Conceito: Memória Compartilhada (Shared Memory)

Trata-se de um 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" (*attached*) ao processo. Ao ser ligada a um processo, a região de memória criada passa a fazer **parte do seu espaço de endereçamento**. Com isso, o processo pode ler ou escrever no segmento, de acordo com as as permissões definidas na operação de "attachment".

O S.O. oferece chamadas de sistemas para criar regiões de memória compartilhada, mas **não** se envolve diretamente na comunicação entre os processos. Isto é, as regiões e os processos que as utilizam são gerenciados pelo núcleo, mas o acesso ao conteúdo (dados) é feito diretamente pelos processos.

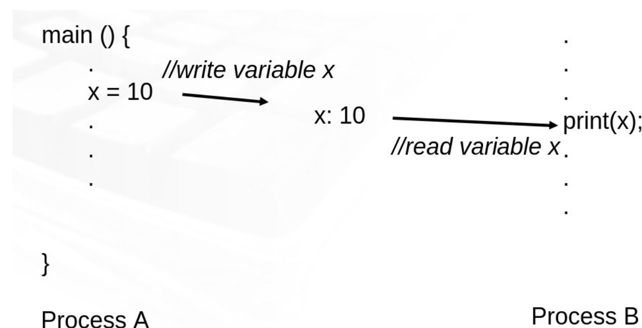

Fig. 1: Se um processo faz alguma modificação na região compartilhada, isso é visto por todos os outros processos que compartilham a região.

As principais vantagens deste tipo de mecanismo de IPC são:

- 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.

Mas há uma desvantagem importante: 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 no UNIX

A criação e uso de um seguimento de memória compartilhada no UNIX se faz por uma sequência de passos:

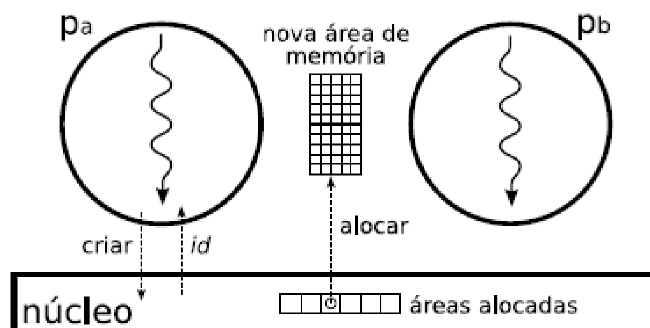

Fig. 2: Passo 1 – Criação de um seguimento de memória "compartilhável"

Passo 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.

Para praticar o "Passo 1" vamos começar usando a seguinte chamada (que ocasiona uma syscall):

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

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

Esta 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.

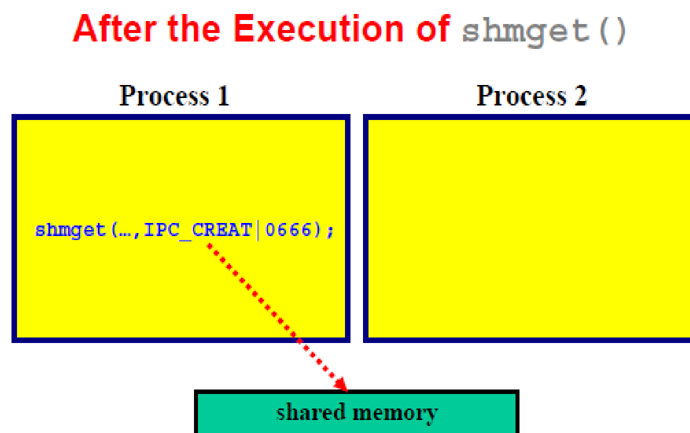

Fig. 3: Criação de um seguimento de memória via **`shmget(…,IPC_CREAT ...)`**

```
/* test_shmget.c()
#include <errno.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>

#define ADDKEY 123
// OBS: SHM_R=0400 SHM_W=200 SHM_R AND SHM_W = 0600

int main() {
      int shmid ; /* identificador da memória comum */
      int size = 1024 ;
      char *path="./" ;
      if (( shmid = shmget(ftok(path,ADDKEY), 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,ADDKEY)) ;
      exit(0);
}
```

TAREFAS:

1) Compile e rode o programa **`test_shmget.c`**. Após executar, rode na bash o comando "ipcs -m"

2) Rode o programa **`test_shmget`** uma segunda vez. O que acontece?

