Comunicação entre Processos com Memória Compartilhada

Mário O. de Menezes
Faculdade de Computação e Informática – FCI
Universidade Presbiteriana Mackenzie

Memória Compartilhada

Uma outra técnica de comunicação entre processos é a **Memória compartilhada**, que é uma área de memória *compartilhada* entre dois ou mais processos.

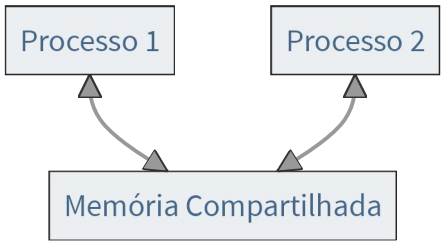
> **Atenção!!**
>
> Veja que isso é não natural! O natural é que as áreas de memórias dos processos sejam totalmente isoladas; um processo não **pode** escrever na área do outro.

Para usar a estratégia de *memória compartilhada* precisamos de suporte do sistema operacional, que vai então criar uma área específica de memória, adicionar esta área às respectivas tabelas de páginas dos processos, de modo que o acesso a esta região não seja considerado ilegal.

Como funciona

Algoritmo

A comunicação entre o Processo 1 e o Processo 2 vai acontecer através da área de memória compartilhada. Nesta área, os processos criam variáveis que são utilizadas para a comunicação.



As etapas são as seguintes:

1. Cria o segmento de memória compartilhada ou utiliza um já criado (`shmget()`).
2. Anexa o processo ao segmento de memória compartilhada já criado (`shmat()`).
3. Desanexa o processo do segmento de memória compartilhada já anexado (`shmdt()`).
4. Controla operações no segmento de memória compartilhada (`shmctl()`).

Funções utilizadas

`shmget()`

Cria ou aloca um segmento de memória compartilhada (*System V*).

```
1  #include <sys/ipc.h>
2  #include <sys/shm.h>
3
4  int shmget(key_t key, size_t size, int shmflg)
```

Os argumentos são os seguintes:

- **`key`** – reconhece o segmento de memória; pode ser um valor arbitrário ou pode ser derivado da biblioteca `ftok()`. Também pode ser `IPC_PRIVATE`, isto é, os processos serão executados no modelo servidor e cliente (relacionamento do tipo *pai-filho*).
- **`size`** – tamanho do segmento de memória a ser alocado, arrendondado a um múltiplo de `PAGE_SIZE`.
- **`shmflg`** – especifica o(s) flag(s) da memória compartilhada requerida, tais como, `IPC_CREAT` (criando um novo segmento), `IPC_EXCL` (usado com `IPC_CREAT` para cria um novo segmento, mas vai falhar se o segmento já existir). Também pode passar as permissões

Funções utilizadas (cont.)

shmat()

Esta função anexa o segmento de memória compartilhada ao espaço de endereçamento do processo chamador.

```
1  #include <sys/types.h>
2  #include <sys/shm.h>
3
4  void * shmat(int shmid, const void *shmaddr, int shmflg)
```

Os argumentos são os seguintes:

- **shmid** – o identificador do segmento de memória compartilhada, é o retorno da função shmget().
- **shmaddr** – para especificar o endereço anexado. Se shmaddr é NULL o sistema, por default, escolhe o endereço adequado para anexar o segmento.
- **shmflg** – especifica os flag(s) requeridos de memória compartilhada, tais como SHM_RND (arredondamento do endereço para SHMLBA) ou SHM_EXEC (permite o conteúdo do segmento ser executado), ou SHM_RDONLY (anexa o segmento somente para leitura) ou SHM_REMAP (substitui o mapeamento existente na faixa especificada por shmaddr e continua até o final do segmento).

Esta função retorna o endereço do segmento de memória compartilhada anexado quando tem sucesso, ou -1 em caso de falha. O motivo da falha deve ser verificado na variável errno ou com a função perror().

Funções utilizadas (cont.)

