

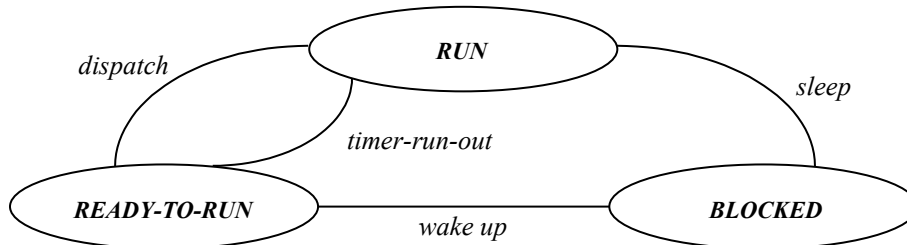
### ***Gestão do Processador***

1. A modelação do ambiente de multiprogramação através da activação e desactivação de um conjunto de processadores virtuais, cada um deles associado a um processo particular, supõe que dois factos essenciais relativos ao comportamento dos processos envolvidos sejam garantidos. Quais são eles?
2. Qual é a importância da tabela de controlo de processos (*PCT*) na operacionalização de um ambiente de multiprogramação? Que tipo de campos devem existir em cada entrada da tabela?
3. O que é o *scheduling* do processador? Que critérios devem ser satisfeitos pelos algoritmos que o põem em prática? Quais são os mais importantes num sistema multiutilizador de uso geral, num sistema de tipo *batch* e num sistema de tempo real?
4. Descreva o diagrama de estados do *scheduling* do processador em três níveis. Qual é o papel desempenhado por cada nível? Num sistema de tipo *batch* multiprogramado fará sentido a existência de três níveis de *scheduling*?
5. Os estados *READY-TO-RUN* e *BLOCKED*, entre outros, têm associadas filas de espera de processos que se encontram nesses estados. Conceptualmente, porém, existe apenas uma fila de espera associada ao estado *READY-TO-RUN*, mas filas de espera múltiplas associadas ao estado *BLOCKED*. Em princípio, uma por cada dispositivo ou recurso. Porque é que é assim?
6. Indique quais são as funções principais desempenhadas pelo *kernel* de um sistema de operação. Neste sentido, explique porque é que a sua operação pode ser considerada como um *serviço de excepções*.
7. O que é uma *comutação de contexto*? Descreva detalhadamente as operações mais importantes que são realizadas quando há uma comutação de contexto.
8. Classifique os critérios devem ser satisfeitos pelos algoritmos de *scheduling* segundo as perspectivas sistémica e comportamental, e respectivas subclasses. Justifique devidamente as suas opções.
9. Distinga disciplinas de prioridade estática das de prioridade dinâmica. Dê exemplos de cada uma delas.
10. Num sistema de operação multiutilizador de uso geral, há razões diversas que conduzem ao estabelecimento de diferentes classes de processos com direitos de acesso ao processador diferenciados. Explique porquê.
11. Entre as políticas de *scheduling preemptive* e *non-preemptive*, ou uma combinação das duas, qual delas escolheria para um sistema de tempo real? Justifique claramente as razões da sua opção.
12. Foi referido nas aulas que os sistemas de operação de tipo *batch* usam principalmente uma política de *scheduling non-preemptive*. Será, porém, uma política pura, ou admite excepções?
13. Justifique claramente se a disciplina de *scheduling* usada em Linux para a classe *SCHED\_OTHER* é uma política de prioridade estática ou dinâmica?
14. O que é o *aging* dos processos? Dê exemplos de duas disciplinas de *scheduling* com esta característica, mostrando como ela é implementada em cada caso.
15. Distinga *threads* de processos. Assumindo que pretende desenvolver uma aplicação concorrente usando um dos paradigmas, descreva o modo como cada um afecta o desenho da arquitectura dos programas associados.
16. Indique justificadamente em que situações um ambiente *multithreaded* pode ser vantajoso.
17. Que tipo de alternativas pode o sistema de operação fornecer à implementação de um ambiente *multi-threaded*? Em que condições é que num multiprocessador simétrico os diferentes *threads* de uma mesma aplicação podem ser executados em paralelo?
18. Explique como é que os *threads* são implementados em Linux.

