Fundamentos de Sistemas de Operação

Unix Windows NT Netware MacOS DOS/VS Vax/VMS
Linux Solaris HP/UX AIX Mach
Chorus

Programas, Processos e o SO:

Comunicação entre processos: II

Resumo da aula anterior…

Unix Windows NT Netware MacOS DOS/VS Vax/VMS Linux Solaris HP/UX AIX Mach Chorus

- *O processo, conceito oferecido pelo SO,*
  - *"É um contentor" que tem um CPU, uma memória e periféricos idênticos aos da "máquina real". Cada programa corre nesta máquina virtual como se mais nada existisse…*
  - *Mas um processo tem de comunicar - ler e escrever em periféricos, em ficheiros e com outros processos - por exemplo, usando pipes.*
  - *A comunicação entre processos usando pipes é um caso particular do paradigma de comunicação por troca de mensagens:*
    - *Os processos comunicam enviando e recebendo mensagens;*
    - *Tais operações implicam cópia de dados entre os espaço endereçamento dos processos, usando um meio exterior ao processo, e.g., pipe;*
    - *As operações de troca podem promover a sincronização dos processos.*

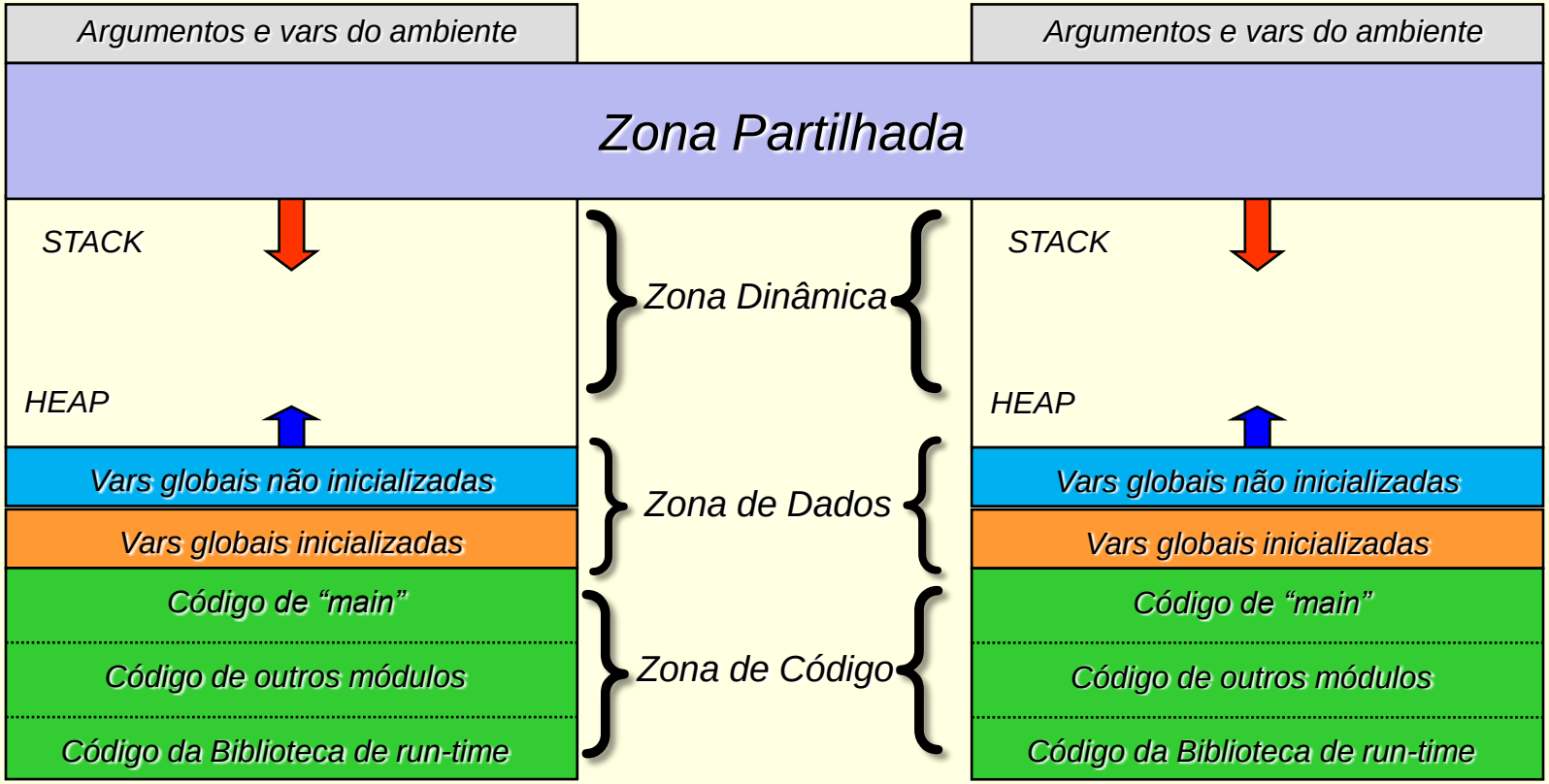
Comunicação por Memória Partilhada (1)

