Parte A (10 valores)

1. Explique porque é que hoje em dia o sistema de operação dos computadores pessoais supõe, quase invariavelmente, uma organização de memória de tipo memória virtual.

2. A maioria dos processos num sistema computacional têm uma duração limitada: nascem, desenvolvem-se e morrem. Apresente um diagrama de blocos que descreve esta situação. Admita uma política de 'scheduling' do processador num só nível. Mostre ainda, esquematicamente, como, a partir do interpretador de comandos (shell), são lançados novos processos. Tome como exemplo o que se passa em Unix.

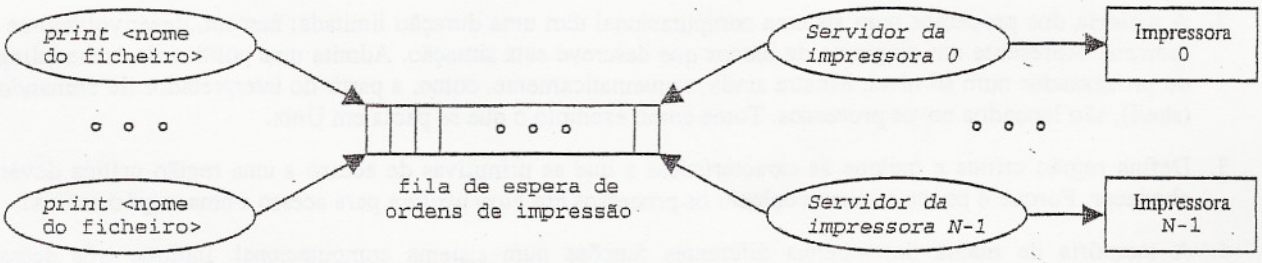
3. Defina região crítica e indique as características a que as primitivas de acesso a uma região crítica devem obedecer. Porque é pouco eficiente colocar os processos em *busy waiting* para acesso a uma região crítica?

4. A memória de massa desempenha diferentes funções num sistema computacional. Indique três dessas funções e explique porque é que é cada vez mais comum construir a memória de massa com múltiplas unidades de disco magnético.

5. Mostre como se estabelece o desacoplamento dos processos utilizadores com os dispositivos de entrada-saída de tipo caracter. Na elaboração da sua resposta, descreva claramente o modelo de interacção e as estruturas de dados mínimas que estão envolvidas.

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 uma situação de erro (não há impressoras activas no parque, por exemplo).

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 estática */
             /* são armazenadas as localizações das ordens de impressão */
```

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);
```

Operações de manipulação de ficheiros

```
int open (char *nome_fic, int flags);
          assuma que o parâmetro flags é sempre igual a O_RDONLY
          a função devolve um file descriptor (fd) em caso de sucesso
               ou -1, em caso de erro
unsigned int read (int fd, void *buffer, unsigned int nbytestoberead);
          a função devolve o n. de bytes que foram efectivamente lidos
void close (int fd);
```

Escrita na impressora

```
void print (unsigned int printnumber, int nbytestobewritten, void *buffer);
```

1. Os processos *servidor de impressora* limitam-se a imprimir o conteúdo do ficheiro cujo nome está na ordem de impressão. Note que eles são processos autónomos do sistema de operação, associados a um utilizador especial, de nome *printspooler*, que tem características absolutamente semelhantes a qualquer outro utilizador convencional do sistema.
   Explique nestas condições que tipo de permissões têm que ter os ficheiros a imprimir para poderem ser processados pelos processos *servidor de impressora*. Justifique claramente a sua resposta. (Tome como exemplo, se quiser, o mecanismo de protecção de ficheiros do Unix).

2. O modelo de interacção apresentado supõe um critério de serviço absolutamente *democrático*. A decisão de impressão de um dado ficheiro baseia-se unicamente na sua posição na fila de espera e, portanto, na ordem de chegada. Imagine que se pretende privilegiar o acesso à impressão de alguns ficheiros, criando-se um sistema de dois níveis de prioridade: alta e baixa, em que os ficheiros de mais alta prioridade serão sempre impressos primeiro. Mostre, justificando, que alterações têm que ser introduzidas para que tal seja possível.

   A primitiva 'print_func' terá nestas circunstâncias a forma seguinte:

```
char *print_fic (char *nome_fic, int prior);
```

em que os valores 0 e 1 da variável *prior* significam, respectivamente, prioridade alta e prioridade baixa.

3. Construa a primitiva '`fifo_in`'.

4. Construa o código do processo *servidor de impressora* que é lançado pelo comando

```
run_printer <número de ordem da impressora> .
```