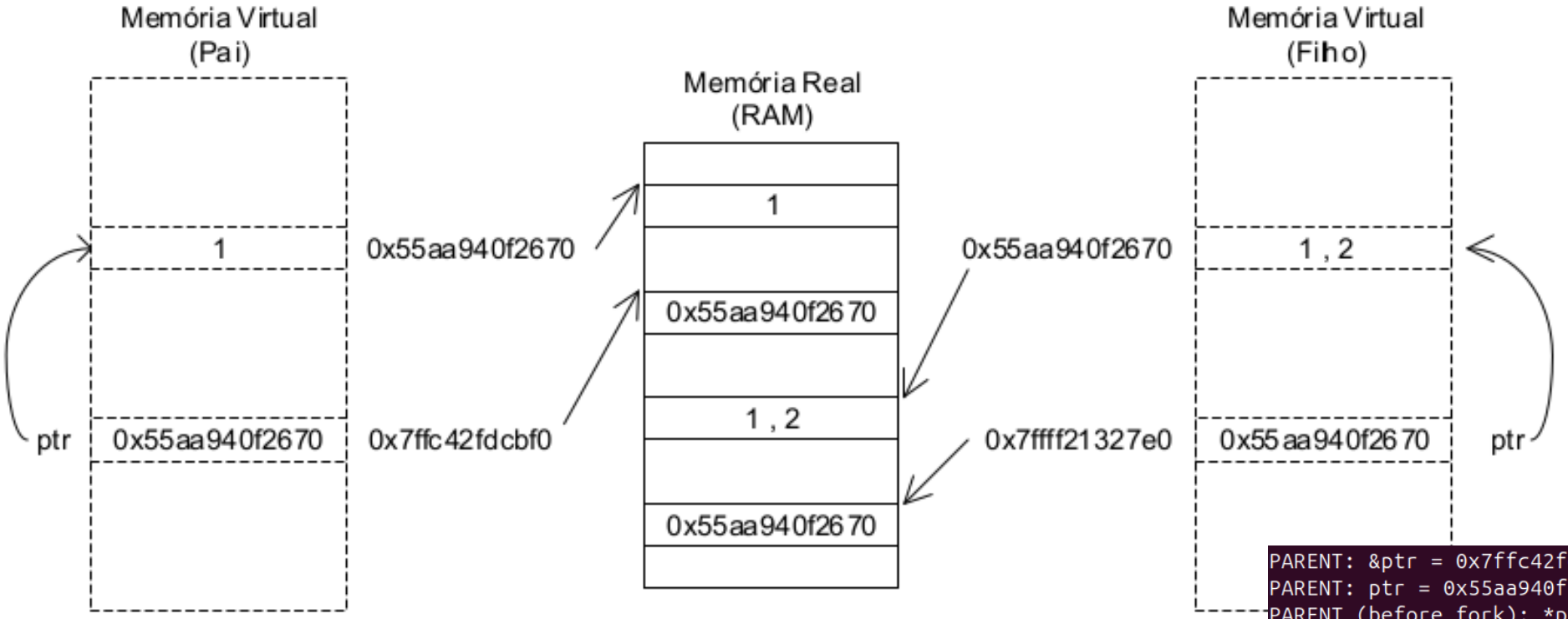
Memória Partilhada

Duplicado... mas não partilhado 🍴 💔

```c
// Program 2.1
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main(){
    int *ptr;
    printf("PARENT: &ptr = %p\n", &ptr); // show the virtual address of the pointer variable
    // reserve a dynamic memory block for one integer, and get a pointer to it
    ptr = (int*)malloc(sizeof(int));
    printf("PARENT: ptr = %p\n", ptr); // show the virtual address of the dynamic memory block
    // write something in the dynamic memory block and show it
    *ptr = 1; printf("PARENT (before fork): *ptr = %d\n", *ptr);

    if(fork()==0) { // create a CHILD that is going to access its own copy of the block
        printf("CHILD: &ptr = %p\n", &ptr); // the virtual address of the pointer is still the same!
        printf("CHILD: ptr = %p\n", ptr); // and the virtual address of the block is still the same!
        // write something diferent in the dynamic memory block and show it
        *ptr = 2; printf("CHILD (before exit): *ptr = %d\n", *ptr);
        free(ptr); exit(0);
    }

    // wait for the CHILD to die and then show the block content
    wait(NULL); printf("PARENT (after wait): *ptr = %d\n", *ptr);
    free(ptr); return(0);
}
```

```
PARENT: &ptr = 0x7ffc42fdcbf0
PARENT: ptr = 0x55aa940f2670
PARENT (before fork): *ptr = 1
CHILD: &ptr = 0x7ffc42fdcbf0
CHILD: ptr = 0x55aa940f2670
CHILD (before exit): *ptr = 2
PARENT (after wait): *ptr = 1
```

