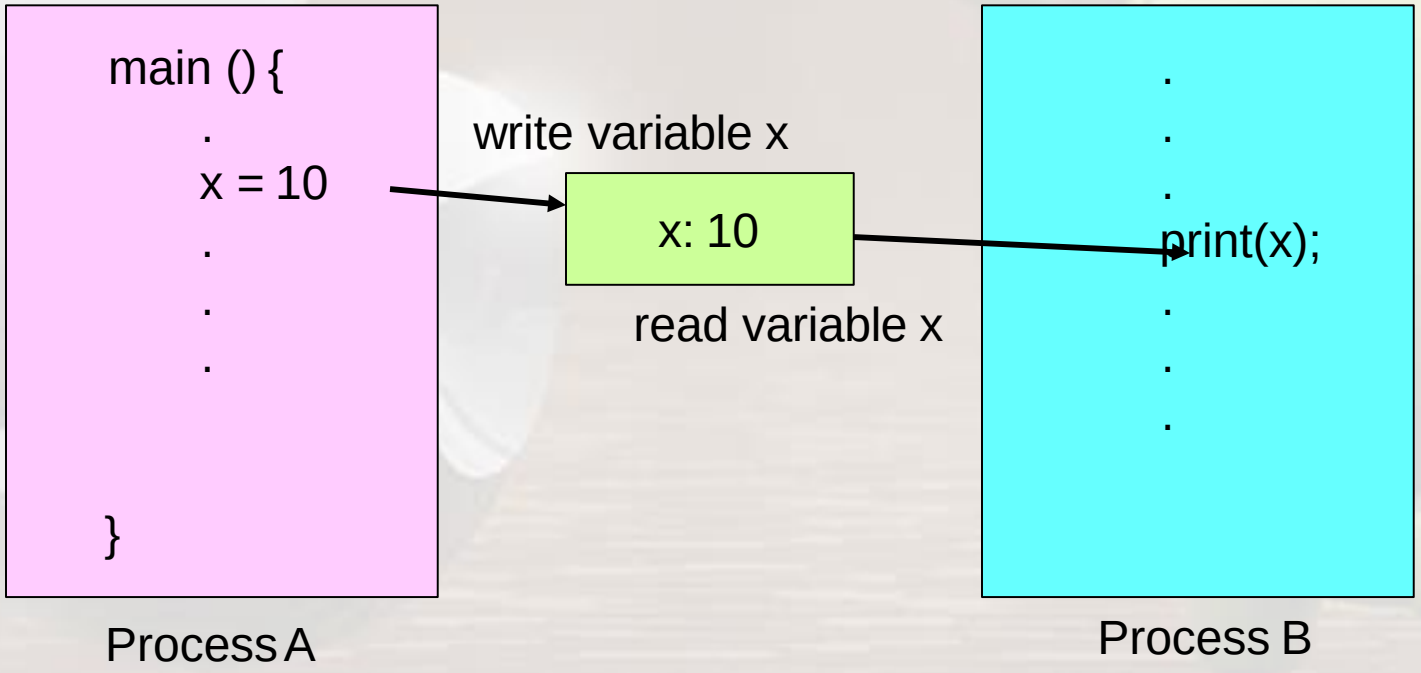
SISTEMAS OPERACIONAIS

Inter-process Communication (IPC)
Memória Compartilhada (Shared Memory)

Shared Memory (1)

- Mecanismo de IPC que cria uma região de memória que pode ser compartilhada por dois ou mais processos.
  - Após a criação, a região deve ser ligada ao processo. Ao ser ligada a um processo, a região de memória criada passa a fazer parte do seu espaço de endereçamento.
  - O processo pode então ler ou gravar no segmento, de acordo com as permissões definidas na operação de “attachment”.
- O S.O. oferece chamadas para criar regiões de memória compartilhada, mas não se envolve diretamente na comunicação entre os processos.
- As regiões e os processos que as utilizam são gerenciados pelo núcleo, mas o acesso ao conteúdo é feito diretamente pelos processos.

