Sistemas de Operação

Teste 1 *7 de Janeiro de 2005*

Parte A (*10 valores*)

1. Os sistemas de operação actuais apresentam um ambiente de interacção com o utilizador de características eminentemente gráficas. Contudo, quase todos eles fornecem em alternativa um ambiente de interacção baseado em linhas de comando. Qual será a razão principal deste facto?

2. 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.

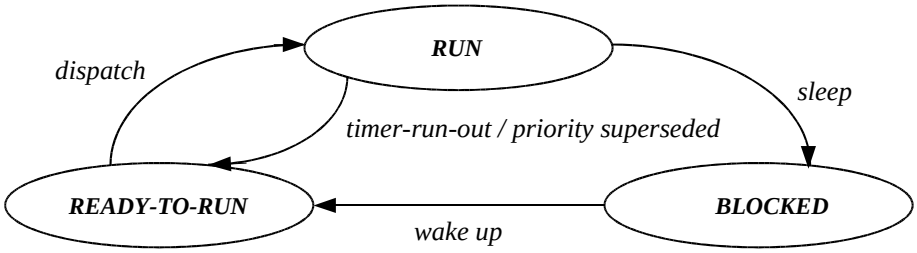
3. As políticas de *prevenção de deadlock no sentido lato* baseiam-se na transição do sistema entre estados ditos *seguros*. O que é um estado seguro? Qual é o princípio que está subjacente a esta definição?

4. Explique qual é o papel desempenhado pela memória de massa num sistema computacional onde está instalado um sistema de operação contemporâneo de uso geral.

5. 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.

Parte B (*10 valores*)

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



Suponha que foi implementada uma disciplina de *scheduling* semelhante à do Linux onde estão contempladas as classes *SCHED_FIFO* e *SCHED_OTHER*.

A classe *SCHED_FIFO* tem dois níveis de prioridade estática. Os processos desta classe executam até terminarem ou bloquearem. O processador só lhes é retirado se um processo da mesma classe, com prioridade mais elevada, passa entretanto ao estado *READY-TO_RUN*.

A classe *SCHED_OTHER* comporta os processos normais do sistema de operação. O processador só lhes é atribuído se não houver processos da classe *SCHED_FIFO* no estado *READY-TO_RUN*. Usa um algoritmo baseado em créditos no estabelecimento da prioridade relativa e na definição da janela de execução.

No instante de recreditação *i*, a prioridade do processo *j*, $CPU_j(i)$, equivalente ao número de créditos de execução que lhe são atribuídos, é calculada pela fórmula seguinte

$$CPU_j(i) = \frac{CPU_j(i-1)}{2} + PBase_j \quad , \qquad (1)$$

em que $CPU_j(i\text{-}1)$ representa o número de créditos não usados pelo processo no intervalo de recreditação *i*-1 e $Pbase_j$ a prioridade base do processo (no intervalo 1 a 20).

O *scheduler* calendariza para execução o processo com mais créditos (maior prioridade). Sempre que ocorre uma interrupção do *RTC*, o processo perde um crédito. Quando o número de créditos atinge o valor zero, o processo perde a posse do processador por esgotamento da janela de execução e outro processo é calendarizado.

Uma nova operação de recreditação acontece sempre que todos os processos da classe no estado *READY-TO_RUN* têm créditos nulos. A operação vai envolver todos os processos existentes, mesmo aqueles que estão correntemente no estado *BLOCKED*.

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 */
                       class,               /* classe de scheduling */
                       prior;              /* nível de prioridade /
                                    créditos de execução disponíveis */
          unsigned char intreg[20];        /* contexto do processador */
          unsigned long addspace;    /* endereço da região de memória
               principal onde está localizado o espaço de endereçagem
                           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;
```

e as variáveis globais descritas abaixo

```
static PCT_ENTRY pct[100];         /* tabela de controlo de processos */
static FIFO sff_rtr[2],    /* array das filas de espera dos processos
                       prontos a serem executados da classe SCHED_FIFO */
            soth_rtr[K];   /* array das filas de espera dos processos
                      prontos a serem executados da classe SCHED_OTHER */
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

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

Manipulação de semáforos

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

Transição de estado dos processos

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

Processamento de créditos

```
void recredit (void);
BOOLEAN RTCservice (void);
```

1. Qual é o tamanho *K* do *array* de *FIFOs* que implementa as filas de espera dos processos prontos a serem executados da classe *SCHED_OTHER*? Justifique adequadamente a sua resposta.
   Sugestão – Tenha em conta os valores gerados sucessivamente pela equação (1) e o facto de que o número de créditos é armazenado em vírgula fixa.

2. Explique que tipo de processos da classe *SCHED_OTHER* são privilegiados pelo algoritmo baseado em créditos, usado no estabelecimento da prioridade relativa e na definição da janela de execução. Justifique adequadamente a sua resposta.

3. Construa a primitiva *RTCservice* que constitui a rotina de serviço à interrupção do *RTC*. Note que é esta rotina que detecta o esgotamento da janela de execução atribuída ao processo da classe *SCHED_OTHER*. Assuma que a rotina devolve *TRUE*, se a janela se esgotou, e *FALSE*, em caso contrário.

4. Construa a primitiva *dispatch*.