplicação de FIFO's: Dicionário

- Construção de um sistema em arquitectura **cliente-servidor** que permita a um processo obter a tradução de algumas palavras

Servidor

- Mantém a informação que constitui o dicionário
- Não deve ser lançado mais do que uma vez em simultâneo
- A informação do dicionário não tem que ser repetida por vário processos que manipulam texto
- As sua únicas tarefa são: esperar perguntas, procurar a tradução e enviar respostas

Cliente

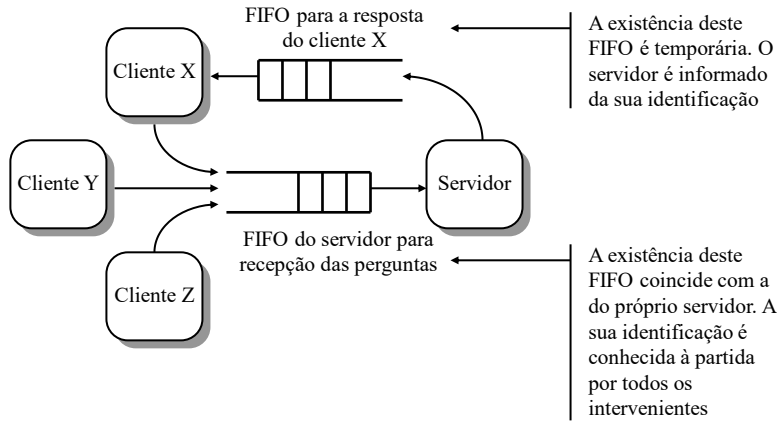
- Efectua perguntas ao servidor e obtém as respostas (tradução)
- Não tem que se preocupar como é que o dicionário funciona – apenas como se utiliza

Nota: Este exemplo também se encontra descrito num documento separado

Esse documento contém nova explicação acerca de *named pipes* e uma solução melhor e mais simples que a dada aqui. Recomenda-se a sua leitura

FIFO's – Exemplo de um dicionário cliente-servidor

Esquema da interacção entre os vários processos (servidor e clientes)



FIFO's – Exemplo de aplicação a um caso concreto

Situação a ter em atenção - 1

- Este exemplo inclui uma situação que se não for devidamente acautelada pode levar a que os vários processos fiquem à espera uns dos outros eternamente (situação comum a muitos problemas)
- Devido à semântica de bloqueio dos FIFOs e à ordem pela qual os vários opens e read/writes são efectuados (tal como visto nas aulas)
  - O cliente tem que enviar nome do seu pipe na pedido de forma a que o servidor saiba para onde deve enviar a resposta
  - Deve criar o pipe antes de efectuar o pedido pois o nome previsto pode já estar ocupado
  - Se abrir também esse o pipe para leitura fica bloqueado pois o servidor ainda não o abriu para escrita (nem sequer o conhece)

FIFO's – Exemplo de aplicação a um caso concreto

Resolução

- Só abrir o pipe da resposta para leitura depois de enviar o pedido
  - Ou seja, sendo possível, manipular a ordem das operações referidas acima, desde que possível no(s) programa(s) que se estão a construir. Nem sempre é possível mudar a ordem das operações.)
- Efectuar algumas operações sobre o pipe de forma não bloqueantes e posteriormente mudar a semântica do pipe para bloqueantes
  - Função **`fcntl`**

  Neste caso
  - Abrir o pipe como não bloqueante – o processo prossegue
  - Mudar logo de seguida para bloqueante

Neste exemplo ambas as estratégias funcionariam. Vai ser usada a segunda solução para exemplificar a função **`fcntl`** e de seguida a primeira solução é apresentada de forma simplificada (apenas as partes relevantes do algoritmo)

