
















Gestão da Memória

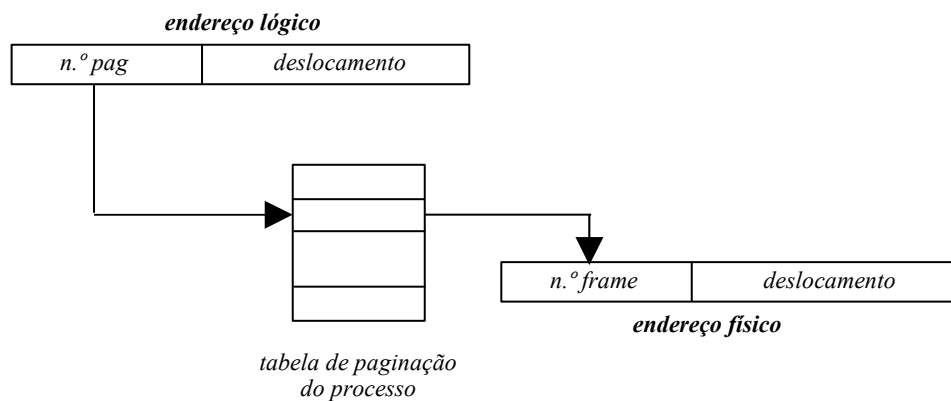
-  1. Descreva os diferentes níveis em que se estrutura a memória de um sistema computacional, caracterizando-os em termos de capacidade, tempo de acesso e custo. Explique, face a isso, que funções são atribuídas a cada nível.
-  2. Qual é o princípio que está subjacente à organização hierárquica de memória? Dê razões que mostrem porque é que a aplicação de tal princípio faz sentido.
-  3. Assumindo que o papel desempenhado pela gestão de memória num ambiente de multiprogramação se centra, sobretudo, no controlo da transferência de dados entre a memória principal e a memória de massa, indique quais são as actividades principais que têm que ser consideradas.
-  4. Porque é que a imagem binária do espaço de endereçamento de um processo é necessariamente relocável num ambiente de multiprogramação?
-  5. Distinga *linkagem estática* de *linkagem dinâmica*. Qual é a mais exigente? Justifique a sua resposta.
-  6. Assuma que um conjunto de processos cooperam entre si partilhando dados residentes numa região de memória, comum aos diferentes espaços de endereçamento. Responda justificadamente às questões seguintes
 - em que região do espaço de endereçamento dos processos vai ser definida a área partilhada?
 - será que o endereço lógico do início da área partilhada é necessariamente o mesmo em todos os processos?
 - que tipo de estrutura de dados em Linguagem C tem que ser usada para possibilitar o acesso às diferentes variáveis da área partilhada?
-  7. Distinga relativamente a um processo *espaço de endereçamento lógico* de *espaço de endereçamento físico*. Que problemas têm que ser resolvidos para garantir que a gestão de memória num ambiente de multiprogramação é eficiente e segura?
-  8. Caracterize a organização de memória designada de *memória real*. Quais são as consequências decorrentes deste tipo de organização?
-  9. Descreva detalhadamente o mecanismo de tradução de um *endereço lógico* num *endereço físico* numa organização de memória real.
-  10. O que é que distingue a *arquitectura de partições fixas* da *arquitectura de partições variáveis* numa organização de memória real? Indique quais são as vantagens e desvantagens de cada uma delas.
-  11. A organização de memória real conduz a dois tipos distintos de fragmentação da memória principal. Caracterize-os e indique a que tipo de arquitectura específica cada um está ligado.
-  12. Entre os métodos mais comuns usados para reservar espaço em memória principal numa arquitectura de partições variáveis, destacam-se o *next fit* e o *best fit*. Compare o desempenho destes métodos em termos do grau e do tipo de fragmentação produzidos e da eficiência na reserva e libertação de espaço.
-  13. Caracterize a organização de memória designada de *memória virtual*. Quais são as consequências decorrentes deste tipo de organização?
-  14. Indique as características principais de uma *organização de memória virtual*. Explique porque é que ela é vantajosa relativamente a uma *organização de memória real* no que respeita ao número de processos que correntemente coexistem e a uma melhor ocupação do espaço em memória principal.
-  15. O que é que distingue a *arquitectura paginada* da *arquitectura segmentada / paginada* numa organização de memória virtual? Indique quais são as vantagens e desvantagens de cada uma delas.

