



Laboratório de Pesquisa em Redes e Multimídia

Inter-process Communication (IPC)

Comunicação entre processos (3)

Memória Compartilhada (*Shared Memory*)

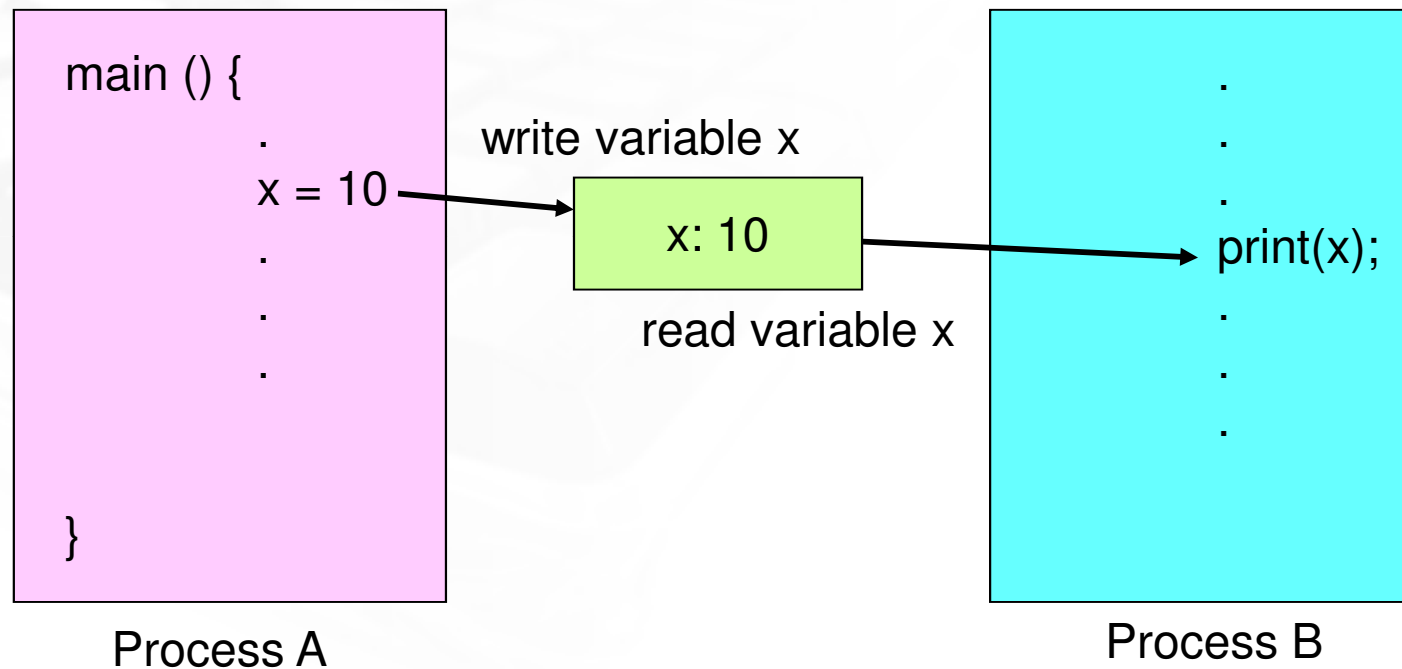


Universidade Federal do Espírito Santo
Departamento de Informática

Memória Compartilhada (*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 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.

Memória Compartilhada (2)