FIFO's – Exemplo de aplicação a um caso concreto

Situação a ter em atenção - 2

- Existido apenas um cliente (situação perfeitamente razoável) e esse cliente terminando, deixa de existir algum processo com o pipe do servidor aberto para escrita (situação que pode acontecer frequentemente)
  - Assim que isso acontece, as leituras do servidor no seu pipe (leituras de perguntas) deixam de bloquear (como se passasse a não bloqueante).
  - Os reads retornam logo e entra-se numa situação de espera activa

    (Lê – não tem nada – volta a ler – repetir)
    - Espera activa é proibida em todo o lado, em particular em SO
- Resolução
  - O servidor abre o seu próprio pipe para leitura, garantindo assim que há sempre "alguém" com o pipe aberto para leitura (técnica "keep-alive")
  - A ordem destas aberturas também deve ser ponderada para evitar situações como a descrita em "situação a ter em atenção – 1"

FIFO's – Exemplo de um dicionário cliente-servidor

Algoritmo simplificado do processo **cliente** *típico*

```
Abre o FIFO do servidor para escrita
Cria o FIFO para receber a resposta
Abre o FIFO das respostas para leitura
Repete durante uma certa condição
    | Obtém palavra a traduzir
    | Constrói pergunta = palavra + nome do FIFO para a resposta
    | Envia a pergunta (escreve no FIFO do servidor)
    | Fica à espera da resposta (efectua uma leitura no FIFO para a resposta)
Fecha o FIFO do servidor
Fecha o FIFO para as respostas
Remove o FIFO das respostas
```

→ "típico" porque os clientes podem fazer outras coisas além de obter traduções

FIFO's – Exemplo de um dicionário cliente-servidor

Parte do algoritmo relativo à abertura do pipe não-bloqueante-e-depois-passar-a-bloqueante

```
Abre o FIFO do servidor para escrita
Cria o FIFO para receber a resposta
Abre o FIFO das respostas para leitura
Repete durante uma certa condição
  | …..
```

Abre o FIFO das respostas para leitura como não-bloqueante
- a abertura não bloqueia apesar do servidor ainda não ter aberto este pipe para a operação inversa)
- a utilização de pipes não bloqueante é mais complicada por isso vai-se mudar para bloqueante agora que já foi aberto)

Muda FIFO para bloqueante com a função **`fcntl`**

FIFO's – Exemplo de um dicionário cliente-servidor

Algoritmo simplificado do processo **servidor**

```
Cria o FIFO do servidor para obter as perguntas
Repete durante uma certa condição
    Obtém a próxima pergunta (efectua uma leitura no FIFO do servidor)
    Obtém a tradução da palavra
    Obtém a identificação do FIFO para a resposta: vinha junto com a pergunta
    Abre o FIFO do cliente que enviou a pergunta = FIFO para a resposta
        Envia a pergunta (escreve no FIFO para a resposta )
    Fecha o FIFO para a  resposta
Fecha o FIFO do servidor
Remove o FIFO do servidor
```

→ Ao invés do(s) cliente(s), o servidor apenas efectua traduções (não se aplica a palavra "típico" como no caso do cliente – só há um servidor a correr).

FIFO's – Exemplo de um dicionário cliente-servidor

Padrão escolhido para a identificação dos vários FIFO's

- FIFO do servidor para enviar/receber as perguntas

  `/tmp/dict_fifo`

  – Este nome é previamente estabelecido de modo a que os clientes o possam abrir

- FIFO de cada cliente para receber as respostas

  `/tmp/resp_pid_fifo`

  – ***`pid`*** será o valor do PID do cliente

  → Cada cliente tem o seu próprio FIFO e não ocorrem conflitos

  – Não é necessário definir este nome à partida: o servidor será informado do nome juntamente com cada pergunta que recebe

FIFO's – Exemplo de um dicionário cliente-servidor

Ficheiro **"dict.h"**

```
/* ficheiro header necessário aos clientes e servidor */
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <limits.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <ctype.h>

/* ficheiro correspondente ao FIFO do servidor
#define SERVER_FIFO "/tmp/dict_fifo"

/* ficheiro correspondente ao FIFO do cliente n (n -> "%d")
#define CLIENT_FIFO "/tmp/resp_%d_fifo"
```

O código aqui apresentado foi testado. No entanto, pode ter havido erros na transcrição e formatação. Este aspecto deve ser tomado em consideração e qualquer dúvida comunicada ao docente.

