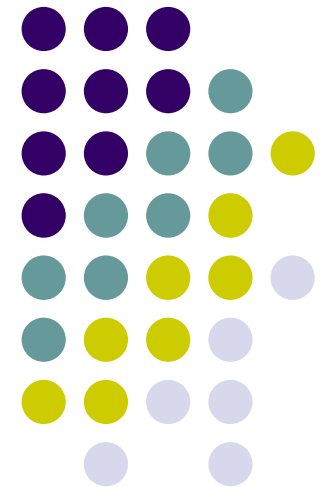


Comunicação entre Processos

Memória Compartilhada

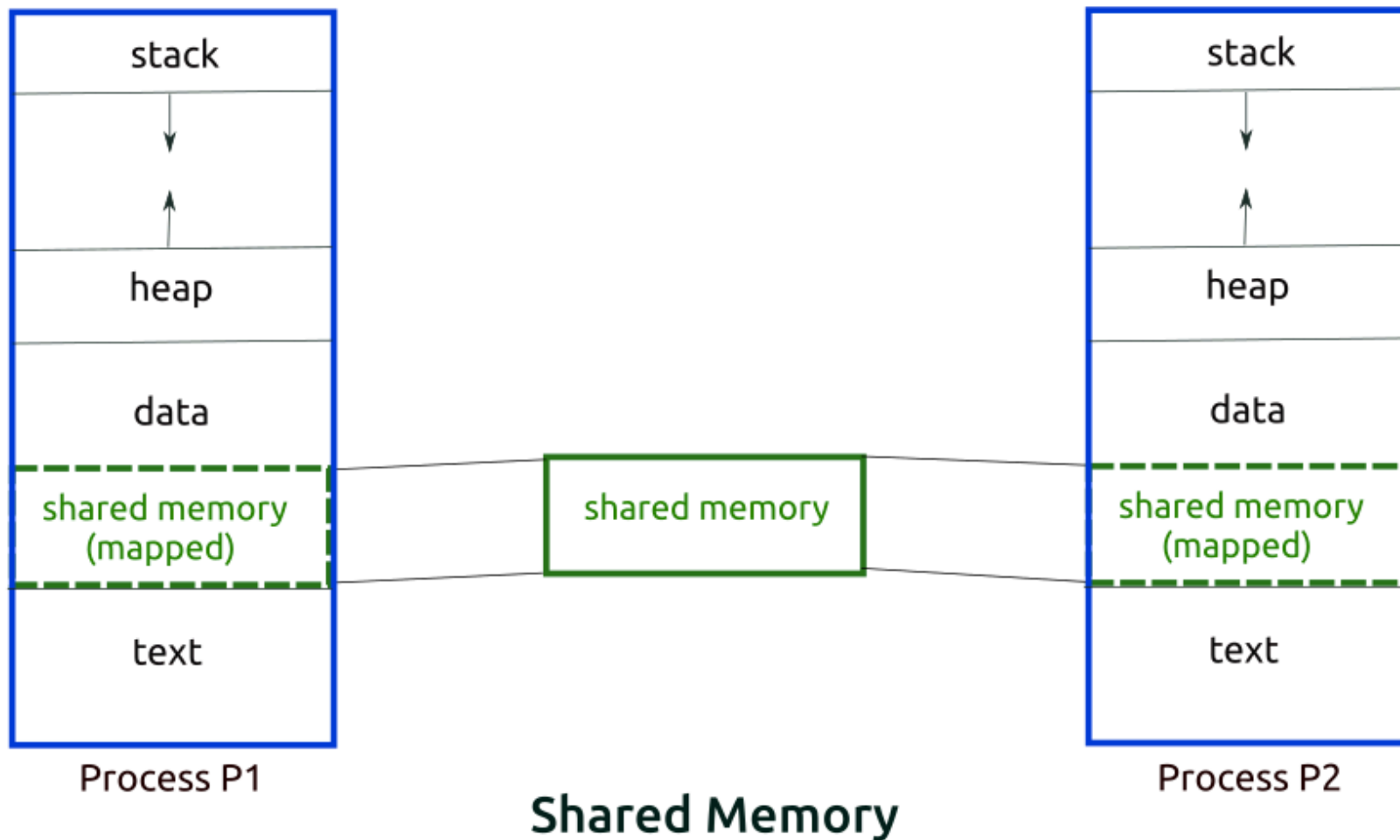


Métodos de comunicação entre processos

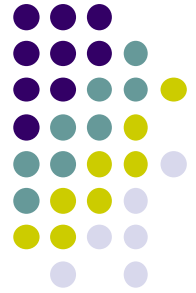


- **Memória compartilhada**
- Sinais
- Pipes
- Troca de mensagens

Memória Compartilhada

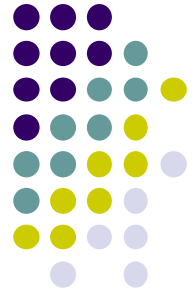


Memória compartilhada



- Permite a comunicação entre processos apenas lendo e/ou escrevendo em uma região de memória
- Forma mais rápida de comunicação entre processos
- Não necessita chamadas ao sistema (system call) para comunicação
- Ausência de suporte por parte do kernel em sincronização
- É a solução para que processos pai e filho acessem a mesma posição de memória.

Modelo



- Um processo aloca a memória
 - A memória é alocada em múltiplos do tamanho da página do sistema (tipicamente 4kB)
- Outros processos se conectam (**attach**) a esta memória
- Processos podem se comunicar escrevendo e lendo dessa memória
- Ao término
 - Todos os processos se desconectam (**detach**) da memória
 - Um processo libera a memória alocada

Alocação da memória

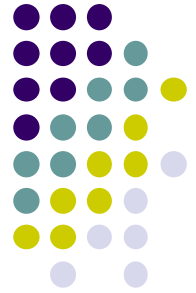


- Alocar espaço em memória utilizando shmget()
 - Definido em <sys/shm.h>

```
int shmget(key_t key, size_t size, int shmflg);
```

- Onde:
