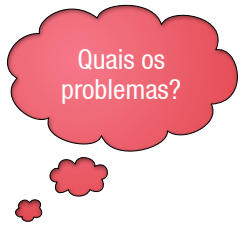
Memória

Performance em Sistemas Ciberfísicos

Memória

Performance em Sistemas Ciberfísicos



MENTIMETER

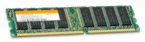
```java
class A {
    private int m = 123;
    public void imprime() { System.out.println(m); }
}

public static void main(String[] args) {

    int[] v = new int[100];
    for (int i = 0; i <= 100; ++i)
        System.out.println(i + " -> " + v[i]);

    A a = null;
    a.imprime();

    f(100, v);
    System.out.println("FIM");
}
```

```java
public static void f(int n, int [] v) {
    if (n > 0) {
        int[] v2 = new int[v.length * 2];
        for (int i = 0; i < 2 * n; ++i)
            v2[i] = v[i/2];
        f(n-1,v2);
    }
}
```

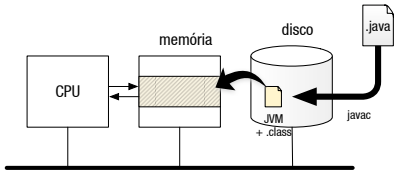
2

Introdução

Memória



- Memória:
  - um dos mais importantes recursos do computador ➔ requer cuidados especiais.
  - O processo precisa estar na memória para ser executado:
    - "arquitetura von Neumann"



- A quantidade de memória exigida pelos processos tem crescido rapidamente:
  - década de 80:
    - universidades usavam sistema de tempo compartilhado com 4 MB
  - hoje:
    - Microsoft Office: 4GB (recomendado, 64 bit)

3

Introdução

Memória

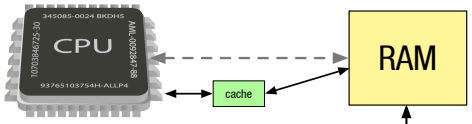
- Memórias têm **capacidades**:
  - expressas em *número de bytes*
- **"Palavras"**:
  - 8, 16 ou 32 bits
- **Endereços** de memória:
  - tamanho dependente do número de palavras armazenadas
  - 2^n palavras → n bits
- **Acesso**:
  - aleatório (qualquer posição diretamente)
- **Desempenho**:
  - taxa de transferência:
    - e.g., bytes por segundo

4

Hierarquia de Memórias

Performance em Sistemas Ciberfísicos

- Para desempenho:
  - memória deve acompanhar desempenho da CPU
  - *"trade-off"*: capacidade, taxa de transferência, custo
- "Hierarquia" de memórias:
  - grande quantidade de memória principal (RAM)
    - volátil, velocidade e custo médios
  - pequena quantidade de memória cache ⟹ gerenciamento ao nível de HW
    - volátil, muito rápida, cara (ex: i7: 12MB de cache)
  - memória secundária (e.g., discos, memória *flash*, *solid-state drives*)
    - não volátil, dezenas de centenas de GB, velocidade e custo baixos
- O Sistema Operacional deve:
  - Coordenar a utilização destas memórias
  - Abstrair esta hierarquia em um modelo útil



RAM

cache

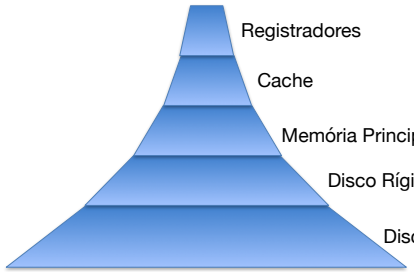
Mem. Secundária

localidade de referência

CPU é ordens de magnitude mais rápida que memória RAM!

Hierarquia de memória

Memória



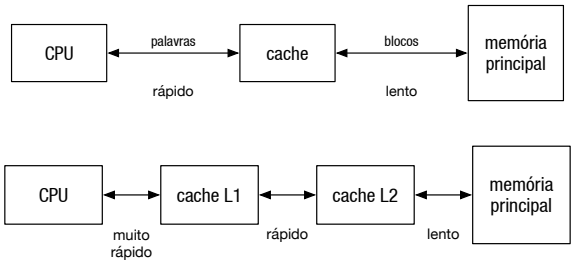
Registradores	desconsiderado	Maior custo Maior banda Menor latência
Cache	~1 ns latência (~25.000.000MB/s)	
Memória Principal	~12.5 ns latência (~25.600MB/s)	Menor custo Menor banda Maior latência Maior armazenamento
Disco Rígido	~7000000 ns latência (~70MB/s)	
Disco Ótico, Pendrive, ...		

Arquitetura da Memória Cache

A Memória Cache

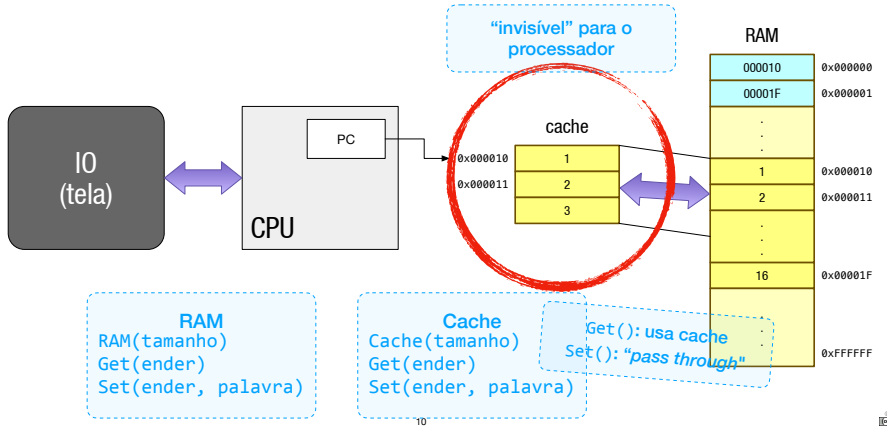
Memória

- Contém cópia de porções da memória principal.