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

Memória Compartilhada (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

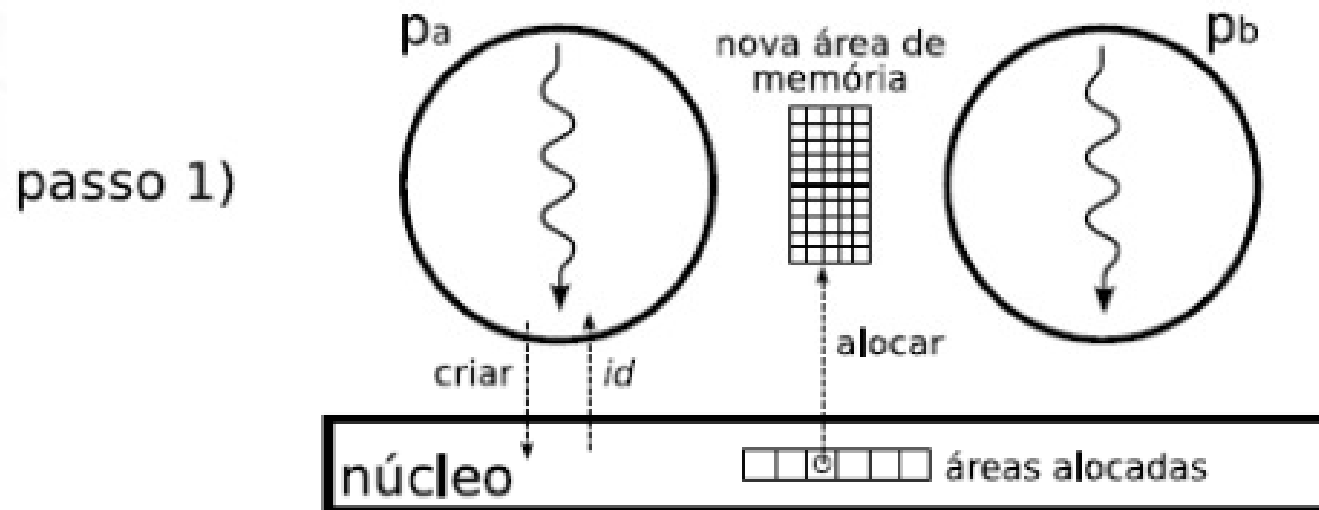
- Diferentemente dos *pipes*, é possível acessar uma parte específica de uma estrutura de dados que está sendo comunicada.

■ Desvantagens:

- Não existe um mecanismo automático (implícito) de sincronização, podendo exigir, por exemplo, o uso de semáforos para controlar ou inibir condições de corrida.

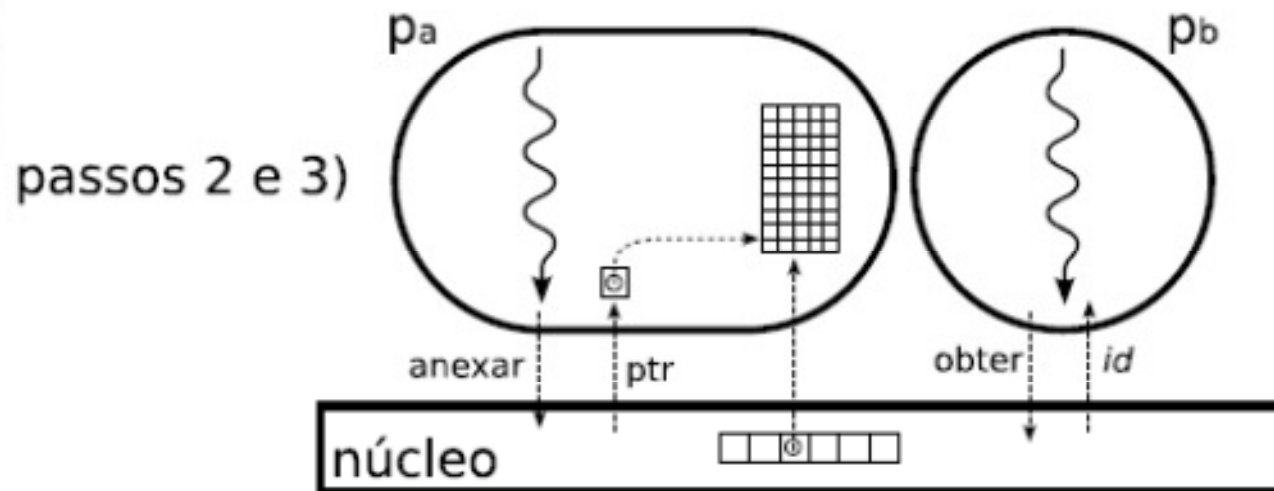
Criação e uso de uma área de memória compartilhada (1)

- Pode ser resumida na seguinte seqüência 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.



