



**Universidade Tecnológica Federal do Paraná – UTFPR**  
**Coordenação de Ciência da Computação - COCIC**  
**Ciência da Computação**

# **BCC34G – Sistemas Operacionais**

---

**Prof. Rogério A. Gonçalves**  
*[rogerioag@utfpr.edu.br](mailto:rogerioag@utfpr.edu.br)*

# Aula 010

---

- Comunicação entre processos
- Pipe, fifo, sinais.
- Memória compartilhada.

# Comunicação entre Processos

- Processo precisar se comunicar
- Processos podem ser independentes ou cooperativos.
- O ***independente*** não compartilha dados com outros processos.
- O ***cooperativo*** pode afetar ou ser afetado por outros processos em execução no sistema.

# Comunicação Interprocessos

- Vários mecanismos de IPC originaram-se do tradicional IPC do UNIX.
  - Permitem que os processos troquem informações.
- Alguns são mais apropriados para determinadas aplicações.
  - Por exemplo, aqueles que se comunicam em uma rede ou trocam mensagens breves com outras aplicações locais.

# Sinais

- Um dos primeiros mecanismos de comunicação interprocessos disponíveis em sistemas UNIX.
- São interrupções de software que notificam ao processo que um evento ocorreu, são utilizados pelo núcleo.
- Permitem somente o envio de uma palavra de dados (código do sinal (1 a 64)) para outros processos.
- Não permitem que processos especifiquem dados para trocar com outros processos.

# Sinais

- Dependem do SO e das interrupções geradas por software suportadas pelo processador do sistema.
- Quando ocorre um sinal, o SO determina qual processo deve receber o sinal e como esse processo responderá ao sinal.

# Quando sinais são gerados

- Criados pelo núcleo em resposta a interrupções e exceções, os sinais são enviados a um processo ou *thread*.
- Em consequência da execução de uma instrução (como falha de segmentação)
- Em um outro processo (como quando um processo encerra outro) ou em um evento assíncrono.

# Sinais POSIX

<i>Sinal</i>	<i>Tipo</i>	<i>Ação default</i>	<i>Descrição</i>
1	SIGHUP	Abortar	Detectada interrupção de comunicação do terminal ou morte do processo controlador
2	SIGINT	Abortar	Interrupção de teclado
3	SIGQUIT	Descarregar	Sair do teclado
4	SIGILL	Descarregar	Instrução ilegal
5	SIGTRAP	Descarregar	Rastro/armadilha de ponto de ruptura
6	SIGABRT	Descarregar	Abortar sinal da função <b>abort</b>
7	SIGBUS	Descarregar	Erro de barramento
8	SIGFPE	Descarregar	Exceção de ponto flutuante
9	SIGKILL	Abortar	Sinal de matar
10	SIGUSR1	Abortar	Sinal 1 definido pelo usuário
11	SIGSEGV	Descarregar	Referência inválida à memória
12	SIGUSR2	Abortar	Sinal 2 definido pelo usuário
13	SIGPIPE	Abortar	Pipe rompido: escrever para pipe com nenhum leitor
14	SIGALRM	Abortar	Sinal de temporizador da função <b>alarm</b>
15	SIGTERM	Abortar	Sinal de encerramento
16	SIGSTKFLT	Abortar	Falha de pilha no co-processador
17	SIGCHLD	Ignorar	Filho parado ou encerrado
18	SIGCONT	Continuar	Continuar se estiver parado
19	SIGSTOP	Parar	Parar processo
20	SIGTSTP	Parar	Parar digitado no dispositivo de terminal



# Sinais POSIX

- Estão definidos 64 sinais.

```
rogerio@guarani:~$ kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
 6) SIGABRT     7) SIGBUS     8) SIGFPE      9) SIGKILL     10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM     15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN    22) SIGTTOU    23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS     34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
```

- Um *kill -9 pid* tem o mesmo efeito de um *kill -SIGKILL pid*

# Tratamento de Sinais

- Processos podem:
  - **Capturar:** especificando uma rotina que o SO chama quando entrega o sinal.
  - **Ignorar:** Neste caso depende da ação padrão do SO para tratar o sinal.
  - **Mascarar um sinal:** Quando um processo mascara um sinal de um tipo específico, o SO não transmite mais sinais daquele tipo para o processo até que ele desbloqueie a máscara do sinal.