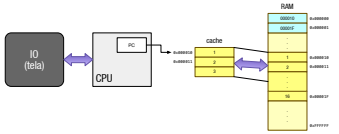


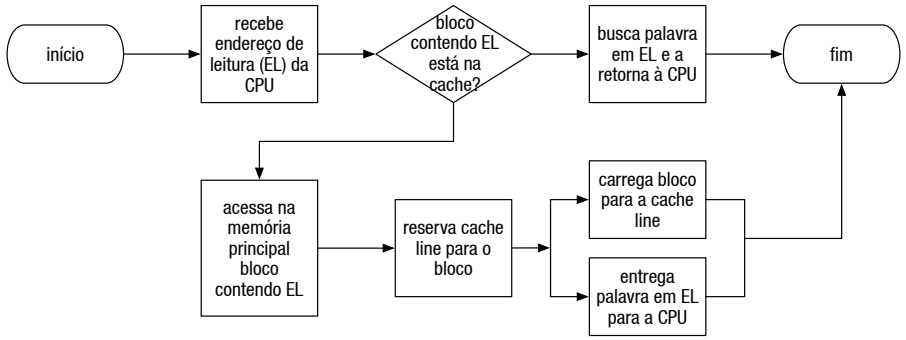
Exercício Prático
(contabilização de presenças!)

Implementação de Cache “básica”
Memória



Operação de Leitura
Memória

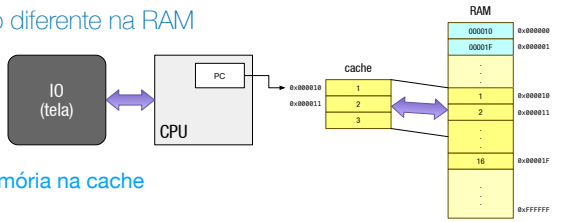




paralelo

Problemas com “cache básica”
Cache

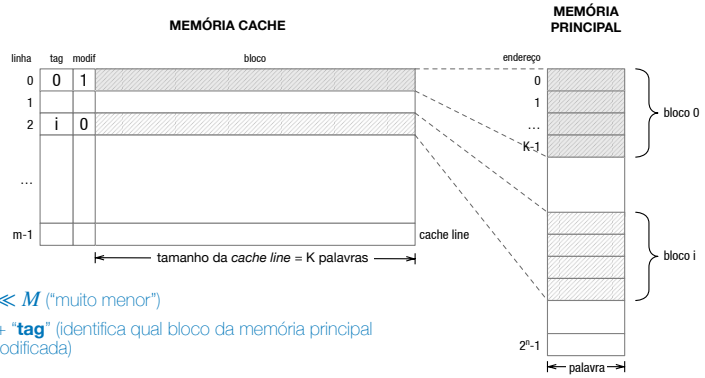
- “multiprogramação"
  - vários programas rodando “ao mesmo tempo”
  - cada um ocupando um espaço diferente na RAM
    - muitos “CACHE MISSES!”
- solução:
  - cache-lines:
    - várias pequenas porções da memória na cache



Memória Cache

Memória

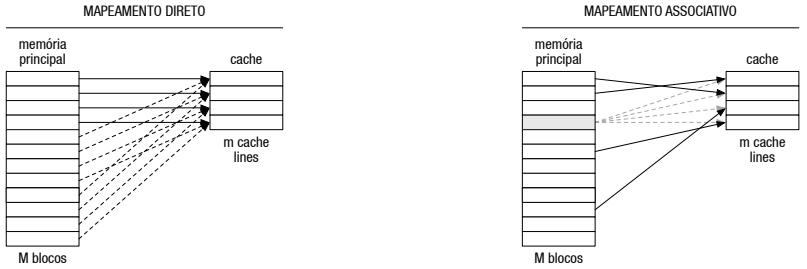
- Memória principal:
  - 2^n palavras endereçáveis
  - endereços: **n** bits
  - **M** blocos de **K** palavras
    - $M = \frac{2^n}{K}$
- Cache:
  - **m** blocos ("*cache lines*"): $m \ll M$ ("muito menor")
  - cada *cache line*: **K** palavras + "**tag**" (identifica qual bloco da memória principal está na *cache line*) (+ **flag** modificada)
  - tamanho da cache = $m \cdot K$
- A cada instante, **m** blocos da memória principal habitam as **m** cache lines.



