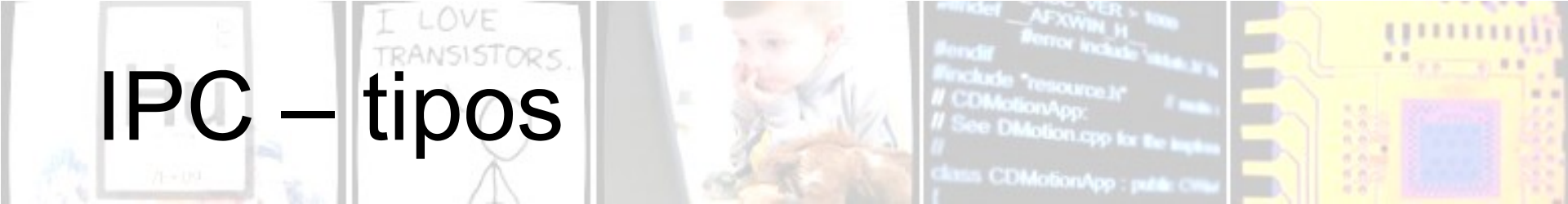




Sistemas de Tempo Real

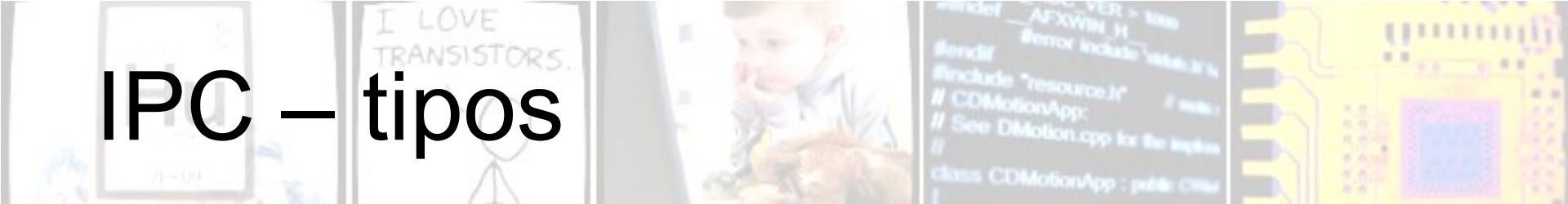
Multitarefa – técnicas de programação
(continuação)



IPC – tipos

» A comunicação de dados pode ser:

- Unidirecional: um primeiro processo envia mensagem para um segundo processo;
- Bidirecional: Quando acontece a possibilidade de envio de mensagens de um primeiro processo para um segundo e vice e versa;