⇦ Previamente estabelecido!

FIFO's – Exemplo de um dicionário cliente-servidor

Ficheiro **"dict.h"** (continuação)

```
/* tamanho máximo de cada palavra */
#define TAM_MAX 50

/* estrutura da mensagem correspondente ao um pedido */
typedef struct {
   pid_t pid_cliente;
   char  palavra[TAM_MAX];
} pergunta_t;

/* estrutura da mensagem correspondente a uma resposta */
typedef struct {
   char palavra[TAM_MAX];
} resposta_t;
```

FIFO's – Exemplo de um dicionário cliente-servidor

Código do cliente (1)

```
#include "dict.h"

int main() {
   int s_fifo_fd;   /* identificador do FIFO do servidor */
   int c_fifo_fd;   /* identificador do FIFO deste cliente */
   pergunta_t perg; /* mensagem do "tipo" pergunta */
   resposta_t resp; /* mensagem do "tipo" resposta */
   char buffer[80]; /* para a leitura da palavra a traduzir */
   char c_fifo_fname[25];  /* nome do FIFO deste cliente */
   long fflags;
   int read_res;
```


FIFO's – Exemplo de um dicionário cliente-servidor

Código do cliente (2)

Abre o FIFO do servidor para as perguntas

- Se não conseguir, não é possível comunicar com o servidor (não está a correr ?) e termina

```
s_fifo_fd = open(SERVER_FIFO, O_WRONLY); /* bloqueante */
if (s_fifo_fd == -1) {
   fprintf(stderr, "\nO servidor não está a correr\n");
   exit(EXIT_FAILURE);
}
```

O nome do ficheiro de FIFO do servidor é conhecido

FIFO's – Exemplo de um dicionário cliente-servidor

Código do cliente (3)

Cria um FIFO para receber a resposta do servidor

- O nome deste FIFO será enviado ao servidor juntamente com a pergunta
  - Já não é tão crítico como no caso do FIFO do servidor
  - No entanto, interessa obter um nome que tenha uma alta probabilidade de não estar já tomado

    Ideia: Concatenação de "**/tmp/resp_**" com o **PID** com "**_fifo**"

```
perg.pid_cliente = getpid();
sprintf(c_fifo_fname, CLIENT_FIFO, perg.pid_cliente);
if (mkfifo(c_fifo_fname, 0777) == -1) {
   fprintf(stderr, "\nErro no FIFO para a resposta (1)");
   close(s_fifo_fd); exit(EXIT_FAILURE);
}
```


FIFO's – Exemplo de um dicionário cliente-servidor

Código do cliente (4)

Abre já o FIFO para receber a resposta do servidor

- Caso contrário o servidor deitaria a resposta fora
- No entanto, não pode ser com semântica bloqueante, senão ficar-se-ia já bloqueado aqui
  - Abre-se como não bloqueante agora, e depois modifica-se para bloqueante (interessa ficar bloqueado à espera da resposta)

```
c_fifo_fd = open(c_fifo_fname, O_RDONLY | O_NONBLOCK);
if (c_fifo_fd == -1) {
   fprintf(stderr, "\nErro no FIFO para a resposta (2)\n");
   close(s_fifo_fd);
   unlink(c_fifo_fname);
   exit(EXIT_FAILURE);
}
```

FIFO's – Exemplo de um dicionário cliente-servidor

Código do cliente (5)

Muda-se a semântica do FIFO das repostas para bloqueante

- Simplifica as leituras
- A alternativa seria o algoritmo ter que efectuar a espera pela resposta de uma forma explicita analisando o resultado da leitura (+ complexo)

```
fflags = fcntl(c_fifo_fd, F_GETFL);
fflags ^= O_NONBLOCK; /* inverte a semântica de bloqueio */
fcntl(c_fifo_fd, F_SETFL, fflags); /* bloqueante = on */
```