Função de Mapeamento

Memória Cache

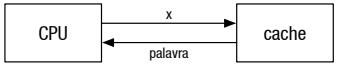
- Uma vez que m < M:
  - necessidade de mapeamento: blocos de memória principal → cache lines



Mapeamento Direto

Memória Cache

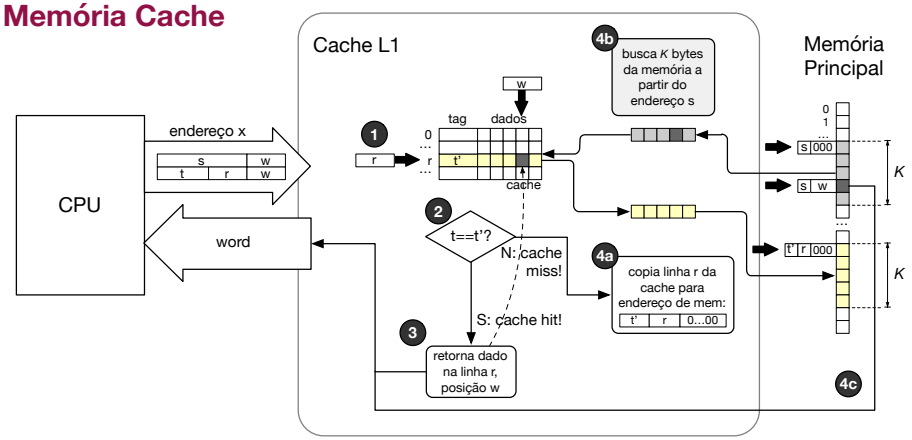
- CPU solicita acesso ao endereço **x** de memória:
- **x** dividido em bits:
  - **w**: uma das palavras da cache line
    - se cache line = 64 palavras, w possui 6 bits
  - **r**: índice da cache line
    - se cache possui 128 cache lines, r possui 7 bits
  - **t**: tag formada dos bits restantes de x
    (identifica qual bloco está atualmente na cache line)
    - se x é de 24 bits, t possui 11 bits
  - **s**: número do bloco da mem. principal
    (concatenação de **t** e **r**)
    - 18 bits, no exemplo



endereço x: 24bits (16M)		
s		w
t	**r**	**w**
tag	# linha	word
0-2047	0-127	0-63
(restante)	m	K
11 bits	7 bits	6 bits

Mapeamento Direto

Memória Cache



Exercício

Exercício

Mapeamento Direto

DADOS

K (palavras na cache line) = 64
Tamanho da cache = 8K
Tamanho da memória principal = 16M

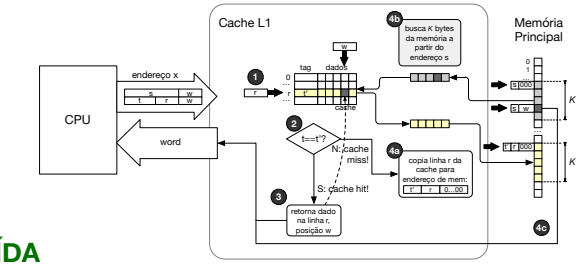
ENTRADA

Endereço x

x = 10560325

1 manual-mente

2 programa Java



SAÍDA

1. Endereços de início e fim do bloco de x na memória principal (s)
2. Índice da *cache line* de x (r)
3. *Tag* da *cache line* de x (t)
4. Posição de x na *cache line (w)*

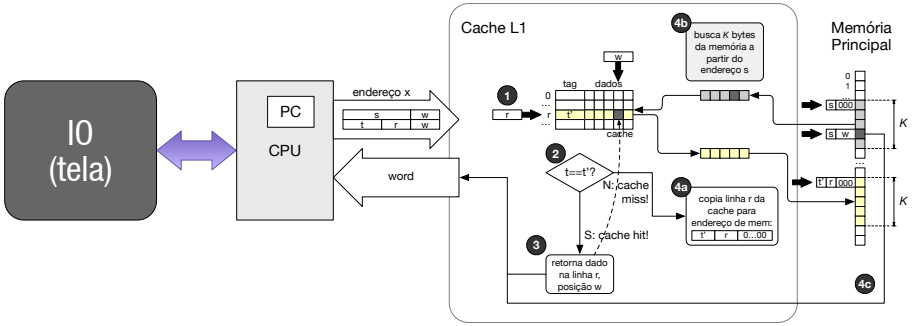
endereço x: 24bits (16M)		
s		w
t	r	w
tag	# linha	word
0-2047	0-127	0-63
(restante)	m	K
11 bits	7 bits	6 bits

Trabalho 1 (Blackboard)

Cache com Mapeamento Direto na Arquitetura von Neumann

Implementação de Cache com Mapeamento Direto

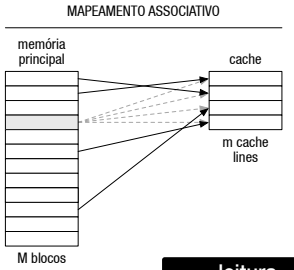
Arquitetura von Neumann



Mapeamento Associativo

Memória cache

- Blocos de RAM podem ser carregados em qualquer cache line.
- "tags": identificam bloco da RAM
- Para identificar se bloco está na cache:
  - controle lógico examina simultaneamente tags de todas as cache lines.
- Desvantagem:
  - circuitaria complexa (p/ analisar em paralelo)!
- Normalmente adotado:
  - map. direto + associativo = **Mapeamento Associativo por Conjunto**



leitura recomendada

Gerência de Memória pelo Sistema Operacional

Memória - Experimentos

Performance em Sistemas Ciberfísicos

pos + endereço do erro? → bss, pilha

- E se extrapolarmos o vetor?
  - C/C++:

```
const int VECSIZE = 10;
const int LOOPSIZE = 10000;

int v[VECSIZE];

for (int i=0; i<LOOPSIZE; ++i)
    cout << i << "\t" << v[i] << endl;
```

C++

Java

```
int VECSIZE = 10;
int LOOPSIZE = 10000;

int [] v = new int[VECSIZE];

for (int i=0; i<LOOPSIZE; ++i)
    System.out.println(i + "\t" + v[i]);
```

Experimento: Blackboard

Introdução (cont.)