Shared Memory (2)



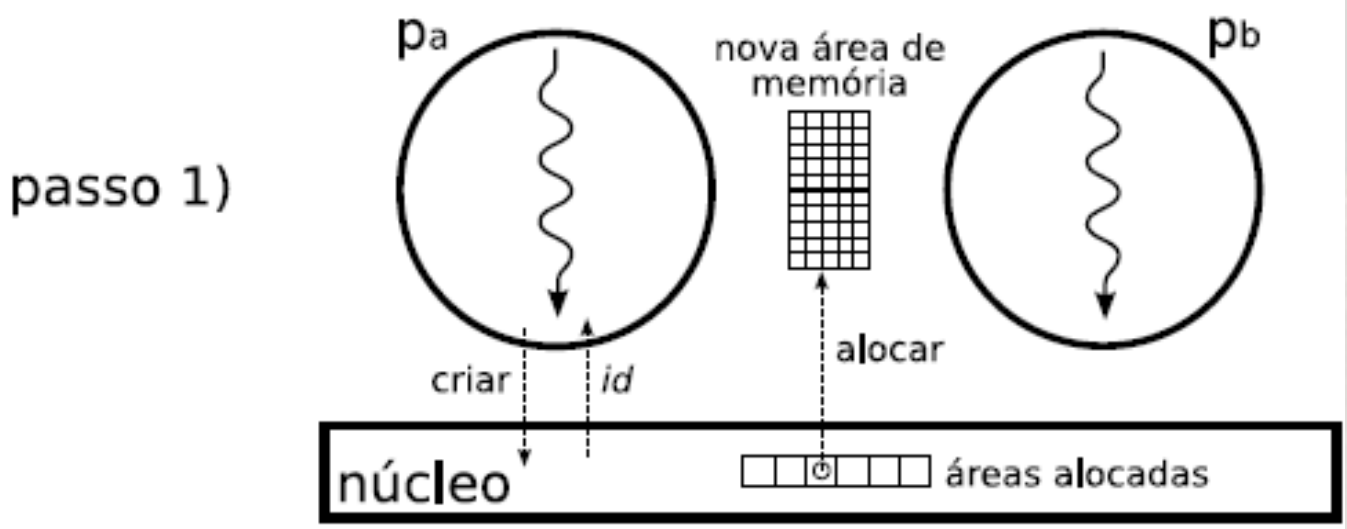
- Se um processo faz alguma modificação na região compartilhada, isso é visto por todos os outros processos que compartilham a região.

Shared Memory (3)

- Vantagens:
  - Eficiência
  - É a maneira mais rápida para dois processos efetuarem uma troca de dados.
  - Os dados não precisam ser passados ao kernel para que este os repasse aos outros processos. O acesso à memória é direto.
  - Acesso randômico
  - Diferentemente dos *pipes*, é possível acessar uma parte específica de uma estrutura de dados que está sendo comunicada.
- Desvantagens:
  - Não existe um mecanismo automático (implícito) de sincronização, podendo exigir, por exemplo, o uso de semáforos para controlar ou inibir condições de corrida.

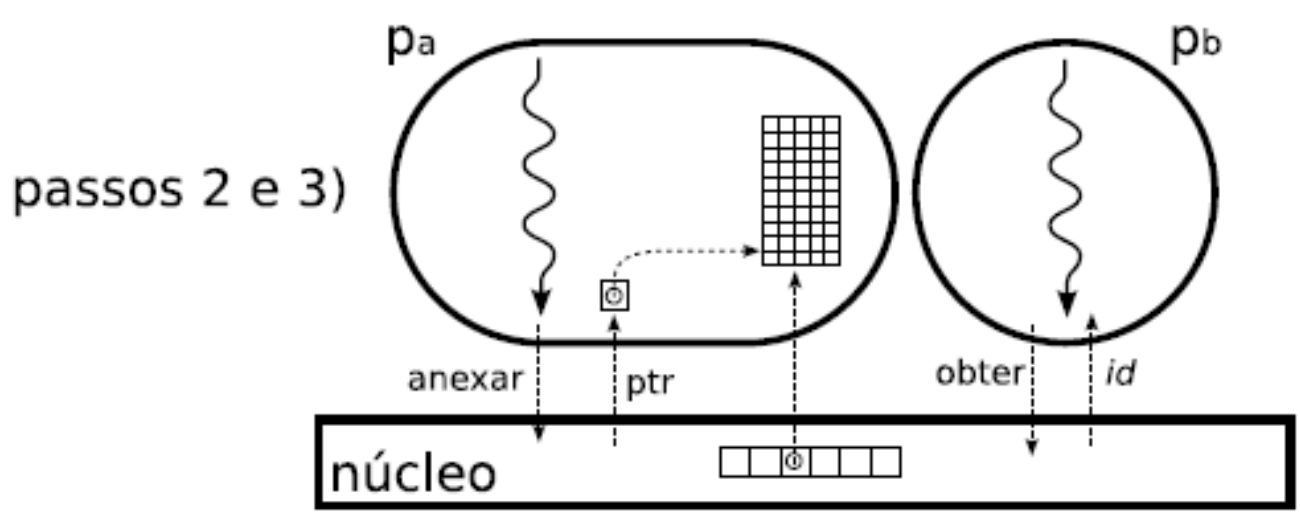
Criação e uso de uma área de memória compartilhada (1)