Na TAREFA 2 você verificou que o programa sinaliza um erro. Para entender isso melhor, precisamos primeiramente detalhar os parâmetros da função **`shmget`.**

- **`key`**: Esta "chave de acesso" define um identificador único no sistema para a área de memória que se quer criar ou à qual se quer ligar. Uma chave nada mais é do que um valor inteiro longo. Ela é utilizada para identificar uma estrutura de dados que vai ser referenciada por um programa. Portanto, todos os processos que quiserem se conectar à área de memória criada devem usar a mesma chave de acesso **`key`**. É do tipo `long`, então qualquer número pode ser usado como chave. Mas para evitar que dois processos "não relacionados" corram o risco de usar a mesma chave, o mais indicado é criar chaves 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 qdo pretende-se fazer um "attachment" ao segmento… mas controlem-se… nós só faremos isso nos Passos 2 e 3).

Voltando à TAREFA 2, observem a chamada **`shmget`** usada no test_shmget:

```
shmget(ftok(path,ADDKEY), size, IPC_CREAT|IPC_EXCL|SHM_R|SHM_W)) == -1)
```

Vamos por partes:

- **`ftok(path,ADDKEY)`:** Esta é uma chamada de sistema usada por programadores para gerar chaves únicas. A função ftok() usa o nome do arquivo apontado por **`path`**, que é único no sistema, como uma cadeia de caracteres, e o combina com um identificador **`ADDKEY`** para gerar uma chave do tipo **`key_t`** no sistema IPC.
- **`IPC_EXCL`**: aqui é fácil… estamos dizendo por SO que queremos criar um segumento NOVO. Se ele já existir, o SO deve retornar -1.

Agora vocês conseguem entender por que houve o erro na TAREFA 2? Nesse caso, como vocês já haviam criado o seguimento, na primeira vez que vocês rodaram test_shmget, na segunda vez que vocês rodaram o mesmo código, o seguimento já existia (PERCEBAM… o seguimento é criado e alocado na RAM pelo Sistema Operacional… quando o processo "criador" morre, o seguimento continua existindo! Mas neste caso, ele não foi anexado ainda a nenhum processo… acalmen-se! Faremos isso!).

Continuando a parte prática…

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** da chamada **shmget()** em que o segmento foi criado).

Já os campos no membro **shm_perm** são os seguintes:

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

A chamada de sistema **shmctl()** que permite examinar ou modificar as informações relativas ao segmento de memória compartilhada: ela 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);
```

TAREFA:

3) Compile e rode o programa **test_shmctl.c** (lembre-se de alterar a variável path para o mesmo usado na TAREFA 1). Após executar, rode na bash o comando “ipcs -m”. Você ainda vê o segmento que foi criado na lista? O que aconteceu com ele?

```
/* arquivo test_shmctl.c */

#include <errno.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#define ADDKEY 123
struct shmid_ds buf ;
int main() {
  char path[] = "nome_de_arquivo_existente" ;
  int shmid ;
  int size = 1024 ;
/* recuperação do identificador do segmento associado à chave 123 */
  if (( shmid = shmget(ftok(path,ADDKEY),size,0)) == -1 ) {
        perror ("Erro shmget()") ;
        exit(1) ; }
```

