Sistemas Operacionais

Gestão de memória - Conceitos

Prof. Carlos Maziero

DInf UFPR, Curitiba PR

Abril de 2019

Conteúdo

1 Tipos de memória

2 A memória de um processo

3 Alocação de variáveis

4 Atribuição de endereços

Tipos de memória

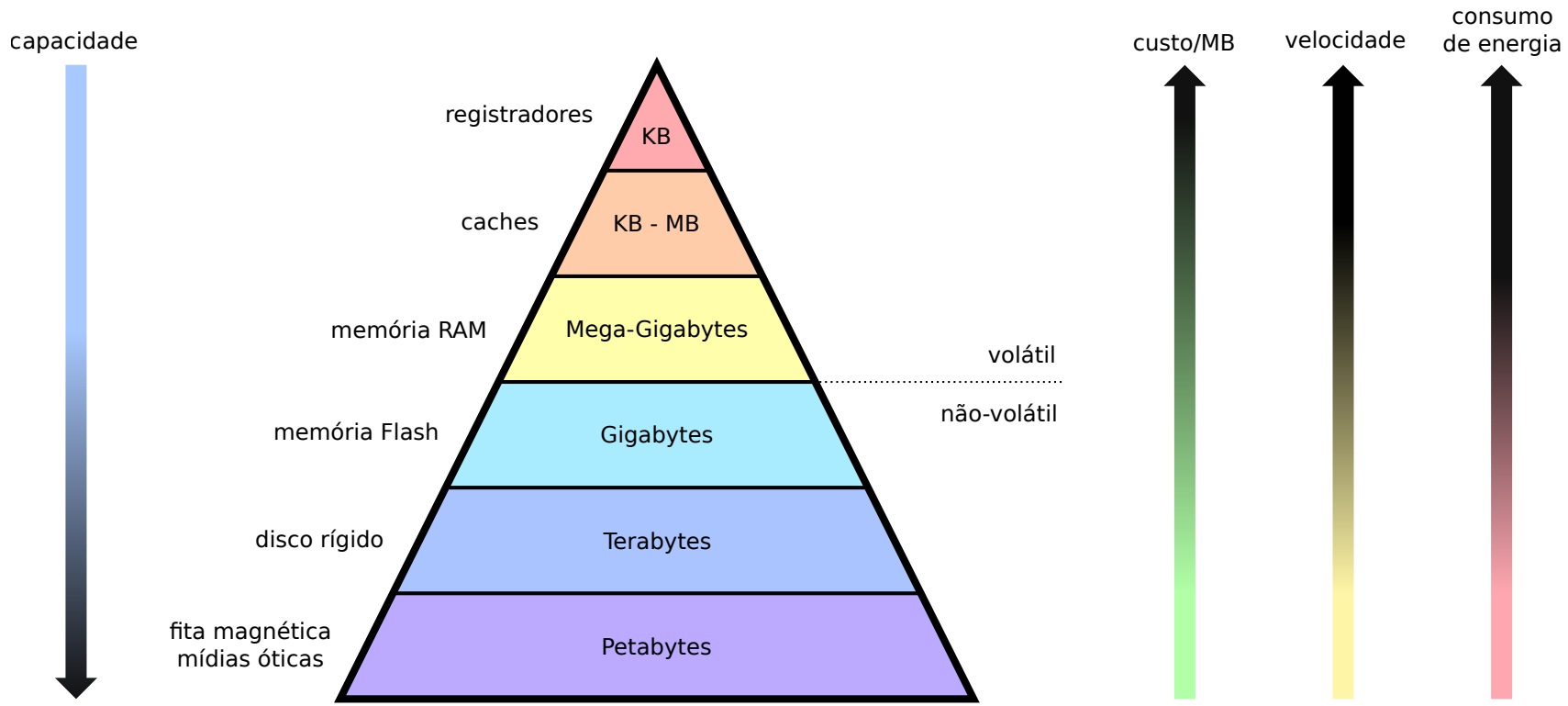
Dispositivos:

- registradores, caches
- RAM
- SSD, HD
- pendrive
- CD, DVD
- fita magnética

Características:

- capacidade
- velocidade
- latência
- custo
- consumo
- volatilidade

Hierarquia de memória



Capacidades distintas

Meio	**Tempo de acesso**	**Taxa de transferência**
Cache L2	1 ns	1 GB/s (1 ns/byte)
Memória RAM	60 ns	1 GB/s (1 ns/byte)
Memória *flash* (NAND)	2 ms	10 MB/s (100 ns/byte)
Disco rígido SATA	5 ms (desloc. da cabeça de leitura e rotação do disco)	100 MB/s (10 ns/byte)
DVD-ROM	de 100 ms a vários minutos (gaveta aberta ou leitor sem disco)	10 MB/s (100 ns/byte)