Performance em Sistemas Ciberfísicos

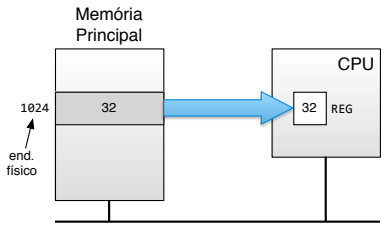
- O papel do sistema operacional no gerenciamento da abstração da hierarquia de memórias:
  - controlar quais partes da memória que estão em uso e quais estão livres;
  - reservar memória para processos;
  - liberar memória de processos;
  - tratar do problema de *swapping* memória principal-disco (quando há falta de espaço de memória);
  - lidar com memória virtual.

Gerência de Memória SEM abstração

Gerência de Memória SEM ABSTRAÇÃO

Performance em Sistemas Ciberfísicos

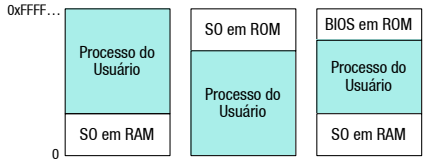
- Cada programa considera a memória física tal qual ela de fato é:
  - `MOV REG, [1024]`
    - Carrega o registrador REG com o conteúdo da memória física na posição 1024.
- Sistemas monoprogramados geralmente não fazem abstração de memória.



Memória em Sistemas Monoprogramados

Performance em Sistemas Ciberfísicos

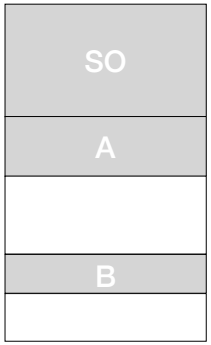
- Um único processo na memória a cada instante (além do próprio SO):
  - Computadores de grande porte e minicomputadores (obsoleto).
  - Computadores portáteis, alguns *smart phones* e sistemas embarcados.
  - PCs (exemplo: MS-DOS);
    - ROM: BIOS (Basic Input Output System)
    - [Atualmente: UEFI ➔ www.uefi.org]



Memória em Sistemas Multiprogramados

Memória (SO)

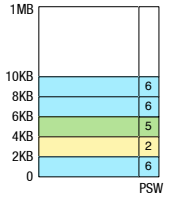
- Multiprogramação =
  - vários programas rodando “ao mesmo tempo”
- Como fica a memória?
  - opção 1:
    - manter na memória 1 programa por vez (alternando com o dispositivo de armazenamento)
    - inviável!
  - opção 2:
    - manter todos os programas que estão rodando na memória*



Problemas para a memória com a multiprogramação

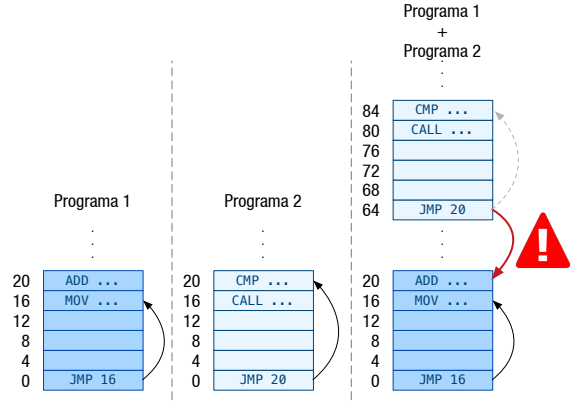
Performance em Sistemas Ciberfísicos

- Proteção de memória é essencial em sistemas multiprogramados.
- Exemplo:
  - Suporte em hardware para a proteção de memória. IBM 360:
    - 1 MB de RAM dividida em blocos de 2 KB
    - 512 chaves de proteção de 4 bits cada uma
    - PSW (Program Status Word): chave de 4 bits atribuída a cada processo
    - O HW interrompe qualquer tentativa de acesso a uma área de memória cujo código de proteção é diferente do PSW do processo corrente.



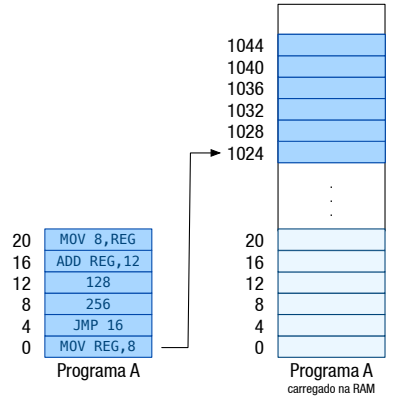
O Problema da Relocação

Performance em Sistemas Ciberfísicos



Exercício — RELOCAÇÃO

Faça a relocação do programa A no endereço de memória RAM indicado.



Relocação Estática
Memória

- Programa principal) + Funções + Bibliotecas:
  - carregados em um único espaço contíguo de endereçamento
- O "*linker*" (usado na compilação do programa):
  - deveria conhecer o endereço a partir do qual o programa deverá ser carregado na memória a fim de ajustar os endereços relativos gerados
  - No entanto…
    - esta informação só estará disponível no momento em que o programa for executado!
- Solução:
  - Recalcular os endereços do programa no momento em que for carregado na memória ➔ RELOCAÇÃO ESTÁTICA

Relocação Estática (cont.)
Memória

- Linker:
  - deve incluir no código binário informações sobre quais bytes são endereços de memória "relocáveis" e quais são códigos de operações, constantes ou outros itens que não devem ser relocados.
- A relocação estática é onerosa e complexa…
  - raramente usada