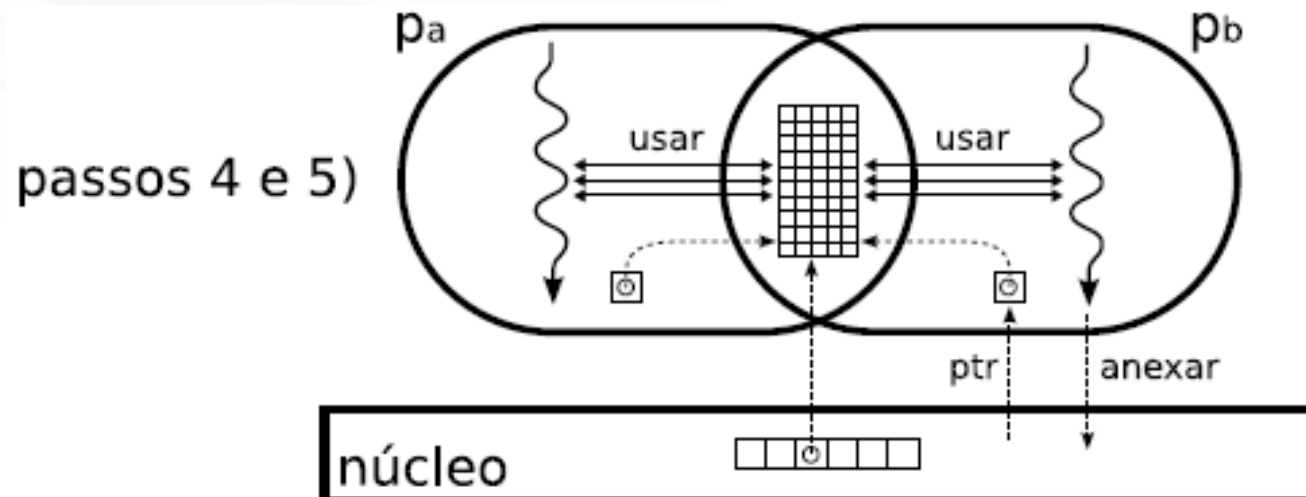
Criação e uso de uma área de memória compartilhada (2)

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`.

```
shmids = shmget(  
    key_type key,          /* chave de identificação  
    int size,             /* tamanho do segmento */  
    int shmflag)         /* flags de permissão */
```

- A função é encarregada de buscar o elemento especificado pela chave de acesso `key` na estrutura `shmids` (ver adiante) e, caso esse elemento não exista, criar um novo segmento de memória compartilhada com tamanho em bytes igual a `size`.
- Em caso de sucesso, a função devolve o identificador do segmento de memória compartilhada, caso contrário retorna -1.

Criação de Memória Compartilhada: `shmget()` (2)

- O primeiro argumento, `key`, deve ser criado através da função `ftok()`, mas existem outras maneiras, dependendo do que se quer fazer.
- O segundo argumento, `size`, é o tamanho em bytes do segmento de memória partilhada.
- Por último, o argumento `shmflg` especifica as permissões do segmento através de um OR bit-a-bit com a constante `IPC_CREAT`, caso se queira criar o segmento.
- Se ao último argumento for atribuído o valor 0 (zero), então pretende-se fazer uma ligação ("attachment") ao segmento, supondo que ele já foi criado.

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`”:
 - 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; SomeKey = 1234;`

Criação de Memória Compartilhada: `shmget()` (4)

- 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:
 - 1) `SomeKey = ftok("/home/zegonc/somefile", 'b')`
 - 2) `shmget(ftok(path, (key_t)key, ..., ...))`

Criação de Memória Compartilhada: shmget() (5)

- Pedir ao sistema que gere uma chave privada.
 - Para isso, usa-se a constante `IPC_PRIVATE` (ou 0) no campo `key`. Neste caso, o kernel gerará uma chave e somente o processo proprietário terá acesso ao segmento de memória compartilhado.
 - Exemplo: `shmid = shmget (IPC_PRIVATE , ... , ...)`
 - 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).
 - Por causa disso, processos não relacionados geralmente não usam `IPC_PRIVATE`.

Criação de Memória Compartilhada: `shmget()` (6)

- Os flags de permissões `shmflag`:
 - Define as permissões de acesso à área compartilhada em três níveis: *owner*, *group*, *other*, da mesma maneira usada na manipulação regular de arquivos.
 - É definido pela combinação, através do operador lógico OU, de diferentes constantes pré-definidas (ex: `IPC_CREAT` e `IPC_EXCL`) juntamente com as permissões de acesso ("mode flags", ex:).
 - Ex: `shmget (... , ... , IPC_CREAT | IPC_EXCL | 0600)`
- `IPC_CREAT`:
 - cria o segmento de memória identificado pela chave `key`. Caso já exista um segmento compartilhado associado a esta chave, anexa o segmento ao espaço de endereçamento do processo.
- `IPC_EXCL`:
 - opção usada em conjunto com `IPC_CREAT` para garantir que retorna erro se o segmento identificado pela chave `key` já existe (e não o anexe, como faz a opção `IPC_CREAT`).