- Pode ser resumida na seguinte sequência de passos
  - 1 - O processo p_a solicita ao núcleo a criação de uma área de memória compartilhada, informando o tamanho e as permissões de acesso; o retorno dessa operação é um identificador (id) da área criada.



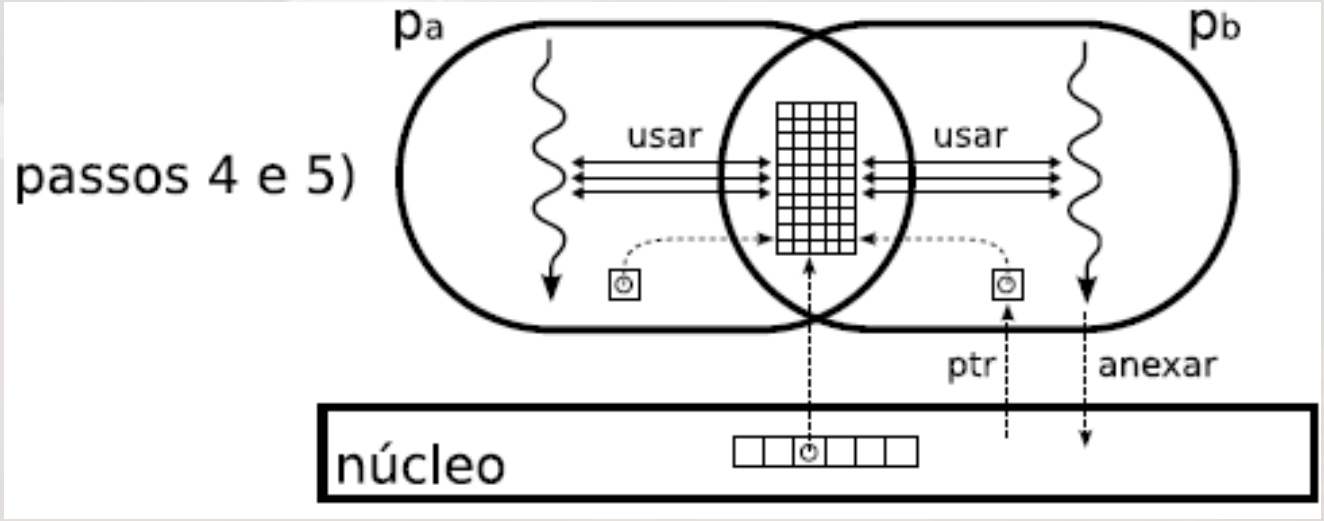
Criação e uso de uma área de memória compartilhada (2)

- 2 - O processo p_a solicita ao núcleo que a área recém-criada seja anexada ao seu espaço de endereçamento.
  - Esta operação retorna um ponteiro para a nova área de memória, que pode então ser acessada pelo processo.
- 3 - O processo p_b obtém o identificador id da área de memória criada por p_a.