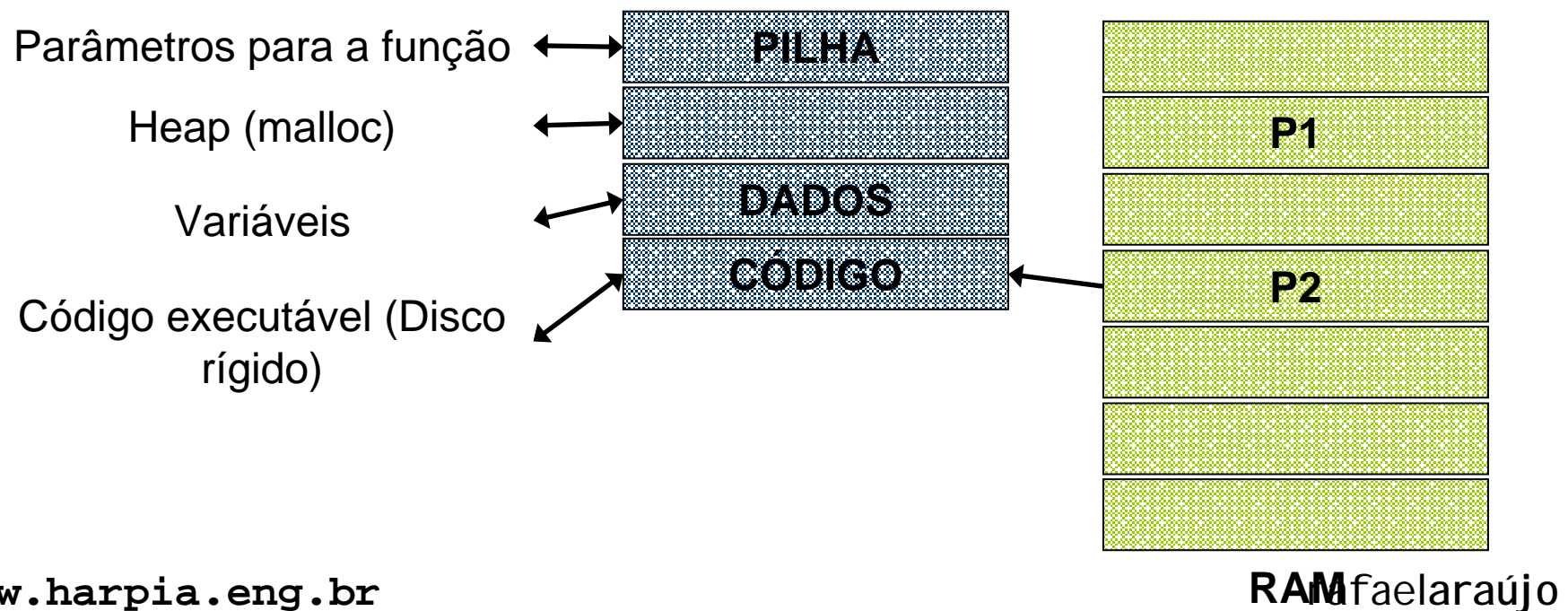
IPC – tipos

» Pode-se definir dois tipos de comunicação:

- Síncrona: o processo que inicia a comunicação fica bloqueado esperando por uma resposta do processo que recebe a mensagem;
- Assíncrona: o processo que inicia a comunicação (enviando uma mensagem) e continua o seu processo normalmente;

Memória compartilhada

» Como fazer com que o conteúdo de variáveis declaradas em P1 possam ser utilizadas por P2?





Memória compartilhada

- » S.O. protege a área de memória não permitindo acesso por outro processo.
 - Deve-se criar uma área de memória compartilhada para que os processos possam acessar.

Memória compartilhada – linguagem C

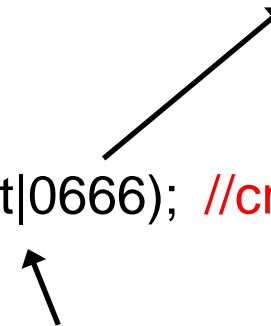
- » Um processo pode criar um segmento de memória compartilhada e suas estruturas de controle através da função `shmget()`.
- » Durante essa criação, os processos devem definir:
 - as permissões de acesso ao segmento de memória compartilhada;
 - o tamanho de bytes do segmento e;
 - a possibilidade de especificar que a forma de acesso de um processo ao segmento será apenas em modo leitura.
- » Para poder ler e escrever nessa zona de memória, é necessário estar de posse do identificador (ID) de memória comum, chamado `shmid`. Este identificador é fornecido pelo sistema (durante a chamada da função `shmget()`) para todo processo que fornece a chave associada ao segmento.
- » Após a criação de um segmento de memória compartilhada, duas operações poderão ser executadas por um processo:
 - acoplamento (*attachment*) ao segmento de memória compartilhada, através da função `shmat()`;
 - desacoplamento da memória compartilhada, utilizando a função `shmdt()`.
- » O acoplamento à memória compartilhada permite ao processo de se associar ao segmento de memória: ele recupera, executando `shmat()`, um ponteiro apontando para o início da zona de memória que ele pode utilizar, assim como todos os outros ponteiros para leitura e escrita no segmento.
- » O desacoplamento da memória compartilhada permite ao processo desassociar-se de um segmento quando ele não desejar mais utilizá-lo. Após esta operação, o processo perde a possibilidade de ler ou escrever neste segmento de memória compartilhada.

Memória compartilhada

» Processo 1 (P1):

```
#include <sys/ipc.h>
#include <sys/shm.h>
#define chave (key_t) 789
main(){
    int id,res,*valor;
    id = shmget(chave,1000*sizeof(int),ipc_creat|0666); //criar área de
    memória compartilhada (M.C.)
    if(id == -1){
        //não conseguiu criar M.C.}
    valor = shmat(id, 0, 0); //igual a ponteiro alocado por malloc. 0 e 0
    indicam início da M.C.
}
```

adm	grupo	outros
6	6	6
110	110	110



OU bit a bit

Memória compartilhada

» Processo 2 (P2):

```
define chave (key_t) 789
```

```
main(){
```

```
    int id,res,*valor;
```

```
    id = shmget(chave,1000*sizeof(int),0); //acessar a área  
    de memória compartilhada (M.C.)
```

```
    valor = shmat(id, 0, 0);
```

```
    res = *valor;
```

```
    ...
```

```
}
```

Para solicitar o endereço da área de memória compartilhada.