 - key: chave de identificação da memória ou IPC_PRIVATE para gerar um identificador novo
 - size: quantidade MÍNIMA de memória a ser alocada
 - shmflg: modo de criação – normalmente 0 (zero) – mais detalhes no próximo slide
- Retorna:
 - Um identificador (IPC ID) da área alocada em caso de sucesso
 - -1 em caso de erro

Flags de modo de criação



- `<sys/ipc.h>`
 - **IPC_CREAT** : Create entry if key does not exist.
 - **IPC_EXCL** : Fail if key exists.
 - **IPC_NOWAIT** : Error if request must wait.
- `<sys/stat.h>`
 - **S_IRWXU** : Read, write, execute/search by owner.
 - **S_IRUSR** : Read permission, owner.
 - **S_IWUSR** : Write permission, owner.
 - **S_IXUSR** : Execute/search permission, owner.
 - **S_IRWXG** : Read, write, execute/search by group.
 - **S_IRGRP** : Read permission, group.
 - **S_IWGRP** : Write permission, group.
 - **S_IXGRP** : Execute/search permission, group.
 - **S_IRWXO** : Read, write, execute/search by others.
 - **S_IROTH** : Read permission, others.
 - **S_IWOTH** : Write permission, others.
 - **S_IXOTH** : Execute/search permission, others.

Anexar memória compartilhada (attach)

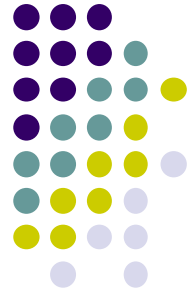


- Usar `shmat()` para anexar o segmento já criado com `shmget()`
 - Definido em `<sys/shm.h>`

```
void *shmat(int shmid, const void *shmaddr,  
int shmflg);
```