Criação e uso de uma área de memória compartilhada (3)

- 4 - O processo p_b solicita ao núcleo que a área de memória seja anexada ao seu espaço de endereçamento e recebe um ponteiro para o acesso à mesma.
- 5 - Os processos p_a e p_b acessam a área de memória compartilhada através dos ponteiros informados pelo núcleo.



Acesso à Memória Compartilhada

- O acesso a memória partilhada é feito por funções com prefixo *shm*.
  - `#include <sys/types.h>`
  - `#include <sys/ipc.h>`
  - `#include <sys/shm.h>`

- Chamadas de sistema:
  - **`shmget()`:** cria zona de memória compartilhada.
  - **`shmat()`:** liga ("*attach*") zona de memória compartilhada ao espaço de endereçamento do processo.
  - **`shmdt()`:** desliga ("*detach*") zona de memória compartilhada do espaço de endereçamento do processo.
  - **`shmctl()`:** desaloca a memória compartilhada ou controla o acesso à zona de memória compartilhada.

Criação de Memória Compartilhada: shmget() (1)

- shmget() é a função usada para criar uma área de memória compartilhada de tamanho size.

```
shmid = shmget(key_type key, int size, int shmflag);
```

  - key: chave de identificação
  - size: tamanho do segmento
  - shmflag: flags de permissão

- A função é encarregada de buscar o elemento especificado pela chave de acesso key, caso esse elemento não exista:
  - pode criar um novo segmento de memória compartilhada OU
  - retornar erro (em função do campo shmflag).
  - Em caso de sucesso, a função devolve o identificador do segmento de memória compartilhada, caso contrário retorna -1.

Criação de Memória Compartilhada: shmget() (2)

- **key:** pode ser criado por meio da função ftok()
- **size:** é o tamanho em bytes do segmento de memória partilhada
- **shmflag**: especifica as permissões do segmento por meio de um OR bit-a-bit:
  - `IPC_CREAT`: caso se queira criar o segmento, caso ele já não exista
  - `IPC_EXCL`: caso se queira exclusivamente criar o segmento (se ele já existir a função retornará -1)
  - `0---:` flags de permissão acesso rwx para usuário-grupo-outros (ex:0664)
  - Pode-se usar constantes pré-definidas... ex: `SHM_R (~0400); SHM_W (~0200)`
    - Ex: `shmget(..., ..., IPC_CREAT|IPC_EXCL|0640)`
- Se shmflg for 0 (zero), a função retorna o id do segmento já existente (deve ser usado quando pretende-se fazer um *attachment* ao segmento).

Criação de Memória Compartilhada: shmget() (3)

- Chave de acesso *key*:
  - Define um identificador único no sistema para a área de memória que se quer criar ou à qual se quer ligar.
  - Todos os processos que quiserem se conectar a área de memória criada devem usar a mesma chave de acesso key.
  - É do tipo long, então qualquer número pode ser usado como chave.

- Existem três maneiras de se gerar a chave de acesso “key”:
  - 1 - Definindo um valor arbitrário
    - Problema: dois programas não relacionados podem definir o mesmo valor de chave embora cada um esteja querendo referenciar segmentos diferentes.
    - Ex: `key_t someKey = 1234;`

