Parte A (10 valores)

1. Explique porque é que, mesmo no caso dos computadores pessoais, o sistema de operação tem que admitir multiprogramação. Dê pelo menos dois exemplos onde esta necessidade se torna patente.

2. A introdução de critérios de prioridade nos algoritmos de 'scheduling' do processador permite diferenciar a qualidade do atendimento de diferentes classes de processos. Mostre neste contexto quais são as características principais que distinguem as políticas de prioridade estática das de prioridade dinâmica. Dê exemplos de cada uma e explique justificadamente qual escolheria para um sistema multi-utilizador de uso geral.

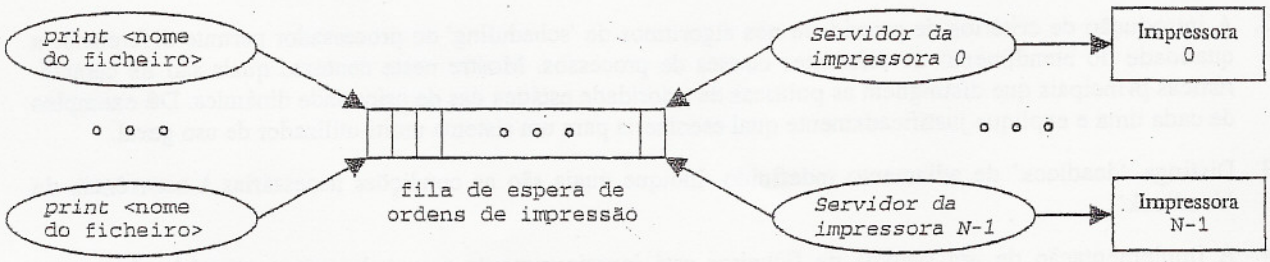
3. Distinga 'deadlock' de adiamento indefinido. Indique quais são as condições necessárias à ocorrência de 'deadlock'.

4. A implementação de um sistema de ficheiros está invariavelmente associada com a memória de massa. Contudo, há pelo menos uma função desempenhada pela memória de massa no âmbito do sistema de operação que pode e deve (por questões de eficácia) ser implementada fora do sistema de ficheiros. Indique-a e dê razões para tal procedimento.

5. Porque é que num ambiente multiprogramado é fundamental desacoplar a comunicação dos processos utilizadores com os diferentes dispositivos de entrada-saída? Justifique a sua resposta descrevendo o modelo de interacção com dispositivos de tipo caracter.

Parte B (10 valores)

A figura abaixo apresenta a organização geral de um sistema de gestão de um parque de impressoras.



O comando de impressão disponibilizado aos utilizadores tem a sintaxe seguinte

```
print <nome do ficheiro> .
```

Este comando associa o segundo argumento da linha com o nome do ficheiro a imprimir e invoca a primitiva

```
char *print_fic (char *nome_fic);
```

que introduz a ordem de impressão na fila de espera e devolve o nome da impressora onde o ficheiro está a ser impresso, ou a indicação de que não há impressoras activas no parque.

O comando termina escrevendo no dispositivo de saída 'standard' o valor devolvido.

O papel dos servidores de impressora, que são processos de sistema, é retirar sucessivamente as ordens de impressão da fila de espera e proceder à impressão do conteúdo do ficheiro na impressora associada.

Como se pode constatar, está-se perante um modelo produtor-consumidor simples em que as sucessivas instanciações do comando `print <...>` constituem os processos produtores e os servidores das impressoras constituem os processos consumidores.

A interacção estabelecida supõe a existência de uma memória de tipo FIFO como meio de comunicação (fila de espera de ordens de impressão), de dois pontos de sincronização para cada processo produtor

- os processos produtores bloqueiam quando a fila de espera de ordens de impressão está cheia;
- os processos produtores bloqueiam enquanto aguardam a indicação da impressora onde a impressão está a ser efectuada;

e de um ponto de sincronização para cada processo consumidor

- os processos consumidores bloqueiam enquanto não há ordens de impressão.

Admita que foram definidas as estruturas de dados

Ordem de impressão

```
typedef struct
        { char *nomefic,      /* nome do ficheiro que contém a informação */
                                                        /* a ser impressa */
          unsigned int imp_ind,  /* n. de ordem da impressora que está a */
                                                    /* ser usada na impressão */
                       espera;    /* identificação do semáforo de espera */
                                                    /* pelo nome da impressora */
        } PRINT_ORD;
```

Memória de tipo FIFO

```
#define   K    20
typedef struct
        { PRINT_ORD *ord[K];  /* caracterização da área de armazenamento */
          unsigned int pin,        /* ponto de inserção do próximo valor */
                       pout;       /* ponto de retirada do próximo valor */
        } FIFO;                              /* implementação semi-estática */
```

Fila de espera de ordens de impressão

```
typedef struct
        { FIFO mem;                                  /* área de armazenamento */
          unsigned int acesso;  /* identificação do semáforo de garantia */
                                          /* de acesso com exclusão mútua */
        } WAIT_QUEUE;
```

Controlo de impressão

```
typedef struct
        { WAIT_QUEUE fesp;                                  /* fila de espera */
          unsigned int ha_espaco,          /* identificação do semáforo de */
                        /* indicação de que há espaço na fila de espera */
                       ha_trabalho;        /* identificação do semáforo de */
                        /* indicação de que há impressões a serem feitas */
        } PRINT_CONT;
```

Caracterização de impressora

```
typedef struct
        { unsigned int stat;              /* estado da impressora: 0-ACTIVA */
                                                        /* 1-EM MANUTENÇÃO */
          char *nome;                      /* identificação da impressora */
        } IMP_DESC;
```

Parque de impressoras

```
#define   N    5
typedef struct
        { unsigned int nimp;                    /* n. de impressoras activas */
          IMP_DESC imp[N];,        /* descrição das impressoras existentes */
          unsigned int acesso;  /* identificação do semáforo de garantia */
                                  /* de acesso com exclusão mútua */
        } IMP_POOL;
```

e as variáveis globais descritas abaixo:

```
static PRINT_CONT pct;                          /* controlo de impressão */
static IMP_POOL impp;,                          /* parque de impressoras */
```

Finalmente, as primitivas seguintes estão também disponíveis

Reserva e libertação de espaço em memória dinâmica

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

Inserção e retirada de informação na FIFO

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

Manipulação de semáforos

```
unsigned int sem_create (void);          /* o campo val é colocado a zero */
void sem_destroy (unsigned int sem_id);
void sem_down (unsigned int sem_id);
void sem_up (unsigned int sem_id);
```

Manipulação de strings

```
unsigned int strlen (char *str);
void strcp (char *sourcestr, char *deststr);
```

1. No arranque do sistema têm que ocorrer inicializações. Indique justificadamente com que valores o campo *val* dos semáforos *espera*, *ha_espaco* e *ha_trabalho* devem ser inicializados.
2. O modelo de interacção apresentado constitui uma variante do utilizado no *problema dos barbeiros sonolentos*. Neste caso, contudo, não há risco de adiamento indefinido. Mostre porquê.
3. Construa a primitiva 'fifo_out'.
4. Construa a primitiva 'print_fic'.