Unix Windows NT Netware MacOS DOS/VS Vax/VMS Linux Solaris HP/UX AIX Mach Chorus

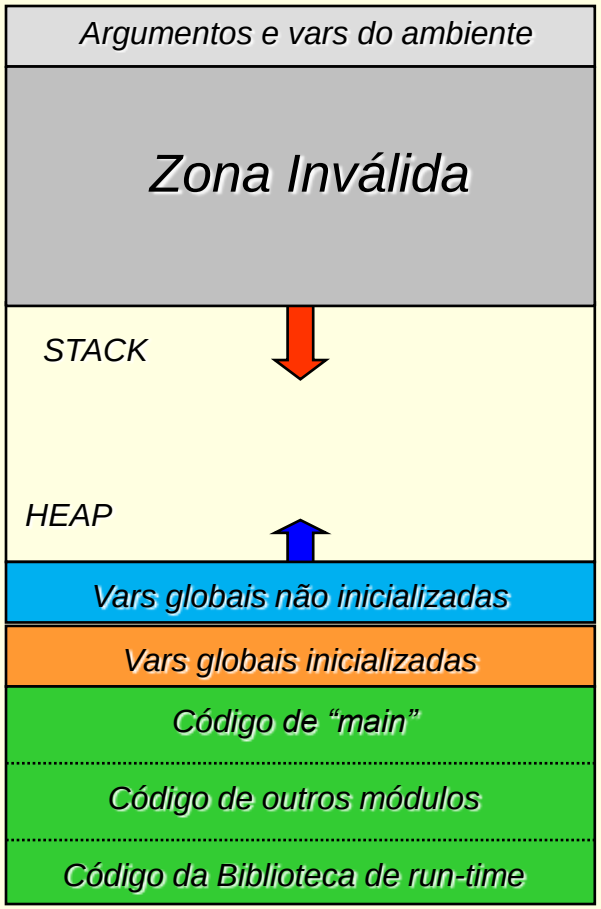
- *Processos partilham uma mesma zona de memória*
  - *Uma zona de memória (partilhada) é projectada ("mapped") no espaço de endereçamento (EE) dos processos [Nota: os EE não têm de ser idênticos, i.e., os programas podem ser diferentes];*
  - *Os "**objectos**" (e.g., dados) definidos na zona "**existem**" simultaneamente nos dois EE, podendo ser acedidos pelos processos sem operações de cópia entre eles*
  - *Nos acessos, não há garantia de sincronização, e.g.:*
    - *Como sabe o Processo B que o A já mudou o valor de uma variável?*
    - *Que acontece se A aceder a uma estrutura de dados que está a ser alterada "no momento" por B?*

Comunicação por Memória Partilhada (2)

- *Mapa dos Espaços de Endereçamento (hipotético)*



Memória Partilhada no Unix (1)



- *Criação de uma zona ("segmento") de Memória Partilhada*

```
int shmget(key, size, flags)
```

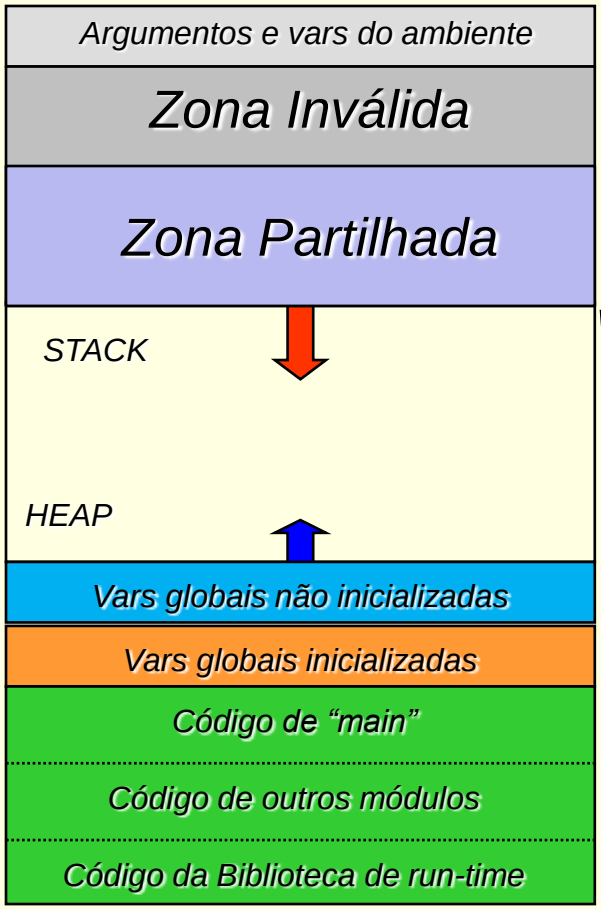
e.g.,

```
shmID= shmget(123, 100, ...)
```