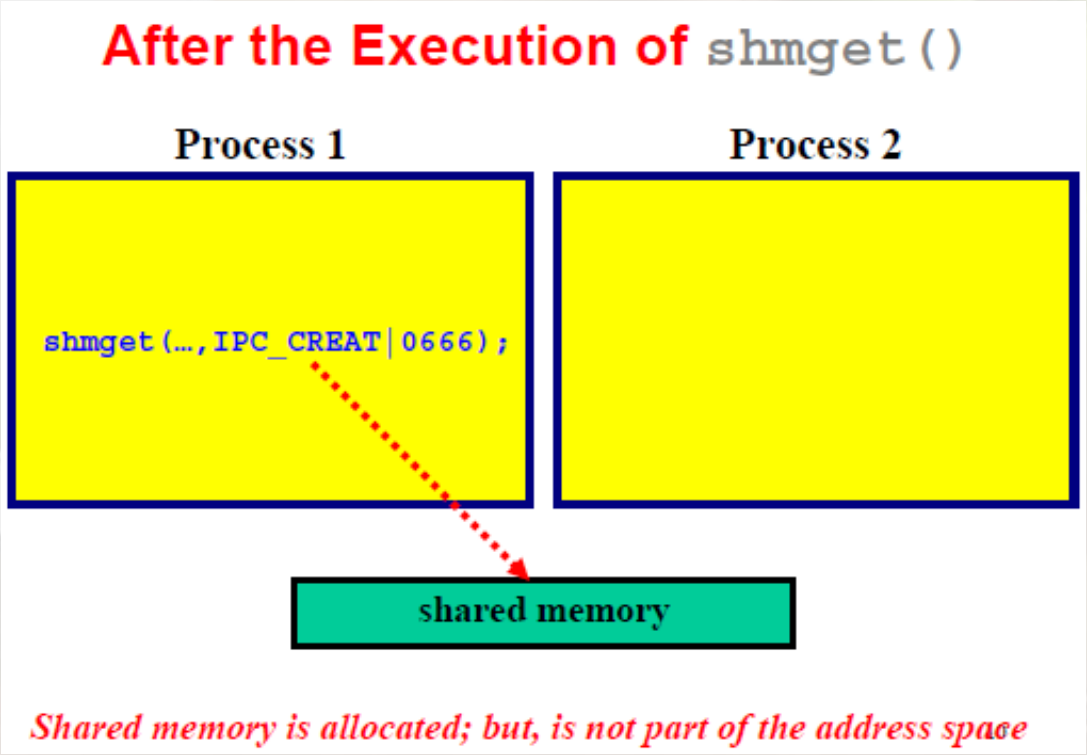
Criação de Memória Compartilhada: shmget() (4)

- 2 - Usando a função `ftok(char *path, int ID)`
  - A função `ftok()` usa alguma informação sobre o arquivo referenciado no argumento *path (p. ex: número do seu *i-node* e *device number* do sistema de arquivo que o contém) juntamente com o valor do campo ID (usualmente um “char” arbitrário qualquer, como “A” ou “x”) para gerar uma chave única (e pública) para `shmget()`.
  - Programas que quiserem acessar a mesma área devem gerar a mesma chave. Para isso, eles devem passar os mesmos parâmetros para ftok().
  - Exemplos:
    - `someKey = ftok(“/home/zegonc/somefile”, ‘b’)`
    - `shmget(ftok(path, id), ..., ...)`

Criação de Memória Compartilhada: shmget() (5)

- 3 - Pedir ao sistema que gere uma chave privada.
  - Usar a constante `IPC_PRIVATE` (ou 0) no campo key
    - Ex: `shmid = shmget (IPC_PRIVATE , … , …)`
  - O kernel gerará uma chave e somente o processo proprietário terá acesso ao segmento de memória compartilhado.
  - Se os outros processos não são descendentes diretos do processo criador, deve-se usar outra forma de IPC para transmitir a chave (por exemplo, salvar a chave em um arquivo).
  - Processos não relacionados geralmente não usam `IPC_PRIVATE`.

Criação de Memória Compartilhada: shmget() (5)



Exemplo 1

- test_shmget.c
  - após executar, rode “ipcs -m”

Lançando duas vezes a execução do programa, tem-se o seguinte resultado:

```
euler:~> test_shmget
Identificador do segmento: 36096
Este segmento e associado a chave unica: 2063804629
euler:~> ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x7b0328d5 36096      saibel     600        1024       0

euler:~> test_shmget
Erro no shmget: File exists
```

Estrutura de Dados da Shared Memory (1)

- Quando um novo segmento de memória é criado, é criada uma estrutura shmid_ds:

```
struct shmid_ds {
   struct ipc_perm shm_perm;/* operation permissions */
   int shm_segsz;           /* size of segment (bytes) */
   time_t shm_atime;        /* last attach time */
   time_t shm_dtime;        /* last detach time */
   time_t shm_ctime;        /* last change time */
   unsigned short shm_cpid; /* pid of creator */
   unsigned short shm_lpid; /* pid of last operator */
   short shm_nattch;        /* no. of current attaches */
};
```

