Tipo de Prova: sem consulta **Modelo de uma prova de exame**
Duração: 2 horas **7.Janeiro.2009**
Cotação máxima: 20 valores (10 da nota final!)
Estrutura da prova: Parte I (escolha múltipla, 25%); Parte II (mais convencional, 50%); Parte III (predominantemente de aplicação, 25%)

PARTE I: Escolha múltipla [5 val.]

Utilização: para cada pergunta só há uma resposta correcta; indique-a (com a letra correspondente) na folha de respostas, completando uma tabela semelhante à que se segue; se não souber a resposta correcta, nada preencha ou faça um traço nessa alínea.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Cotação: cada resposta certa vale 1 ponto; cada resposta errada vale – 0,5 ponto (note o sinal menos!); cada resposta ambígua, ininteligível ou não assinalada vale 0 ponto. O total é 15 pontos (que irão equivaler a 5 valores da nota final da prova).

1. As variáveis de ambiente costumam ser utilizadas para fornecer

A) informação comum a um conjunto de programas.
B) informação específica a cada instância de um programa.
C) informação opcional a um determinado tipo de programas.

2. O *software* de um sistema computacional é constituído por dois tipos de componentes:

A) programas de compilação e programas de edição.
B) programas de sistema e programas de aplicação.
C) programas de programação e programas de utilização.

3. Um sistema operativo é um "simulador de máquina virtual". Isso quer dizer que:

A) facilita a programação de aplicações.
B) facilita a utilização de aplicações.
C) facilita o teste de aplicações.

4. Concorrência de processos quer dizer

A) execução "simultânea" de vários processos, provavelmente competindo pelos mesmos recursos.
B) execução simultânea de vários processos em vários processadores.
C) execução "simultânea" de vários processos de um mesmo utilizador.

5. Os três principais estados de um processo reconhecido pelo sistema operativo são:

A) prontidão, bloqueamento, execução.
B) dormência, bloqueamento, execução.
C) prontidão, bloqueamento, sinalização.

6. Em Unix, quando um processo pretende executar um programa diferente do inicial, terá sempre de invocar a chamada `fork()`.

A) Sim, essa é uma característica da gestão de processos em Unix. Depois do `fork()`, poderá invocar-se `execve()` ou uma variante.
B) Não, se se pretender descontinuar o processo inicial. Basta, então, invocar-se `execve()` ou uma variante.
C) Não é necessário usar `fork()`, basta invocar a chamada ao sistema `exec()`.

7. Deve optar-se por programação com *threads* relativamente à programação por processos,

A) quando se pretender escrever um servidor.
B) quando se pretender facilitar a troca de informação entre as partes do programa.
C) quando se pretender aumentar o isolamento entre as partes do programa.

8. Uma condição de competição (*race condition*) surge quando dois ou mais processos

A) tentam partilhar de forma eficiente um mesmo conjunto de recursos.

B) tentam partilhar de forma sincronizada um mesmo conjunto de recursos.

C) tentam partilhar de forma arbitrária um mesmo conjunto de recursos.

9. Um processo fora da zona crítica não pode impedir outro de entrar na região.

A) Sim, pode, reservando-se, desde logo, o acesso futuro.

B) Sim e até é conveniente nos sistemas de exclusão mútua.

C) Não, não pode, pois estaria a tornar o sistema ineficiente e com tendência a encravar.

10. Uma das estratégias para se lidar com encravamentos ao nível do sistema operativo preconiza

A) detectar-se o problema e tentar resolvê-lo, eventualmente terminando alguns processos.

B) evitar-se o problema, planeando sempre tudo com os utilizadores.

C) prevenir-se o problema, negando o acesso ao sistema a alguns dos utilizadores.

11. Considere um sistema em que a memória, numa dada ocasião, apresenta os seguintes espaços vazios (*holes*), por ordem crescente de posição: 10K, 4KB, 20KB, 18KB, 7KB e 9KB. Nessa situação, foram feitos 3 pedidos sequenciais de memória: 12KB, 10KB e 9 KB, após o que a lista ordenada de espaços vazios ficou: 4KB, 9KB 7 KB e 9 KB. Qual foi o método de alocação utilizado?

A) Primeiro a servir (*first fit*).

B) Melhor a servir (*best fit*).