`shmdt()`

```c
#include <sys/types.h>
#include <sys/shm.h>

int shmdt(const void *shmaddr)
```

A chamada de sistema acima realiza a desanexação do segmento de memória especificado pelo argumento **`shmaddr`**, que deve ser o mesmo retornado pela chamada de sistema `shmat()`. Esta chamada retorna 0 no sucesso e -1 em caso de falha. O motivo da falha deve ser verificado na variável `errno` ou com a função `perror()`.

Funções utilizadas (cont.)

`shmctl()`

```
1  #include <sys/ipc.h>
2  #include <sys/shm.h>
3
4  int shmctl(int shmid, int cmd, struct shmid_ds *buf)
```

Esta chamada de sistema realiza operações de controle do segmento de memória compartilhada. Os seguintes argumentos devem ser passados:

- **`shmid`** – o identificador do segmento de memória compartilhado, é o `id` retornado pela chamada `shmget()`.
- **`cmd`** – é o comando para realizar a operação de controle requerida no segmento de memória compartilhada. Os valores para `cmd` são apresentados a seguir.
- **`buf`** – ponteiro para a estrutura de memória compartilhada chamada `shmid_ds`. Os valores nesta estrutura serão utilizados ou para a definição (*set*) ou para retorno (*get*).

Funções utilizadas (cont.)

`shmctl()` (cont.)

Os valores para `cmd` são:

- **`IPC_STAT`** – copia a informação dos valores atuais de cada membro da estrutura `shmid_ds` para o ponteiro passado em `buf`. Este comando requer permissão de leitura no segmento de memória compartilhada.
- **`IPC_SET`** – define o ID do usuário, ID do grupo do proprietário, permissões, etc, apontados pela estrutura `buf`.
- **`IPC_RMID`** – marca o segmento para ser destruído; acontecerá após o último processo desanexá-lo.
- **`IPC_INFO`** – retorna a informação acerca dos limites e parâmetros do segmento de memória compartilhada na estrutura apontada por `buf`.
- **`SHM_INFO`** – retorna a estrutura `shm_info` contendo informação acerca dos recursos do sistema consumidos pela memória compartilhada.

Funções utilizadas (cont.)

`shmctl()` (cont.)

O valor de retorno desta chamada depende do comando passado. Em caso de sucesso de `IPC_INFO` e `SHM_INFO` ou `SHM_STAT`, retorna o índice ou identificador do segmento de memória compartilhada, ou 0 para as outras operações; e -1 em caso de falha. Causa da falha deve ser verificada na variável `errno` ou com a função `perror()`.

Exemplo

Algoritmo

Vamos examinar o seguinte programa exemplo, descrito nas seguintes etapas:

- Criamos dois processos, um para escrita no segmento de memória compartilhada (`shm_write.c`) e outro para leitura do segmento de memória compartilhada (`shm_read.c`).
- O programa realiza escritas no segmento de memória compartilhada através do processo de escrita (`shm_write.c`) e leitura no segmento de memória compartilhada pelo processo de leitura (`shm_read.c`).
- No segmento de memória compartilhada, o processo de escrita cria um segmento de memória compartilhada de 1K (e flags) e anexa a memória compartilhada.
- O processo de escrita escreve 5 vezes o alfabeto de “A” até “E” em cada um dos 1023 bytes no segmento de memória compartilhada. O último byte significa o final do buffer.

Exemplo (cont.)

Algoritmo (cont.)

- O processo de leitura vai ler do segmento de memória compartilhada e escrever na saída padrão.
- As ações dos processos de leitura e escrita são realizadas simultaneamente.
- Após completar a escrita, o processo de escrita atualiza para indicar que completou a escrita no segmento de memória compartilhada (com a variável `complete` na estrutura `shmseg`).
- O processo de leitura realiza a leitura do segmento de memória compartilhada e mostra na saída até que ele receba a indicação de que o processo de escrita completou (com a variável `complete` na estrutura `shmseg`).
- O programa realiza leituras e escritas por algum tempo para simplificação e para evitar um loop infinito e complicar desnecessariamente.