Criação de Memória Compartilhada: shmget() (7)

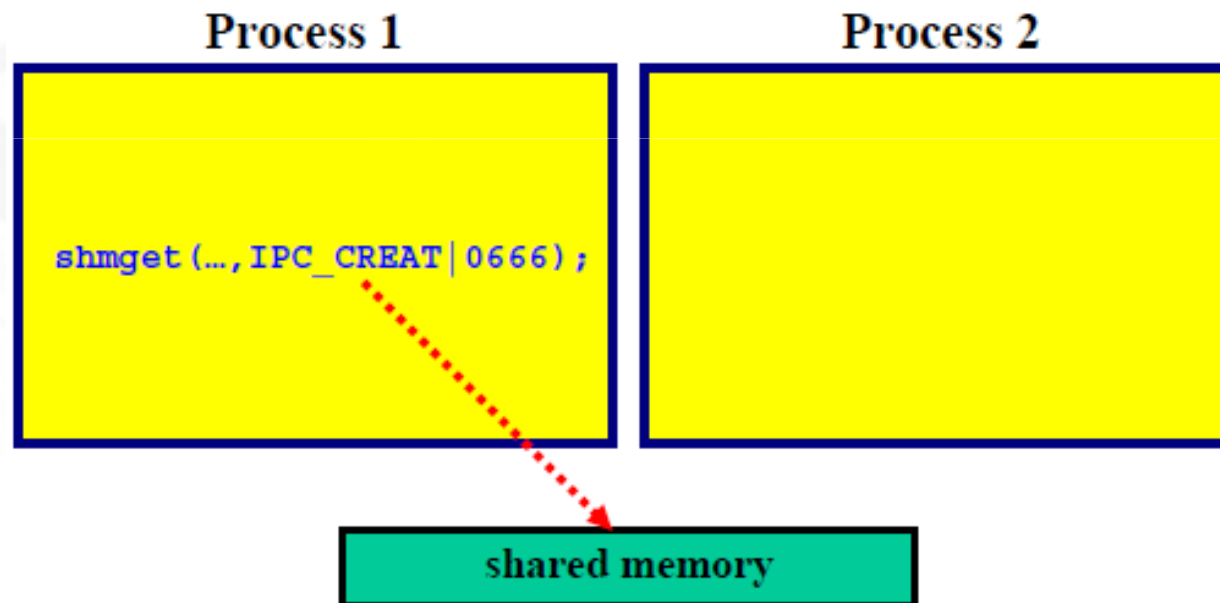
- Os flags de permissões `shmflag`: (cont.)
 - Mode Flags: flag de acesso de 9 bits divididos em 3 bits para *owner*, 3 para *group* e 3 para *other*, como na manipulação de arquivos. Por exemplo, se definido o valor decimal 664 teremos como valor binário associado 110 110 100, que corresponde a rw- rw- r--.
 - Podem ser usadas constantes pré-definidas (ver: man 2 stat). Por exemplo, `S_IRUSR` e `S_IWUSR` para permissão de leitura e escrita para o *owner* do segmento e `S_IROTH` e `S_IWOTH` para dar permissão de leitura e escrita a *other*.

- Exemplos:

```
key_t key; int shmid;
key = ftok("/home/beej/somefile3", 'R');
shmid = shmget(key, 1024, 0644 | IPC_CREAT);
struct Data { int a; double b; char x; }; int ShmID;
shmid = shmget(IPC_PRIVATE, sizeof(struct Data), IPC_CREAT | 0666)
```

Criação de Memória Compartilhada: shmget (8)

After the Execution of `shmget()`



Shared memory is allocated; but, is not part of the address space

Criação de Memória Compartilhada: `shmget` (9)

- Resumindo, as seguintes regras gerais devem ser observadas na criação de um segmento compartilhado:
 - `key` deve conter um valor identificando o segmento (diferente de `IPC_PRIVATE`);
 - `shmflg` deve conter os direitos de acesso desejados para o segmento, e ainda a constante `IPC_CREAT`.
 - Caso se deseje testar a existência ou não de um segmento associado a uma chave especificada, deve-se adicionar (OU lógico) a constante `IPC_EXCL` ao argumento `shmflg`. A chamada `shmget` retornará erro se esse segmento existir.
 - Se ao argumento `flags` da chamada for atribuído o valor 0 (zero), então pretende-se fazer uma ligação ("attachment") ao segmento e não uma criação de segmento.

