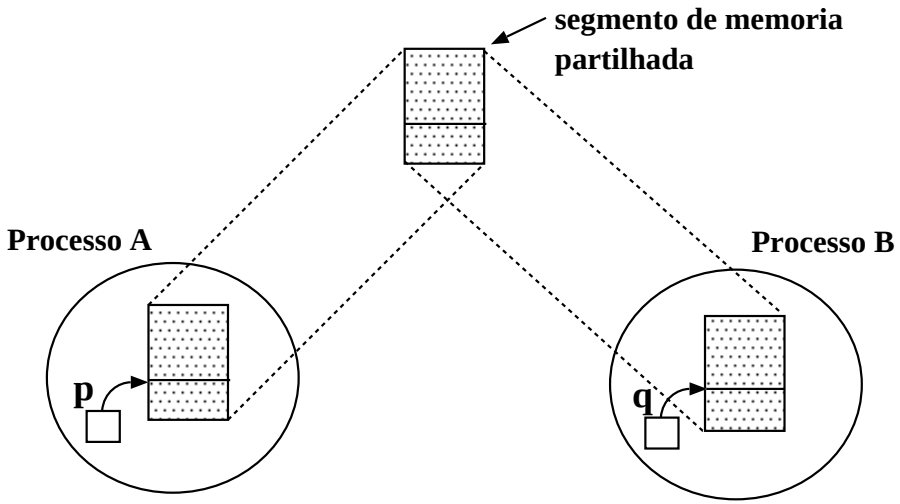
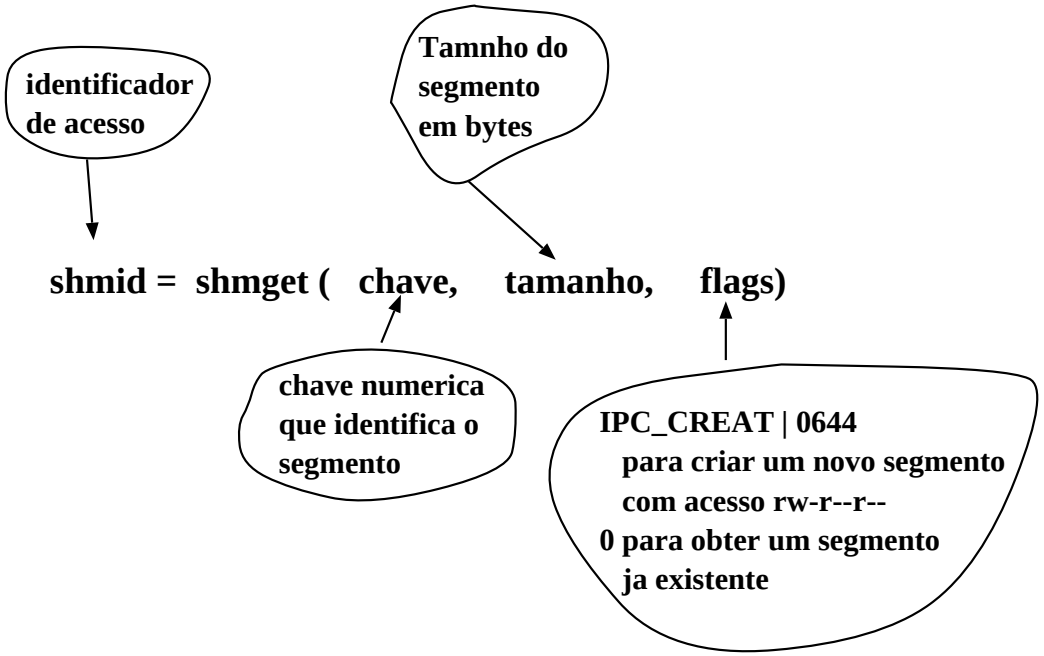
# Memória partilhada em Unix SysV

A forma mais geral de comunicação entre processos é através de memória partilhada.

