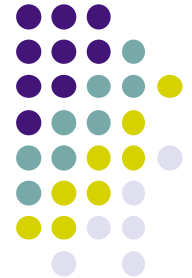
Comunicação entre Processos

Named Pipes (FIFO)

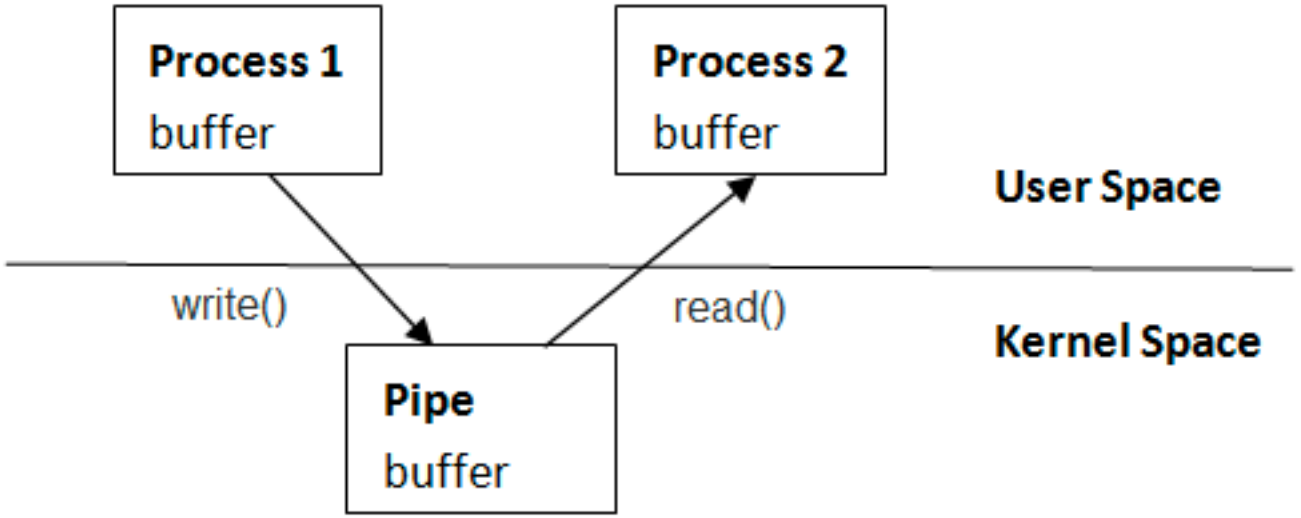
Named pipe (FIFO)

- FIFO permite que dois processos quaisquer se comuniquem
- É um tipo especial de arquivo visível no sistema de arquivos
- Na troca de dados através do FIFO o núcleo passa os dados internamente sem escrita no sistema de arquivos. O nome do FIFO serve apenas de referência.
- O controle de acesso é determinado pelos bits `rwx`
- O FIFO persiste dados além do processo que o criou. Para removê-lo execute `rm`

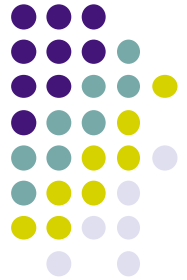
Named Pipe (FIFO)



- Troca de dados é através de um buffer alocado no núcleo.
- Dados não são estruturados.

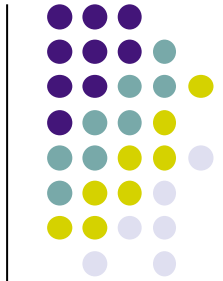


FIFO



- O FIFO precisa ser aberto nas duas pontas (para leitura e escrita) antes de qualquer transmissão de dados
- Normalmente, a chamada a open() bloqueia até que a outra ponta também abra o FIFO.
- FIFOs também podem ser abertos em modo não-bloqueante.
  - Nesse caso, o open para read-only será bem sucedida mesmo se nenhum processo tiver ainda aberto o FIFO para escrita.

Criando FIFOs na shell



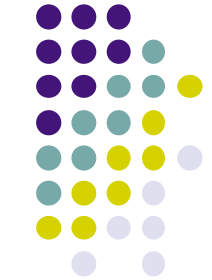
- FIFO's são criadas pelo comando mkfifo

$ mkfifo nome

- 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

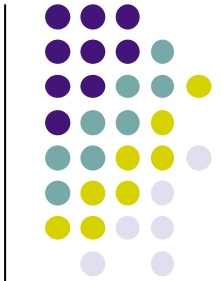


- Em um programa C, podemos utilizar a seguinte chamada do sistema (system call)
  - Definido em: <sys/stat.h>

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

  - Onde:
    - filename : nome da FIFO a ser criada
    - mode : bits com as permissões de criação
      - S_IRUSR S_IWUSR S_IRGRP S_IWGRP S_IROTH S_IWOTH ...
  - Retorna:
    - Em caso de sucesso, 0 (zero)
    - Em caso de erro, -1

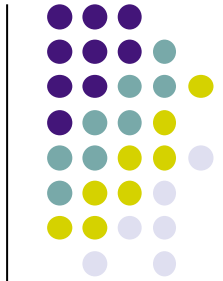
Criando uma FIFO



```c
#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$
```

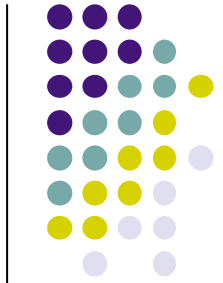
Utilizando a FIFO



```
int main (void)
{
   int fifo;
   if ((fifo = open("minhaFifo", "w")) < 0)
   {
                puts ("Erro ao abrir a FIFO para escrita"); return -1;
   }
   write(fifo, &c, sizeof(c);
   close (fifo);
   return 0;
}
```