Problema

- » Se P2 for mais rápido que P1 vai ler endereços que ainda não foi escrita por P1.
- » Esse controle deve ser feito pelo programador:
 - Pode-se efetuar uma struct para marcar o que pode ser lido ou não.



Memória compartilhada

- » Mecanismo rápido porque após a criação da M.C. o acesso é direto (ponteiros):
 - Todo o controle de acesso é realizado pelo programador.
- » O compartilhamento de uma região de memória entre dois ou mais processos (executando programas) corresponde a maneira mais rápida para efetuarem uma troca de dados.
- » A zona de memória compartilhada (denominada segmento de memória compartilhada) é utilizada por cada um dos processos como se ela fosse um espaço de endereçamento que pertencesse a cada um dos programas.
- » O compartilhamento de memória permite aos processos trabalhar sob um espaço de endereçamento comum em memória virtual.
- » Em consequência, este mecanismo é dependente da forma de gerenciamento da memória; isto significa que as funcionalidades deste tipo de comunicação interprocessos são fortemente ligadas ao tipo de arquitetura (máquina) sobre a qual a implementação é realizada.



Memória compartilhada

» Soluções:

- Sleep em P2 para atrasar a leitura;
- P1 escreve os dados como struct:

```
struct qualquer{  
    int permissao;  
    int dados;  
}
```

A collage of images related to electronics and programming. On the left, a whiteboard has the text 'I LOVE TRANSISTORS.' and a simple stick figure drawing. In the center, a young child is looking at a small electronic component. On the right, there is a screenshot of C++ code and a close-up of a yellow printed circuit board (PCB) with a blue integrated circuit (IC) mounted on it.

Memória compartilhada

» P1:

```
valor → pode = 1;  
valor → dados = le_sensor();  
valor++;
```

» P2:

```
while(1){  
    if(valor → pode==1){  
        //processa valor → dados  
        valor → pode =0;  
    }  
    valor++;  
}
```



Filas de mensagens

- » Uma fila de mensagens é uma estrutura usada pelos processos para comunicação de mensagens de tamanho fixo.
- » O acesso à fila é “bloqueante”, ou seja, um processo que tenta colocar uma mensagem em uma fila cheia terá de esperar até que surjam vagas;
- » O processo que deseja receber mensagens de uma fila vazia terá de esperar até que uma mensagem seja enviada para a fila.
- » Definição:
 - *Buffers* limitados acessados por processos que se comportam como produtores e consumidores de mensagens. Sendo uma região crítica, deve ser protegida de eventuais condições de disputa. Utiliza-se semáforos em sua implementação.



Filas de mensagens

- » A comunicação interprocessos por mensagens é realizada pela troca de dados, armazenados no sistema, sobre a forma de arquivos.
- » Cada processo pode emitir ou receber mensagens durante uma comunicação.
- » Existe um limite imposto para o tamanho dos blocos a serem inseridos na fila, bem como o tamanho máximo total de todos os blocos em todas as filas no sistema.
- » Em Linux, por exemplo, dois valores são definidos para esses limites: MSGMAX (4096) e MSGMNB (16834), os quais definem, respectivamente, o tamanho máximo em bytes de uma mensagem individual e o tamanho máximo da fila.



Filas de mensagens – princípio

- » Da mesma forma que a M.C. uma fila de mensagem é associada a uma chave de acesso única – uma representação numérica no sistema.
- » Esta chave é utilizada para definir e obter um identificador da fila de mensagens.
- » Procedimentos:
 - Um processo que deseja enviar uma mensagem deve inicialmente obter o identificador da fila msgid, utilizando para isso a função msgget.
 - Então utiliza-se a função msgsnd() para armazenar sua mensagem (a qual está associada a um tipo de dados), dentro de um arquivo.
 - De maneira similar, se um processo deseja ler uma mensagem, ele deve primeiramente buscar o identificador da fila (através da função msgget()), para depois ler a mensagem através da função msgrcv().