- Nela são mantidas as informações sobre o segmento
  - ex: permissões de acesso definidas pelo parâmetro shmflg

Estrutura de Dados da Shared Memory (2)

- Os campos no membro shm_perm são os seguintes:

```
struct ipc_perm
{
   key_t key;
   ushort uid;       /* owner euid and egid */
   ushort gid;
   ushort cuid;      /* creator euid and egid */
   ushort cgid;
   ushort mode;      /* lower 9 bits of shmflg */
   ushort seq;       /* sequence number */
};
```

Examinando a Memória Compartilhada: shmctl() (1)

- A função `shmctl()` é utilizada para examinar e modificar as informações relativas ao segmento de memória compartilhada.

- Permite ao usuário receber informações relacionadas ao segmento, definir o proprietário ou grupo, especificar permissões de acesso e, adicionalmente, destruir o segmento.

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmctl(int shmid,int cmd, struct shmid_ds *buf);
```

Examinando a Memória Compartilhada: shmctl() (2)

- **cmd** pode conter:
  - `IPC_RMID` (0): O segmento de memória será destruído.
    - O usuário deve ser o proprietário, o criador, ou o super-usuário para realizar esta operação. Todas as outras operações em curso sobre esse segmento irão falhar.
  - `IPC_SET` (1): alterar informações sobre a memória compartilhada
    - Os novos valores são copiados da estrutura apontada por buf. A hora da modificação é também atualizada ;
  - `IPC_STAT` (2): é usada para copiar a informação sobre a memória compartilhada
    - Copia para a estrutura apontada por buf;

- O super-usuário pode ainda evitar ou permitir o swap do segmento compartilhado usando os valores `SHM_LOCK` (3), para evitar o swap, e `SHM_UNLOCK` (4), para permitir o swap.

Exemplo 2

- test_shmctl.c

```
euler:~/> test_shmctl

ESTADO DO SEGMENTO DE MEMORIA COMPARTILHADA 35968
ID do usuario proprietario: 1145
ID do grupo do proprietario: 1000
ID do usuario criador: 1145
ID do grupo criador: 1000
Modo de acesso: 384
Tamanho da zona de memoria: 1024
pid do criador: 930
pid (ultima operacao): 0

euler:~> ipcs -m
------ Shared Memory Segments --------
key shmid owner perms bytes nattch status
```

Ligação à Memória Compartilhada: shmat() (1)

- Depois de criado, é necessário ligar o segmento de memória compartilhada ao espaço de endereçamento do processo.

- O processo usa a função `shmat()` para se ligar a um segmento de memória existente. A função retorna um ponteiro para a memória alocada e esta torna-se parte do espaço de endereçamento do processo.