A memória dinâmica (e já agora, a memória estática) é privada relativamente ao processo! 😱




```
PARENT: &ptr = 0x7ffc42fdcbf0
PARENT: ptr = 0x55aa940f2670
PARENT (before fork): *ptr = 1
CHILD: &ptr = 0x7ffc42fdcbf0
CHILD: ptr = 0x55aa940f2670
CHILD (before exit): *ptr = 2
PARENT (after wait): *ptr = 1
```

Mapa de memória do processo pai e do processo filho – output do Programa 2.1

Solução? Usar memória partilhada! 

1. Alocar um segmento de memória partilhada → **`shmget`**
2. Associar (mapear) o segmento ao espaço de memória do processo → **`shmat`**
3. Libertar o mapeamento ao terminar → **`shmdt`**
4. Remover o segmento de memória → **`shmctl`**

shmget – shared memory get

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

- Cria/obtém memória partilhada; devolve ID ou -1 em caso de erro
- **key** identifica o segmento de memória a ser acedido por outros processos
- Key/chave **IPC_PRIVATE** → cria um novo segmento privado
- Chave hexadecimal (e.g., **0x1234abcd**) → cria um novo segmento, se ainda não existir (**IPC_CREAT** em **shmflg**); sem **IPC_EXCL**, devolve um segmento existente se a chave existir
- **size** → tamanho do segmento em bytes
- **shmflg** (e.g., **00600**) → permissões de acesso; combinar com **|**
  - **IPC_CREAT** → cria segmento; **IPC_EXCL** → falha se a chave já existir

shmget – shared memory get

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

- Diretivas necessárias: **<sys/ipc.h>**, **<sys/shm.h>**
- `shmid = shmget(IPC_PRIVATE, N*sizeof(int), 00600); // private memory for N ints`
- `shmid = shmget(0x00000001, sizeof(int), 00600 | IPC_CREAT | IPC_EXCL); // non-private shared memory for one int, fails if key exists`

shmat – shared memory attach

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

- Associa o segmento **shmid** ao espaço de endereços do processo; devolve o endereço ou -1 em caso de erro
- **shmaddr** → endereço virtual pedido; usualmente **NULL** para ser o SO a escolher; endereços indicados pelo utilizador podem ser ignorados, se indisponíveis
- **shmflg**: **0** = leitura/escrita, **SHM_RDONLY** = apenas leitura; em qualquer caso, prevalecem as permissões definidas pelo criador via **shmget**
- Um processo pode associar-se **várias vezes** ao mesmo segmento, obtendo endereços virtuais diferentes
- `ptr = (int*)shmat(shmid, NULL, 0); // attach segment, OS chooses the address`

shmdt – shared memory detach

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

- Quebra a associação entre o endereço virtual **shmaddr** e o segmento de memória partilhada; devolve **0** em caso de sucesso, **-1** em caso de erro
- Nota: depois de **exit** ou **return** no **main**, a desassociação ocorre automaticamente

shmctl – shared memory control

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

- Executa operações de controlo (identificadas por **cmd**) sobre o segmento de memória partilhada (**shmid**); retorna **0** em caso de sucesso, **-1** em caso de erro. Para libertar o segmento, fazer **cmd** = **IPC_RMID** e **buf** = **0** (a remoção efetiva ocorre após o último processo se desligar)

Comandos Linux para memória partilhada 💻

- Listar os segmentos de memória partilhada (**i**nter-**p**rocess **c**ommunication status –**m**emory):

```
ipcs -m
```

- Remover um segmento:

```
ipcrm -m id  # por ID do segmento

ipcrm -M key # por chave
```