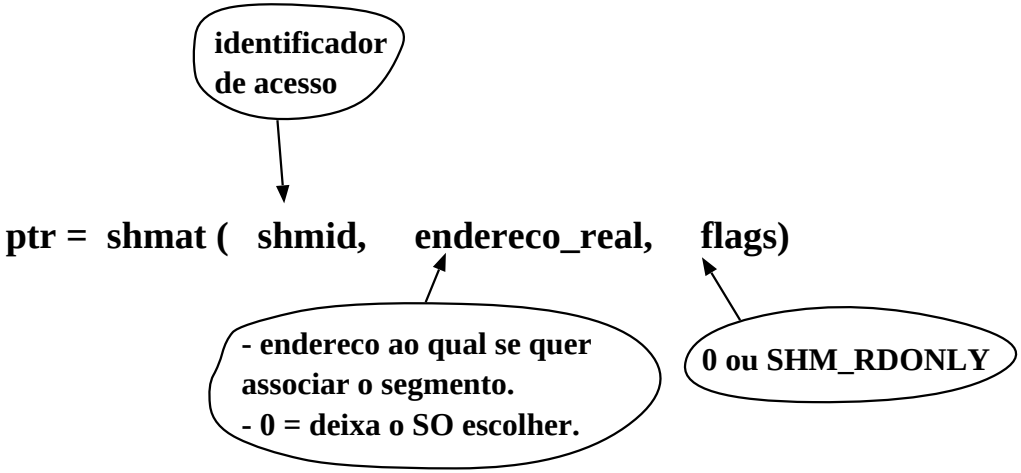



→ Criação de um segmento de memória partilhada:


identificador de acesso
Tamnho do segmento em bytes
shmid = shmget ( chave, tamanho, flags)
chave numerica que identifica o segmento
IPC_CREAT | 0644
para criar um novo segmento com acesso rw-r--r--
0 para obter um segmento ja existente

# Memória partilhada (cont.)

→ Ligar um segmento de memória partilhada a um endereço virtual no espaço de endereçamento do processo:




→ Desligar o segmento de memória partilhada do espaço do processo:

`int shmdt(ptr)` – o segmento deixa de ficar associado ao endereço local ao processo, `ptr`.

→ Remover o segmento de memória partilhada:

`int shmctl(shmid, cmd, buffer)` – caso `cmd` seja `IPC_RMID`, remove o segmento de memória partilhada do sistema.

→ O número de segmentos de memória partilhada que é possível criar é limitado. O SO fornece dois comandos para lidar com segmentos:

- `ipcs` – vêr que segmentos estão a ser usados;
- `ipcrm` – remover um segmento.

# Memória partilhada (cont.): `ipcs`

```
------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 0          root       777        139264     2
0x00000000 65537      ines       600        393216     2          dest
0x00000000 98306      root       777        4096       2
0x00000000 131075     root       777        4096       2
0x00000000 163844     ines       600        393216     2          dest
0x00000000 196613     ines       600        393216     2          dest
0x00000000 229382     root       777        4096       2
0x00000000 262151     root       777        4096       2
0x00000000 294920     root       777        4096       2
```

# Exemplo: shmget() e shmat()

Exemplo de 2 programas (um servidor e outro cliente) que comunicam através de memória partilhada.

```
#include <sys/ipc.h>
#include <sys/shm.h>
#define SHMSZ  27
main() {
  char c, *shm, *s;
  int chave= 5678, shmid;

  shmid= shmget(chave, SHMSZ, (IPC_CREAT|0666));
  shm= (char *)shmat(shmid, NULL, 0);

  s= shm;                  /* escreve info em memória */
  for (c='a'; c<='z'; c++)
    *s++= c;
  *s= '\0';                /* espera até que outro proc.
                              altere o 1o. char em memória */
  while (*shm != '*')
    sleep(1);
  shmdt(shmid);  /* liberta segmento */
  exit(0);
}
```

# Exemplo shmget() e shmat() (cont.)

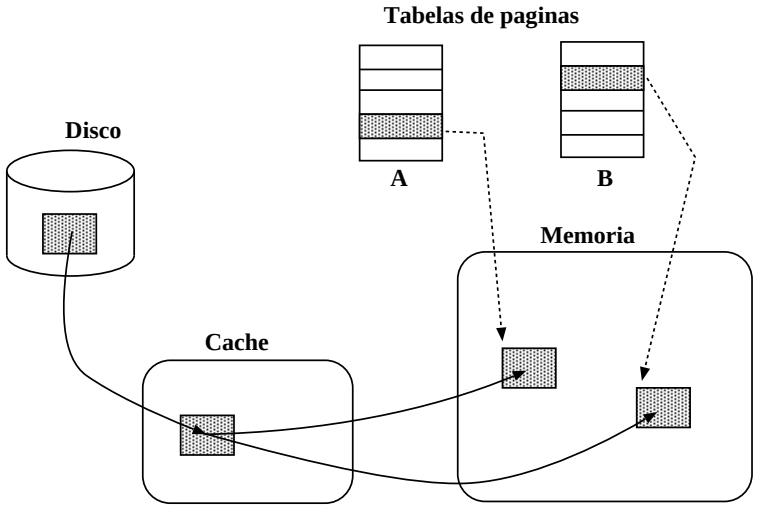
Programa para ler da memória partilhada:

```
#include <sys/ipc.h>
#include <sys/shm.h>
#define SHMSZ  27
main()
{
  char c, *shm, *s;
  int chave= 5678, shmid;

  shmid= shmget(chave, SHMSZ, 0666);
  shm= (char *)shmat(shmid, NULL, 0);

  for (s=shm; *s!='\0'; s++) /* lê da memória partilhada*/
    putchar(*s);
  putchar('\n');
  *shm='*';          /* alterar o 1o. caracter em memória */
  exit(0);
}
```