Linguagem C

» Função:

- `msgget();`

» Biblioteca:

- `# include <sys/types.h>`
- `# include <sys/ipc.h>`
- `# include <sys/msg.h>`

» Sintaxe:

- `int msgget (key_t key, int msgflg);`

Linguagem C – msgget()

- » Retorna o identificador *msqid* da fila, ou -1 em caso de erro.
- » É utilizada para criar uma nova fila de mensagens, ou para obter o identificador da fila *msqid* de uma fila de mensagens existente no sistema.
- » Esta função recebe dois parâmetros: *key* é a chave indicando uma constante numérica representando a fila de mensagens; *msgflg* é um conjunto de flags especificando as permissões de acesso sobre a fila. Sintaxe:
 - `int msgget (key_t key, int msgflg);`



Linguagem C

» Função:

- `msgsnd()`;

» Biblioteca:

- `#include <sys/types.h>`
- `#include <sys/ipc.h>`
- `#include <sys/msg.h>`

» Sintaxe:

- `int msgsnd (int msqid, struct msgbuf *msgp, int msgsz, int msgflg);`

The background features a collage of images: a whiteboard with the text 'I LOVE TRANSISTORS.', a person working with electronic components, a computer screen displaying code, and a yellow printed circuit board (PCB).

Linguagem C – msgsnd()

- » Retorna 0 (zero) se a mensagem for colocada na fila e -1 em caso de erro.
- » Permite a inserção de uma mensagem na fila.
- » A estrutura da mensagem é limitada de duas maneiras: primeiramente, ela deve ser menor que o limite estabelecido pelo sistema e deve respeitar o tipo de dado estabelecido pela função que receberá a mensagem.



Linguagem C

» Função:

- `msgrcv();`

» Biblioteca:

- `# include <sys/types.h>`
- `# include <sys/ipc.h>`
- `# include <sys/msg.h>`

» Sintaxe:

- `int msgrcv(int msqid, struct msgbuf *msgp, int msgsz, long msgtyp, int msgflg);`

The background features a collage of four images: a whiteboard with the text 'I LOVE TRANSISTORS.', a young child, a snippet of C++ code including 'class CDMotorApp', and a yellow printed circuit board (PCB).

Linguagem C – msgrcv()

- » Retorna o número de bytes da mensagem extraída da fila, ou -1 em caso de erro.
- » Retira uma mensagem da fila.
- » A função vai armazenar a mensagem lida numa estrutura apontada por msgp.

Exemplo – criando fila

```
# include <stdio.h>
# include <ctype.h>
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/msg.h>
# include <errno.h>

# define CXMSG (key_t)123

struct msg
{
    long tipo;
    char conteudo [1024];
};

main ()
{
    int idcaixa, status;
    struct msg mens1, mens2;
    struct msqid_ds info_fila;

    idcaixa = msgget(CXMSG, IPC_CREAT|0666);
```

```
    if (idcaixa == -1)
    {
        printf ("Nao conseguiu criar fila\n");
        perror("");
        return;
    }
    printf ("FILHA CRIADA COM SUCESSO idfila = %d\n", idcaixa);
    mens1.tipo = 1;
    strcpy (mens1.conteudo, "oooooooooooooooooooo");
    status = msgsnd (idcaixa, &mens1, strlen (mens1.conteudo),
IPC_NOWAIT);
    if (status == -1)
    {
        printf ("Nao conseguiu escrever na fila\n");
        perror ("erro ");
        return;
    }
    printf ("ESCREVEU NA FILA \n");
}
```

Exemplo – lendo a fila

```
# include <stdio.h>
# include <ctype.h>
# include <sys/types.h>
# include <sys/ipc.h>
# include <sys/msg.h>
# include <errno.h>

# define CXMSG (key_t) 123

struct msg
{
    long tipo;
    char conteudo [1024];
};

main ()
{
    int idcaixa, status;
    struct msg mens1, mens2;
```

```
idcaixa = msgget(CXMSG, 0);
if (idcaixa == -1)
{
    perror("Erro no msgget");
    return;
}
printf ("Identificador da fila obtido - idfila =
%d\n", idcaixa);
status = msgrcv (idcaixa, &mens1, strlen
(mens1.conteudo), 1, IPC_NOWAIT);
if (status == -1)
{
    perror ("erro no msgsnd ");
    return;
}
}
```



M.C. x Filas

» Memória compartilhada

- só pode direcionar se criar *struct* e fizer o controle manual na recepção;
- Gerenciamento é feito pelo programa: mais rápido porque a escrita é direto na memória, porém mais sujeito a inconsistências.

» Filas:

- permitem direcionar os dados para o processo de destino (campo tipo da mensagem);
- São gerenciadas pelo SO: mais lento porque precisa usar via função, porém mais imune a inconsistência.

» Se a quantidade de dados for pequena, usar Fila; se for grande, usar M.C..