## ***Parte A*** (10 valores)

1. O que são *chamadas ao sistema?* Dê exemplos válidos para o Unix (recorde que o Linux não é mais do que uma implementação específica do Unix). Explique qual é a sua importância no estabelecimento de um *interface de programação de aplicações* (API).

2. Distinga disciplinas de prioridade estática das de prioridade dinâmica. Dê um exemplo de cada uma delas.

3. 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.

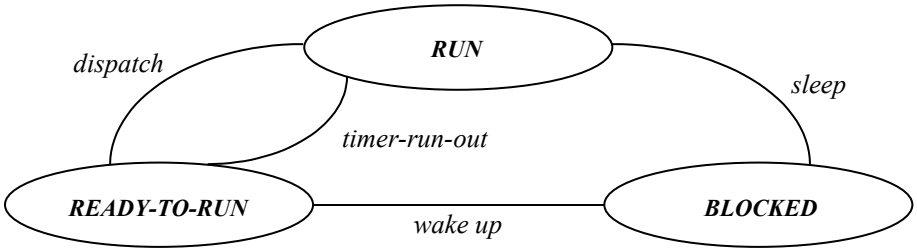
4. As políticas de *prevenção de deadlock no sentido lato* baseiam-se na transição do sistema entre estados ditos *seguros*. O que é um estado seguro? Qual é o princípio que está subjacente a esta definição? Dê um exemplo de uma política com estas características.

5. 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?

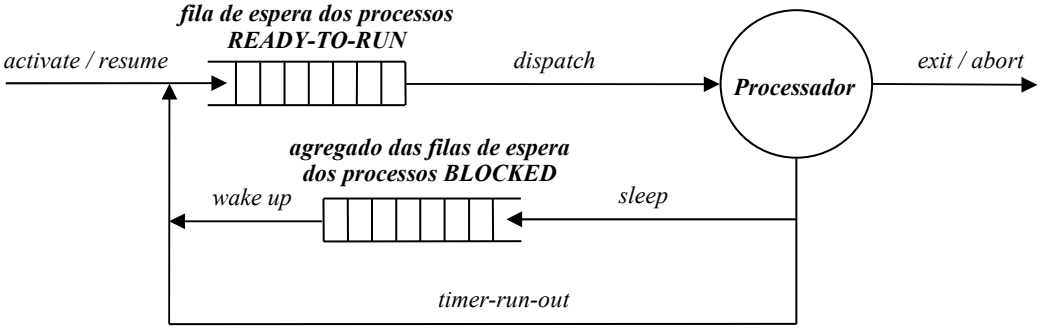
   que tipo de estrutura de dados em Linguagem C tem que ser usada para possibilitar o acesso às diferentes variáveis da área partilhada?

## ***Parte B*** (10 valores)

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




Suponha que foi implementada uma disciplina de *scheduling* que valoriza o critério de justiça.




O que esta disciplina tem de mais característico é que procura efectuar a valorização do critério de justiça da maneira mais estrita possível. Para o entender, considere um sistema computacional ideal que tem a particularidade de poder executar em paralelo qualquer número de processos, sendo a velocidade de processamento inversamente proporcional ao número de processos que coexistem num dado instante. Assim, suponha um processo cujo tempo de processamento nesse sistema computacional é de $x$ segundos, se forem lançados $N$ processos semelhantes, o tempo de execução de cada um deles passa a ser de $Nx$ segundos, mas em nenhum momento qualquer deles permanece inactivo aguardando a atribuição do processador. Trata-se, portanto, da valorização máxima possível!