16. Descreva detalhadamente o mecanismo de tradução de um *endereço lógico* num *endereço físico* numa organização de memória virtual paginada.
17. Explique porque é que hoje em dia o sistema de operação dos computadores pessoais supõe, quase invariavelmente, uma organização de memória de tipo *memória virtual*.
18. Porque é que a *área de swapping* desempenha papéis diferentes nas organizações de *memória real* e de *memória virtual*?
19. Considere uma organização de memória virtual implementando uma *arquitectura segmentada / paginada*. Explique para que servem as tabelas de *segmentação* e de *paginação* do processo. Quantas existem de cada tipo? Descreva detalhadamente o conteúdo das entradas correspondentes.
20. Qual é a diferença principal existente entre a divisão do espaço de endereçamento de um processo em *páginas* e *segmentos*? Porque é que uma *arquitectura segmentada pura* tem pouco interesse prático?
21. As bibliotecas de rotinas *linkadas dinamicamente (DLLs)* são ligadas ao espaço de endereçamento de diferentes processos em *run time*. Neste contexto, responda justificadamente às questões seguintes
 - que tipo de código tem que ser gerado pelo compilador para que esta ligação seja possível?
 - este mecanismo de ligação pode ser utilizado indiferentemente em organizações de *memória real* e de *memória virtual*?
22. Quer numa *arquitectura paginada*, quer numa *arquitectura segmentada / paginada*, a memória principal é vista operacionalmente como dividida em *frames*, onde pode ser armazenado o conteúdo de uma página de um processo. Porque é que é conveniente impor que o tamanho de cada *frame* seja uma potência de dois?
23. Foi referido que nem todos os *frames* de memória principal estão disponíveis para substituição. Alguns estão *locked*. Incluem-se neste grupo aqueles que contêm as páginas do *kernel* do sistema de operação, do *buffer cache* do sistema de ficheiros e de um ficheiro mapeado em memória. Procure a aduzir razões que justifiquem esta decisão para cada um dos casos mencionados.
24. O que é o *princípio da optimalidade*? Qual é a sua importância no estabelecimento de algoritmos de substituição de páginas em memória principal?
25. O que é o *working set* de um processo? Conceba um método realizável que permita a sua determinação.
26. Dê razões que expliquem porque é que o *algoritmo do relógio*, numa qualquer das suas variantes, é tão popular. Descreva uma variante do *algoritmo do relógio* em que são consideradas as quatro classes de *frames* características do *algoritmo NRU*.
27. Como distingue a estratégia *demand paging* da *prepaging*? Em que contexto é que elas são aplicadas?
28. Assuma que a política de substituição de páginas numa dada organização de memória virtual é de âmbito *global*. Explique detalhadamente como procederia para detectar a ocorrência de *thrashing* e como procederia para a resolver.

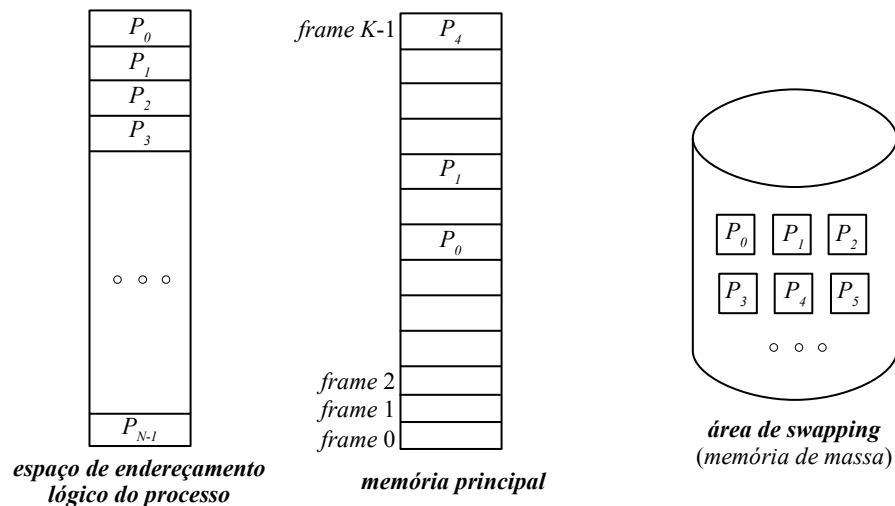
Problema técnico

Considere uma organização de memória virtual implementando uma *arquitectura paginada* em que as páginas têm o tamanho de 4KB.

A figura abaixo representa de uma maneira simplificada o mecanismo de tradução de um *endereço lógico* num *endereço físico* que é realizado sempre que ocorre um acesso à memória. Assuma endereços de 32 bits representados por variáveis de tipo **unsigned long**.



A figura seguinte procura ilustrar a distribuição das páginas do espaço de endereçamento de um processo num dado instante, parte delas está localizada em memória principal.



O algoritmo de substituição de páginas utilizado é o *NRU (Not Recently Used)* e é aplicado numa estratégia de âmbito global.