C) Pior a servir (*worst fit*).

12. Um sistema de memória virtual:

A) tem de utilizar paginação com segmentação.

B) pode utilizar paginação com segmentação.

C) não pode utilizar paginação com segmentação.

13. Os discos rígidos utilizados nos PCs modernos costumam

A) ter uma componente de *software* de controlo embutida.

B) poder aceder facilmente ao processador principal.

C) ter uma interface USB.

14. Muitos dos pedidos a dispositivos de E/S são bloqueantes por não poderem ser normalmente atendidos com a rapidez necessária à operação do processador. Por isso

A) é importante que se faça uso de transferências directas para a memória (DMA) via *threads*.

B) é importante que o processo requisitante faça uma espera activa até chegar a resposta.

C) é importante que uma interrupção avise o sistema quando a resposta chegar, a fim de o processo requisitante ser acordado.

15. Os directórios, ao contrário dos ficheiros regulares, costumam ser implementados por uma estrutura interna que traduz a sua estrutura de dados.

A) Não, porque os ficheiros regulares também costumam ser implementados pelo sistema operativo de forma estruturada.

B) Não, porque a implementação de directórios costuma ser semelhante à de ficheiros regulares, apenas as chamadas ao sistema são diferentes.

C) Sim, por isso é que existem chamadas ao sistema específicas de directórios, e.g. `opendir()`.

PARTE II: Mais convencional [10 val.]

Utilização: cada resposta, a apresentar na folha de respostas, deve ser (brevemente) justificada.
Cotação: mostrada em cada pergunta.

1. [0,5 val.] Comente a veracidade da seguinte afirmação: «*O utilitário make é um interpretador de programas especiais (makefiles), só é utilizável em sistemas do tipo Unix e apenas para a compilação de programas.*»

2. [1 val.] Quais das seguintes instruções só devem poder ser executadas no "modo de supervisão" (*kernel mode*)?

a) desactivação de interrupções;
b) leitura do relógio interno;
c) escrita do relógio interno.

3. **[1 val.]** Diga se cada programador pode escrever o código para guardar e repor o estado de um processo quando este cede e retoma o uso do CPU.

4. **[1 val.]** Explique o conceito de *thread*, distinguindo-o de *processo*.

5. **[1,5 val.]** Num sistema operativo em que o escalonamento do processador é do tipo "à vez" (*round-robin*) correm vários processos, maioritariamente *I/O bound*. Admitindo que: Q = *quantum* atribuído a cada processo; C = duração média da comutação de contexto; B = duração média dos picos de processamento (*CPU-bursts*) dos processos, discuta o efeito da fixação dos seguintes valores para Q no desempenho global do sistema:
   a) $Q = C$;
   b) $Q \approx B$;
   c) $Q > B$;

6. **[1 val.]** Um processo envia um sinal a outro processo mediante a chamada `kill()`. O que poderá suceder ao processo alvo?

7. **[1 val.]** A solução de acesso a zonas críticas exemplificada no segmento de código seguinte tem um problema. Identifique-o.

```
        int i;      // process number: 0 or 1
        int turn;   // shared variable for access to critical region
enter_region(i)
        { while (turn != i) ; } // wait turn
leave_region(i)
        { turn = 1-i; }         // give turn away
```

8. **[1 val.]** Apresente uma vantagem da estratégia de comunicação síncrona (~ bloqueante) relativamente à assíncrona e vice-versa.

9. **[1 val.]** Considere um sistema que implementa memória paginada com (apenas!) 4 molduras (*page frames*). A tabela mostra a informação de paginação acessível ao sistema operativo na altura em que uma decisão de libertação de *frame* tem de ser tomada.
   Qual das páginas mostradas dará lugar à nova página, quando o algoritmo usado pelo sistema operativo para a substituição for:
   a) o da "página não usada recentemente" (NRU, Not Recently Used Page);
   b) o do "relógio" (~ FIFO com 2ª chance)?

quadro	**pág.**	**carregada há (*clock ticks*)**	**referenciada há (*clock ticks*)**	**R flag**	**M flag**
0	35	126	280	1	0
1	12	230	265	0	1
2	9	140	270	0	0
3	87	110	285	1	1

10. **[1 val.]** O programa de particionamento de discos rígidos usado em Unix (`fdisk`) revelou a seguinte informação sobre o disco de um computador: «`Disk /dev/hda: 20.0 GB, 20020396032 bytes; 16 heads, 63 sectors/track, 38792 cylinders`».
    a) Qual é o tamanho dos sectores, em *bytes*?
    b) A geometria do disco aparenta ser bem definida e uniforme. Provavelmente, poder-se-ia construir um *driver* do disco capaz de controlar até a movimentação e posicionamento físicos das cabeças de leitura de forma a se poder ler um sector específico. Está correcta esta afirmação?