```
void *shmat(int shm_id, void *shm_ptr int flag);
   shm_id: ID do segmento obtido via shmget() */
   shm_pt: Endereço do acoplamento do segmento */
   Flag: Igual a SHM_RONLY, caso só leitura, ou 0 (zero), caso
   contrário */
```

Ligação à Memória Compartilhada: shmat() (2)

- **shm_ptr**:
  - É um ponteiro que especifica aonde, no espaço de endereçamento do processo, se quer mapear (acoplar) a memória compartilhada.
  - Se for especificado `0` (`NULL`), o usual, o sistema escolhe ele mesmo um endereço disponível para acoplar o segmento no espaço de endereços do processo.

- **flags**:
  - Se igual a `SHM_RND`: indica ao sistema que o endereço especificado no segundo argumento deve ser arredondado (p/ baixo) para um múltiplo do tamanho da página.
  - Se igual a `SHM_RDONLY`: indica que o segmento será read only.
  - Se igual a `0` (zero): indica leitura e escrita.

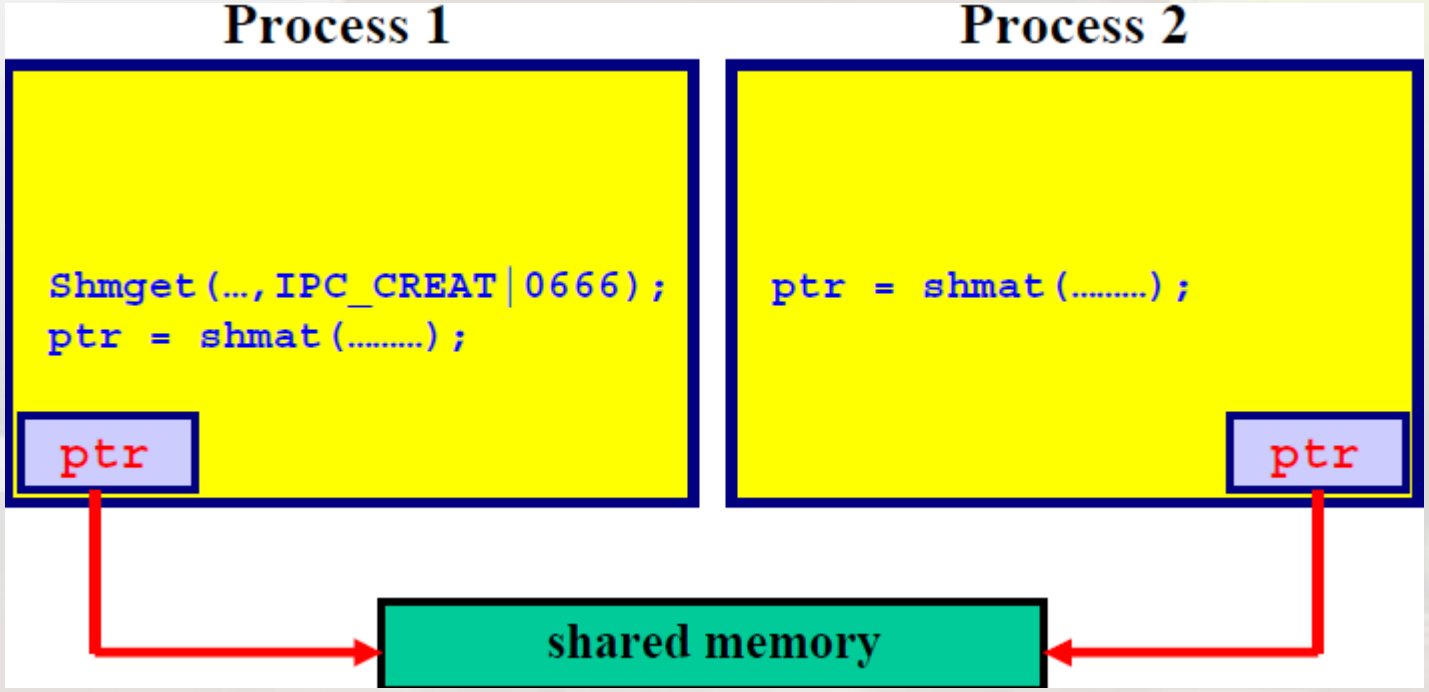
Ligação à Memória Compartilhada: shmat() (3)

- Exemplo:
  - Deste modo, consegue-se ter um ponteiro para todo o segmento de memória

```
p = (int *)shmat(shmid, NULL, 0)
```

```
key_t key;
int shmid;
char *data;
key = ftok("/home/beej/somefile3",'R');
shmid = shmget(key, 1024, 0644 | IPC_CREAT);
data = shmat(shmid, (void *)0, 0);
```

Ligação à Memória Compartilhada: shmat() (4)



Exemplo 3

- test_shmat.c
- test_shmat2.c
  - exemplo de utilização de shmat() escrita num segmento de memoria compartilhada
  - Suponha que um segmento de memória compartilhada tenha sido criado anteriormente através do programa test_shmget.
  - Este programa test_shmat vai reacoplar um processo ao segmento e escrever na memória comum uma cadeia de caracteres.
  - O programa test_shmat2 (a seguir) irá então se acoplar à mesma área de memória e ler seu conteúdo.

Exemplo 4

```
1  #include <stdio.h>
   #include <unistd.h>
   #include <stdlib.h>
   #include <sys/types.h>
   #include <sys/ipc.h>
6  #include <sys/shm.h>
```

```
   int main (int argc, char *argv[])
   {
      key_t key;  /* chave que identifica a area no sistema */
11    int shmid, value, *ptr;