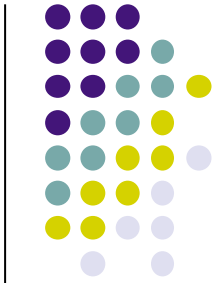
```
int main (void)
{
   int fifo;
   char ch;
   if ((fifo = open("minhaFifo", "r")) < 0)
   {
                puts ("Erro ao abrir a FIFO para escrita"); return -1;
   }
   while (read(fifo, &ch, sizeof(ch)) > 0) putchar (ch);
   close (fifo);
   return 0;
}
```

Problemas com FIFO



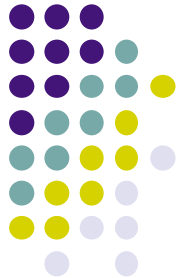
- Um programa não deve abrir uma FIFO para leitura e escrita.
- Se houver necessidade de comunicação bidirecional, devemos utilizar duas FIFOs, uma para cada sentido de transferência de dados.
- Outra forma (não recomendada) seria abrir a FIFO para leitura em um processo e escrita no outro, fechar e re-abrir no sentido contrário.

Trabalhando com FIFO non-blocking



- Ao abrir o canal, especificar 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

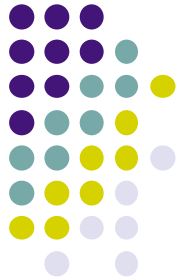


```
fifo$ ./leFifoNonBlocking
Abrindo FIFO
Começando a ler...
Fim da leitura
fifo$
```

```c
#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;
}
```

Verificando se FIFO já não está criado

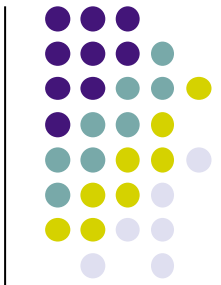
Abrindo FIFO para escrita sem bloqueio



```
#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$
```

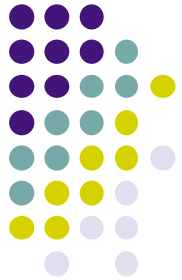
O mesmo programa, com bloqueio



```
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$
```

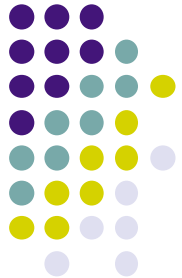
Perguntas?

Exercícios



1. Abra duas seções de terminais
   - Na primeira, execute um programa que fica em loop lendo de uma FIFO para depois escrever na saída padrão (tela)
   - Na segunda, execute um programa que fica lendo da entrada padrão (teclado) e depois escreve na mesma FIFO

Exercícios

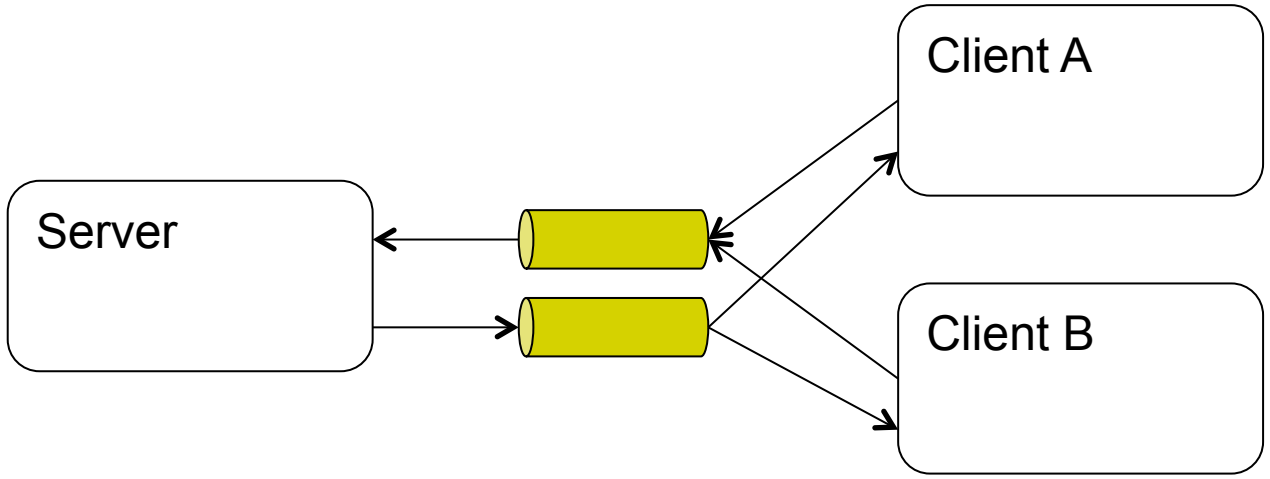


2. Escreva um programa que primeiro cria uma FIFO e em seguida cria dois processos filho que escrevem uma string na FIFO.

    O pai dá um waitpid em seguida lê as strings desse FIFO e exibe na tela

Exercícios

3. Escreva um servidor e um cliente usando duas FIFOs: uma para enviar strings para o servidor, e outra para transmitir as respostas de volta para os clientes. O servidor deverá rodar em background e transformar cada palavra recebida de minusculas para maiúsculas.



Obs: execute os seus clientes em terminais diferentes.