`flags ^= O_NONBLOCK`

- Deixa os outros bits como estavam e inverte o da *flag* que controla a semântica de bloqueio através de uma operação de ou exclusivo

FIFO's – Exemplo de um dicionário cliente-servidor

Código do cliente (6)

Obtém a garantia de um `'\0'`

```
perg.palavra[TAM_MAX-1] = '\0';
```

Ciclo principal:

a) Obtém a pergunta
b) Envia a pergunta
c) (Espera e) obtém a resposta

```
while (1) {  /* "fim" termina */
   /* ---- a) OBTEM PERGUNTA ---- */
   printf("Palavra a traduzir ->");
   scanf("%s",buffer);
   if (!strcasecmp("fim",buffer)) break;
   strncpy(perg.palavra,buffer,TAM_MAX-1);
```

FIFO's – Exemplo de um dicionário cliente-servidor

Código do cliente (7)

```
      /* ---- b) ENVIA A PERGUNTA ---- */
      write(s_fifo_fd, & perg, sizeof(perg));
      /* ---- c) OBTEM A RESPOSTA ---- */
      read_res = read(c_fifo_fd, & resp, sizeof(resp));
      if (read_res == sizeof(resp))
         printf("Traducao -> %s\n", resp.palavra);
      else
         printf("Sem resposta ou resposta incompreensivel
                                              [%d]\n",read_res);
   }
```

Verificações efectuadas

- **Se o tamanho da resposta lida for diferente do tamanho da estrutura de uma resposta, então assume-se que não é uma resposta correcta**
- **Pode-se terminar o servidor escrevendo "fim" para o seu FIFO**

FIFO's – Exemplo de um dicionário cliente-servidor

Código do cliente (8)

Termina, libertando os recursos detidos

- Fecha o FIFO deste cliente
- Fecha o FIFO do servidor
  - A utilização do FIFO do servidor por parte de outros clientes não é afectada
- Remove o FIFO deste cliente

```
   close(c_fifo_fd);
   close(s_fifo_fd);
   unlink(c_fifo_fname);
} /* fim da função main */
```

- A remoção do FIFO é feita com recurso a uma função do sistema de ficheiros

  `unlink(<nome do ficheiro>)`

FIFO's – Exemplo de um dicionário cliente-servidor

Código do servidor (1)

```
#include "dict.h"
#include <signal.h>

#define NPALAVRAS 7       /* Número de palavras conhecidas */

char * dicionario[NPALAVRAS][2] = {  /*  O dicionário      */
   { "memory",    "memória" },       /*  é constituído     */
   { "computer",  "computador" },    /*  por uma matriz    */
   { "close",     "fechar" },        /*  bidimensional de  */
   { "open",      "abrir" },         /*  ponteiros para    */
   { "read",      "ler" },           /*  caractere.        */
   { "write",     "escrever" },      /*  [i][0] = palavra  */
   { "file",      "ficheiro" } };    /*  [i][1] = tradução */

int s_fifo_fd, c_fifo_fd, keep_alive_fd; /* desc. de fich. */
```


FIFO's – Exemplo de um dicionário cliente-servidor

Código do servidor (2)

```
int main() {
   pergunta_t perg;   /* mensagem do "tipo" pergunta */
   resposta_t resp;   /* mensagem do "tipo" resposta */

   int read_res,i;
   char c_fifo_fname[50];
   char * aux;

   signal(SIGINT,  sig_int);
```

- A chamada **`signal()`** vai servir para associar uma função ao sinal SIGINT (interrupção via teclado) para se poder terminar o programa de forma adequada
  A função associada ao sinal (**`sig_int(int)`**) é descrita mais adiante

FIFO's – Exemplo de um dicionário cliente-servidor

Código do servidor (3)