Exemplo 1

```
/* test_shmget() */
#include <errno.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#define KEY 123

int main() {
    int shmid ; /* identificador da memória comum */
    int size = 1024 ;
    char *path="nome_de_arquivo_existente" ;

    if (( shmid = shmget(ftok(path, (key_t)KEY), size,
        IPC_CREAT|IPC_EXCL|SHM_R|SHM_W)) == -1) {
        perror("Erro no shmget") ;
        exit(1) ;
    }
    printf("Identificador do segmento: %d \n",shmid) ;
    printf("Este segmento e associado a chave unica: %d\n",
        ftok(path, (key_t)KEY)) ;
    exit(0);
}
```

OBS: SHM_R=0400 SHM_W=200
SHM_R AND SHM_W = 0600

Exemplo 1 (cont.)

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
```

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

Exemplo 2

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

    /* cria uma chave unica */
    key = ftok ("/tmp/reference", 'S');
    /* 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);
    }

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

    while (1) {
        /* escreve um valor aleatorio na area compartilhada */
        value = random () % 1000;
        (*ptr) = value;
        printf ("Wrote value %i\n", value);
        sleep (1);

        /* le e imprime o conteudo da area compartilhada */
        value = (*ptr);
        printf ("Read value %i\n", value);
        sleep (1);
    }
}
```

Estrutura de Dados da Shared Memory (1)

- Quando um novo segmento de memória é criado, as permissões de acesso definidas pelo parâmetro `shmflg` são copiadas no membro `shm_perm` da estrutura `shmid_ds` que define efetivamente o segmento compartilhado.
- A estrutura `shmid_ds` é a seguinte:

```
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 */
};
```

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)

- O argumento `cmd` pode conter os seguintes valores:
 - `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)`: dá ao identificador do grupo, ao identificador do usuário e aos direitos de acesso, os valores contidos no campo `shm_perm` da estrutura apontada por `buf`. A hora da modificação é também atualizada (membro `shm_ctime`);
 - `IPC_STAT (2)`: é usada para copiar a informação sobre a memória compartilhada no buffer `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*.

```
/* arquivo test_shmctl.c */
#include <errno.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define KEY 123
struct shmctl_ds buf ;
int main() {
    char *path = "nome_de_arquivo_existente" ;
    int shmctl ;
    int size = 1024 ;
    /* recuperação do identificador do segmento associado à chave 123 */
    if ( ( shmctl = shmctl(ftok(path, (key_t)KEY), size, 0)) == -1 ) {
        perror ("Erro shmctl()") ;
        exit(1) ; }
    /* recuperação das informações relativas ao segmento */
    if ( shmctl(shmctl, IPC_STAT, &buf) == -1){ perror("Erro shmctl()") ;
        exit(1) ;}
    printf("ESTADO DO SEGMENTO DE MEMORIA COMPARTILHADA %d\n", shmctl) ;
    printf("ID do usuario proprietario: %d\n", buf.shm_perm.uid) ;
    printf("ID do grupo do proprietario: %d\n", buf.shm_perm.gid) ;
    printf("ID do usuario criador: %d\n", buf.shm_perm.cuid) ;
    printf("ID do grupo criador: %d\n", buf.shm_perm.cgid) ;
    printf("Modo de acesso: %d\n", buf.shm_perm.mode) ;
    printf("Tamanho da zona de memoria: %d\n", buf.shm_segsz) ;
    printf("pid do criador: %d\n", buf.shm_cpid) ;
    printf("pid (ultima operacao): %d\n", buf.shm_lpid) ;
    /* destruicao do segmento */
    if ( (shmctl(shmctl, IPC_RMID, NULL)) == -1){ perror("Erro shmctl()") ;
        exit(1) ; }
    exit(0);
}
```

Exemplo 3

Neste exemplo, supõe-se que o segmento de memória compartilhada tem a chave de acesso 123 utilizada no Exemplo 1.

Exemplo 3 (cont.)