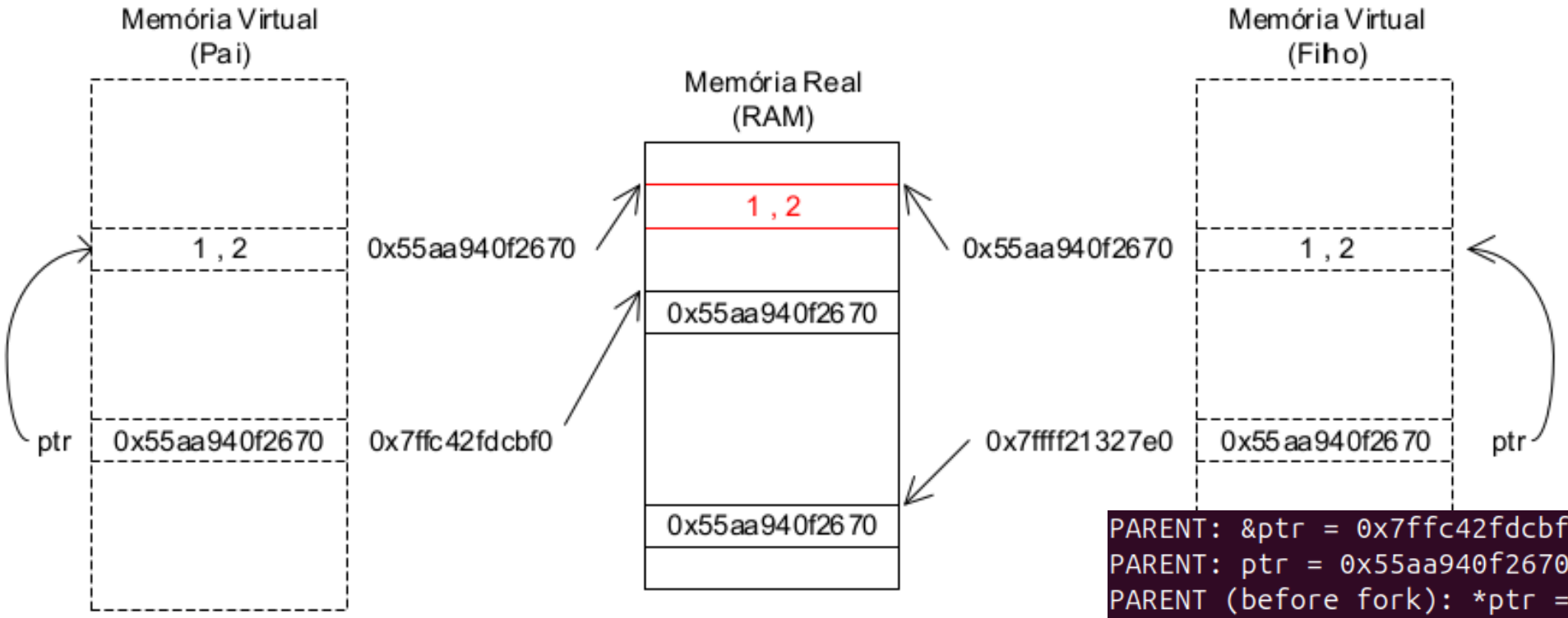
- Nota: a remoção só ocorre após o último desligamento do segmento

👉 Exemplo de memória partilhada

```c
// Program 2.2
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/shm.h>

int main(){
    int shmid, *ptr;
    printf("PARENT: &ptr = %p\n", &ptr); // show the virtual addresses of the pointer variable
    // create a private shared memory zone for one integer
    shmid = shmget(IPC_PRIVATE, sizeof(int), 00600);
    ptr = (int*)shmat(shmid, NULL, 0); // attach a virtual address (chosen by the OS) to the zone
    printf("PARENT: ptr = %p\n", ptr); // show the virtual address of the shared memory zone
    // write something in the shared memory zone and show it
    *ptr = 1; printf("PARENT (before fork): *ptr = %d\n", *ptr);

    if(fork()==0){ // create a CHILD that is going to access the shared memory zone
        printf("CHILD: &ptr = %p\n", &ptr); // show that the virtual address of the pointer is still the same
        printf("CHILD: ptr = %p\n", ptr);   // show that the virtual address of the zone is still the same
        // write something diferent in the shared memory zone and show it
        *ptr = 2; printf("CHILD (before exit): *ptr = %d\n", *ptr);
        shmdt(ptr); exit(0); // detach from the shared memory zone and exit
    }

    // wait for the CHILD to die and then show the shared memory content
    wait(NULL); printf("PARENT (after wait): *ptr = %d\n", *ptr);
    // detach from the shared memory zone, ask its removal and exit
    shmdt(ptr); shmctl(shmid, IPC_RMID, 0); return(0);
}
```

```
PARENT: &ptr = 0x7ffc42fdcbf0
PARENT: ptr = 0x55aa940f2670
PARENT (before fork): *ptr = 1
CHILD: &ptr = 0x7ffc42fdcbf0
CHILD: ptr = 0x55aa940f2670
CHILD (before exit): *ptr = 2
PARENT (after wait): *ptr = 2
```

Memória partilhada via primitivas. Resolvido! 😎




```
PARENT: &ptr = 0x7ffc42fdcbf0
PARENT: ptr = 0x55aa940f2670
PARENT (before fork): *ptr = 1
CHILD: &ptr = 0x7ffc42fdcbf0
CHILD: ptr = 0x55aa940f2670
CHILD (before exit): *ptr = 2
PARENT (after wait): *ptr = 2
```

Mapa de memória do processo pai e do processo filho – output do Programa 2.2

Exercícios

1. Codifique um programa que envolva dois processos – pai e filho – e recorra a memória partilhada para sincronização de forma a garantir o seguinte comportamento: a) o processo pai mostra o seu PID depois do filho mostrar o seu PID; b) o comportamento reverso de a).

