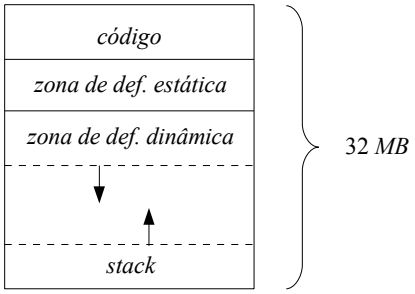
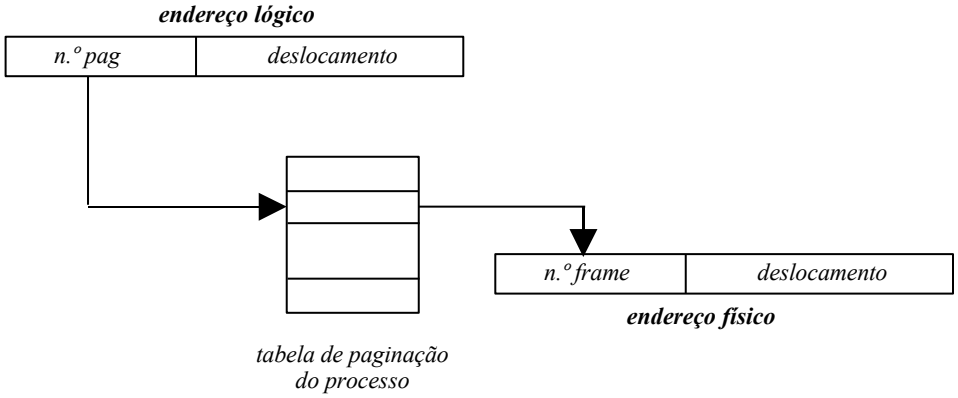
Parte B (10 valores)

Considere uma organização de memória virtual implementando uma *arquitectura paginada* em que as páginas constituintes têm o tamanho de 4 KB. Considere ainda que os endereços são formados por palavras de 32 bits e que o *espaço de endereçamento lógico* de cada processo é fixado em 32 MB e é organizado como a figura abaixo mostra.

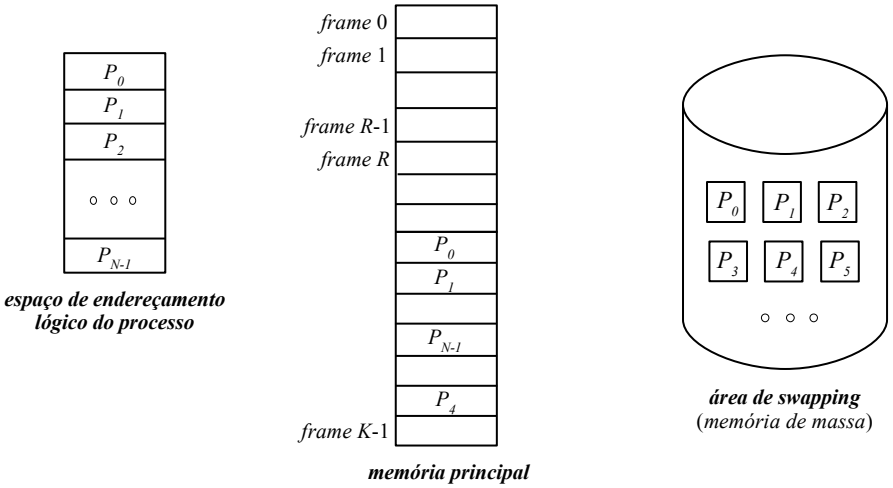


Note que o *linker* associa a cada uma das três primeiras regiões (código, zona de definição estática e zona de definição dinâmica) o início de uma nova página e ao *stack* o fim da última página. Para simplificar, assuma que esta organização não permite a definição de regiões de partilha de código, nem de dados.

O mecanismo de tradução de um *endereço lógico* num *endereço físico*, que é realizado sempre que ocorre um acesso à memória, é descrito pelo diagrama seguinte.



Recorde que as páginas do *espaço de endereçamento lógico* de um processo que coexiste estão distribuídas num dado instante pela *área de swapping* (a sua totalidade) e pela memória principal (eventualmente, só parte delas).



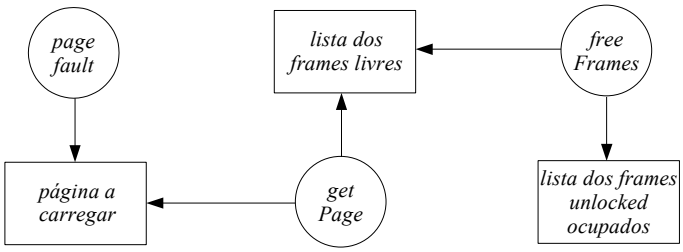
Assuma que os *R* primeiros *frames* da memória principal estão reservados para o núcleo do sistema de operação e para as estruturas de dados associadas e estão permanentemente *locked*. Os *frames* restantes, em número *K-R*, estão *unlocked* e o seu conteúdo pode ser, portanto, modificado ao longo do tempo. Assuma ainda que a *área de swapping* constitui uma partição em memória de massa formada por *T* blocos, cada um com uma capacidade de armazenamento de 4 KB.

