Tipo de Prova: sem consulta **Exame de Recurso da Época Normal**
Duração: 2 horas **18.Fevereiro.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 sua 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. Os parâmetros da linha de comando de um executável costumam ser utilizados para fornecer

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

2. Um interpretador de comandos (*shell*):

A) tem, necessariamente, de ser usado através de um terminal.
B) permite a comunicação gráfica com um sistema operativo.
C) permite ao utilizador comunicar com um sistema operativo.

3. Recomenda-se que a arquitectura de um sistema operativo moderno seja:

A) modular e com um núcleo pequeno, executando no modo de supervisão.
B) modular e com um núcleo abarcando todos os serviços, executando no modo de supervisão.
C) monolítica e com um núcleo cujas partes inter-comunicam por mensagens.

4. Um processo executa num computador moderno como que de uma forma contínua, ocupando sempre o processador.

A) Sim, do ponto de vista do processo.
B) Sim, mas só no caso dos processos especiais.
C) Sim, mas apenas quando só há um utilizador.

5. Um processo reconhecido pelo sistema operativo, pode passar

A) do estado de prontidão ao de execução.
B) do estado de bloqueamento ao de execução.
C) do estado de prontidão ao de bloqueamento.

6. Na utilização de `pthread_create()`, tem de se especificar uma função onde irá começar a execução do novo *thread*.

A) É certo, mas essa função tem de ter um protótipo do tipo: `void* function (void*)`.
B) É certo, mas essa função apenas necessita de receber como argumento um apontador para `void`.
C) É certo, mas não há qualquer restrição ao tipo do valor de retorno da função.

7. Em geral, só haverá problemas de competição (*races*) em processos concorrentes

A) pertencentes a um mesmo utilizador; nos outros casos, o sistema operativo dá a necessária protecção.
B) do tipo multi-*thread*, devido à partilha de segmentos de memória!
C) cujo código foi mal planeado, mal programado ou mal sincronizado.

8. Um programa não deve ter mais de uma zona crítica.

A) Sim, uma só zona crítica é o máximo admissível.
B) Não há limite para o número de zonas críticas, mas deverá ser o menor possível.
C) Não há limite para o número de zonas críticas, mas só no caso de se usarem monitores.

9. Uma situação de encravamento resolve-se por terminação de todos os processos envolvidos.

A) Sim, apesar de poder bastar terminar apenas um deles.

B) Não, basta remover um deles da fila de execução.

C) Não, pode esperar-se um tempo suficiente para que o engarrafamento acabe.

10. Uma entrada de uma tabela de páginas de um sistema de memória virtual contém, entre outros dados:

A) um número de moldura, um bit de presença e um bit de alteração.

B) um número de moldura, um número de página e um bit sinalização.

C) um número de página, um bit de falha de página e um bit sinalização.

11. Num sistema de memória virtual, a escolha do algoritmo de substituição de página é especialmente importante por questões de

A) economia do espaço de memória disponível.

B) melhoria do desempenho do sistema.

C) minimização do desgaste das cabeças de leitura do disco.

12. O controlo da informação sobre nomes e permissões de acesso a dispositivos de E/S é efectuado pelo

A) *device driver* dos dispositivos.

B) *software* independente dos dispositivos.

C) *software* ligado ao equipamento.

13. A formatação de baixo-nível de discos rígidos basicamente consiste

A) em se efectuar a laser infravermelho uma série de traços na camada superior do disco magnético.

B) em se escrever determinada informação magnética nas partes periféricas e centrais do disco.

C) em se efectuar a escrita do preâmbulo (ou cabeçalho) de cada sector do disco magnético.

14. Os ficheiros costumam ser guardados nos discos em blocos de igual tamanho. O valor do bloco afecta a rapidez de leitura e a eficiência de armazenamento:

A) Sim, pois grandes blocos melhoram tal eficiência.

B) Sim, pois grandes blocos melhoram tal rapidez.

C) Não, tais parâmetros não são muito afectados pelo valor do bloco.

15. Os directórios, tal como os ficheiros regulares, costumam ser implementados por uma estrutura interna que traduz a sua estrutura de dados.

A) Não, os ficheiros regulares não costumam ser implementados de forma estruturada.

B) Não, os directórios não costumam ser implementados de forma estruturada.

C) Sim, por isso é que ambos utilizam a chamada ao sistema `open()`.

PARTE II: Mais convencional [10 val.]

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

1. [1 val.] Apresente uma vantagem e uma desvantagem da utilização da técnica de ligação (*linking*) dinâmica de programas relativamente à ligação estática.