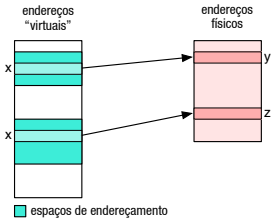
Gerência de Memória SEM ABSTRAÇÃO
Memória

- A gerência de memória sem abstração está obsoleta em computadores de uso geral.
- Porém…
  - geralmente usam endereçamento direto de memória:
    - sistemas embarcados;
    - smart cards;
    - software de eletrodomésticos, etc.
  - (neste caso, os programas instalados são conhecidos e não variam)

Gerência de Memória COM ABSTRAÇÃO

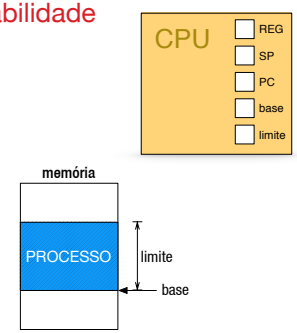
Espaços de Endereçamento
Memória — SO

- Relocação e proteção são problemas relacionados à multiprogramação.
- Melhor solução = abstração de memória:
  - Espaço de Endereçamento:
    - conjunto de endereços que um processo pode usar para endereçar memória.
  - Cada processo tem o seu próprio espaço de endereçamento.
    - Um mesmo endereço x é mapeado em diferentes localizações físicas em diferentes espaços de endereçamento.



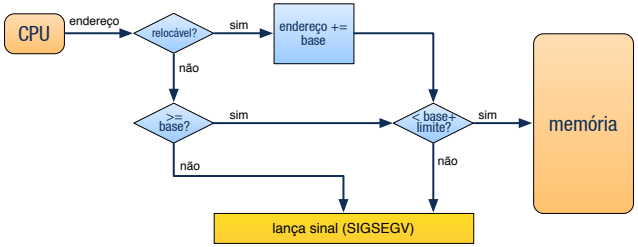
Proteção
Memória — SO

- Sistemas multiprogramados não devem permitir que processos acessem memória reservada para um outro processo ⇒ estabilidade
- Registradores da CPU (*hardware*) :
  - `base` e `limite`
- Quando o escalonador escolhe um processo, os registradores `base` e `limite` são carregados a partir da entrada na tabela de processos referente ao processo escolhido:
  - `base` = início da região de memória do processo
  - `limite` = tamanho da região



Proteção (cont.)
Memória — SO

- Cada endereço relocável gerado pelo programa é somado à base (relocação) e verificado se está dentro da partição alocada ao processo (proteção).



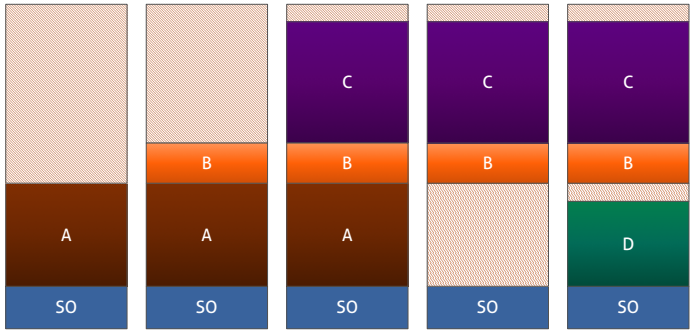
Proteção (cont.)
Memória — SO

- O *hardware* protege ambos os registradores de tentativas de modificação vindas de programas de usuários:
  - (instrução especial somente em modo *kernel*)
- Por que são `base` e `limite` são implementados como registradores (*hardware*)?
  - Comparação e soma devem ser feitas para cada endereço gerado:
    - grande impacto negativo no desempenho do sistema.

Swapping
Memória — SO

- Sistemas multiprogramados:
  - Geralmente a memória é insuficiente para rodar os processos necessários.
- Solução?
  - Enviar os processos excedente para o disco: "***swapping***".
  - Outra opção: Memória Virtual
    - diferença: partes dos processos podem ir para o disco

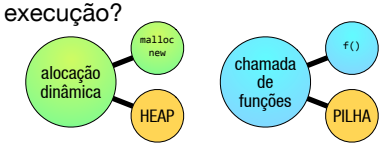
Swapping ao longo do tempo
Memória — SO



Tendência: criação de espaços vazios ("buracos") na memória.

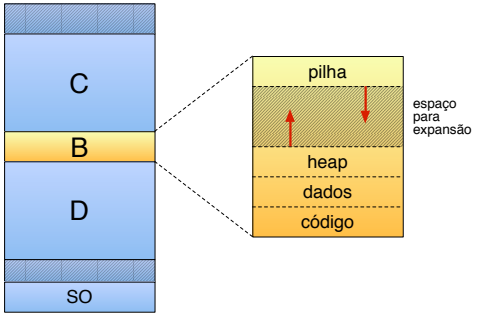
Swapping
Memória — SO

- Possível solução:
  - Compactação:
    - Oneroso (mesmo que em RAM)
- Em que situações a quantidade de memória exigida por um processo pode aumentar durante sua execução?



- Se não houver espaço adjacente disponível, o SO pode tentar deslocar o processo para um segmento maior…
  - deslocamentos ⇒ lentidão

Alocação de Espaço Extra para a Expansão
Memória — SO



Alocação de Espaço Extra para a Expansão (cont.)

Memória — SO

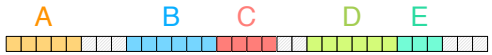
- Se o espaço previsto para a expansão for insuficiente:
  - o processo pode ser deslocado para outro espaço livre maior (se houver);
  - outro processo pode ser enviado ao disco para aguardar a liberação de espaço de memória que lhe seja suficiente; ou
  - o processo pode ser simplesmente abortado.