Num sistema computacional real, com um só processador, só um processo de cada vez é executado. Os restantes aguardam na fila de espera dos processos *READY-TO-RUN* a sua oportunidade de execução. É, no entanto, viável organizar uma disciplina de *scheduling* que aproxime o tipo de operacionalidade do sistema real à providenciada pelo sistema ideal. Para o verificar, imagine que, associado a cada processo, existem duas informações temporais que são inicializadas a zero quando o processo é criado. A primeira, medida em tiques do relógio de tempo real, indica o acumulado de tempo que o processo aguarda inactivo pela atribuição do processador; a segunda, medida em tiques do relógio de tempo virtual (um tique deste relógio corresponde a $N$ tiques do relógio de tempo real, em que $N$ é o número de processos que coexistem), indica aproximadamente o tempo de execução do processo como se ele estivesse a ser executado no sistema ideal. A diferença entre o primeiro e o segundo tempo é usada para se obter uma estimativa do grau de justiça presente: processos para os quais este valor é maior não têm garantido a sua fracção de tempo processador.

A fila de espera dos processos *READY-TO-RUN* não constitui, por isso, neste caso um FIFO, sendo ordenada por ordem decrescente desta diferença, e o processador é sempre atribuído ao processo para o qual a diferença é maior.

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 */
                       waittime, /* acumulado do tempo que o processo
        aguarda no estado READY-TO-RUN pela atribuição do processador */
                       virttime;                     /* tempo virtual */
          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 info1,                      /* valor armazenado */
                     info2;              /* chave de ordenação (CAM) */
        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 */
           *actual;            /* ponteiro para o nó que acabou de ser
        referenciado por uma operação de tipo cam_get_* - inicializado
                                          a pstart por cam_get_start */
    unsigned long n;                                /* tamanho da CAM */
  };
typedef struct cam CAM;
```

*Semáforo*

```
typedef struct
        { unsigned 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 CAM redtorun;                /* fila de espera dos processos no
                                                 estado READY-TO-RUN */
static unsigned int pindex;           /* índice da entrada da PCT que
                           descreve o processo que detém o processador */
static unsigned int timeslotocup;    /* fracção de ocupação da janela
        temporal de execução por um processo - o valor é inicializado
             a T e decrementado a cada tique do relógio de tempo real */
static unsigned int virttimeslot;    /* duração da janela de temporal
         de execução medida em tempo virtual - o valor é inicializado
       a zero e incrementado a cada N tiques do relógio de tempo real */
static unsigned int realtovirtconv;          /* registo auxiliar para
           conversão dos tiques do relógio de tempo real em tiques do
                                          relógio de tempo virtual */
```

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);
BINODE *fifo_out (FIFO *fifo);
BOOLEAN fifo_empty (FIFO *fifo);
```

*Inserção e retirada de nós na CAM e rotinas auxiliares* (os nós são ordenados pelo campo *info2*)

```
void cam_in (CAM *cam, BINODE *val);             /* inserção de um nó */
BINODE *cam_out (CAM *cam);                      /* retirada de um nó */
          retira sempre o nó cujo campo info2 é maior,
          se a CAM estiver vazia, devolve NULL
BINODE *cam_get_first (CAM *cam);           /* leitura do primeiro nó */
          se a CAM estiver vazia, devolve NULL
BINODE *cam_get_next (CAM *cam);            /* leitura do nó seguinte */
          se a CAM estiver vazia ou não houver mais nós, devolve NULL
BOOLEAN cam_empty (CAM *cam);
unsigned int cam_size (CAM *cam);            /* n. de nós armazenados */
```

*Transição de estado dos processos*

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

Assuma que existe sempre pelo menos um processo a ser executado.

1. Proponha uma estrutura de dados de nome FIFO que defina um *FIFO* dinâmico, formado por nós de tipo BINODE.

2. Que alterações teriam que ser introduzidas na estrutura de dados *PCT_ENTRY*, se se pretendesse introduzir o *scheduling* de nível alto (*gestão do ambiente de multiprogramação*)? Justifique adequadamente a sua resposta.

3. Construa a primitiva que actualiza os campos *waittime* e *virttime* das entradas da tabela de controlo de processos referentes a todos os processos não bloqueados

```
void update_temporal_data (void);
```

4. Construa a primitiva *wakeup*.