2. [1 val.] Explique a diferença em programa e processo.

3. [1 val.] Um programa multi-*thread* de processamento intensivo foi posto a correr em duas máquinas: uma, mono-processador; outra, multi-processador. Curiosamente, verificou-se que a rapidez de execução do programa era semelhante. Tente explique o facto.

4. [1 val.] Num sistema POSIX, diversos processos escrevem mensagens de igual tamanho (512 B) num canal do tipo FIFO. Diversos outros processos lêem tais mensagens do "outro extremo" do FIFO.

a) Que cuidados deverão ter tido os programadores das aplicações que deram origem aos processos para evitar situações de competição no acesso ao FIFO?

b) Que cuidados deverão ter tido os programadores do sistema com o mesmo objectivo?

5. [1 val.] Uma das formas de lidar com situações de encravamento (*deadlock*) consiste em evitá-las através de uma cuidadosa atribuição de recursos. Um dos algoritmos apresentados nas aulas que faz uma alocação daquele tipo é o "algoritmo do banqueiro", esquematizado na figura ao lado para um exemplo de 4 entidades (A, B, C, D) que vão necessitar de um total de 22 recursos do mesmo tipo, dispondo o sistema de apenas 10 em simultâneo.
Mostre que, se a partir da situação inicial a) se chegar à situação b), então o sistema está num estado seguro, pelo que ainda não há risco de encravamento. Pelo contrário, se se chegar à situação c), então o sistema poderá encravar.

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

a) Free: 10

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

b) Free: 2

	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

c) Free: 1

6. [1 val.] A figura mostra o esquema de um endereço lógico de 32 bits de uma posição de memória de um sistema de memória virtual paginada do tipo "multi-nível".

10b	10b	12b
PT1 #	PT2 #	OFF

a) Qual é o tamanho de página, o número de entradas de cada tabela de páginas e o número total de tabelas de páginas que o sistema comporta?

b) Afinal, qual é a vantagem de um sistema multi-nível, relativamente a um sistema de páginas simples?

7. [1 val.] A figura ao lado, retirada das folhas de apoio às aulas teóricas, ilustra as classes de páginas em memória que servem de base ao algoritmo de remoção de páginas "não usada recentemente" (NRU, Not Recently Used). Será mesmo possível haver páginas, não referenciadas mas modificadas, pertencentes à classe 2 (2ª linha)?

Eviction Podium	**R**	**M**
1st	0	0
2nd	0	1
3rd	1	0
4th	1	1

8. [1 val.] Explique como pode o sistema operativo conseguir avisar diversos processos do decorrer de diferentes intervalos de tempo (e.g. especificados pela chamada `alarm()`), com base num único relógio.

9. [1 val.] Considere a informação respeitante a um ficheiro, que deve ser mantida pelo sistema operativo (os também designados atributos de um ficheiro). Apresente 5 desses atributos.

10. [1 val.] Um ficheiro é, normalmente, dividido em blocos que são armazenados no disco do computador e o sistema operativo administra esses blocos essencialmente por uma das seguintes técnicas de alocação: contígua, por lista ligada simples, por lista ligada em tabela (FAT, File Allocation Table) e por nós de indexação (*i-nodes*). Apresente uma das técnicas e refira uma vantagem e uma desvantagem da sua utilização.

PARTE III: Predominantemente de aplicação [5 val.]

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

1. [3 val.] O seguinte programa utiliza um canal (*pipe*), criado pelo processo pai, para comunicar com os processos-filho. Na *pipe* criada, `fd[0]` contém o descritor de acesso para leitura e `fd[1]` para escrita.
(Nota: para efeitos desta prova, o código está ligeiramente simplificado.)

```
 1. #define NCHILD 20
 2. int fd[2];
 3.
 4. void child() {
 5.   int n,i;
 6.   char msg[20];
 7.
 8.   n = read(fd[0], msg, sizeof(msg));
 9.   if (n == 0) printf("Read failed\n");
10.  else if(n < 0) perror("Read error");
11.  else {
12.       close(fd[0]);
13.       sscanf(msg, "%d", &i);
14.      printf("child: %d\n", i);
15.       exit (i);
16.  }
17.  exit (-1);
18.}
19.
20.int main(int argc, char *argv[]) {
21.
22.  pid_t pid;
23.  int i=0, fd[2], st, n;
24.  char msg[20];
25.
26.  pipe(fd);
27.
28.  while(i<NCHILD) {
29.       pid = fork();
30.       if (pid == 0)
31.         child();
32.       else if (pid < 0 && errno == EAGAIN)
33.         usleep(1000);
34.      else if (pid < 0 && errno != EAGAIN)
35.        execlp("echo", "echo", "-e", "\a", NULL);     // faz "beep"
36.       else
37.         i++;
38.  }
39.
40.  for (i=0; i<NCHILD; i++) {
41.       sprintf(msg, "%d", i);
42.       n = write(fd[1], msg, strlen(msg)+1);
43.       if (n == 0) printf("write failed\n");
44.       else if (n < 0) perror("write error");
45.  }
46.
47.  for (i=0; i<NCHILD; i++) {
48.      /* completar */
49.      printf("returned: %d\n", /* completar */);
50.  }
51.
52.  return 0;
53.}
```