# Tratamento de Sinais

- Um processo/*thread* pode tratar um sinal
  1. ***Capturando o sinal***
    - quando um processo capta um sinal, chama o tratador para responder ao sinal.
  2. ***Ignorando o sinal***
    - os processos podem ignorar todos, exceto os sinais SIGSTOP e SIGKILL.
  3. ***Executando a ação default***
    - Ação definida pelo núcleo para esse sinal.

# Tratamento

- **Ações default**
  - **Abortar:** terminar imediatamente.
  - **Descarga de memória:** copia o contexto de execução antes de sair para um arquivo do núcleo (*memory dump*).
  - **Ignorar.**
  - **Parar** (isto é, suspender).
  - **Continuar** (isto é, retomar).

# Bloqueio de Sinais

- Um processo ou *thread* pode bloquear um sinal.
- O sinal não é entregue até que o processo/*thread* pare de bloqueá-lo.
- Enquanto o tratador de sinal estiver em execução, os sinais desse tipo são bloqueados por *default*.
- Ainda é possível receber sinais de um tipo diferente (não bloqueados).
- Os sinais comuns não são enfileirados.
- Os sinais de tempo real podem ser enfileirados.

**Exemplo:** `sinais (so-aula-010-cod.zip)`

# Mecanismos de passagem de mensagem

- *Pipe:*
  - Permite a criação de filas de processos;  
**ps -aux | grep rogerio**
  - Saída de um processo é a entrada de outro;
  - Existe enquanto o processo existir;
- *Fifo (Named pipe):*
  - Extensão de pipe;
  - Continua existindo mesmo depois que o processo terminar;
  - Criado com chamadas de sistemas;

# Pipes

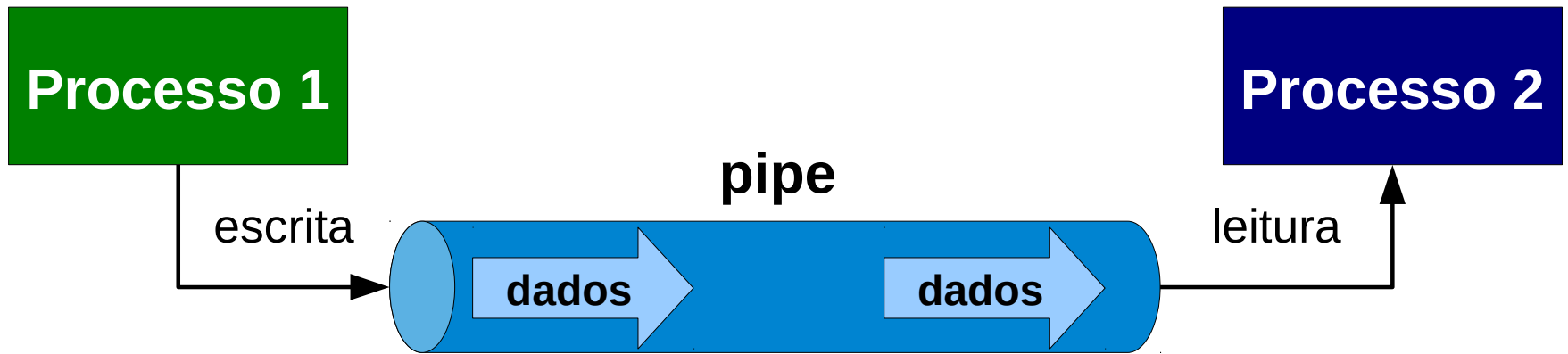
- O processo produtor escreve dados para o pipe. Depois disso, o processo consumidor lê dados do pipe na ordem “primeiro a entrar, primeiro a sair”.
- Quando um pipe é criado, um inode que aponta para o buffer do pipe (página de dados) é criado.

# Pipes

- O acesso ao pipe é controlado por descritores de arquivo.
  - Podem ser passados entre processos relacionados (por exemplo, pai e filho).
- Pipes nomeados (FIFOs).
  - Podem ser acessados por meio da árvore de diretório.
- Limitação: *buffer* de tamanho fixo.