```
/* recuperação das informações relativas ao segmento */
  if ( shmctl(shmid,IPC_STAT,&buf) == -1){ perror("Erro shmctl()") ;
       exit(1) ;}
  printf("ESTADO DO SEGMENTO DE MEMORIA COMPARTILHADA %d\n",shmid) ;
  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(shmid, IPC_RMID, NULL)) == -1){ perror("Erro shmctl()") ;
       exit(1) ; }
  exit(0);
}
```

Observando o código do programa você observa na parte final que o segmento cujos dados de controle foram acessados foi apagado/destruído utilizando-se a chamada: **`shmctl(shmid, IPC_RMID, NULL)`**. Nesse caso o 2º. parâmetro (**`cmd`**) contém **`IPC_RMID`** (equivale a 0), o que significa que o segmento de memória será destruído. Mas para que a chamada seja executada, 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. Os outros valores possíveis para cmd são:

- **`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`**;

Obs.: 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.

IMPORTANTE: Quando **`shmctl(shmid, IPC_RMID, NULL)`** é usado, o segmento só é removido quando o último processo que está ligado a ele é finalmente desligado dele. Além disso, 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.

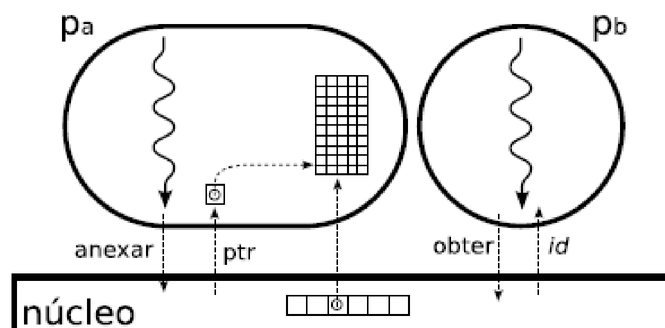

Fig. 4: Passos 2 e 3 – *Attachment* do segmento de memória ao processo p_a

Passo 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 (*ptr*), que pode então ser acessada pelo processo.

Passo 3) O processo p_b obtém o identificador (*id*) da área de memória criada por p_a.

Bom, agora que vocês já sabem criar, verificar e destruir um seguimento de memória "compartilhável", vamos ver como anexá-lo ao espaço de endereçamento de um processo. Para isso, no Passo 2, temos que primeiramente obter o id do segumento… isso vocês já sabem como obter: usando **`shmget`**, seja para criar um novo segmento (a função retorna justamente o id desse segmento criado) seja passando a chave de um seguimento criado anteriormente.

Mas para executar o Passo 3 precisamos de uma outra chamada de sistema:

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

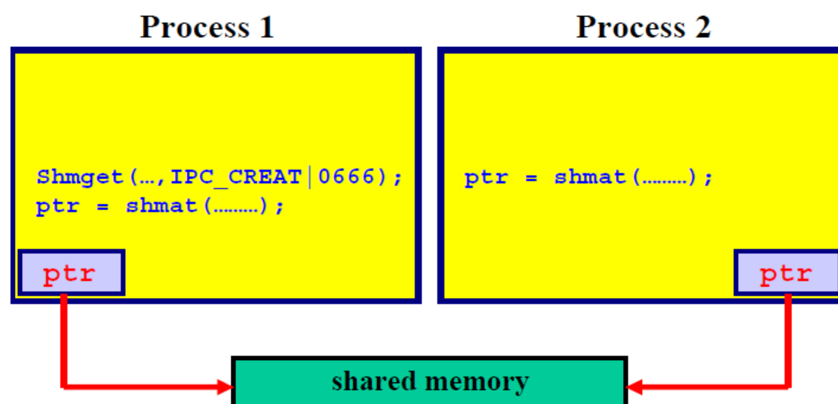

Fig. 5: Ligação à Memória Compartilhada usando shmat()

O processo usa a função **`shmat()`** para se ligar a um segmento de memória existente referenciado pelo identificador **`shm_id`**. A função retorna um **ponteiro** para a memória alocada e esta torna-se parte do espaço de endereçamento do processo. Os outros dois argumentos são:

- **`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 SO escolhe ele mesmo um endereço disponível (dentro da faixa de HEAP) 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.

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 */
```

TAREFA:

4) Rode o programa **`test_shmget`** (para criar o seguimento de memória novamente). Agora compile e rode o programa **`test_shmat.c`** (lembre-se de alterar a variável **`path`** para o mesmo valor usado na TAREFA 1). Após executar, rode na bash o comando "ipcs -m" e observe a coluna **`nattch`**. O que ela mostra?

Passo 4) O processo p_b solicita ao núcleo que a área de memória seja anexada ao seu espaço de endereçamento e recebe um ponteiro (*ptr*) para o acesso à mesma.

Passo 5) Os processos p_a e p_b acessam a área de memória compartilhada através dos ponteiros informados pelo núcleo.