a) A variável `fd` tem mesmo de ser global?

b) Explique a necessidade da existência das linhas 32-33, onde EAGAIN significa «*cannot allocate sufficient memory to copy the parent's page tables and allocate a task structure for the child*».

c) Complete o código das linhas 47-50, em que o processo pai deve aguardar pela terminação de cada um dos processos filho e imprimir o seu código de terminação.

d) Diga se é de esperar que na consola surjam impressos os números em sequência, tal como:

```
child: 0
child: 1
...
child: 19
returned: 0
...
returned: 19
```

e) [1 val] Devido a um erro na concepção, o programa ao ser executado por vezes escreve:

```
child: 0
...
Read failed
returned: 255
…
```

Porque razão falha o *read*? Apresente as modificações necessárias de modo a que todos os processos-filho recebam correctamente a mensagem através da *pipe*.

2. [2 val.] O seguinte programa simula o problema dos filósofos-"morfadores". Estes estão sentados a uma mesa circular, com um prato à frente e um garfo de cada lado do prato e nunca falam uns com os outros. O garfo esquerdo de um filósofo é o garfo direito do filósofo que se encontra à sua esquerda. Para pensar, um filósofo tem que comer! Para comer, um filósofo tem que apanhar um garfo, a seguir o outro, e comer durante um intervalo aleatório de tempo; depois vai pousar cada um dos garfos, e pensar até ter fome de novo, recomeçando o ciclo.

```
 1. #define NPHIL 10
 2. pthread_mutex_t mut[NPHIL];
 3.
 4. void *philosopher(void *arg) {
 5.   int phil = * (int *) arg ;
 6.   int left_fork = phil;
 7.   int right_fork = (phil + 1) % NPHIL;
 8.
 9.   while(1) {
10.       pthread_mutex_lock(&mut[left_fork]);
11.       printf("phil=%d got left fork\n", phil);
12.
13.       pthread_mutex_lock(&mut[right_fork]);
14.       printf("phil=%d got right fork\n", phil);
15.
16.       printf("phil=%d eating...\n", phil);
17.       usleep(rand()%1000);
18.
19.       pthread_mutex_unlock(&mut[left_fork]);
20.       pthread_mutex_unlock(&mut[right_fork]);
21.
22.       printf("phil=%d thinking...\n", phil);
23.       usleep(rand()%1000);
24.   }
25. }
26. int main() {
27.   int i;
28.   int arg[NPHIL];
29.   pthread_t tid;
30.
31.   setbuf(stdout, NULL);
32.
33.   for(i=0; i< NPHIL; i++)
34.       pthread_mutex_init(&mut[i], NULL);
35.
36.   if (pthread_create(&tid, NULL, BigBrother, NULL) != 0)
37.       exit(-1);
38.
39.   for(i=0; i< NPHIL; i++) {
40.       arg[i] = i;
41.       if (pthread_create(&tid, NULL, philosopher, (void *) &arg[i]) != 0)
42.         exit(-1);
43.   }
44.   pthread_exit(NULL);
45.   return 0;
46. }
```

a) Qual é a função da instrução da linha 31? É ou não indispensável ao bom funcionamento do programa?

b) A *thread* principal faz `pthread_exit()` sem esperar que as *threads* por ela criadas terminem. Isso não fará terminar imediatamente o processo e as *threads* nele contidas?

c) Argumente da necessidade ou não de usar aqui *threads* do tipo *detached*.

d) O programa apresenta risco de encravamento. Corrija o programa, com um mínimo de alterações, de modo a que isso não aconteça e descreva a ideia subjacente à sua solução.

e) [1 val] Sem recorrer à solução que propôs na alínea anterior e sem alterar significativamente o código fornecido, escreva o código de uma *thread* supervisora de nome BigBrother que force a terminação do processo quando detectar que o encravamento ocorre há já pelo menos 30 segundos.

JMMC, JFSC