Justifique todas as suas respostas.

I

1. **[1,5 Valores]** Explique a necessidade da existência de instruções privilegiadas no suporte à realização do sistema operativo.
2. **[1,5 Valores]** Discuta as implicações na variação do *time quantum* adoptado pelo sistema operativo no suporte ao escalonamento de processos/tarefas.
3. **[1,5 Valores]** A familia de sistemas operativos Windows permite a atribuição de prioridades às tarefas. Considere um programa composto por várias tarefas, que competem por um recurso e onde apenas uma pode estar dentro do recurso. Um programador afirmou que para garantir o acesso ao recurso por parte de uma tarefa era apenas necessário subir a prioridade dessa tarefa. Comente a afirmação do programador.

II

1. **[1,5 Valores]** Compare os mecanismos de sincronismo, da `API Win32`, `Critical Section` e `Mutex`. Discuta, de forma justificada, as vantagens, desvantagens e eficiência das duas alternativas.

2. **[1,5 Valores]** Suponha que em ambiente `Win32` necessita de disponibilizar um conjunto de funções, que mantêm estado entre diferentes invocações (recorrem à utilização de variáveis estáticas e globais). Apresente uma solução, utilizando os mecanismos disponibilizados pela `API Win32`, que permita utilizar essas funções de forma correcta num ambiente de várias tarefas.

3. **[2 Valores]** Diga qual o resultado (incluindo mensagens apresentadas na consola) da execução do programa apresentado.

```
int* p;

DWORD filtro1() {return EXCEPTION_CONTINUE_SEARCH;}

DWORD filtro2() {return EXCEPTION_EXECUTE_HANDLER;}

int _tmain(int argc, TCHAR* argv[]) {
  __try {
    __try {
      __try {
        p = NULL;
      }
      __finally { _tprintf( TEXT("%d"), (*p) ); }
    }
    __except ( filtro1() ) { _tprintf(TEXT("Handler 1"));}
  }
  __except ( filtro2() ) { (*p) = 5; }
  _tprintf( TEXT("Ola") );
  return 0;
}
```

III

1. Considere uma arquitectura de suporte à gestão de memória paginada, a dois níveis, utilizando um endereçamento virtual de 32 bits e páginas de 4 KBytes. Os processos P1 e P2, em execução nesse sistema, possuem as estruturas de suporte à paginação com as tabelas de páginas apresentada na figura 1. Cada tabela ocupa a dimensão de uma página e cada entrada nas tabelas ocupa a dimensão de 4Bytes.

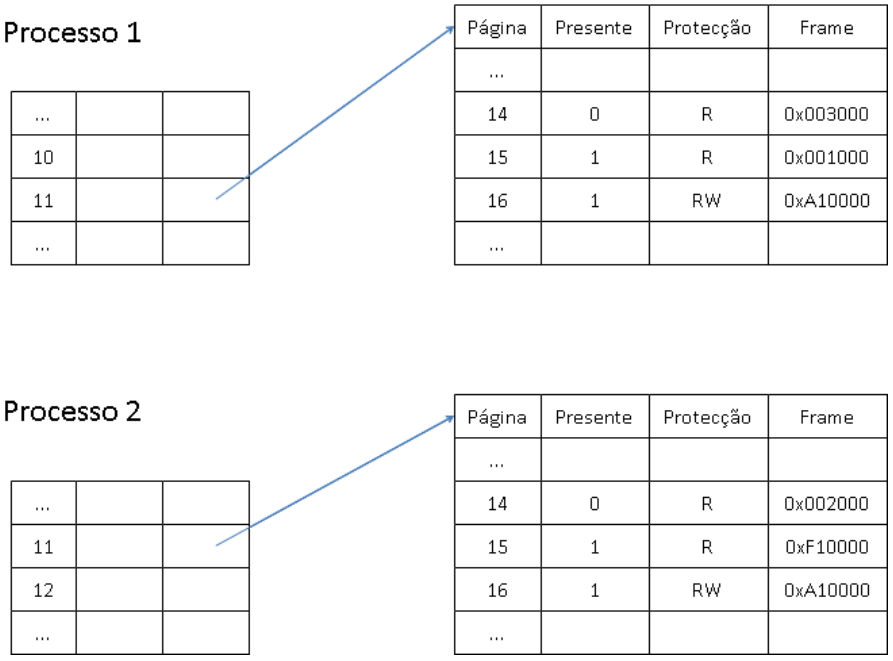

Figura 1 – Estruturas de suporte à paginação dos processos P1 e P2

a) **[2 Valores]** Suponha que em ambos os processos o endereço virtual `0x02C104FA` está associado a um apontador para inteiro (por exemplo `ptrInt`). Indique o valor da posição apontada por `ptrInt` após a execução sequencial (P1 seguido de P2) dos seguintes troços de códigos:

P1	P2

`...`
`(*ptrInt) = 5;`
`...`
	`...`
	`++(*ptrInt);`
	`printf("%d", (*ptrInt) );`
	`...`

b) **[1,5 Valores]** Suponha que processo P2 efectua uma escrita no endereço `0x02C0E000`. Indique todas as acções realizadas.

2. **[1,5 Valores]** Indique, justificando convenientemente as suas afirmações, as razões que conduzem à adopção de estruturas de suporte à paginação (tabelas de páginas) organizadas em múltiplos níveis que implicam um maior número de acessos suplementares à memória para a conversão do endereço virtual no endereço físico.

