Tipo de Prova: sem consulta **Exame Normal da Época Normal**
Duração: 2 horas **26.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. O comando "`gcc -Wall -o exc font.c`" foi invocado num dos computadores das salas de informática da FEUP. Nas operações efectuadas para a criação do executável `exc`,

A) foi utilizada uma biblioteca estática.

B) foi utilizada uma biblioteca dinâmica.

C) não foi utilizada qualquer biblioteca.

2. Um sistema operativo é um "gestor de recursos". Isso quer dizer que:

A) permite, por exemplo, a partilha da memória disponível no computador.

B) faz a gestão dos recursos que as aplicações lhe submeterem.

C) é apropriado a programas gestão (de bases de dados, por exemplo).

3. O modo de aplicação (*user mode*) de um processador é o modo como a maior parte do código executa por ser:

A) mais poderoso, para o utilizador, do que o modo de supervisão (*kernel mode*).

B) mais compacto, em termos de memória, do que o modo de supervisão (*kernel mode*).

C) mais seguro, para o sistema, do que o modo de supervisão (*kernel mode*).

4. Sistemas multi-programação e sistemas multi-utilizador,

A) são sinónimos, pois têm o mesmo significado.

B) são antagónicos, pois não podem ambos qualificar um dado sistema operativo.

C) são coisas diferentes, apesar de normalmente virem associados.

5. Um processo tem associado:

A) um programa, uma zona de memória e um ficheiro.

B) um programa, uma zona de memória e uma entrada na tabela de processos.

C) um programa, um processador e um dono.

6. Na sequência da invocação da chamada `fork()` o novo processo irá começar imediatamente a executar.

A) Não, pois o processo-pai é sempre o primeiro a executar.

B) Sim, se for escolhida a opção certa para `fork()`.

C) Não é possível afirmar isso com certeza.

7. Na programação com *threads* POSIX, tem de se associar a cada *thread* uma rotina.

A) Sim e essa rotina poderá, ou não, ser diferente para cada *thread*!

B) Sim e essa rotina terá necessariamente de ser diferente para cada *thread*!

C) Tal não é necessário, pois cada novo *thread*, por omissão, fica associado a `main()`.

8. Uma região crítica consiste num segmento de código que

A) pode ser executado por vários processos em simultâneo.
B) só pode ser executado em simultâneo por vários *threads*.
C) não pode ser executado em simultâneo.

9. As soluções de *software* para o problema de exclusão mútua

A) não são totalmente eficazes; só uma solução de *hardware* resolve o problema.
B) só são eficazes se forem do tipo "espera activa".
C) são eficazes, mas podem ser pouco eficientes sem o concurso do sistema operativo.

10. Um encravamento (*deadlock*) típico dá-se quando vários processos se vêm impossibilitados de prosseguir a sua actividade normal por

A) terem esgotado o seu *quantum* de processador.
B) estarem aguardando a libertação do processador.
C) necessitarem de recursos detidos por diferentes processos do grupo.

11. Num sistema de memória virtual paginada, cabe à Unidade de Gestão de Memória (MMU) do processador a indicação

A) do tamanho de página apropriado à situação.
B) da moldura a esvaziar, no caso de se dar uma falha de página.
C) do endereço real de uma referência à memória.

12. É muito útil a um programador de aplicações o conhecimento de que o seu sistema operativo alvo usa

A) paginação.
B) segmentação.
C) *swapping*.

13. O sistema de acesso aos dispositivos de entrada e saída de um computador é organizado em camadas executando operações diferentes. Essa organização

A) é necessária, para esse tipo de dispositivos.
B) é normal, para todo o sistema operativo.
C) é opcional, porque depende do uso de interrupções.

14. Para permitir que diversos processos sejam avisados do decorrer de diferentes intervalos de tempo, o sistema operativo

A) costuma utilizar um temporizador para cada processo em espera.
B) pode manter uma lista ligada de estruturas de dados, ordenada pelo prazo de "despertar" mais próximo.
C) a cada *tick* do temporizador único, percorre a tabela de processos, procurando os que têm de ser avisados.

15. Dos seguintes nomes completos de ficheiro, qual deles é um nome absoluto?

A) `/lib/dict`.
B) `./lib/dict`.
C) `../lib/dict`.

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. [1 val.] Qual é a utilidade do serviço `atexit()`? Seria importante estender este serviço para o caso em que um programa em execução é subitamente interrompido por um sinal?

2. [1 val.] Indique 3 aspectos que distingam um processo de um *thread*.

3. [0,5 val.] Por que razão se recomendou várias vezes, nas aulas práticas e teóricas, a utilização da invocação «`setbuf(stdout, NULL);`» no desenvolvimento de programas?

4. [1 val.] O uso de monitores não é absolutamente necessário para se controlar o acesso a zonas críticas. Tal pode ser conseguido com mutexes, por exemplo. Mas, então, qual é o grande interesse dos monitores?

5. [1,5 val.] Suponha uma situação de encravamento que envolve 3 processos.

a) Represente-a esquematicamente com um grafo idêntico aos mostrados nas aulas.
b) Estarão os processos necessariamente no estado "bloqueado"?
c) Como poderia o problema ser resolvido com um mínimo de custos?