O algoritmo de substituição de páginas utilizado é uma qualquer das variantes do *algoritmo do relógio* e é aplicado numa estratégia de âmbito global.

Quando ocorre uma *falha de página* (*page fault*), o processo P que a originou é bloqueado pela rotina de serviço à excepção e o processo de sistema *getPage*, encarregado de transferir a página em falta da *área de swapping* e de a armazenar num *frame* livre da memória principal, é acordado. Findo o que o processo P é colocado de novo na fila de espera dos processos *READY-TO-RUN* para que possa eventualmente continuar a sua execução.

O sistema mantém sempre uma lista de *frames* livres para que a transferência de uma página em falta possa ser posta em marcha o mais cedo possível. Ao longo do tempo, esta lista vai reduzindo o seu tamanho. Quando o número de *frames* livres se torna mínimo, o processo de sistema *getPage* acorda um outro processo de sistema *freeFrames*, cuja função é libertar *frames* de memória principal até que o número de *frames* livres atinja de novo o valor máximo, usando o algoritmo de substituição estabelecido.

Assuma que os processos de sistema *getPage* e *freeFrames* têm uma prioridade de execução superior à de qualquer outro processo existente, sendo *getPage* o mais prioritário dos dois, e que os seus espaços de endereçamento estão localizados na região do núcleo do sistema de operação.



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[D];          /* contexto do processador */
          TB_PAG pag[NP];           /* tabela de paginação do processo */
          unsigned int waitPage;          /* id do semáforo de bloqueio
                             do processo quando ocorre uma falta de página */
        } 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

```
struct node
  { unsigned int pct_index;            /* índice da entrada da PCT que
                             descreve o processo a que pertence o frame */
    unsigned long npag;   /* n.º da página do espaço de endereçamento,
                                                          se aplicável */
    unsigned long nframe;                  /* n.º do frame associado,
                                                          se aplicável */
    struct node *ant,                   /* ponteiro para o nó anterior */
                *next;                  /* ponteiro para o nó seguinte */
  };
typedef struct node NODE;
```

Lista

```
typedef struct
  { NODE *pstart;                   /* ponteiro para o início da lista */
    unsigned long n;                        /* tamanho actual da lista */
  } LIST;
```

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 NODE frame[K];        /* tabela de estado dos frames da memória */
static LIST b_frm,              /* lista dos frames unlocked ocupados,
                                     organizada como um FIFO circular */
            f_frm;                        /* lista dos frames livres,
                                     organizada como um FIFO circular */
static NODE pageInfo;          /* buffer de comunicação com o processo
                                getPage quando ocorre uma falta de página */
static unsigned int waitForPageFault,    /* id do semáforo de bloqueio
          do processo getPage enquanto aguarda por uma falta de página */
                    waitForFrame,        /* id do semáforo de bloqueio
                do processo getPage enquanto aguarda por um frame livre */
                     waitForWork;        /* id do semáforo de bloqueio
     do processo freeFrames enquanto aguarda por iniciar o seu trabalho */
```

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 retiradade nós na LIST

```
void fifo_in (LIST *list, NODE *val);
void fifo_out (LIST *list, NODE **val_p);
          se o nó não existir, *val_p = NULL
BOOLEAN fifo_empty (LIST *list);
```

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);
```

Manipulação de semáforos

```
void sem_down (unsigned int sem_id);
void sem_up (unsigned int sem_id); .
```

1. Qual é o tamanho da tabela de paginação de cada processo definida em termos de número de entradas? Justifique claramente a sua resposta.

2. Descreva o conteúdo das quatro primeiras entradas da tabela de paginação de um processo cujas páginas associadas apresentam actualmente as características seguintes
   - o seu armazenamento na *área de swapping* é feito, respectivamente, nos blocos n.º $E3D_{16}$, $F54A_{16}$, 125_{16} e $5C54_{16}$;
   - a primeira e a terceira páginas estão residentes em memória principal nos frames 3195_{16} e $4A5A_{16}$;
   - a terceira página foi modificada e ambas foram referenciadas no último intervalo de tempo monitorado.

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

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

4. Construa a primitiva que implementa o algoritmo de substituição de um *frame* em memória principal usando o *algoritmo de relógio* na *variante da segunda oportunidade*. A primitiva deve devolver o n.º do *frame* escolhido para substituição.

```
unsigned long second_chance_locate (void);
```