Resultado da Execução

```
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)

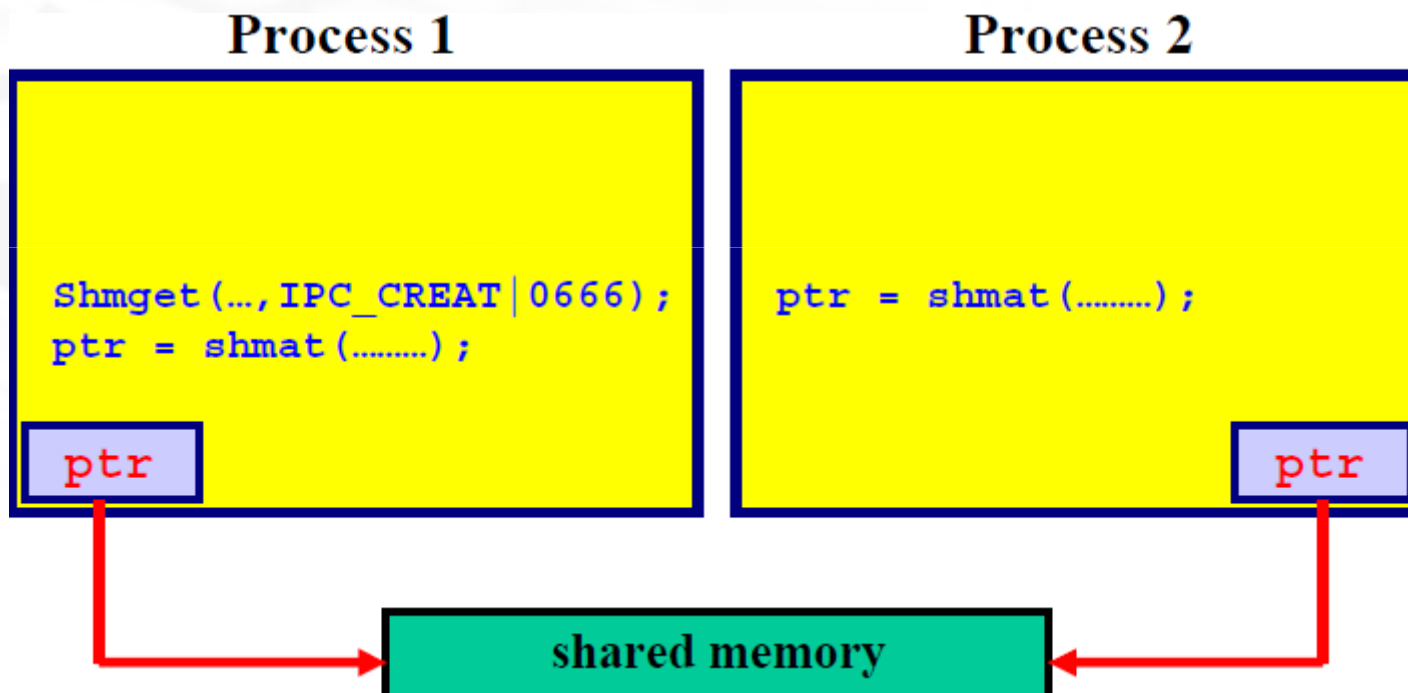
- Depois de criado, é necessário ligar o segmento de memória compartilhada ao espaço de endereçamento do processo.
- O processo usa a função `shmat()` para se ligar a um segmento de memória existente. A função retorna um ponteiro para a memória alocada e esta torna-se parte do espaço de endereçamento do processo.

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

Ligação à Memória Compartilhada: `shmat()` (2)

- Segundo argumento (`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.
- Terceiro argumento (`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)



Ligação à Memória Compartilhada: `shmat()` (4)

- Exemplos:

```
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); /* Deste modo, consegue-se
                                     ter um ponteiro para todo
                                     o segmento de memória */
```

Exemplo 4

```
/* test_shmat.c */
/* exemplo de utilizacao de shmat()
 * escrita num segmento de memoria compartilhada */
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#define KEY 123
#define KEY 123
#define MSG "Mensagem escrita na memoria comum"
int main() {
    int shmid ; /* identificador da memoria comum */
    int size = 1024 ;
    char *path="nome_de_arquivo_existente" ;
    char *mem ;
    int flag = 0;
/* recuperacao do shmid */
    if ((shmid = shmget(ftok(path,(key_t)KEY), size,0)) == -1) {
        perror("Erro no shmget") ;
        exit(1) ; }
    printf("Sou o processo com pid: %d \n",getpid()) ;
    printf("Identificador do segmento recuperado: %d \n",shmid) ;
    printf("Este segmento e associado a chave unica: %d\n",ftok(path,(key_t)KEY)) ;
/* acoplamento do processo a zona de memoria */
    if ((mem = shmat (shmid, 0, flag)) == (char*)-1){
        perror("acoplamento impossivel") ;
        exit (1) ; }
/* escrita na zona de memoria compartilhada */
    strcpy(mem,MSG);
    exit(0);
}
```

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 (cont.)