IV

1. **[2,5 Valores]** Considere que se pretende determinar a média de espectadores em todos os jogos do Mundial 2010. Assuma, que tem disponível a função: `int ObterNumeroEspectadores(char *nomeEstadio);` que quando invocada vai contactar o estádio indicado iniciando um processo de contagem de espectadores que é síncrono e, previsivelmente, demorado. Os nomes de todos os estádios estão descritos num *array* `nomeEstadios`. Apresente um programa que imprima na consola a média de espectadores, em que a obtenção da média de espectadores pelos estádios seja realizada em paralelo. Indique, adicionalmente, as principais vantagens deste tipo de implementação concorrente.

2. **[3 Valores]** Pretende-se simular o comportamento do ruído produzido pelas claques das equipas de futebol que utilizam vuvuzelas quando assistem aos treinos. Para esse efeito pretende-se que desenvolva um mecanismo de sincronismo, que implementa a interface em anexo, que sera utilizado no contexto de uma aplicação que suporta a simulação. Nessa simulação cada adepto será representado por uma tarefa. A simulação assume que existem 3 tipos de adeptos:

```
class IClaqueApoio {
public:
  virtual int Entrar_Com_Vuvuzela()= 0;
  virtual int Entrar_E_Aluga_Vuvuzela()= 0;
  virtual int Entrar()= 0;

  virtual int Sair_Entrega_Vuvuzela ()= 0;
  virtual int Sair_Vuvuzela()= 0;
  virtual int Sair()= 0;
};
```

   - os querem utilizar a sua vuvuzela;
   - os querem utilizar uma vuvuzela mas não têm;
   - os que querem assistir ao treino sem vuvuzela;

   Na entrada do estádio existe uma banca de apoio às claques (mecanismo de sincronismo a implementar) que aluga vuvuzelas aos adeptos que as pretendam utilizar. Os adeptos só podem entrar no estádio, onde se realizam os treinos, verificando-se as seguintes condições:

   - Lotação do estádio não alcançada
   - Máximo de vuvuzelas no estádio inferior a um limite (para os adeptos que utilizem vuvuzelas)
   - Existência de vuvuzelas no quiosque de aluguer (para os adeptos que pretendam alugá-las)

   Apresente uma implementação do mecanismo de sincronismo tendo por base o mecanismo de sincronismo semáforo ().

Nuno Oliveira e Carlos Gonçalves

ANEXO

```
BOOL WINAPI CreateProcess(
  __in          LPCTSTR lpApplicationName,
  __in_out      LPTSTR lpCommandLine,
  __in          LPSECURITY_ATTRIBUTES
                              lpProcessAttributes,
  __in          LPSECURITY_ATTRIBUTES
                               lpThreadAttributes,
  __in          BOOL bInheritHandles,
  __in          DWORD dwCreationFlags,
  __in          LPVOID lpEnvironment,
  __in          LPCTSTR lpCurrentDirectory,
  __in          LPSTARTUPINFO lpStartupInfo,
  __out         LPPROCESS_INFORMATION
                              lpProcessInformation
);

VOID WINAPI ExitProcess(
  __in          UINT uExitCode
);

BOOL WINAPI GetExitCodeProcess(
  __in          HANDLE hProcess,
  __out         LPDWORD lpExitCode
);

HANDLE WINAPI CreateThread(
  __in    LPSECURITY_ATTRIBUTES lpThreadAttributes,
  __in    SIZE_T dwStackSize,
  __in    LPTHREAD_START_ROUTINE lpStartAddress,
  __in    LPVOID lpParameter,
  __in    DWORD dwCreationFlags,
  __out   LPDWORD lpThreadId
);

VOID WINAPI ExitThread(
  __in          DWORD dwExitCode
);

BOOL WINAPI GetExitCodeThread(
  __in          HANDLE hThread,
  __out         LPDWORD lpExitCode
);

DWORD WINAPI WaitForSingleObject(
  __in          HANDLE hHandle,
  __in          DWORD dwMilliseconds
);

DWORD WINAPI WaitForMultipleObjects(
  __in          DWORD nCount,
  __in          const HANDLE* lpHandles,
  __in          BOOL bWaitAll,
  __in          DWORD dwMilliseconds
);

void WINAPI InitializeCriticalSection(
  __out        LPCRITICAL_SECTION lpCriticalSection
);

void WINAPI EnterCriticalSection(
  __in_out     LPCRITICAL_SECTION lpCriticalSection
);

void WINAPI LeaveCriticalSection(
  __in_out     LPCRITICAL_SECTION lpCriticalSection
);

void WINAPI DeleteCriticalSection(
  __in_out     LPCRITICAL_SECTION lpCriticalSection
);

HANDLE WINAPI CreateMutex(
  __in     LPSECURITY_ATTRIBUTES lpMutexAttributes,
  __in     BOOL bInitialOwner,
  __in     LPCTSTR lpName
);

BOOL WINAPI ReleaseMutex(
  __in          HANDLE hMutex
);

HANDLE WINAPI CreateSemaphore(
  __in LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
  __in LONG lInitialCount,
  __in LONG lMaximumCount,
  __in LPCTSTR lpName
);

BOOL WINAPI ReleaseSemaphore(
  __in          HANDLE hSemaphore,
  __in          LONG lReleaseCount,
  __out         LPLONG lpPreviousCount
);

HANDLE WINAPI CreateEvent(
  __in     LPSECURITY_ATTRIBUTES lpEventAttributes,
  __in     BOOL bManualReset,
  __in     BOOL bInitialState,
  __in     LPCTSTR lpName
);

BOOL WINAPI SetEvent(
  __in          HANDLE hEvent
);

BOOL WINAPI ResetEvent(
  __in          HANDLE hEvent
);
```