Teste de Setembro 5 de Setembro de 2001

Comece por escrever no cabeçalho do papel de prova a sua identificação (nome e número mecanográfico) em duas folhas separadas. A primeira conterá as suas respostas à parte A e, a segunda, à parte B.

Tenha em conta que:

- deve responder sempre de uma forma precisa e concisa;
- pode alterar a ordem das questões, desde que o sinalize de um modo claro.

Parte A – (10 valores)

(1 valor por questão)

1 - O que é que caracteriza um sistema operativo como um sistema operativo de utilizadores múltiplos?

2 - Porque é que a ocorrência de 'busy waiting' é nociva nas soluções de acesso a uma região crítica com exclusão mútua?

3- O que é uma 'flag' de 'locking'? Mostre como é que elas podem ser implementadas usando instruções de 'test-and-set' ('read-modify-write').

4- A que características gerais deve obedecer a política de 'scheduling' do disco?

5- Distinga 'scheduling preemptive' de 'scheduling non-preemptive'.

6- Qual é a função de um 'device-driver'?

7- Em que consistem as políticas de prevenção de 'deadlock' no sentido lato. Dê um exemplo de uma delas aplicada ao problema do jantar dos filósofos

8- O que é uma organização de memória virtual? Que tipo de exigências de 'hardware' são necessárias à sua implementação.

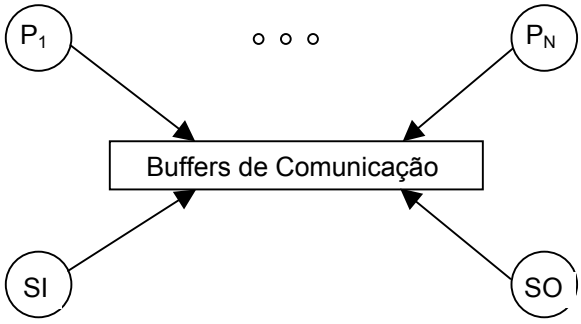
9- O que é o 'working set' de um processo. Explique como é que ele pode ser usado na implementação de uma política eficaz de substituição de blocos.

10- Indique as principais características do modelo 'cliente-servidor'. Tome como base da sua explicação um servidor de ficheiros.

Parte B – (10 valores)

(2,5 valores por questão)

Considere um sistema de desacoplamento de acesso a 'ports' série, ligados a dispositivos de entrada-saída 'standard', como é indicado na figura abaixo.



Os processos P_1, ..., P_N são processos de utilizador convencionais que pretendem ler e/ou escrever dados nos dispositivos de entrada-saída, enquanto os processos SI e SO são processos de sistema, activados por interrupção, que transferem directamente 'bytes' 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 seu processo. Para isso, existem os comandos seguintes:

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 liberta 'port').

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,       /* identificação do semáforo indicativo
                                 de não existência de dados de entrada */
              fo_full;        /* identificação do semáforo indicativo
                                 de que o 'fifo' de dados de saída está
                                 cheio */
          BOOLEAN tx_idle;    /* sinalização de fim das interrupções
                                 associadas à transmissão */
        } COMM_BUFFER;
```

Existem ainda as estruturas de dados seguintes:

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

```
typedef struct
        { int access;       /* identificação do semáforo de imposição
                               de acesso com exclusão mútua à estrutura
                               de dados interna */
          int userid;       /* identificação do utilizador associado */
          int sess_act;     /* n.º de ordem da sessão que está presente-
                               mente associada ao dispositivo de I/O */
          struct
          { int combid;   /* n.º de ordem do 'buffer' de comunicação
                               referente à sessão; -1 em caso contrário */
            int pid;      /* identificação do processo associado */
          } sess[10];     /* lista das sessões existentes */
        } IO_ENV;
```

Gestão global dos 'buffers' de comunicação

```
typedef struct
        { int access;       /* identificação do semáforo de imposição
                               de acesso com exclusão mútua à estrutura
                               de dados interna */
          int nt_buff;      /* n.º 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 descritas a seguir:

Leitura / escrita de dados (processos utilizador)

```
void Read_Byte (char *carp, int mode); /* mode=0 não bloqueante, */
void Write_Byte (char car, int mode);  /*     =1 bloqueante      */
```

Inibição / desinibição de interrupções

```
void Interrupt_Enable (void);
void Interrupt_Disable (void);
```

Manipulação de semáforos

```
int Sem_Creat (void);      /* devolve o identificador do semáforo que
              foi criado (o semáforo é inicializado a vermelho) */
void Sem_Destroy (int sid);
void Sem_Down (int sid);
void Sem_Up (int sid);
```

Manipulação do 'FIFO'

```
void Fifo_init (FIFO *f);
BOOLEAN Fifo_Full (FIFO *f);
BOOLEAN Fifo_Empty (FIFO *f);
void Fifo_IN (FIFO *f, char car);
void Fifo_OUT (FIFO *f, char *carp);
```

Leitura / escrita de dados dos 'ports' série

```
void Read_Port (char *carp, int port_id);
void Write_Port (char car, int port_id);
```

Funções auxiliares

```
int  Get_Port_Id  (void);  /* devolve o número do port associado ao
                              utilizador corrente; -1 caso não exista */
int  Get_Buff_Id  (void);  /* devolve o número do buffer associado ao
                              processo corrente; -1 caso não exista */
int Aloc_Buffer(void);    /* devolve o n.º de ordem de um 'buffer' de
              comunicação ou, -1, se não houver qualquer disponível */
int getpid(void);  /* devolve o identificador do processo corrente. */
```

e que foram definidas as variáveis

```
#define T   8          /* n.º de dispositivos físicos de entrada-saída */
#define M   64         /* n.º 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) Explique a necessidade dos semáforos `access` nas estruturas `IO_ENV` e `IO_MANAG`.

b) Construa a função de atendimento de interrupção que transfere um byte do 'port' para o 'buffer' de comunicação, cuja interface é:

```
void Transfer_Byte_In(int port_id);
```

em que `port_id` é o n.º do dispositivo físico de entrada-saída. Considere que durante a execução das rotinas de atendimento todas as interrupções estão inibidas.

c) Construa a função `Read_Byte` que permite a um processo do utilizador ler do 'buffer' de comunicação respectivo em modo bloqueante ou não bloqueante.

d) Construa a função `End_Session` que é chamada quando se pretende terminar a sessão associada ao processo corrente. A função devolve o número da sessão libertada e -1 em caso de erro, a sua interface é:

```
int End_Session(int port_id);
```