Gerenciamento de Memória Livre

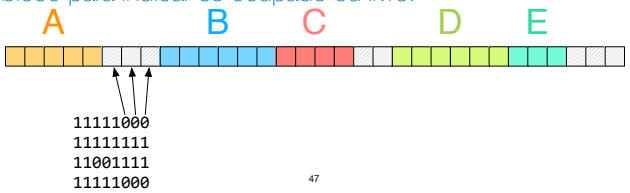
Mapa de bits

Memória — SO

- Bloco de memória = unidade de alocação de memória



- Mapa de bits:
  - 1 bit por bloco para indicar se ocupado ou livre:



Mapa de bits (cont.)

Memória — SO

- Tamanho dos blocos de memória:
  - Blocos pequenos:
    - Necessidade de muitos bits ➔ uso ineficiente da memória
  - Blocos grandes:
    - Memória sub-utilizada

- Mapa de bits:
  - Vantagem: simplicidade
  - Desvantagem: lentidão na localização de k blocos consecutivos livres

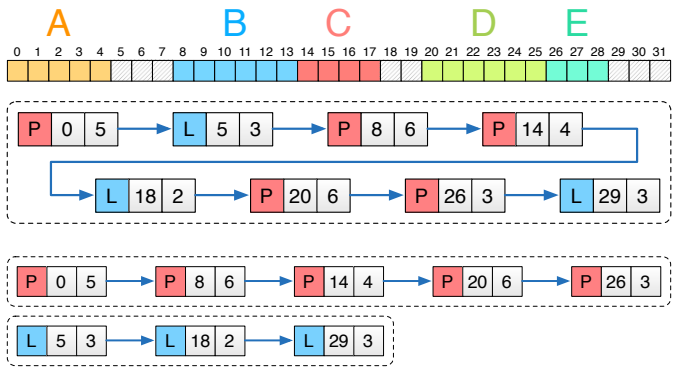
Listas Ligadas (p/ gerência de memória)

Memória — SO

- Lista ligada para indicar os segmentos de memória:
  - Livres L
  - Ocupados P
- Cada nó com os campos:
  - Segmento livre L ou ocupado P
  - Início do segmento
  - Tamanho do segmento (em blocos)
  - Próximo segmento

Listas Ligadas (p/ gerência de memória)

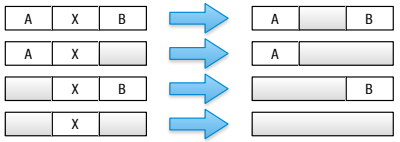
Memória — SO



Listas Ligadas

Memória — SO

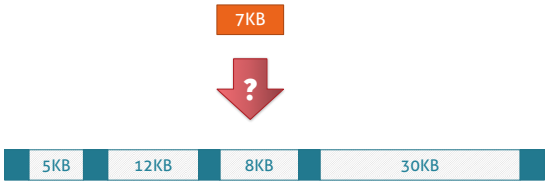
- Ordenação pelo campo de início do segmento:
  - Agiliza atualização
- Ordenação por outros campos é também possível (e.g., tamanho dos segmentos, listas separadas de blocos L e blocos P, etc.)
  - Dificulta a atualização das listas:



Algoritmos de alocação de memória

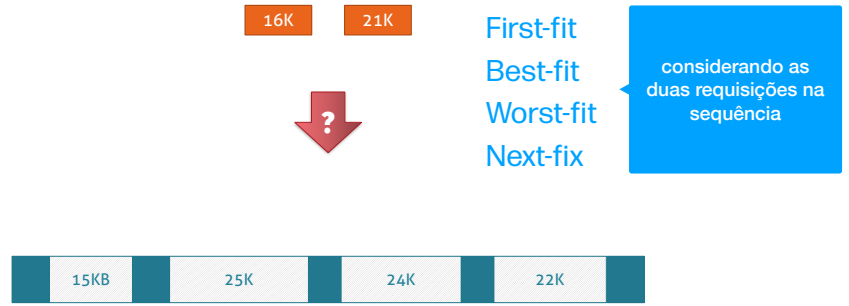
Memória — SO

- Sequência de segmentos livres:
  - 5KB, 12KB, 8KB, 30KB
- Em qual dos segmentos livres alocar 7KB?
  - First-fit
    - **12KB**
  - Best-fit
    - **8KB**
  - Worst-fit
    - **30 KB**
  - (Next-fit)



Exercício

Indique qual bloco livre será alocado para cada solicitação, considerando o algoritmo.



First-fit
Best-fit
Worst-fit
Next-fix

considerando as duas requisições na sequência

Memória Virtual

Memória Virtual

Memória — SO

- Necessária quando o tamanho da memória principal (RAM) não for suficiente para armazenar um processo.
- Solução?
  - Manter na memória apenas as partes do processo sendo utilizadas no momento.
- Isto é possível devido à propriedade da localidade de referência típica na maioria dos programas.