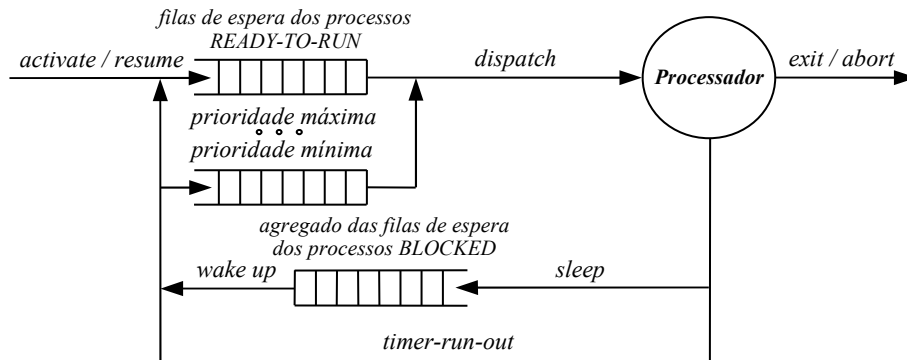
19. O principal problema da implementação de *threads*, a partir de uma biblioteca que fornece primitivas para a sua criação, gestão e *scheduling* no nível utilizador, é que quando um *thread* particular executa uma *chamada ao sistema* bloqueante, todo o processo é bloqueado, mesmo que existam *threads* que estão prontos a serem executados. Será que este problema não pode ser minimizado?
20. Num ambiente *multithreaded* em que os *threads* são implementados no nível utilizador, vai haver um par de *stacks* (sistema / utilizador) por *thread*, ou um único par comum a todo o processo? E se os *threads* forem implementados ao nível do *kernel*? Justifique.

## Problema técnico

A figura apresenta o diagrama de transição de estados do *scheduling* de baixo nível do processador



Suponha que foi implementada uma disciplina de *scheduling* com 4 níveis de prioridade (0- representando a prioridade máxima, 3 – representando a prioridade mínima), como a figura abaixo ilustra.



Admita ainda 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 */
    prior;                 /* nível de prioridade */
    unsigned char intreg[K]; /* contexto do processador */
    unsigned long addspace; /* endereço da região de memória
                           principal onde está localizado o espaço de endereçamento
                           do processo (organização de memória real) */
} PCT_ENTRY;
  
```

Nó de lista biligada

```

struct binode
{
    unsigned int info;           /* valor armazenado */
    struct binode *ant,         /* ponteiro para o nó anterior */
    *next;                     /* ponteiro para o nó seguinte */
};

typedef struct binode BINODE;
  
```

*FIFO*

```
typedef struct
{ BINODE *pin_val,          /* ponteiro para o ponto de inserção */
  *pout_val;                /* ponteiro para o ponto de retirada */
} FIFO;
```

*Semáforo*

```
typedef struct
{ int val;                  /* valor de contagem */
  FIFO queue;               /* fila de espera dos processos bloqueados */
} SEMAPHORE;
```

e as variáveis globais descritas abaixo

```
static SEMAPHORE sem[200];          /* array de semáforos */
static PCT_ENTRY pct[100];          /* tabela de controlo de processos */
static FIFO redtorun[4];             /* array das filas de espera dos processos
                                     prontos a serem executados */
static unsigned int pindex;          /* índice da entrada da PCT que
                                     descreve o processo que detém o processador */
```

Finalmente, as primitivas seguintes estão também disponíveis:

*Activação e inibição das interrupções*

```
void interrupt_enable (void);
void interrupt_disable (void);
```

*Salvaguarda e restauro do contexto do processador*

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

*Reserva e libertação de espaço em memória dinâmica*

```
void *malloc (unsigned int size);
void free (void *pnt);
```

*Inserção e retirada de nós na FIFO*

```
void fifo_in (FIFO *fifo, BINODE *val);
void fifo_out (FIFO *fifo, BINODE *val);
```

*Transição de estado dos processos*

```
void dispatch (void);
void timerrunout (void);
void sleep (unsigned int sem_index);
void wakeup (unsigned int sem_index);
```

Assuma ainda que o mecanismo de prioridades estabelecido faz variar a prioridade de cada processo de uma forma circular, entre o valor máximo e o valor mínimo, sempre que ele é agendado para execução.

1. Qual é a vantagem do valor armazenado no campo *info* do *array* de FIFOs que implementa as filas de espera dos processos prontos a serem executados, ser o índice da entrada da tabela de controlo de processos associado com um dado processo e não o seu *pid*?
2. Que alterações teriam que ser introduzidas na estrutura de dados *PCT\_ENTRY*, se se pretendesse introduzir o *scheduling* de nível médio (*gestão da memória principal*)? Justifique adequadamente a sua resposta.
3. Construa as primitivas que fazem o *create*, o *destroy*, o *down* e o *up* de um semáforo.

```
unsigned int sem_create (void);
void sem_destroy (unsigned int sem_index);
void sem_down (unsigned int sem_index);
void sem_up (unsigned int sem_index);
```

4. Construa as primitivas *dispatch*, *timerrunout*, *sleep* e *wakeup*.