A memória de um processo

A memória de cada processo é organizada em várias áreas:

TEXT: **código** binário (executável)

DATA: variáveis globais/estáticas inicializadas

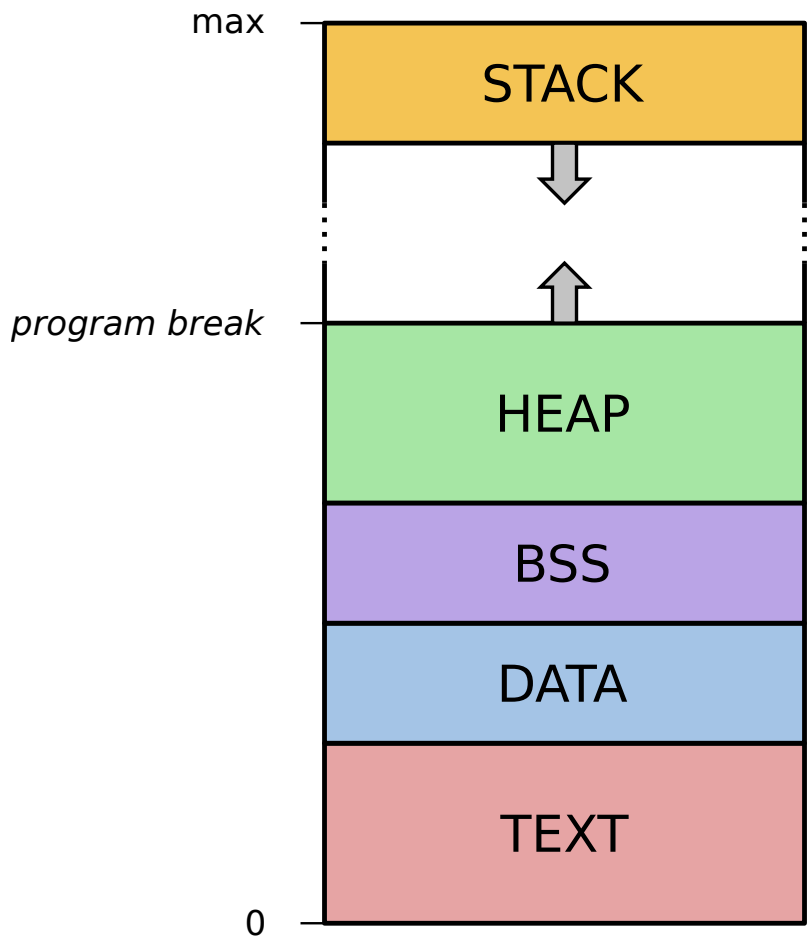
BSS: *Block Started by Symbol*, variáveis globais não inicializadas

HEAP: **dados alocados dinamicamente** (`malloc`); tem tamanho variável (ponteiro *program break*).

STACK: **pilha** de execução; tem tamanho variável e cresce "para baixo"

Outras áreas: bibliotecas dinâmicas, pilhas das *threads*, etc

Áreas de memória



dados usados por funções (parâmetros, variáveis locais, endereços de retorno)

área livre

variáveis dinâmicas (malloc/free)

variáveis não-inicializadas

variáveis inicializadas

código binário do programa

Áreas de memória

Áreas de memória de um *Hello, World!*

```
# pmap -x 27505

27505:   /usr/bin/hello

Address           Kbytes  Mode   Mapping

0000000000400000     808  r-x--  /usr/bin/hello  (TEXT)
00000000006c9000      12  rw---  /usr/bin/hello  (DATA)
00000000006cc000       8  rw---  [ anon ]        (BSS)
000000000092e000     140  rw---  [ anon ]        (HEAP)
00007ffe6a5df000     132  rw---  [ stack ]       (STACK)
```

Alocação de variáveis

Alocar: Reservar espaço na memória

`long counter` : reservar 8 bytes na RAM

Várias formas de alocação:

- Alocação **estática**
- Alocação **automática**
- Alocação **dinâmica**

Alocação estática

Espaço é reservado pelo compilador:

Para variáveis usadas durante toda a execução:

- Variáveis globais
- Variáveis locais estáticas (`static int`)

Alocação pode ser feita em áreas distintas:

- Em `DATA` para variáveis inicializadas
- Em `BSS` para variáveis não-inicializadas

Alocação estática

```
1  #include <stdio.h>
2
3  int soma = 0 ;
4
5  int main ()
6  {
7    int i ;
8
9    for (i=0; i<1000; i++)
10     soma += i ;
11   printf ("Soma de inteiros até 1000: %d\n", soma) ;
12
13   return (0) ;
14 }
```