Memória Virtual

Memória — SO

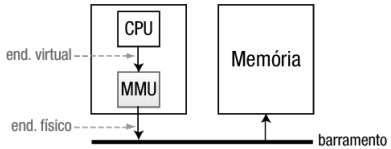
Localidade de referência:

- Localidade espacial:
  - **A tendência do programa em referenciar endereços de memória próximos daqueles que foram recentemente acessados (devido ao processamento sequencial de instruções, ou acesso sequencial a estruturas de dados, etc.).**
- Localidade temporal:
  - **A tendência do programa em acessar os mesmos endereços de memória acessados recentemente em um futuro próximo (devido a laços).**

Memória Virtual

Memória — SO

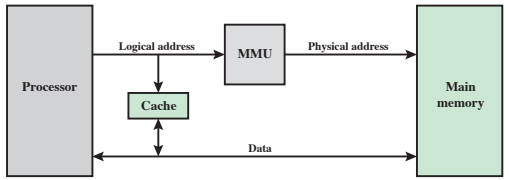
- Separação entre o Espaço de Endereçamento Virtual e o Espaço de endereçamento físico
- Endereços virtuais ⇒ Endereços físicos
  - Mapeamento feito pela MMU (Unidade de Gerenciamento de Memória)



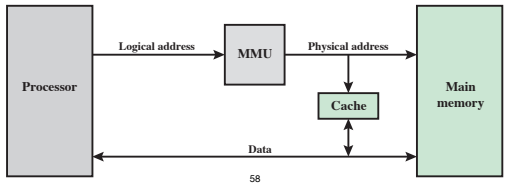
57

MMU e a Cache

Memória



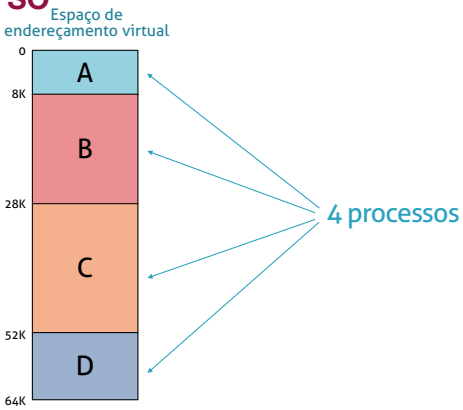
(a) Logical Cache



58

Memória Virtual

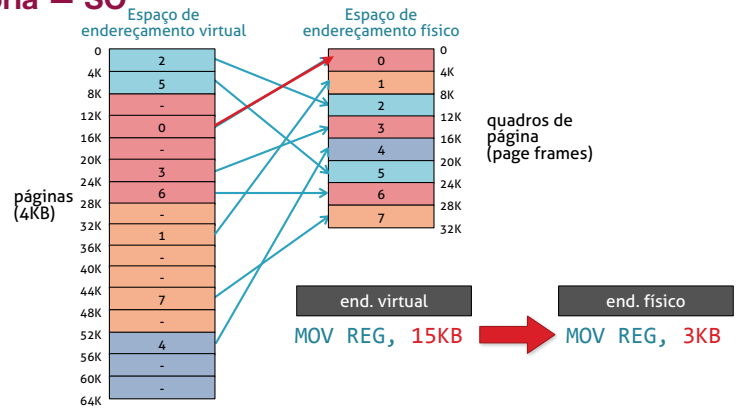
Memória — SO



59

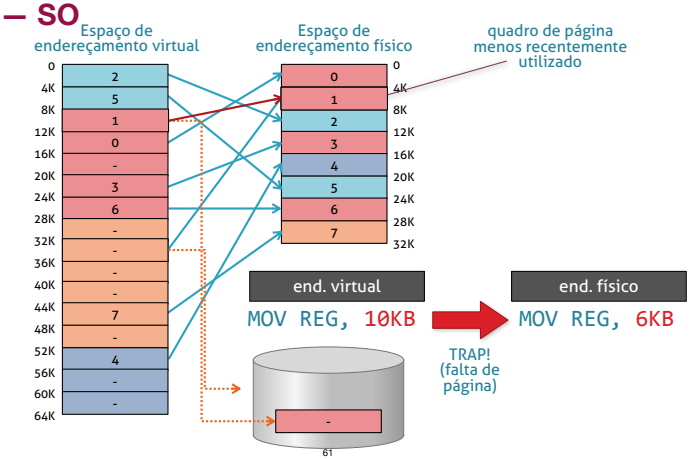
Memória Virtual (mapeamento)

Memória — SO

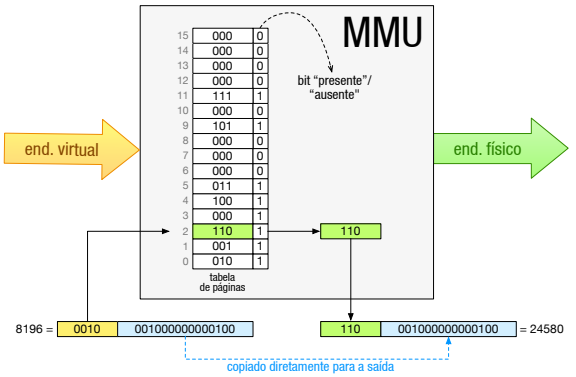


60

Memória Virtual (falta de página)



Mapeamento MMU



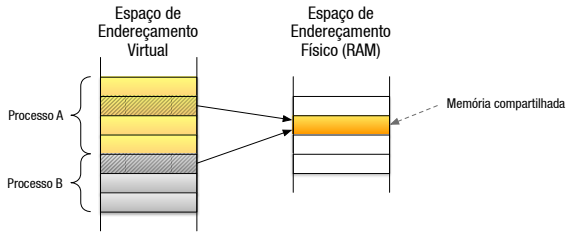
Vantagens da Memória Virtual

Há várias vantagens de separar o espaço de endereçamento virtual de um processo dos endereços físicos (RAM):

1. Processos são isolados uns dos outros (e do kernel) impedindo-os de acessar memória indevida.
   - Entradas na tabela de páginas de cada processo apontam para diferentes quadros de página físicos (ou quadros de páginas na área de swap).

