Sistemas de Operação
Teste 1 *6 de Janeiro de 2006*

Parte A (*10 valores*)

1. Os sistemas de operação de uso geral actuais são tipicamente *sistemas de operação de rede*. Faça a sua caracterização..

2. 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.

3. 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?

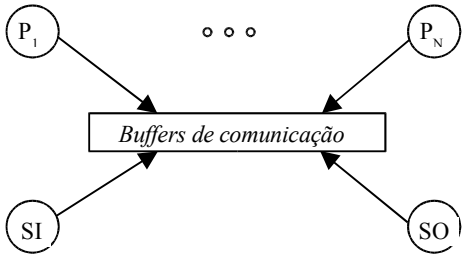
4. Distinga relativamente a um processo *espaço de endereçamento lógico* de *espaço de endereçamento físico*. Que problemas têm que ser resolvidos para garantir que a gestão de memória num ambiente de multiprogramação é eficiente e segura?

5. Distinga as diferentes políticas de *prevenção de deadlock no sentido estrito*. Dê um exemplo ilustrativo de cada uma delas numa situação em que um grupo de processos usa um conjunto de blocos de disco para armazenamento temporário de informação.

Sistemas de Operação
Teste 1 *6 de Janeiro de 2006*

Parte B (*10 valores*)

Considere uma organização de desacoplamento no acesso a *ports série*, associados com dispositivos de entrada-saída *standard*, como é indicado na figura.



Os processos P_1, ... , P_N são *processos utilizador* convencionais que lêem e/ou escrevem valores nos dispositivos de entrada-saída *standard*, enquanto os processos SI e SO são *processos de sistema*, activados por interrupção, que transferem directamente caracteres entre os *ports série* e os *buffers de comunicação*.

Cada utilizador do sistema computacional está ligado a um e um só *terminal* (dispositivo físico de entrada-saída), mas ele ou ela tem a possibilidade de definir até a um máximo de 10 *terminais virtuais* (sessões) sobre esse dispositivo, cada um deles associado à execução de um processo diferente.

Existem para isso os comandos seguintes que são lidos do dispositivo de entrada

ESC C - criação de uma nova sessão (se possível)
ESC N - passagem à sessão seguinte (de um modo rotativo)
ESC D - anulação da sessão actual (com passagem à sessão seguinte de um modo rotativo; se a sessão actual for única não acontece nada).

Os *buffers de comunicação* têm uma organização do tipo indicado abaixo

```
typedef struct
        { FIFO f_in,                          /* fifo de entrada de dados */
               f_out;                           /* fifo de saída de dados */
          int fi_empty, /* id do semáforo de controlo da entrada de dados */
              fo_full;    /* id do semáforo de controlo da saída de dados */
          BOOLEAN tx_idle;         /* sinalização de fim das interrupções
                    associadas com o registo de transmissão do port série */
        } COMM_BUFFER;
```

Existem ainda as estruturas de dados seguintes

Ambiente (associado a cada dispositivo físico de entrada-saída)

```
typedef struct
        { int combid;  /* n.º de ordem do buffer de comunicação da sessão;
                                                        -1 em caso contrário */
          int pid;                            /* id do processo associado */
        } SESSION;

typedef struct
        { int access;        /* id do semáforo de imposição de acesso com
                                exclusão mútua à estrutura de dados interna */
          int userid; /* id do utilizador associado; -1 em caso contrário */
          int sess_act;                /* n.º de ordem da sessão que está
                                  presentemente associada com o port série */
          SESSION sess[10];              /* lista das sessões existentes */
        } IO_ENV;
```

Gestão global dos *buffers de comunicação*

```
typedef struct
        { int access;       /* id do semáforo de imposição de acesso com
                               exclusão mútua à estrutura de dados interna */
          int nt_buff;            /* n. total de buffers de comunicação */
          int n_buff;                        /* n.º de buffers ocupados */
          int buff_map[ML];        /* bitmap da ocupação dos M buffers de
                                      comunicação (ML = M/(8*sizeof(int))) */
        } IO_MANAG;
```

Assumindo que são conhecidas as primitivas

Leitura / escrita de dados no dispositivo virtual de entrada-saída (*processos utilizador*)

```
void readNBytes (int n, char buff[]);
void writeNBytes (int n, char buff[]);
```

Leitura / escrita de dados nos *ports série*

```
void readData (char *carp, int port_id);
void writeData (char car, int port_id);
```

Manipulação de semáforos

```
int semCreate (void);       /* devolve o id do semáforo que foi criado
                                 (o semáforo é inicializado a vermelho) */
void semDestroy (int semid);
void semDown (int semid);
void semUp (int semid);
```

Manipulação do FIFO

```
void fifoInit (FIFO *f);
BOOLEAN fifoFull (FIFO *f);
BOOLEAN fifoEmpty (FIFO *f);
void fifoIN (FIFO *f, char car);
void fifoOUT (FIFO *f, char *carp);
```

Funções auxiliares

```
int getPortId (void);         /* devolve o número do port associado ao
                                 utilizador corrente; -1 caso não exista */
int getBuffId (void);       /* devolve o número do buffer associado ao
                                 processo corrente; -1 caso não exista */
int allocBuffer(void);       /* devolve o n.º de ordem de um buffer de
                  comunicação ou, -1, se não houver qualquer disponível */
void freeBuffer(int buffid);   /* liberta o buffer de comunicação cujo
                                                  n. de ordem é fornecido */
int getpid(void);                /* devolve o id do processo corrente */
void refreshScreen(int sessionid);  /* imprime no dispositivo de saída
                      standard toda a informação previamente impressa */
```

e que foram definidas as variáveis

```
#define T   8              /* n. de dispositivos físicos de entrada-saída */
#define M   64                            /* n. de buffers de comunicação */
#define ML  M/(8*sizeof(int))
static IO_MANAG man;
static IO_ENV env[T];
static COMM_BUFFER buf[M];
```

responda às questões seguintes:

a) Defina o tipo de dados FIFO como uma memória estática com capacidade máxima de armazenamento de K bytes.

b) Explique qual é o papel desempenhado pelos campos `fi_empty` e `fo_full` do tipo de dados `COMM_BUFFER`. A que valores devem ser inicializados os campos `val` dos semáforos associados.

c) Construa a função `readNBytes` que permite ao processo utilizador ler dados do dispositivo virtual de entrada-saída.

d) Construa a função `Next_Session` que é chamada quando se pretende passar à sessão seguinte associada com o terminal. O seu interface é

```
void Next_Session(void);
```