

# Sistemas Operacionais

Aula 10 – Gestão de Memória – Uso da memória

Prof. Igor da Penha Natal

# Conteúdo

- 1 Organização do espaço de endereços
- 2 A memória de um processo
- 3 Alocação de variáveis
- 4 Atribuição de endereços

# Espaço de endereçamento

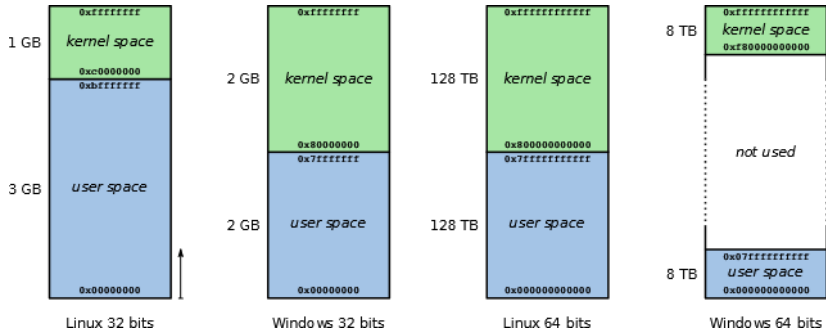
Na maioria dos sistemas operacionais atuais:

- O espaço de endereçamento usa endereços de 32 ou 64 bits
- O início do espaço de endereçamento é do processo
- O final do espaço de endereçamento é do núcleo

O núcleo está no espaço de endereçamento do processo:

- Páginas do núcleo são protegidas por flags da *page table*
- Endereços de páginas frequentes do núcleo ficam na TLB

# Espaço de endereçamento



Layout sendo revisado devido ao bug *Meltdown* (2018)

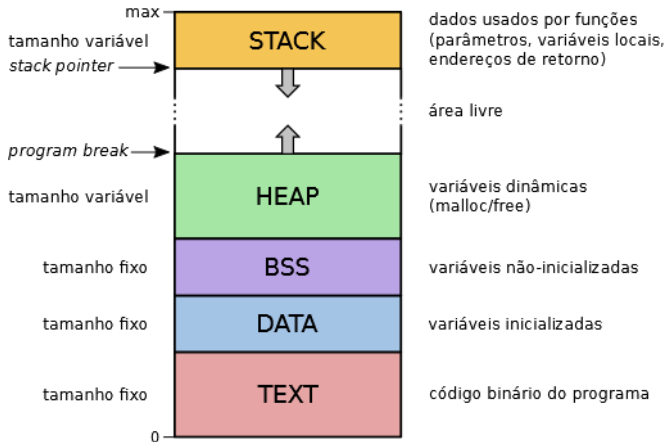
No Linux: *Kernel Page-Table Isolation* (KPTI)

# 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



# Áreas de memória

Áreas de memória de um processo simples (*Hello, World!*)

1	# pmap 27505	# 27505: PID do processo		
2				
3	Address	Kbytes	Mode	Mapping
4				
5	00000000000400000	808	r-x--	/usr/bin/hello (TEXT)
6	000000000006c9000	12	rw---	/usr/bin/hello (DATA)
7	000000000006cc000	8	rw---	[ anon ] (BSS)
8	0000000000092e000	140	rw---	[ anon ] (HEAP)
9	00007ffe6a5df000	132	rw---	[ stack ] (STACK)

ASLR – *Address Space Layout Randomization*

# Alocação de variáveis

Declaração de uma variável:

- Reserva (aloca) um espaço na memória
- Associa uma referência (nome) a esse espaço

1	long	counter ;	// aloca 8 bytes
2	char	c[100] ;	// aloca 100 bytes

Várias formas de alocação:

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



# Alocação estática

Espaço da variável 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 ; // alocação estática
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/procedimentos/métodos:

- 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 fatorial (int n) // alocação automática
4  {
5      long parcial ;    // alocação automática
6
7      printf ("inicio: n = %d\n", n) ;
8
9      if (n < 2)
10         parcial = 1 ;
11     else
12         parcial = n * fatorial (n - 1) ;
13
14     printf ("final : n = %d, parcial = %ld\n", n, parcial) ;
15     return (parcial) ;
16 }
17
18 int main ()
19 {
20     printf ("Fatorial (4) = %ld\n", fatorial (4)) ;
21 }
  
```

# 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 da recursão
7  final : n = 2, parcial = 2    // ...
8  final : n = 3, parcial = 6
9  final : n = 4, parcial = 24
10
11 Fatorial (4) = 24            // retorno final
  
```

# Alocação dinâmica

Requisição explícita pelo processo:

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

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

Símbolos soma, i, main e printf representam **endereços**



# Programa compilado

Ao compilar, os símbolos são convertidos em endereços:

```

1  $ gcc -Wall -c -O0 soma.c ; objdump -d soma.o
2
3  0000000000000000 <main>:
4      0:  55                      push  %rbp
5      1:  48 89 e5                mov   %rsp,%rbp
6      4:  48 83 ec 10             sub   $0x10,%rsp
7      8:  c7 45 fc 00 00 00 00    movl  $0x0,-0x4(%rbp)
8      f:  eb 2f                  jmp   40 <main+0x40>
9      11:  8b 15 00 00 00 00       mov   0x0(%rip),%edx
10     17:  8b 45 fc                mov   -0x4(%rbp),%eax
11     1a:  01 d0                  add   %edx,%eax
12     1c:  89 05 00 00 00 00       mov   %eax,0x0(%rip)
13     22:  8b 15 00 00 00 00       mov   0x0(%rip),%edx
14     28:  8b 45 fc                mov   -0x4(%rbp),%eax
15     2b:  89 c6                  mov   %eax,%esi
16     ...

```

# Atribuição de endereços

A tradução [*símbolo*  $\rightarrow$  *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