      /* cria uma chave unica */
      key = ftok ("/tmp/reference", 'S');
16    /* abre a area de memoria (se nao existir , cria a area) */
      shmid = shmget (key, sizeof(int), IPC_CREAT | 0660) ;
      if (shmid == -1) {
         perror ("shmget");
         exit (1) ;
21    }

      /* associa a area ao espaco de enderecamento do processo */
      ptr = shmat (shmid, 0, 0) ;
      if ( (int) ptr == -1) {
26       perror ("shmat");
         exit (1);
      }

      while (1) {
31       /* escreve um valor aleatorio na area compartilhada */
         value = random () % 1000 ;
         (*ptr) = value ;
         printf ("Wrote value %i\n", value) ;
         sleep (1);
36
         /* le e imprime o conteudo da area compartilhada */
         value = (*ptr) ;
         printf ("Read value %i\n", value);
         sleep (1) ;
41    }
   }
```

Desconectando/Excluindo Memória Compartilhada (1)

- Para obter informações sobre o segmento compartilhado, deve ser passado `IPC_STAT` como segundo parâmetro e um ponteiro para uma `struct shmid_ds` como terceiro parâmetro.

- Para remover um segmento, passa-se `IPC_RMID` como segundo parâmetro e `NULL` como terceiro parâmetro. O segmento só é removido quando o último processo que está ligado a ele é finalmente desligado dele.

- Cada segmento compartilhado deve ser explicitamente desalocado usando `shmctl` após o seu uso para evitar problemas de limite máximo no número de segmentos compartilhados.
  - A invocação de `exit()` e `exec()` desconeta os segmentos de memória mas não os extingue.

Exemplo 5

- test_shmdt.c
  - allocate, attach, write, detach, reattach, read, detach, deallocate

Exemplo 6

- Comunicação com o Filho

```
void main(int argc, char *argv[])
{
  int   ShmID, *ShmPTR, status;
  pid_t pid;

  ShmID = shmget(IPC_PRIVATE,4*sizeof(int),IPC_CREAT|0666);
  ShmPTR = (int *) shmat(ShmID, NULL, 0);
  ShmPTR[0] = atoi(argv[0]);  ShmPTR[1] = atoi(argv[1]);
  ShmPTR[2] = atoi(argv[2]);  ShmPTR[2] = atoi(argv[3]);
  if ((pid = fork()) == 0) {
     Child(ShmPTR);
     exit(0);
  }
  wait(&status);
  shmdt((void *) ShmPTR);  shmctl(ShmID, IPC_RMID, NULL);
  exit(0);
}
```

15

```
void Child(int  SharedMem[])
{
   printf("%d %d %d %d\n",  SharedMem[0],
        SharedMem[1], SharedMem[2], SharedMem[3]);
}
```

Por que `shmget()` e `shmat()` não são necessárias no processo filho??

Os Comandos ipcs e ipcrm

- O comando ipcs provê informação sobre IPC, incluindo os segmentos compartilhados.
  - Usa-se o flag –m para obter informações sobre memória compartilhada.
  - O exemplo abaixo ilustra que o segmento de 1627649 está em uso:

```
   % ipcs -m
------ Shared Memory Segments -------
key        shmid      owner      perms      bytes      nattch
status
0x00000000 1627649    user       640        25600      0
```

- Se este segmento tiver sido esquecido por um programa, ele pode ser removido usando o comando ipcrm, como mostrado abaixo:

```
% ipcrm shm 1627649
```

Shmem x mmap

- On modern operating systems mmap(2) is typically preferred to the System V IPC Shared Memory facility.

- The main difference between System V shared memory (**shmem**) and memory mapped I/O (**mmap**) is that SystemV shared memory is **persistent**
  - unless explicitly removed by a process, it is kept in memory and remains available until the system is shut down. mmap'd memory is not persistent between application executions (unless it is backed by a file).

Referências

- Slides adaptados de Roberta Lima Gomes (UFES)

- Bibliografia
  - Kay A. Robbins, Steven Robbins, *UNIX systems programming: communication, concurrency, and threads*. Prentice Hall Professional, 2003 - 893 pages
    - Capítulo 15