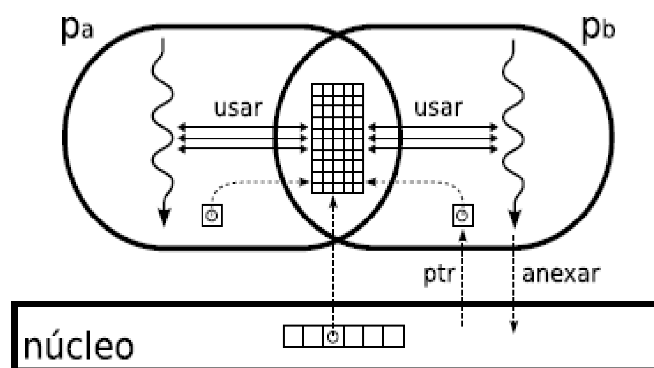

Fig. 4: Passos 5 e 6 – Attachment do segmento de memória ao processo p_b

Para executar os passos 4 e 5, basta que o processo p_b obtenha o id do seguimento de memória (usando **`shmget()`**) e depois faça o attachmento (usando **`shmat`**).

TAREFAS:

4) Compile e rode o programa **`test_shmat2.c`** (lembre-se de alterar a variável **`path`** para o mesmo usado na TAREFA 1). Após executar, antes que os processos morram, rode na bash o comando "ipcs -m" e observe a coluna **`nattch`**. O que ela mostra?

5) Agora rode novamente **`test_shmat`**. Antes que ele "morra", rode **`test_shmctl`** e verifique se o segmento foi excluido (use ipcs -m , você vai ver que ele ainda está lá, mas observe a coluna "chave"). Por fim, tente rodar **`test_shmat2`** e veja se ele conseguiu anexar o segumento ao seu espaço de endereçamento.

Na TAREFA 5, o que aconteceu é que o SO não permite excluir um segmento enquanto ele estiver "attachado" a um processo. Nesse caso, quando um processo tenta destruir um segmento de memória, o SO, ao invés de destruí-lo, torna-o "PRIVATE" (vejam que ao fazer um ipcs -m a coluna "chave" fica zerada)…. Mas o que é isso?!?

Uma forma de se criar um seguimento de memória "compartilhável" é NÃO associar nenhuma chave a ele, criando assim um seguimento "PRIVATE". O processo criador recebe em retorno da chamada shmget() o *id* desse segumento e com esse id é possível fazer o attachment. Observem que, como não

há chave associada, para que outros processos possam acessar este mesmo segumento eles devem ter o seu *id*. Quando o processo criador do segumento fizer fork(), seu descentede (s) poderão acessar o segmento. Geralmente, apenas processos relacionados (ex: que têm "ancestrais em comum) usam segmentos PRIVATE. A grande vantagem desse tipo de segmento é que torna-se impossível um processo não relacionado com o processo criador, por coincidência (ou intencionalmente!), usar uma mesma chave e acabar acessando um seguimento de memória compartilhado que ele não deveria .

Voltando à TAREFA 5, como o SO tornou o segmento PRIVATE outros processos que tente anexar este mesmo segmento (como o caso do **`test_shmat2`**) não o conseguirão.

TAREFA:

6) Agora altere o código de **`test_shmat`** e **`test_shmat2`** de forma que ambos os processos, após realizarem o attachment, imprimam o endereço de memória retornado pelo **`shmat()`**. Eles são iguais ou diferentes? Diferentes… mas por quê? Não deveriam ser iguais já que eles endereçam o a mesma posição de memória!?

Para entender o que aconteceu na TAREFA 6 vocês têm que usar o conceito de Memória Virtual. Na verdade esses endereços retornados por **`shmat()`** em cada processo correspondem a endereços lógicos (ou endereços virtuais), que, por meio a MMU (acessando a tabela de páginas de cada processo), serão convertidos em um MESMO endereço físico!

[EXTRA]

Os programas "normais" não precisam se preocupar com endereços físicos enquanto são executados em um espaço de endereço virtual com todas as suas conveniências. Além disso, nem todos os endereços virtuais têm um endereço físico, eles podem pertencer a arquivos mapeados ou a páginas trocadas. No entanto, às vezes, pode ser interessante ver esse mapeamento, mesmo no território do usuário.

Para este propósito, o kernel Linux expõe seu mapeamento para userland através de um conjunto de arquivos no /proc . A documentação pode ser encontrada em:

https://www.mjmwired.net/kernel/Documentation/vm/pagemap.txt

Pequeno resumo:

1. *`/proc/$pid/maps` provides a list of mappings of virtual addresses together with additional information, such as the corresponding file for mapped files.*
2. *`/proc/$pid/pagemap` provides more information about each mapped page, including the physical address if it exists.*

   *Obs: **sudo** is required to read /proc/[PID]/pagemap*