# Pipe



```
terminal
$ ps -aux | grep rogerio
...
```

# Pipe

```
int pipe(int pipefd[2]);  
int pipe2(int pipefd[2], int flags);
```

pipe() cria um pipe, um canal de dados unidirecional que pode ser usado em IPC.

O *array* pipefd é usado para retornar dois descritores de arquivo:

- pipefd[0] referencia o lado de leitura
- pipefd[1] o lado de escrita.

Os dados escritos são colocados em um *buffer* pelo kernel até ser lido pelo lado de leitura.

Se flags é 0, então pipe2() é o mesmo que pipe().

Mais detalhes: ***man pipe***

## Exemplo Pipe

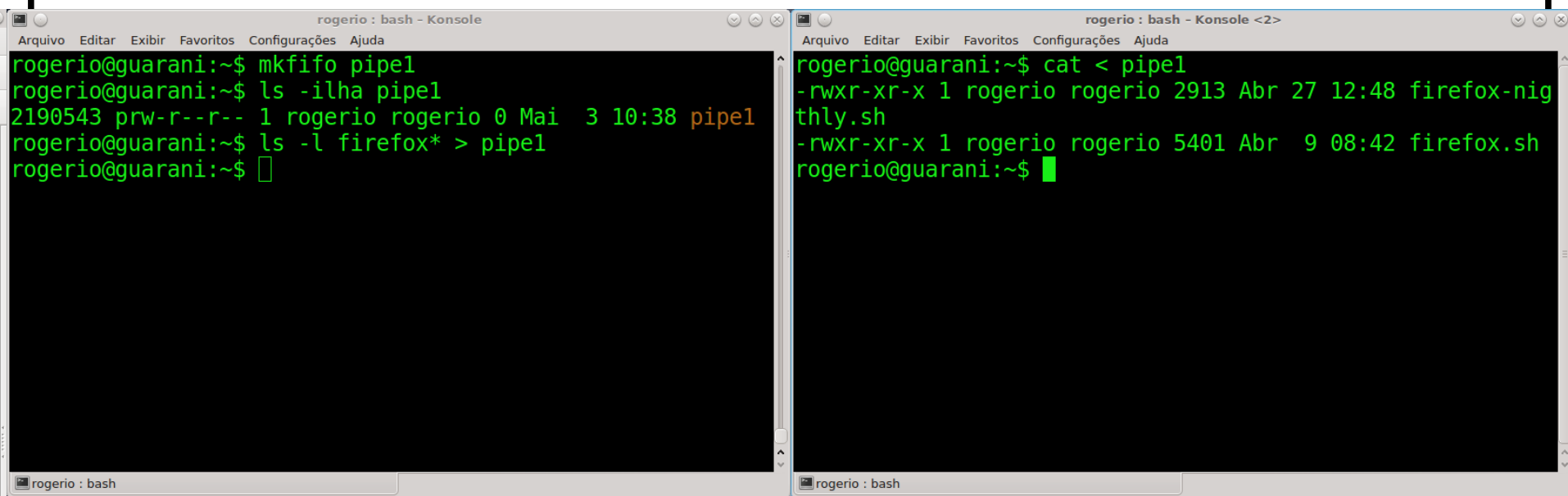
```
int main(void) {
    pid_t pid;
    int mypipe[2];

    /* Criar o pipe. */
    if (pipe(mypipe)) {
        fprintf(stderr, "Falha ao criar o Pipe.\n");
        return EXIT_FAILURE;
    }
    /* Criar o processo filho. */
    pid = fork();
    if (pid == (pid_t) 0) {
        /* No processo filho. */
        close(mypipe[1]);
        read_from_pipe(mypipe[0]);
        return EXIT_SUCCESS;
    } else if (pid < (pid_t) 0) {
        /* pid negativo, falha no fork. */
        fprintf(stderr, "Falha ao executar o Fork.\n");
        return EXIT_FAILURE;
    } else {
        /* Processo pai. */
        close(mypipe[0]);
        write_to_pipe(mypipe[1]);
        return EXIT_SUCCESS;
    }
}
```

**Exemplo: pipe (so-aula-010-cod.zip)**

# Named Pipes (FIFO)

FIFOs são canais nomeados (*named pipes*).