Vantagens da Memória Virtual (cont.)

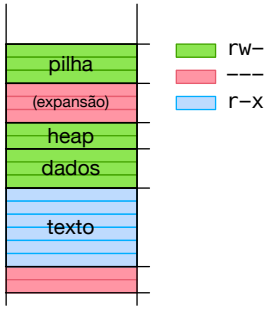
2. Quando necessário, 2 processos podem compartilhar memória.
   - Diferentes páginas podem ser mapeadas no mesmo quadro de página.



Vantagens da Memória Virtual (cont.)

Memória — SO

3. A implementação de esquemas de proteção de memória se tornam mais fáceis.
   - Páginas são marcadas:
     - “readable",
     - “writable",
     - "executable"



Vantagens da Memória Virtual (cont.)

Memória — SO

4. Programadores e ferramentas (como compiladores e *linkers*) não precisam se preocupar com o *layout* físico da memória.
   - Relocação estática: desnecessária

5. Programas são carregados mais rapidamente uma vez que apenas “partes” deles precisam ser trazidos para a memória para começar a execução.

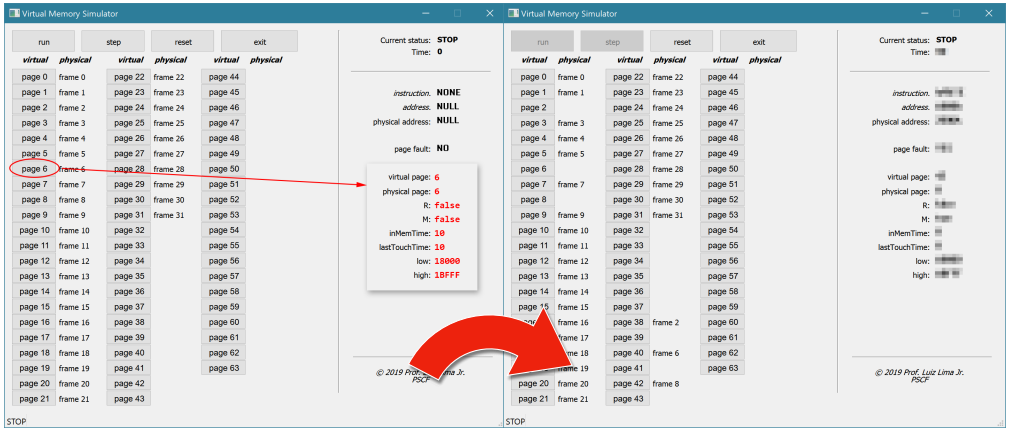
6. O “tamanho virtual” de um processo pode exceder a capacidade de RAM disponível.

66

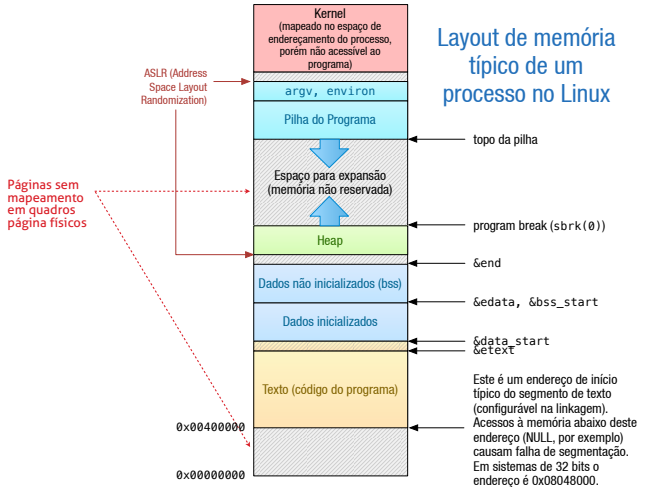
Exercício: Simulador de Memória Virtual (Blackboard)

Simulador de Memória Virtual

Alterar o arquivo **“comandos.txt”** de forma a produzir o seguinte mapeamento na MMU



Layout de Memória de um Processo no Linux



Segmentos de Memória de um Processo

Memória — SO

- Segmento de **texto**
  - Contém instruções em linguagem de máquina do programa
  - Segmento *read-only* (para que o processo não altere acidentalmente suas próprias instruções)
  - "Compartilhável": vários processos podem rodar o mesmo programa (`fork`!)
- Segmento de **dados inicializados**
  - Contém variáveis globais e estáticas inicializadas explicitamente
  - Valores são lidos diretamente do arquivo executável
- Segmento de dados não inicializados (***BSS** – Block Started by Symbol*)
  - Contém variáveis globais e estáticas que não foram explicitamente inicializadas
  - Antes de iniciar o programa, o sistema zera todo o conteúdo desta memória

Segmentos de Memória de um Processo (cont.)

Memória — SO

- **Pilha** do programa
  - Expande e retrai à medida em que funções são chamadas e retornam, respectivamente.
  - Cada elemento na pilha: "*stack frame*":
    - variáveis locais das funções
    - parâmetros de entrada
    - valor de retorno
- ***Heap***
  - Área de memória onde fica a memória alocada dinamicamente em tempo de execução.

Referências

- A. Tanenbaum, “Gerenciamento de Memória” em Sistemas Operacionais Modernos, 4ª edição
- M. Kerrisk, “The Linux Programming Interface — A Linux and UNIX System Programming Handbook”, no starch press, 2010.
- [1] Stallings, W. “Arquitetura e Organização de Computadores”, 10a. Edição, 2018, Pearson.
  - Seções 4.2 e 4.3

74