2. Recorrendo a memória partilhada para sincronização, construa um programa no qual dois processos (pai e filho) contam alternadamente até 15 (mostrando a evolução do contador): o processo pai deverá contar de 1 a 3, dando de seguida a vez ao filho; este deverá contar de 4 a 6, passando a vez novamente ao pai; o pai contará de 7 a 9, dando de seguida a vez ao filho; este deverá contar de 10 a 12, passando a vez novamente ao pai, após o que pode terminar; finalmente, o pai deverá contar de 13 a 15, após o que termina.

Exercícios

3. Codifique um programa em que um processo filho envie inteiros (lidos do teclado) ao seu pai, este soma-lhes uma unidade e devolve-os ao filho, que apresenta o resultado no ecrã; após o envio do inteiro `0` ao pai, o filho não lhe deve enviar mais inteiros, devendo ambos terminar. Recorra exclusivamente a memória partilhada para troca de dados e sincronização.

4. (variante do exercício 3) Considere um processo pai e 2 filhos: o 1º filho criado lê um inteiro do teclado e envia-o ao seu irmão (2º filho criado); este soma uma unidade ao número recebido e envia-o ao pai; por seu turno, este soma uma unidade ao número recebido e envia-o ao 1º filho, que o mostra no ecrã; este comportamento repete-se até que o 1º filho leia o número `0` do teclado, situação em este circuito ainda deve ser feito, mas pela última vez. Recorra apenas a memória partilhada para troca de dados e sincronização.

Exercícios

5. Codifique um programa que se baseie em dois processos para o cálculo da soma dos primeiros 100 números inteiros positivos. O processo pai deve calcular a soma dos primeiros 50 números e o processo filho deve calcular a soma dos últimos 50 números. O processo pai apresentará a soma global. Para sincronização, explore os diversos mecanismos já estudados até este capítulo (incluído); para passagem de dados, utilize apenas memória partilhada.

6. (variante do exercício 5) Codifique um programa para calcular a soma dos primeiros 100 números positivos, à custa de somas parciais: um processo pai será responsável por criar 10 filhos; o primeiro filho calculará a soma parcial `1 + 2 + ... + 10`, o segundo filho calculará a soma `11 + 12 + ... + 20`, etc.; o processo pai calculará a soma final e apresentará o resultado. Para sincronização, explore os diversos mecanismos já estudados até este capítulo (incluído); para passagem de dados, utilize apenas memória partilhada.

Exercícios

7. Construa um programa para calcular a soma dos quadrados dos primeiros 10 números positivos, com base num processo pai e em 10 filhos, devendo o cálculo ser efetuado em Round-Robin (com um processo ativo de cada vez): depois de criar os 10 filhos, o pai passa a vez ao primeiro filho, o qual calcula e acumula o quadrado de 1; por sua vez o primeiro filho passa a vez ao segundo filho, o qual calcula e acumula o quadrado de 2, passa a vez ao terceiro filho, e assim sucessivamente ... até que o décimo filho, depois de calcular e acumular o quadrado de 10, passa a vez ao pai, que apresenta a soma final. Use exclusivamente memória partilhada para resolver o problema.

Exercícios

8. (variante do exercício 1.7.c) Escreva um programa onde um processo pai cria um array `A` de `N` inteiros em memória partilhada e preenche-o com números aleatórios. Caberá depois a `N` processos filhos verificar se cada célula do array é ímpar (um filho criado em n'ésimo lugar analisará a célula de índice n do array). Caberá ao pai apresentar o total de ímpares detetados. Resolva o problema para as seguintes situações: a) o array `A` é preenchido antes da criação dos filhos; b) o array `A` é preenchido depois da criação dos filhos. Na resolução deste problema use exclusivamente memória partilhada para troca de dados e para sincronização entre os vários processos.

Exercícios

9. (variante do exercício 1.11) Escreva um programa que implemente o seguinte cenário: um processo pai cria 10 processos filhos; antes de terminar, cada filho deve mostrar o seu PID; no entanto, primeiro devem ser os filhos de PID par a mostrar o seu PID, e só depois os filhos de PID ímpar; o pai só deve terminar depois de todos os filhos terem mostrado o seu PID. Resolva o problema usando memória partilhada para sincronizar os processos.