- Onde:
 - `shmid`: identificador da área já alocada
 - `shmaddr`: endereço de referência para a alocação da página – normalmente 0 (zero) ou NULL (nulo)
 - `shmflg`: modo como o endereço da página deve ser anexado – normalmente 0 (zero)
- Retorna:
 - O endereço da página em caso de sucesso
 - -1 em caso de falha

Desanexar memória compartilhada (detach)

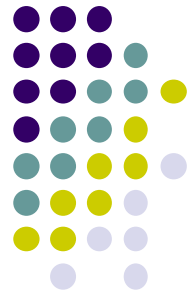


- Desanexa o segmento anexado por `shmat()`
 - Definido em `<sys/shm.h>`

```
int shmdt(const void *shmaddr) ;
```

- Onde:
 - `shmaddr` : endereço do segmento obtido por `shmat()`
- Retorna:
 - 0 (zero) no caso de sucesso
 - -1 em caso de falha

Controle da memória compartilhada



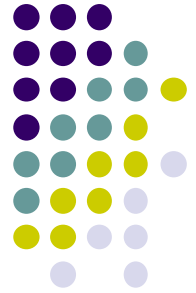
- Informações sobre o segmento anexado com shmctl()
 - Definido em <sys/shm.h>

```
int shmctl(int shmid, int cmd, struct shmid_ds
*buf) ;
```

- Onde:
 - shmid: identificador da área já alocada
 - cmd: comando, que pode ser:
 - IPC_STAT: preenche a estrutura shmid_ds
 - IPC_SET: modifica os atributos shm_perm.uid, shm_perm.gid, shm_perm.mode de acordo com a estrutura shmid_ds
 - SHM_LOCK e SHM_UNLOCK para bloquear e liberar a shared memory
 - **IPC_RMID: remove a memória compartilhada identificada por shmid**
 - buf: ponteiro para a estrutura shmid_ds definida em <sys/shm.h>
- Retorna:
 - 0 em caso de sucesso
 - -1 em caso de falha

Um exemplo com todos os comandos e atributos: <http://goo.gl/bNqpkk>

Estrutura `shmid_ds` e `ipc_perm`



- `<sys/shm.h>`
 - A estrutura `shmid_ds` possui, pelo menos, estes campos:
 - `struct ipc_perm shm_perm` : Operation permission structure.
 - `size_t shm_segsz` : Size of segment in bytes.
 - `pid_t shm_lpid` : Process ID of last shared memory operation.
 - `pid_t shm_cpid` : Process ID of creator.
 - `shmatt_t shm_nattch` : Number of current attaches.
 - `time_t shm_atime` : Time of last *shmat* ().
 - `time_t shm_dtime` : Time of last *shmdt* ().
 - `time_t shm_ctime` Time of last change by *shmctl* ().
- `<sys/ipc.h>`
 - A estrutura `ipc_perm` tem, pelo menos, estes campos:
 - `uid_t uid` : Owner's user ID.
 - `gid_t gid` : Owner's group ID.
 - `uid_t cuid` : Creator's user ID.
 - `gid_t cgid` : Creator's group ID.
 - `mode_t mode` : Read/write permission.



Exemplo:

- O seguinte programa cria uma área de memória compartilhada, representando um número inteiro
- O programa cria um processo filho
- Ambos incrementam a memória compartilhada e apresentam o resultado do incremento
- O filho incrementa em 5 unidades, enquanto o pai em 10

```

#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <unistd.h>
#include <sys/wait.h>
int main (int argc, char *argv[])
{
    int segmento, *p, id, pid, status;
    // aloca a memória compartilhada
    segmento = shmget (IPC_PRIVATE, sizeof (int), IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);
    // associa a memória compartilhada ao processo
    p = (int *) shmat (segmento, 0, 0); // comparar o retorno com -1
    *p = 8752;
    if ((id = fork()) < 0)
    {
        puts ("Erro na criação do novo processo");
        exit (-2);
    }
    else if (id == 0)
    {
        *p += 5;
        printf ("Processo filho = %d\n", *p);
    }
    else
    {
        pid = wait (&status);
        *p += 10;
        printf ("Processo pai = %d\n", *p);
    }

    // libera a memória compartilhada do processo
    shmdt (p);
    // libera a memória compartilhada
    shmctl (segmento, IPC_RMID, 0);

    return 0;
}