Exemplo – Código shm_write.c

```c
/* Filename: shm_write.c */
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

#define BUF_SIZE 1024
#define SHM_KEY 0x1234

struct shmseg {
   int cnt;
   int complete;
   char buf[BUF_SIZE];
};
int fill_buffer(char * bufptr, int size);

int main(int argc, char *argv[]) {
   int shmid, numtimes;
   struct shmseg *shmp;
   char *bufptr;
   int spaceavailable;
   shmid = shmget(SHM_KEY, sizeof(struct shmseg), 0644|IPC_CREAT);
   if (shmid == -1) {
      perror("Shared memory");
      return 1;
   }
```

Exemplo – Código shm_write.c (cont.)

```
    // Attach to the segment to get a pointer to it.
    shmp = shmat(shmid, NULL, 0);
    if (shmp == (void *) -1) {
       perror("Shared memory attach");
       return 1;
    }
    /* Transfer blocks of data from buffer to shared memory */
    bufptr = shmp->buf;
    spaceavailable = BUF_SIZE;
    for (numtimes = 0; numtimes < 5; numtimes++) {
       shmp->cnt = fill_buffer(bufptr, spaceavailable);
       shmp->complete = 0;
       printf("Writing Process: Shared Memory Write: Wrote %d bytes\n", shmp->cnt);
       bufptr = shmp->buf;
       spaceavailable = BUF_SIZE;
       sleep(3);
    }
    printf("Writing Process: Wrote %d times\n", numtimes);
    shmp->complete = 1;
    if (shmdt(shmp) == -1) {
       perror("shmdt");
       return 1;
    }
    if (shmctl(shmid, IPC_RMID, 0) == -1) {
       perror("shmctl");
       return 1;
    }
    printf("Writing Process: Complete\n");
    return 0;
}
```

Exemplo – Código `shm_write.c` (cont.)

```
1  int fill_buffer(char * bufptr, int size) {
2     static char ch = 'A';
3     int filled_count;
4     //printf("size is %d\n", size);
5     memset(bufptr, ch, size - 1);
6     bufptr[size-1] = '\0';
7     if (ch > 122)
8     ch = 65;
9     if ( (ch >= 65) && (ch <= 122) ) {
10       if ( (ch >= 91) && (ch <= 96) ) {
11          ch = 65;
12       }
13    }
14    filled_count = strlen(bufptr);
15    //printf("buffer count is: %d\n", filled_count);
16    //printf("buffer filled is:%s\n", bufptr);
17    ch++;
18    return filled_count;
19 }
```

Exemplo – Código shm_read.c

```c
/* Filename: shm_read.c */
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>

#define BUF_SIZE 1024
#define SHM_KEY 0x1234

struct shmseg {
   int cnt;
   int complete;
   char buf[BUF_SIZE];
};

int main(int argc, char *argv[]) {
   int shmid;
   struct shmseg *shmp;
   shmid = shmget(SHM_KEY, sizeof(struct shmseg), 0644|IPC_CREAT);
   if (shmid == -1) {
      perror("Shared memory");
      return 1;
   }
```

Exemplo – Código `shm_read.c` (cont.)

```
    // Attach to the segment to get a pointer to it.
    shmp = shmat(shmid, NULL, 0);
    if (shmp == (void *) -1) {
       perror("Shared memory attach");
       return 1;
    }

    /* Transfer blocks of data from shared memory to stdout*/
    while (shmp->complete != 1) {
       printf("segment contains : \n\"%s\"\n", shmp->buf);
       if (shmp->cnt == -1) {
          perror("read");
          return 1;
       }
       printf("Reading Process: Shared Memory: Read %d bytes\n", shmp->cnt);
       sleep(3);
    }
    printf("Reading Process: Reading Done, Detaching Shared Memory\n");
    if (shmdt(shmp) == -1) {
       perror("shmdt");
       return 1;
    }
    printf("Reading Process: Complete\n");
    return 0;
}
```

Exercícios

1. Escreva um programa que use memória compartilhada para verificar se uma palavra fornecida pelo usuário é palíndrome ou não.
2. Escreva um programa que use memória compartilhada para inverter uma palavra fornecida pelo usuário.

Referências e outros links

1. What is shm
2. Example 1
3. Example 2
4. Tutorials Point (esta é a referência básica para estes slides.)