The image shows two terminal windows side-by-side. The left window shows the creation of a named pipe named 'pipe1' using the 'mkfifo' command. The right window shows the output of the 'cat' command reading from 'pipe1', displaying the contents of 'firefox-nigthly.sh' and 'firefox.sh'.

```
rogerio@guarani:~$ mkfifo pipe1
rogerio@guarani:~$ ls -ilha pipe1
2190543 prw-r--r-- 1 rogerio rogerio 0 Mai  3 10:38 pipe1
rogerio@guarani:~$ ls -l firefox* > pipe1
rogerio@guarani:~$
```

```
rogerio@guarani:~$ cat < pipe1
-rwxr-xr-x 1 rogerio rogerio 2913 Abr 27 12:48 firefox-nigthly.sh
-rwxr-xr-x 1 rogerio rogerio 5401 Abr  9 08:42 firefox.sh
rogerio@guarani:~$
```

É possível criar pipes nomeados usando o comando `mkfifo`.

# fifo

A mesma chamada de sistema está disponível em linguagem de programação:

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

FIFOs são canais nomeados (*named pipes*).

O que nos diz o manual?

Mais detalhes: ***man fifo e man mkfifo***

**Tarefa:** Faça uma pesquisa sobre fifo e implemente um exemplo de troca de dados entre dois processos.

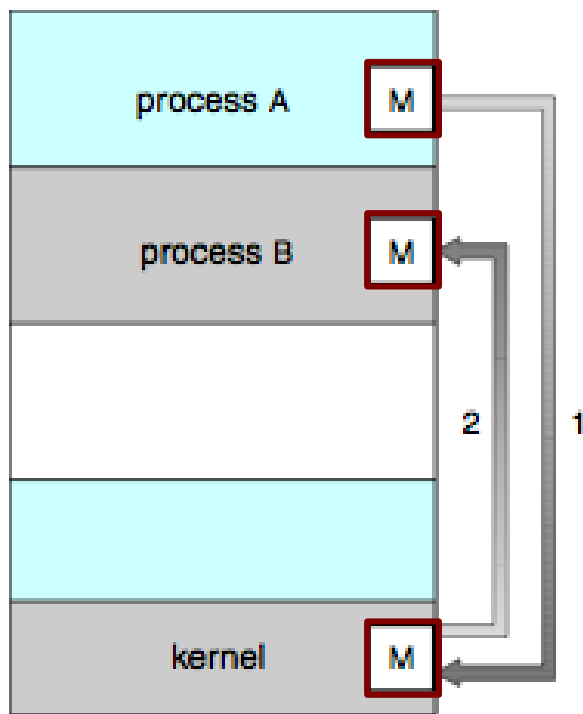
# Ambiente de Cooperação entre Processos

- Motivos:
  - Compartilhamento de informações
  - Agilidade na computação
  - Modularidade
  - Conveniência
- Necessidade de mecanismo para prover a comunicação entre processos
  - (*Interprocess Communication – IPC*)
- Dois modelos:
  - ***Memória compartilhada***
  - ***Troca de mensagem***

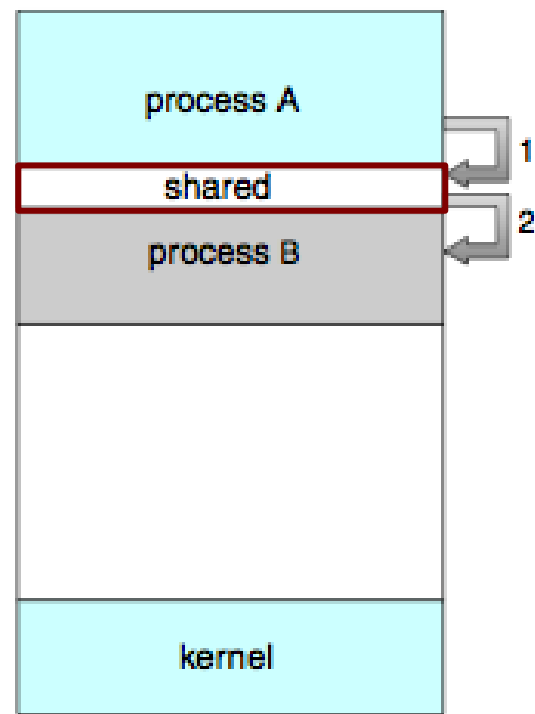
# Modelos de Comunicação entre Processos

- **Memória Compartilhada:** uma região de memória é compartilhada entre os processos.
  - A troca de informações ocorre por leitura/escrita de dados nessa região.
  - Mais rápida, chamadas de sistemas só criam a região.
- **Troca de Mensagem:** Ocorre por meio de troca de mensagens entre os processos.
  - Útil para trocar quantidades menores de dados.
  - Mais fácil de implementar.
  - Normalmente implementado com uso de chamadas de sistema.