Zona Partilhada

O identificador retornado em **`shmID`** *vai ser usado para identificar a zona de MP noutras funções…*

Memória Partilhada no Unix (2)

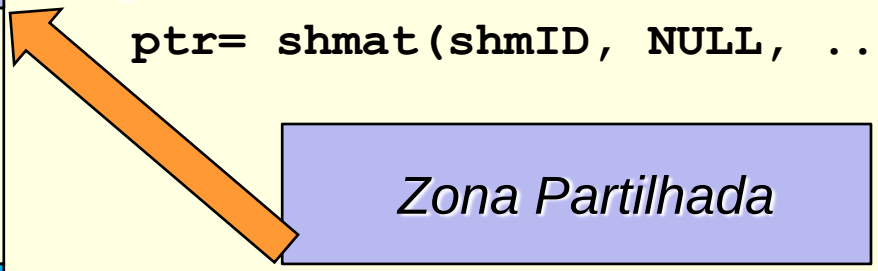


- *Projecção de uma zona de Memória Partilhada no EE de um processo*

```
void *shmat(id, addr, flags)
```

e.g.,

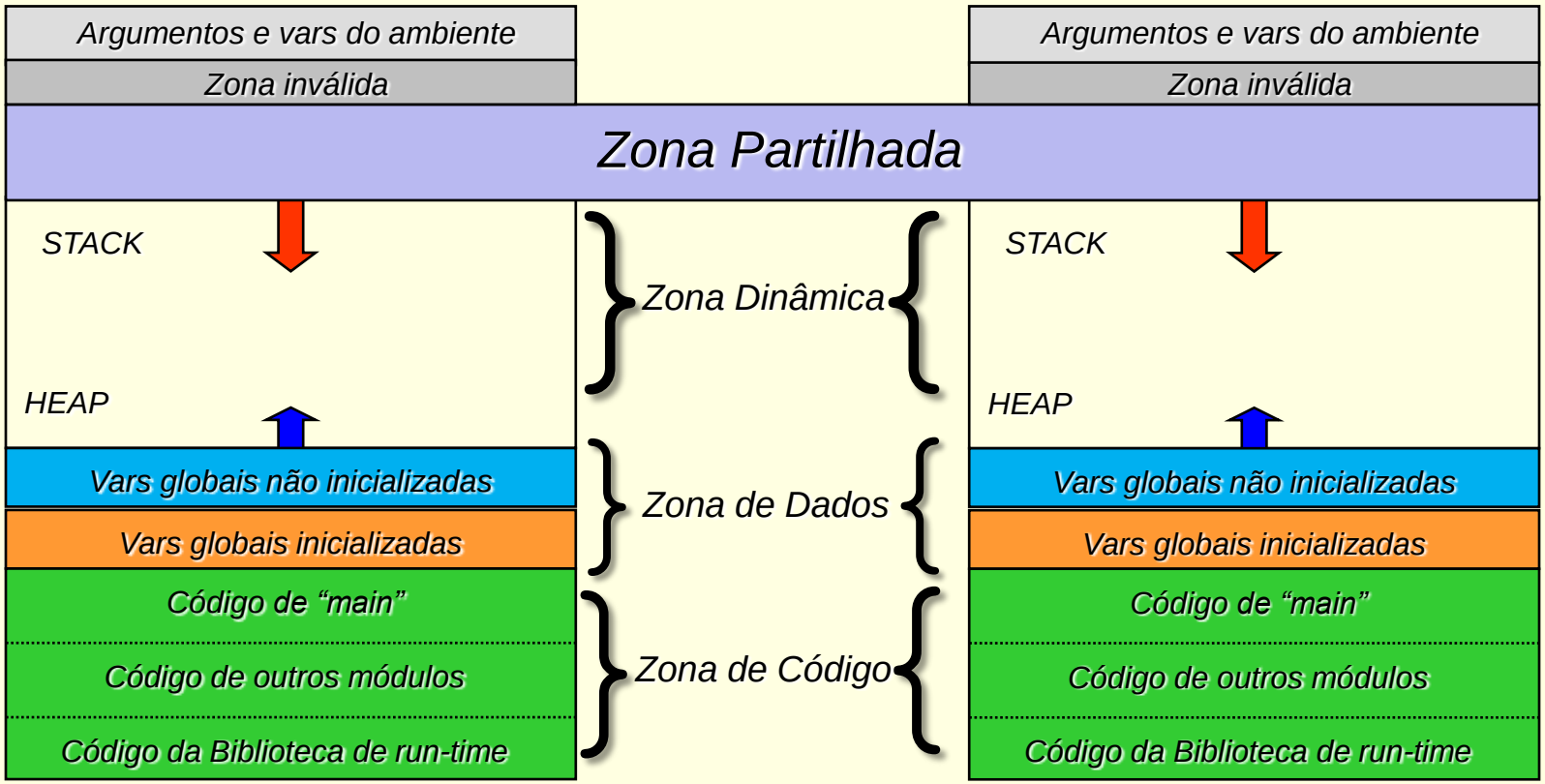
```
ptr= shmat(shmID, NULL, ...)
```



Onde **`ptr`** *é uma variável do programa que fica a apontar para o início da zona de MP; se dois processos executam um* **`shmat()`**, *a situação é a exibida no slide seguinte…*

Memória Partilhada no Unix (3)

- Mapa *(hipotético)* resultante de dois **`shmat()`**



"Criar" um segmento de MP

- **`int shmget(int key, int size, int flags)`**
  - *Retorna o identificador do segmento de MP identificado pela chave única* **`key`**.
    - *Se o segmento não existe e as flags incluem a opção* **`IPC_CREAT`**, *o segmento é criado; se tal opção não é incluída, a chamada falha.*
    - *Se o segmento já existe e as flags incluem não só a opção* **`IPC_CREAT`** *mas também a opção* **`IPC_EXCL`**, *a chamada falha.*
  - *Nota 1: Quando se cria um segmento, deve indicar-se o conjunto de permissões ("modo"); caso contrário não se consegue aceder ao segmento.*
  - *Nota 2: A dimensão* **`size`** *é arredondada para cima ao tamanho de uma página.*

Projectar um segmento de MP no EE

- **`void *shmat(int id, void *addr, int flags)`**
  - *Projecta ("mapeia") o segmento de MP identificado por* **`id`** *no EE do processo e retorna um apontador para o início da região.*
    - *O valor* **`addr`** *permite, em conjunto com o valor de* **`flags`**, *controlar em que zona do EE queremos "mapear" o segmento; usaremos* **`NULL`** *para* **`addr`** *e* **`0`** *para* **`flags`**, *e deixaremos o controle ao Unix.*

Usando MP... experimente!

Programa A

```
#define flags IPC_CREAT|0666

int main()
{ char *pA; int iD;