Cria o FIFO para recepção das perguntas

➔ Se o FIFO já existir termina

– Esse recurso já está ocupado e não vale a pena continuar com um FIFO com outra identificação: os clientes não saberiam para onde enviar as perguntas

```
mkfifo(SERVER_FIFO, 0777);
s_fifo_fd = open(SERVER_FIFO, O_RDONLY); /* bloqueante */
if (s_fifo_fd == -1) {
   fprintf(stderr, "\nErro ao criar/abrir o FIFO");
   exit(EXIT_FAILURE);
}
```


FIFO's – Exemplo de um dicionário cliente-servidor

Código do servidor (4)

Problema:

Quando o último cliente termina, o FIFO do servidor passa a estar na situação de não estar aberto para escrita por nenhum processo

➔ O servidor, que está bloqueado no `read()` à espera da próxima pergunta é desbloqueado com 0 bytes lidos

➔ `read()`'s subsequentes não bloqueiam

Solução

➔ Fecha-se o FIFO e volta-se a abrir novamente (para leitura), ou

➔ Abre-se já este FIFO para escrita, impedindo a situação descrita de ocorrer

- Esta solução é a mais simples
- A semântica de bloqueio aqui é indiferente porque o servidor nunca irá escrever neste FIFO

```
keep_alive_fd = open(SERVER_FIFO, O_WRONLY);
```

FIFO's – Exemplo de um dicionário cliente-servidor

Código do servidor (5)

Ciclo principal

a) obtém a próxima pergunta
b) obtém a tradução (se existir)
c) envia a resposta (o destinatário vinha identificado na pergunta)

```
while (1) {
   /* ---- a) OBTEM PERGUNTA ---- */
   read_res = read(s_fifo_fd, & perg, sizeof(perg));
   if (read_res < sizeof(perg)) {
      if (!strncasecmp("fim",(char *) & perg,3)) break;
      else {
         fprintf(stderr, "\nPergunta incompleta!");
         continue;
      }
   }
```


FIFO's – Exemplo de um dicionário cliente-servidor

Código do servidor (6)

```
   /* ---- b) PROCURA TRADUCAO ---- */

   aux = "DESCONHECIDO";
   for (i=0; i<NPALAVRAS; i++)
      if (!strcasecmp(perg.palavra,dicionario[i][0])) {
         aux = dicionario[i][1];
         break;
      }
   strcpy(resp.palavra,aux);

   /* ---- OBTEM FILENAME DO FIFO PARA A RESPOSTA ---- */

   sprintf(c_fifo_fname, CLIENT_FIFO, perg.pid_cliente);
```

FIFO's – Exemplo de um dicionário cliente-servidor

Código do servidor (7)

```
      /* ---- c) ENVIA RESPOSTA ---- */

      c_fifo_fd = open(c_fifo_fname, O_WRONLY | O_NONBLOCK);
      if (c_fifo_fd != -1) {
         write(c_fifo_fd, & resp, sizeof(resp));
         close(c_fifo_fd); /* FECHA LOGO O FIFO ! */
      }
      else
        fprintf(stderr, "\nNinguem quis a respsta [%s]",
                                              resp.palavra);
   } /* ciclo principal */
```

- É concebivel que o cliente tenha terminado e não queira a resposta. Nessa situação (*bug* ?) o FIFO para a resposta poderia já nem existir. O servidor verifica essa situação e apresenta uma mensagem nesse caso.

FIFO's – Exemplo de um dicionário cliente-servidor

Código do servidor (8)

Termina, libertando os recursos detidos

- Fecha o FIFO do servidor
  - Lembrar que esta aberto duas vezes: para leitura e para escrita
- Remove o FIFO deste cliente
  - Com recurso à função **`unlink()`**

```
   close(keep_alive_fd);
   close(s_fifo_fd);
   unlink(SERVER_FIFO);
} /* fim da função main */
```

FIFO's – Exemplo de um dicionário cliente-servidor

Código do servidor (9)