# Comunicação entre Processos



**Passagem de mensagem**



**Memória compartilhada**



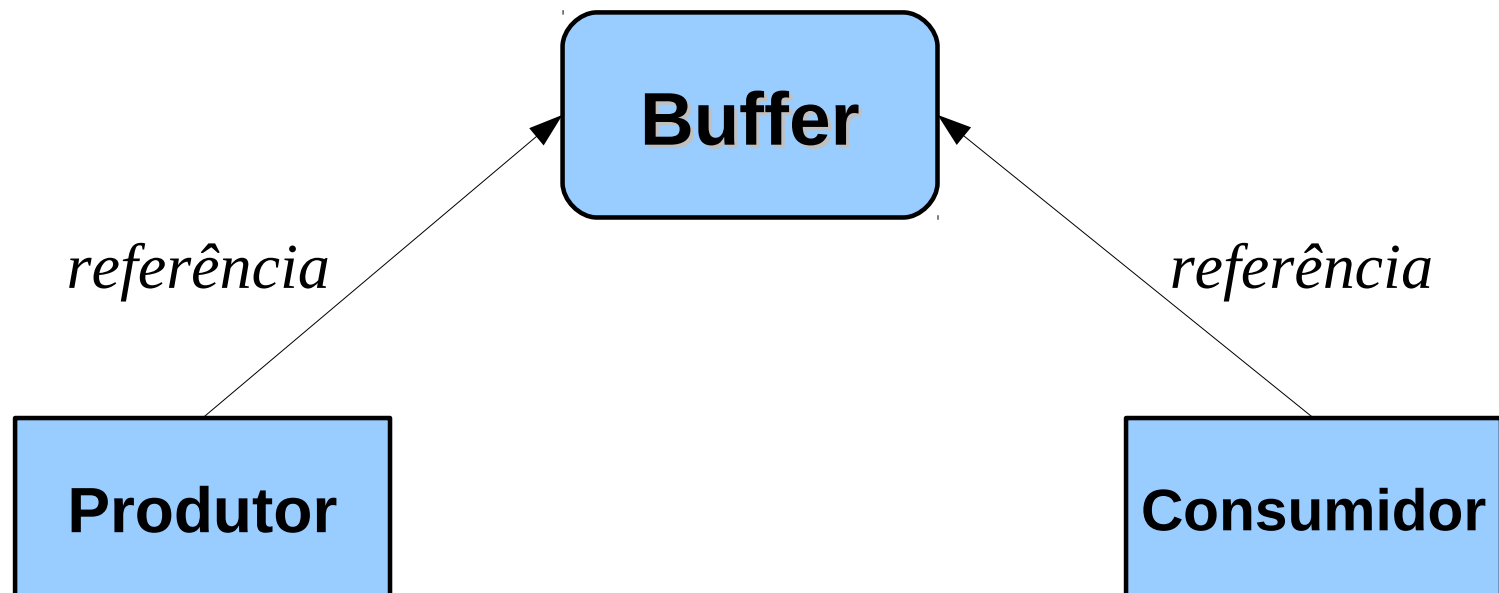
# Memória Compartilhada

- Dois ou mais processos utilizam a região de memória compartilhada, conectando-a no seu espaço de endereçamento.
- Deve se ter a garantia de que os dois processos não estejam gravando dados no mesmo local simultaneamente.
- ***Exemplo:*** Problema Produtor-Consumidor

# Problema Produtor-Consumidor

- Paradigma para processos em cooperação
- **Processo *produtor*** produz informações que são consumidas por um **processo *consumidor***
  - ***Buffer ilimitado*** não impõe limite prático sobre o tamanho do *buffer*
  - ***Buffer limitado*** assume que existe um tamanho de *buffer* fixo