```
/* test_shmat2.c */
/* programa para ler o conteúdo de um segmento de memória
 * compartilhada que foi preenchido anteriormente por outro processo
 */
#include <errno.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#define KEY 123
int main() {
    int shmid ; /* identificador da memória comum */
    int size = 1000 ;
    char *path="nome_de_arquivo_existente" ;
    char *mem ;
    int flag = 0 ;
    /* recuperação do shmid */
    if (( shmid = shmget(ftok(path, (key_t)KEY), size, 0)) == -1) {
        perror("Erro no shmget") ;
        exit(1) ; }
    printf("Sou o processo com pid: %d \n",getpid()) ;
    printf("Identificador do segmento recuperado: %d \n",shmid) ;
    printf("Este segmento e associado a chave unica: %d\n",
    ftok(path, (key_t)KEY)) ;
    /* acoplamento do processo à zona de memória */
    if ((mem = shmat (shmid, 0, flag)) == (char*)-1){
        perror("acoplamento impossivel") ;
        exit (1) ;}
    /* tratamento do conteúdo do segmento */
    printf("leitura do segmento de memória compartilhada:\n");
    printf("\t==>%s\n",mem) ;
    exit(0);
}
```

```
euler:~/> test_shmget
Identificador do segmento: 41600
Este segmento e associado a chave unica: 2063804629
euler:~/> test_shmat
Sou o processo com pid: 1250
Identificador do segmento recuperado: 41600
Este segmento e associado a chave unica: 2063804629
euler:~/> test_shmat2
Sou o processo com pid: 1251
Identificador do segmento recuperado: 41600
Este segmento e associado a chave unica: 2063804629
leitura do segmento de memoria compartilhada:
    ==>Mensagem escrita na memoria comum
euler:~/> test_shmctl
ESTADO DO SEGMENTO DE MEMORIA COMPARTILHADA 41600
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: 1249
pid (ultima operacao): 1251
```

Note que após o lançamento em seqüência dos programas, o processo com `pid = 1249`, correspondente à execução de `test_shmget` cria o segmento de memória. Depois, esse segmento será acessado por dois processos, sendo que o último é aquele com `pid = 1251`, correspondente à execução de `test_shmat2`.

Exemplo 4

(cont.)
Resultado da
Execução

Desconectando/Excluindo Memória Compartilhada (2)

- 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()` desconecta os segmentos de memória mas não os extingue.

Exemplo 5

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>
int main ()
{
int segment_id;
char* shared_memory;
struct shmid_ds shmbuffer;
int segment_size;
const int shared_segment_size = 0x6400;

/* Allocate a shared memory segment. */
segment_id = shmget (IPC_PRIVATE, shared_segment_size, IPC_CREAT|IPC_EXCL|S_IRUSR|S_IWUSR);

/* Attach the shared memory segment. */
shared_memory = (char*) shmat (segment_id, 0, 0);
printf ("shared memory attached at address %p\n", shared_memory);

/* Determine the segment's size. */
shmctl (segment_id, IPC_STAT, &shmbuffer);
segment_size = shmbuffer.shm_segsz;
printf ("segment size: %d\n", segment_size);
```

Exemplo 5 (cont.)

```
/*Write a string to the shared memory segment*/
    sprintf (shared_memory, "Hello, world.");

/*Detach the shared memory segment*/
    shmdt (shared_memory);

/*Reattach the shared memory at a different address*/
    shared_memory = (char*) shmat (segment_id, (void*) 0x5000000, 0);
    printf ("shared memory reattached at address %p\n", shared_memory);

/*Print out the string from shared memory.*/
    printf ("%s\n", shared_memory);

/*Detach the shared memory segment*/
    shmdt (shared_memory);

/*Deallocate the shared memory segment*/
    shmctl (segment_id, IPC_RMID, 0);
    return 0; }
```

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

Exemplo 6: Comunicação com o Filho (cont.)

```
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
```