Alocação automática

Para variáveis usadas em funções e procedimentos:

- Variáveis locais
- Parâmetros de entrada
- Valor de retorno

Alocação é feita na pilha (área `STACK`):

- Áreas alocadas ao invocar a função
- Áreas liberadas ao concluir a função
- Conveniente para chamadas recursivas

Alocação automática

```
1  #include <stdio.h>
2
3  long int fatorial (int n)
4  {
5    long int parcial ;
6
7    printf ("inicio: n = %d\n", n) ;
8    if (n < 2)
9      parcial = 1 ;
10   else
11     parcial = n * fatorial (n - 1) ;
12   printf ("final : n = %d, parcial = %ld\n", n, parcial) ;
13   return (parcial) ;
14 }
15
16 int main ()
17 {
18   printf ("Fatorial (4) = %ld\n", fatorial (4)) ;
19   return 0 ;
20 }
```

Alocação automática

```
1  inicio: n = 4                          // chamada inicial
2  inicio: n = 3                          // chamadas recursivas
3  inicio: n = 2                          // ...
4  inicio: n = 1
5
6  final : n = 1, parcial = 1             // retorno recursivo
7  final : n = 2, parcial = 2             // ...
8  final : n = 3, parcial = 6
9  final : n = 4, parcial = 24
10
11 Fatorial (4) = 24                      // retorno inicial
```

Alocação dinâmica

Requisição explícita pelo processo:

- `malloc(...)` para obter um bloco de N bytes
- `free(...)` para liberar um bloco
- Operador `new()` em linguagens a objetos
- *Garbage collectors*

Usa a área `HEAP`

- Delimitada pelo ponteiro *Program break* (BRK)
- Tamanho da área pode ser ajustado pelo SO

Alocação dinâmica

```
1  char * prt ;                    // ponteiro para caracteres
2
3  ptr = malloc (4096) ;           // solicita um bloco de 4.096 bytes;
4                                  // ptr aponta para o início do bloco
5
6  if (ptr == NULL)                // se ptr for nulo, ocorreu um erro
7    abort () ;                    // e a área não foi alocada
8
9  ...                             // usa ptr para acessar o bloco alocado
10
11 free (ptr) ;                    // libera o bloco alocado na linha 3
```

```
1  Rectangle rect1 = new Rectangle (10, 30) ;
2  Rectangle rect2 = new Rectangle (3, 2) ;
3  Triangle tr1  = new Triangle (3, 4, 5) ;
```

Atribuição de endereços

Como definir os endereços das variáveis?

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  int main ()
5  {
6     int i, soma = 0 ;
7
8     for (i=0; i< 5; i++)
9     {
10       soma += i ;
11       printf ("i vale %d e soma vale %d\n", i, soma) ;
12    }
13    exit(0) ;
14 }
```

Símbolos `soma`, `i`, `main`, `printf` e `exit` representam endereços

Programa compilado

Símbolos são convertidos em endereços:

```
1  0000000000000000 <main>:
2    0:   55                          push  %rbp
3    1:   48 89 e5                    mov   %rsp,%rbp
4    4:   48 83 ec 10                 sub   $0x10,%rsp
5    8:   c7 45 fc 00 00 00 00        movl  $0x0,-0x4(%rbp)
6    f:   eb 2f                       jmp   40 <main+0x40>
7   11:   8b 15 00 00 00 00           mov   0x0(%rip),%edx
8   17:   8b 45 fc                    mov   -0x4(%rbp),%eax
9   1a:   01 d0                       add   %edx,%eax
10  1c:   89 05 00 00 00 00           mov   %eax,0x0(%rip)
11  22:   8b 15 00 00 00 00           mov   0x0(%rip),%edx
12  28:   8b 45 fc                    mov   -0x4(%rbp),%eax
13  2b:   89 c6                       mov   %eax,%esi
14  2d:   bf 00 00 00 00              mov   $0x0,%edi
15  32:   b8 00 00 00 00              mov   $0x0,%eax
16  37:   e8 00 00 00 00              callq 3c <main+0x3c>
17  ...
```

Atribuição de endereços

A tradução [*símbolo* → *endereço*] pode ocorrer:

Na edição: o programador define os endereços.

Na compilação: o compilador define os endereços.

Na ligação: o compilador define símbolos sem endereços; o ligador define os endereços ao construir o executável.

Na carga: o *carregador* carrega o código do processo na memória e define os endereços.

Na execução: endereços acessados pelo processador são convertidos nos endereços efetivos em RAM.

Atribuição de endereços