Admita que foram definidas as estruturas de dados seguintes:

Entrada (simplificada) da Tabela de Controlo de Processos

```
typedef struct
{
    BOOLEAN busy; /* sinalização de entrada ocupada */
    unsigned int pid, /* identificador do processo */
    pstat; /* estado do processo: 0 - RUN;
            1 - BLOCKED; 2 - READY-TO-RUN
            3 - SUSPENDED-BLOCK; 4 - SUSPENDED-READY
            5 - CREATED; 6 - TERMINATED */
    unsigned char intreg[K]; /* contexto do processador */
    unsigned long pag_tb; /* endereço da tabela de paginação
                           do processo em memória principal */
    unsigned int pag_tb_len; /* n.º de entradas da tabela */
} PCT_ENTRY;
```

Entrada (simplificada) da Tabela de Paginação

```
typedef struct
{
    BOOLEAN loaded; /* sinalização de carregamento da página em
                    memória principal */
    unsigned long nframe; /* n.º do frame de armazenamento da
                           página em memória principal */
    unsigned long nblk; /* n.º do bloco de armazenamento da
                        página na área de swapping */
    BOOLEAN access, /* sinalização de acesso à página no
                    último intervalo de monitorização */
    modif; /* sinalização de modificação da página
            desde que foi carregada em memória principal */
} TB_PAG;
```

Nó de lista biligada

```
struct binode
{
    unsigned int pct_index; /* índice da entrada da PCT que
                            descreve o processo a que pertence o nó */
    unsigned long npag; /* n.º da página do espaço de endereçamento */
    unsigned long nobjt; /* n.º do frame ou do bloco de armazenamento */
    struct binode *ant, /* ponteiro para o nó anterior */
    *next; /* ponteiro para o nó seguinte */
};
typedef struct binode BINODE;
```

CAM

```
struct cam
{
    BINODE *pstart; /* ponteiro para o início da CAM */
    unsigned long n; /* tamanho da CAM */
};
typedef struct cam CAM;
```

e as variáveis globais descritas abaixo:

```
static PCT_ENTRY pct[100]; /* tabela de controlo de processos */
static unsigned int pindex; /* índice da entrada da PCT que
                             descreve o processo que detém o processador */
static CAM b_frm, /* lista dos frames unlocked ocupados */
f_frm; /* lista dos frames livres */
static CAM b_blk, /* lista dos blocos ocupados da área de swapping */
f_blk; /* lista dos blocos livres da área de swapping */
```

Finalmente, as primitivas seguintes estão também disponíveis:

Salvaguarda e restauro do contexto

```
void save_context (unsigned int pct_index);
void restore_context (unsigned int pct_index);
```

*Inserção e retirada e pesquisa de nós na CAM (os nós são ordenados pelo campo *nobjt*)*

```
void cam_in (unsigned long nord, CAM *cam, BINODE *val);
void cam_out (unsigned long nord, CAM *cam, BINODE **val_p);
    se o nó não existir, *val_p = NULL
BOOLEAN cam_empty (CAM *cam);
void cam_search (unsigned long nord, CAM *cam, BINODE **val_p);
    se o nó não existir, *val_p = NULL
```

Transferência de páginas de e para a memória principal

```
void swap_in (unsigned long nframe, unsigned long nblk);
void swap_out (unsigned long nframe, unsigned long nblk);
```

1. Assumindo que coexistem presentemente quatro processos no sistema computacional, A, B, C e D, de tamanho, respectivamente, 20MB, 4MB, 1MB e 32 MB, indique qual o tamanho da tabela de paginação de cada um deles. Justifique detalhadamente a sua resposta..
2. Descreva o conteúdo da tabela de paginação de um processo cujo espaço de endereçamento é formado por 4 páginas com as características seguintes
 - o seu armazenamento na *área de swapping* é feito, respectivamente, nos blocos n.º 1023D₁₆, F54A₁₆, 25₁₆ e 5C567₁₆;
 - a primeira e a terceira páginas estão residentes em memória principal nos frames 395₁₆ e 45A₁₆;
 - a terceira página foi modificada e ambas foram referenciadas no último intervalo monitorado.
3. Assuma uma situação hipotética em que só estão residentes em memória principal páginas do espaço de endereçamento de dois processos, A e B, cuja descrição é feita nas entradas 1C₁₆ e 2A₁₆ da tabela de controlo de processos (PCT). As páginas do processo A presentes são a terceira e a quinta, alojadas nos *frames* 211₁₆ e 385₁₆, respectivamente. A página do processo B presente é a oitava, alojada no *frame* 500₁₆. Mostre para a situação indicada como é formada a *lista dos frames ocupados em memória principal*, *b_frm*. Justifique detalhadamente a sua resposta.
4. Explique em que condições (considere todas) o espaço de endereçamento do processo pode estar totalmente fora da memória principal.
5. Construa a primitiva que retira o processo de memória, faz o seu *swapped out* completo, após ele ter terminado. Não se esqueça de actualizar os campos da entrada da tabela de controlo de processos correspondente.

```
void full_swap_out (unsigned int pct_index);
```

6. Construa a primitiva que implementa o algoritmo de substituição de um *frame* em memória principal segundo a estratégia *NRU*. A primitiva deve devolver o n.º do *frame* escolhido para substituição.

```
unsigned long NRU_gen_replace (void);
```

7. Construa uma primitiva que identifica todos os processos que correntemente coexistem, listando para cada um deles o seu *pid*, o seu *estado*, o tamanho do seu espaço de endereçamento e o n.º e a identificação dos *frames* de memória principal que lhe estão atribuídos.

```
void proc_list (void);
```