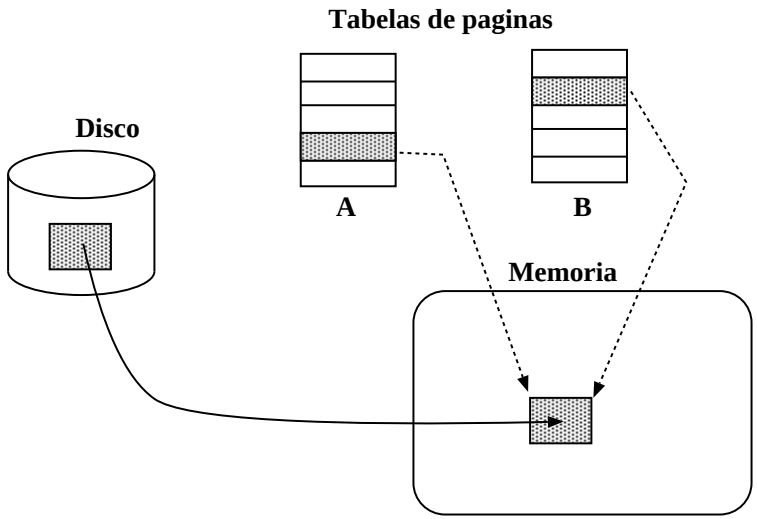
# Mapeamento de Memória em Ficheiros

• Modo tradicional de acesso a ficheiros em Unix: open + (read ou write ou lseek) + close.

Dois processos mapeiam no seu espaço uma cópia da mesma página de um ficheiro:




Dois processos mapeiam a mesma página no seu espaço:




Esta página pode ser mapeada como partilhada ou como privada. Neste caso não há garantia de atomicidade na escrita e leitura do ficheiro (como acontece com read e write).

# Função mmap()

```
aptr= mmap(endereço,tam,prot,flags,fd,offset)
```

onde,

- `aptr` é o endereço onde está colocado o mapeamento.
- `endereço` sugere um endereço em memória para o mapeamento. `0` ou `NULL`: o sistema escolhe.
- `tam` é o tamanho em bytes.
- `prot` é `PROT_READ` ou `PROT_WRITE`
- `flags`: `MAP_SHARED`
- `fd` é o descritor do ficheiro (é necessário abrir o ficheiro antes!)
- `offset` deslocamento no ficheiro onde começar o mapeamento.

Atenção aos passos para cada processo:

- obter o descritor do ficheiro
- fazer o mmap que retorna um apontador
- ler e escrever sobre o ficheiro através do apontador.

# Exemplo com mmap(): rank-sort

```c
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/mman.h>
#define  N     6
#define  SIZE  4*1024*sizeof(int)
int  a[N];
int *b;

main() {
  int childpid, status, fd, i;

  printf("Vector A:\n");
  for(i=0; i<N; i++) {
    a[i]= N-i-1; printf("%d ",a[i]);
  }
  printf("\n");

  fd= open("tmp.mmap",O_RDWR);  /* abre o ficheiro        */
  lseek(fd,SIZE,SEEK_SET);      /* truque para garantir   */
  write(fd,"",1);               /* tamanho do fich é SIZE */

  b=(int *)mmap(0, (N+1)*sizeof(int),
```

```c
void putInPlace(int i) {
  int t, j, rank;

  t= a[i];
  rank= 0;
  for (j=0; j<N; j++)
    if (t>a[j]) rank++;
  b[rank]= t;
  exit(0);
}
```

```
          PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
  for(i=0; i< N; i++)
    if ((childpid = fork()) == 0)
      putInPlace(i);
  for(i=0; i<N; i++)
    wait(&status);
  printf("Vector B:\n");
  for(i=0; i<N; i++)
    printf("%d ",b[i]);
  printf("\n");
}
```