Tratamento do sinal **`SIGINT`**

- Termina o servidor de uma forma adequada libertando os recursos detidos

  Feito da mesma forma como se se tivesse recebido ordem para terminar ("fim" escrito directamente para o FIFO do servidor)

```
void sig_int(int i) {
   fprintf(stderr, "\nServidor de dicionario a terminar
                          (interrompido via teclado)\n\n");
   close(keep_alive_fd);
   close(s_fifo_fd);
   unlink(SERVER_FIFO);
   exit(EXIT_SUCCESS);     /* termina o processo */
}
```


FIFO's – Exemplo de um dicionário cliente-servidor - 2

Modificação para evitar o recurso à função **`fcntl`** - **Cliente**

Abre o FIFO do servidor para escrita
Cria o FIFO para receber a resposta
Abre o FIFO das respostas para leitura
Repete durante uma certa condição
    Obtém palavra a traduzir
    Constrói pergunta = palavra + nome do FIFO para a resposta
    Envia a pergunta (escreve no FIFO do servidor)
    Fica à espera da resposta (efectua uma leitura no FIFO para a resposta)
Fecha o FIFO do servidor
Fecha o FIFO para as respostas
Remove o FIFO das respostas

FIFO's – Exemplo de um dicionário cliente-servidor

Parte do algoritmo de abrir o FIFO das respostas (bloqueante logo de origem)

Após o envio da pergunta ao FIFO das respostas

- Eventualmente bloqueia temporariamente
- Não levanta problemas porque o servidor já tem a identificação desse FIFO e irá abri-lo para escrita e escrever nele

A abertura é efectuada dentro do ciclo principal

- Apenas é necessário testar se o FIFO já estava a aberto pois a sua abertura é agora efectuada dentro do ciclo principal
  (Ou então abrir e fechar mas essa opção é desnecessariamente pesada)

A única alteração necessária é no cliente. A modificação algorítmica é apresentada a seguir. A sua implementação (trivial) é um exercício (que se recomenda)

FIFO's – Exemplo de um dicionário cliente-servidor

Parte do algoritmo que abre o pipe

Não bloqueante, após o envio da pergunta, dentro do ciclo principal

```
Repete durante uma certa condição
    Obtém palavra a traduzir
    Constrói pergunta = palavra + nome do FIFO para a resposta
    Envia a pergunta (escreve no FIFO do servidor)
    Abre o FIFO das respostas para leitura
    Fica à espera da resposta (efectua uma leitura no FIFO para a resposta)
…..
```

```
Verifica se o FIFO já estava aberto (flag, etc.)
    Se não estava
        Abre o FIFO como bloqueante
        (Eventualmente bloquea temporáriamente)
        Assinala (flag? etc.) que o FIFO está aberto
```

FIFO's – Considerações

Recomendações para a resolução de problemas com ***named pipes***

- Planear cuidadosamente a interacção entre os processos
- Identificar o modelo de comunicação mais apropriado (cliente-servidor, caixa de correio, diálogo, etc.)
- Identificar quais os pipes que vão existir e atribuir papéis a cada processo
  - Quem cria/apaga cada pipe
  - Quem lê/escreve em cada pipe
- Responder à pergunta – como sabem os processos para onde enviar a informação?
- Planear cuidadosamente a ordem pela qual as operações de abertura e escrita/leitura são feitas para evitar esperas mútuas
- Definir um protocolo de comunicação entre os intervenientes (significado e estrutura da informação)
  - Qualquer coisa pode ser enviada pelo pipe, mas tanto quem envia como quem recebe têm que saber o que é que está a ser enviado.
  - Se forem enviadas coisas diferentes, a ordem pela qual são enviadas e recebidas é fundamental
- Prever mecanismos de recuperação de mensagens erradas (fora de ordem? incompletas?), clientes ou servidor que não respondem (timeouts?) e terminação ordeira de clientes.
- Não deixar pipes por apagar

Importante: Soluções confusas e pouco intuitivas normalmente estão erradas

DEIS/ISEC Sistemas Operativos – 2025/2026 João Durães

53