# Simulando Memória Compartilhada



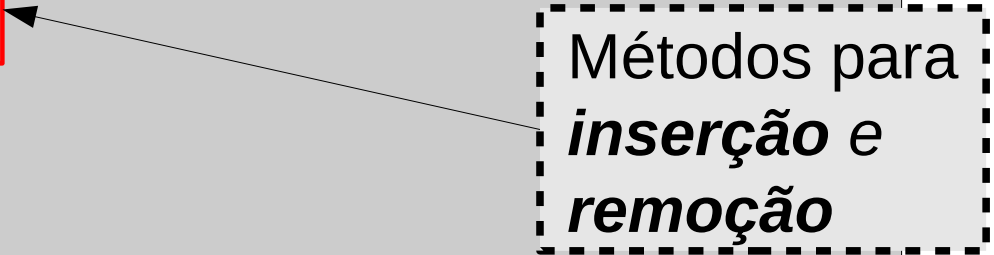
# Simulando Memória Compartilhada

```
#ifndef BUFFER_H_
#define BUFFER_H_

class Buffer {
public:
    int BUFFER_SIZE = 5;
    int count; // Número de itens no buffer.
    int in;    // Posição de inserção, próxima posição livre.
    int out;   // Próxima posição cheia.
    int *buffer;

    Buffer();
    virtual ~Buffer();
    void insert(int item);
    int remove();
    void print();
};

#endif /* BUFFER_H_ */
```



Métodos para ***inserção e remoção***

# Buffer vinculado – método insert()

```
void Buffer::insert(int item) {  
    while(count == BUFFER_SIZE) // Cheio?  
        ; // faça nada.  
  
    // add um item no buffer.  
    count++;  
    buffer[in] = item;  
    // Incremento Circular.  
    in = (in + 1) % BUFFER_SIZE;  
}
```

***Cheio?***

***Incremento Circular***

# Buffer vinculado – método remove()

```
int Buffer::remove() {  
    int item;  
    while(count == 0) // Vazio?  
        ; // faça nada.  
    // remove o item do buffer.  
    count--;  
    item = buffer[out];  
    out = (out + 1) % BUFFER_SIZE;  
  
    return item;  
}
```

*Vazio?*

**Exemplo:** buffer (so-aula-010-cod.zip)

# Memória Compartilhada

- Vantagens
  - Melhora o desempenho de processos que acessam frequentemente dados compartilhados.
  - Os processos podem compartilhar a mesma quantidade de dados que podem endereçar.
- Interface padronizada
  - Memória compartilhada System V
  - Memória compartilhada POSIX
    - Não permite que os processos mudem privilégios de um segmento de memória compartilhada.

# Memória Compartilhada

- Chamadas ao sistema de memória compartilhada do System V.

*Chamada ao sistema de memória  
compartilhada do System V*

shmget

shmat

shmctl

shmdt

*Propósito*

Aloca um segmento de memória compartilhada.

Anexa um segmento de memória compartilhada a um processo.

Muda as propriedades do segmento de memória compartilhada (por exemplo, permissões).

Separa (elimina) um segmento de memória compartilhada de um processo.

**Exemplo: shm (so-aula-010-cod.zip)**



# Implementação de Memória Compartilhada

- Trata a região da memória compartilhada como um arquivo.
- As molduras de página de memória compartilhada são liberadas quando o arquivo é apagado.
- O tmpfs (sistema de arquivo temporário) armazena esses arquivos.
  - As páginas do tmpfs podem ser trocadas.
  - É possível definir as permissões.
  - O sistema de arquivo não exige formatação.

# Troca de Mensagens

- O SO fornece mecanismos para permitir que os processos cooperativos se comuniquem entre si por meio de troca de mensagens.
- Comunicação e sincronização de ações sem compartilhar o mesmo espaço de endereços e é útil em um ambiente distribuído.