PARTE III: Predominantemente de aplicação [5 val.]

Utilização: cada resposta, a apresentar na folha de respostas, deve ser (brevemente) justificada.
Cotação: mostrada em cada pergunta.

1. [2 val.] O programa que se apresenta mais abaixo, `proc.c`, cria um novo processo ao executar.

a) Indique a linha de código onde isso acontece.
b) Apresente a sequência completa de números de linhas cujo código vai ser executado pelo novo processo.
c) Apresente toda a informação que irá ser impressa no terminal referente à variável `var` no caso de `fork()` não falhar.
d) Explique se o acesso à variável global `var` necessita de mecanismos de sincronização; se necessitar, faça as necessárias alterações ao código para que passe a funcionar sem perigo de situações de competição; se não necessitar, proponha uma situação em que o acesso a `var` exija tais mecanismos e apresente-os.

```
1.   /* proc.c */
2.   #include <stdio.h>
3.   #include <unistd.h>
4.   #include <stdlib.h>
5.   int var = 4;
6.
7.   int main()
8.   {
9.   printf("\nMeu PID = %d.\n", getpid());
10.   switch (fork())
11.       {
12.       case -1:
13.             perror ("fork()");
14.             exit (1);
15.       case 0:
16.             printf("\nMeu PID = %d.\n", getpid());
17.             var++;
18.             break;
19.       default:
20.             printf("\nMeu PID = %d.\n", getpid());
21.             var--;
22.             break;
23.       }
24.   printf("\nvar = %d.\n", var);
25.   return 0;
26.   }
```

2. [3 val.] O programa que se apresenta mais abaixo, `thr.c`, ilustra a criação e terminação de *threads*.

a) Escreva um comando que permita compilar e construir o executável `thread`, correspondente ao código `thr.c`.
b) Qual é o número total de *threads* que fazem parte do processo? Diga qual é a rotina que cada um executa.
c) Escreva o código que falta na linha 11, por forma a que seja impressa no ecrã uma mensagem mostrando o valor do parâmetro pela rotina `rot()`.
d) Escreva o código que falta na linha 32, por forma a que seja impressa no ecrã uma mensagem mostrando a identificação de um *thread* e o valor do seu código de terminação.
e) Apresente uma situação alternativa à passagem de informação de terminação dos *threads*, mas que não recorra à variável global `codes`.
f) [1 val.] Suponha que os *threads* criados em `main()` incrementavam uma variável global `val`, inicializada a zero, até que o seu valor atingisse `500000`. Nessa altura, imediatamente todos os *threads* começariam a decrementar a variável global, até o valor inicial ser atingido, e os *threads* terminariam de executar. Altere o programa `thr.c`, por forma a cumprir o especificado.

```
1. /* thr.c */
2. #include <pthread.h>
3. #include <stdio.h>
4. #include <stdlib.h>
```

```
5. #define NTHREADS 3
6.
7. pthread_t codes[NTHREADS];
8.
9. void *rot(void *pn)
10. {
11. /* a completar */
12. pthread_exit(&codes[*(int *) pn]);
13. }
14.
15. int main()
16. {
17.      int i;
18.      int arg[NTHREADS];
19.      pthread_t ids[NTHREADS];
20.
21. for(i=0; i< NTHREADS; i++)
22.      {
23.      arg[i] = i;
24.      if (pthread_create(&ids[i], NULL, rot, (void *) &arg[i]) != 0)
25.            exit(-1);
26.      }
27.
28. for(i=0; i< NTHREADS; i++)
29.      {
30.            pthread_t *code;
31.      pthread_join(ids[i], (void *) &code);
32.      /* a completar */
33.      }
34. return 0;
35. }
```

JMMC, JFSC