```

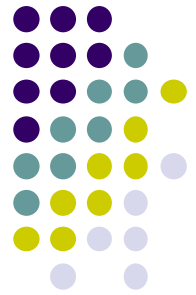


```

~/Documents/ProgramacaoUnix/programas/IPC$ ./shm
Processo filho = 8757
Processo pai = 8767
~/Documents/ProgramacaoUnix/programas/IPC$

```

Outra possibilidade: **shm_open/ mmap / unmmap**



- **shm_open()** retorna um descritor de arquivo que pode ser usado para acessar a memória compartilhada.
- Isso é conveniente pois é possível usar todos os operadores de controle de descritores de arquivos para controlar também a memória compartilhada.
- A única desvantagem é que **shm_open()** requer um nome de arquivo, arquivo que será veinculado a memória compartilhada.
- **mmap()** mapeia a memória compartilhada para dentro do espaço de endereço do processo.
- No final, **unmmap()** desfaz o mapeamento, e no final precisa-se fechar o descritor de arquivo desvinculando o objeto de memória compartilhada usando **close()** e **shm_unlink()**
- Veja o exemplo em: [Using Shared memory in Linux](#)

Perguntas?



1) Soma de matrizes



Faça um programa para somar matrizes de acordo com o seguinte algoritmo

- O primeiro processo irá criar duas matrizes preenchidas e uma terceira vazia em 3 áreas de memória compartilhada.
- Para cada linha da matriz solução, o seu programa deverá gerar um processo para o seu cálculo.

OBS: implemente as matrizes como vetores de tamanho (linha x coluna) e aloque a shared memory para os vetores correspondentes, pois acessar os elementos (i,j) é complexo.



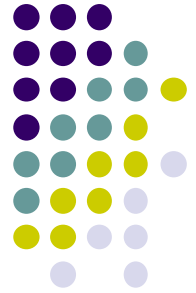
Filho 1	$\begin{pmatrix} 5 & 7 & 9 \end{pmatrix}$	+	$\begin{pmatrix} 5 & 3 & 0 \end{pmatrix}$	=	$\begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$
Filho 2	$\begin{pmatrix} 6 & 3 & 6 \end{pmatrix}$				
Filho 3	$\begin{pmatrix} 3 & 1 & 2 \end{pmatrix}$				

2) Mensagem do Dia



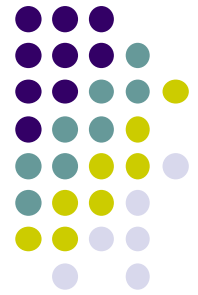
- Faça um programa que:
 - Leia uma mensagem do dia do stdin (ou arquivo)
 - Crie uma memória compartilhada com a chave 8752
 - Salve a mensagem na memória
- Faça um outro programa “cliente” que utilize a mesma chave (8752) e exiba a mensagem do dia para o usuário

3) Busca paralela em vetor



- Faça um programa paralelo (com pelo menos 4 processos) para localizar uma chave em um vetor.
 - Crie uma memória compartilhada com dados numéricos inteiros e desordenados e a divida pelo número de processos
 - Cada processo deve procurar o dado na sua área de memória e informar a posição onde o dado foi localizado.

4) Multiplicação multi-processo



Faça um programa que:

- Tenha um processo pai que abre dois blocos de memória compartilhada, m1 e m2.
- cria dois processos filho (use exec), P1 e P2: estes também fazem attach em m1 ou m2 respectivamente
- Cada um dá um sleep() randômico e escreve um valor int na área compartilhada dele, e avisa o processo pai que um novo valor foi gerado, escrevendo tb um nr de sequencia
- O pai fica em loop verificando se houve um novo valor. Apenas quando ambos P1 e P2 geraram um novo valor, o pai imprime o produto dos valores gerados por P1 e P2

Arquitetura