  iD= shmget(123, 10, flags);

  pA= shmat(iD, NULL, 0);

  *pA= 'A';

  printf("Valor: %c\n", *pA);

  return 0;
}
```

Programa B

```
#define flags 0

int main()
{ char *pB; int iD;

  iD= shmget(123, 10, flags);

  pB= shmat(iD, NULL, 0);

  *pB= 'B';

  printf("Valor: %c\n", *pB);

  return 0;
}
```

Para pensar…

- *Qual o resultado da execução dos programas (slide 10)?*
  - *O Programa A imprime "A" e o Programa B imprime "B"*
  - *O Programa A imprime "B" e o Programa B imprime "A"*
  - *Os dois imprimem "A"*
  - *Os dois imprimem "B"*
  - *Outro resultado…*
- *Explique!*

Dissociar um segmento de MP do EE

- **`int shmdt(void *addr)`**
  - *Dissocia o segmento de MP apontado por* **`addr`** *do EE do processo.*
    - *O apontador tem de apontar exactamente para o local obtido com o* **`shmat()`** *original.*
    - *Se o processo terminar sem ter efectuado esta operação, o Unix fá-la.*

Apagar um segmento de MP

- **`int shmctl(int id, int cmd, struct shmid_ds *buf)`**
  - *Permite efectuar diversas operações sobre o segmento de MP identificado por* **`id`***; apenas estudaremos a remoção do segmento.*
    - *O* **`cmd`** *tem o valor* **`IPC_RMID`**.
    - *O valor de* **`buf`** *é* **`NULL`**.
- *Se se esquecer de remover o segmento na aplicação (ou esta falhar sem executar a chamada), pode usar o comando externo* **`ipcrm`**.

Mais um exemplo...

Programa A

Programa B

```
int saldo= 1000;
```

```
int deposita(int v)
{ int t;
  return(saldo + v);
}

int main() {
  ...
  saldo= deposita(100);
}
```

```
int levanta(int v)
{ int t;
  return(saldo - v);
}

int main() {
  ...
  saldo= levanta(100);
}
```

- *Suponha que os programas (main) executam uma única vez as rotinas deposita (em A) e levanta (em B)…*
  - *Quanto fica em saldo?*

Para pensar…

- *Como sincronizar a execução dos programas A e B (slide anterior) de forma que o resultado seja o correcto?*