# Troca de Mensagens

- Sistema de mensagem – processos se comunicam entre si sem lançar mão de variáveis compartilhadas
- Facilidade de passagem de mensagem oferece **duas operações**:
  - **send(mensagem)** – tamanho da mensagem fixo ou variável
  - **receive(mensagem)**
- Se  $P$  e  $Q$  quiserem se comunicar, eles precisam:
  - estabelecer um *link de comunicação* entre eles
  - trocar mensagens por meio de **send/receive**
- Implementação do *link* de comunicação
  - físico (por exemplo, memória compartilhada, barramento de hardware)
  - lógico (por exemplo, propriedades lógicas)

# Questões de implementação

- Como os links (enlace) são estabelecidos?
- Um link pode estar associado a mais de dois processos?
- Quantos links pode haver entre cada par de processos em comunicação?
- Qual é a capacidade de um link?
- O tamanho de uma mensagem que o link pode acomodar é fixo ou variável?
- Um link é unidirecional ou bidirecional?

# Comunicação direta

- Processos devem nomear um ao outro explicitamente:
  - **send** ( $P$ , *mensagem*) – envia uma mensagem ao processo  $P$
  - **receive**( $Q$ , *mensagem*) – recebe uma mensagem do processo  $Q$
- Propriedades do link de comunicação
  - Links são estabelecidos automaticamente
  - Um link é associado a exatamente um par de processos em comunicação
  - Entre cada par existe exatamente um link
  - O link pode ser unidirecional, mas normalmente é bidirecional

# Comunicação indireta

- As mensagens são direcionadas e recebidas de caixas de correio (também conhecidas como portas)
  - Cada caixa de correio tem uma id exclusiva
  - Os processos só podem se comunicar se compartilharem uma caixa de correio
- Propriedades do link de comunicação
  - Link estabelecido somente se os processos compartilharem uma caixa de correio comum
  - Um link pode estar associado a muitos processos
  - Cada par de processos pode compartilhar vários links de comunicação
  - O link pode ser unidirecional ou bidirecional

# Comunicação indireta

- Operações
  - cria uma nova caixa de correio
  - envia e recebe mensagens por meio da caixa de correio
  - destrói uma caixa de correio
- Primitivos são definidos como:
  - send**(*A, mensagem*) – envia uma mensagem à caixa de correio A
  - receive**(*A, mensagem*) – recebe uma mensagem da caixa de correio A

# Comunicação indireta

- Compartilhamento de caixa de correio
  - $P_1$ ,  $P_2$  e  $P_3$  compartilham caixa de correio A
  - $P_1$  envia;  $P_2$  e  $P_3$  recebem
  - Quem recebe a mensagem?
- Soluções
  - Permite que um link seja associado a no máximo dois processos
  - Permite que somente um processo de cada vez execute uma operação de recepção
  - Permite que o sistema selecione arbitrariamente o receptor.
  - Permite que o sistema selecione arbitrariamente o receptor. Emissor é notificado de quem foi o receptor.



# Sincronismo no envio/recebimento

- ❑ A passagem de mensagens pode ser com bloqueio ou sem bloqueio
- ❑ **Bloqueio** é considerado **síncrono: (blocking)**
  - **Envio com bloqueio** deixa o emissor bloqueado até que a mensagem é recebida
  - **Recepção com bloqueio** deixa o receptor bloqueado até que a uma mensagem esteja disponível
- ❑ **Não bloqueio** é considerado assíncrono: **(nonblocking)**
  - **Envio sem bloqueio** faz com que o emissor envie a mensagem e continue
  - **Recepção sem bloqueio** faz com que o receptor receba uma mensagem válida ou nulo

# Mecanismos de passagem de mensagem

- *Fila de Mensagens: mqueue*
  - É criada uma estrutura, uma fila de mensagens.
- Memória Compartilhada:
  - É criada uma região de memória que é acoplada ao espaço de dados dos processos.
  - Os processos podem trocar informações usando essa região.

# Fila de Mensagens

- Permitem que os processos transmitam informações que são compostas por um tipo de mensagem e por uma área de dados de tamanho variável.
- Armazenadas em filas de mensagens, permanecem até que um processo esteja preparado para recebê-las.
- Processos relacionados podem procurar um identificador de fila de mensagens em um arranjo global de descritores de fila de mensagens.

# Fila de Mensagens

- O descritor de fila de mensagens contém:
- fila de mensagens pendentes;
- fila de processos em espera de mensagens;
- fila de processos em espera para enviar mensagens;
- dados que descrevem o tamanho e o conteúdo da fila de mensagens.

**Exemplo:** `mqueue (so-aula-010-cod.zip)`

# Próxima Aula

- ***Prática sobre Comunicação entre Processos***