6. [1 val.] O seguinte segmento de código, que pretende ver escrita no ecrã a mensagem "`Hello, world!`" exibe uma condição de competição. Identifique-a.

```
1.  int main() {
2.              struct sigaction new;
3.        // prepare sigaction struct new...
4.        sigaction(SIGUSR1, &new, NULL);
5.        switch (fork()) {
6.              case -1: perror ("fork()"); exit (1);
7.              case 0: // filho
8.                        printf ("Hello, ");
9.                        kill (getppid(), SIGUSR1);
10.                        break;
11.             default: // parent
12.                        pause();
13.                        printf("world!");
14.             }
15.  } // main()
```

7. [1 val.] Apresente uma vantagem e um inconveniente de um sistema de memória virtual só com segmentação relativamente a um só com paginação. Um sistema misto (de paginação com segmentação) recolhendo as vantagens de ambos os tipos de técnicas, será a solução "ideal"?

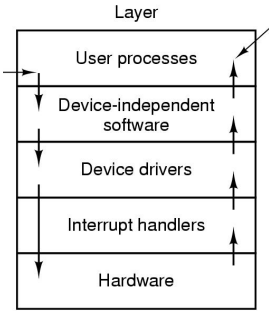
8. [1 val.] Considere um sistema que implementa memória paginada com (apenas!) 4 molduras (*page frames*). A tabela mostra a informação de paginação acessível ao sistema operativo na altura em que uma decisão de libertação de quadro tem de ser tomada.

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

a) Qual das páginas mostradas dará lugar à nova página, quando o algoritmo usado pelo sistema operativo para a substituição for o da "página não usada há mais tempo" (LRU, Least Recently Used Page)?

b) Apresente uma vantagem e um inconveniente da utilização deste algoritmo.

9. [1 val.] A figura ao lado representa o percurso de um pedido feito a um dispositivo de E/S, e da consequente resposta, através do modelo de camadas que estrutura a maior parte dos módulos de E/S dos sistemas operativos actuais.
Diga, exemplificando se quiser, uma tarefa executada por cada uma das camadas mostradas.



10. [1 val.] Por que será que a operação sobre directórios `readdir()`, não tem em Unix uma sintaxe de invocação semelhante à correspondente operação `read()` sobre ficheiros?

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 seguinte programa utiliza uma FIFO para que o processo pai possa comunicar com o processo filho:

```
1. #define FIFON "/tmp/exame-fifo"
2. int main(int argc, char *argv[]) {
3.   pid_t pid;   int fd, n;  char msg[20];
4.   mkfifo(FIFON, 0666);
5.   pid = fork();
6.
7.   if (pid > 0) {
8.     strcpy( msg, "Tás bem?");
9.     fd = open(FIFON, O_WRONLY);
10.    n = write(fd, msg, strlen(msg)+1);
11.    if (n <= 0) perror("write error");
12.    close(fd);
13.
14.  } else if(pid == 0) {
15.    fd = open(FIFON, O_RDONLY);
16.    n = read(fd, msg, sizeof(msg));
17.    if (n == 0) printf("Read failed\n");
18.    else if(n<0) perror("Read error");
19.    else printf("%s\n", msg);
20.    close(fd);
21.
22.  } else {
23.    perror("fork");
24.    execlp("ls", "ls", "/tmp", NULL);
25.    printf("ls terminou\n");
26.  }
27.  /* unlink(FIFON); */
28.  return 0;
29.}
```

a) Apresente toda a informação que irá ser impressa no terminal no caso de `fork()` falhar mas `execlp()` não falhar.

b) É possível que outros processos no sistema leiam e escrevam na FIFO? Que alteração introduziria no programa para que apenas os processos do utilizador o possam fazer?

c) Descreva a sequência de acontecimentos que leva a que na listagem de saída ocorra por vezes a mensagem «`Read failed`»?

d) Descomentando a linha 27, por vezes ocorrem as mensagens «`write error: Bad file descriptor`» e/ou «`Read error: Bad file descriptor`». Qual a sequência de acontecimentos que as provocam? Como o corrigir?

e) [1 val] Se o processo filho não devolver um valor após 30 segundos de execução, deve ser terminado pelo processo pai. Escreva o código-fonte que implemente este requisito.

2. [3 val.] O programa seguinte, quando executado, cria 1000 *threads* que preenchem um *array* global `ring`, criando ainda outra *thread* `watcher` que troca todos os elementos do *array*; finalmente, a *thread* principal imprime todos os elementos do *array.*

```
 1. #define NTHREADS 1000
 2. int ring[NTHREADS];
 3.
 4. void *filler(void *arg) {
 5.   pthread_detach(pthread_self());
 6.   ring[*(int *) arg] = *(int *) arg ;
 7.   return NULL;
 8. }
 9. void *watcher(void *arg) {
10.   int i, tmp;
11.   for (i=0; i<NTHREADS/2; i++) {
12.        tmp = ring[i]; ring[i] = ring[NTHREADS-i-1]; ring[NTHREADS-i-1] = tmp;
13.   }
14.   return NULL;
15. }
16. int main() {
17.   int i; int arg[NTHREADS]; pthread_t tids, tid;
18.
19.   if (pthread_create(&tid, NULL, watcher, NULL) != 0)
20.        exit(-1);
21.
22.   for(i=0; i< NTHREADS; i++) {
23.        arg[i] = i;
24.    /* if (pthread_create(&tids, NULL, filler, (void *) &i) != 0) */
25.        if (pthread_create(&tids, NULL, filler, (void *) &arg[i]) != 0)
26.          exit(-1);
27.   }
28.
29.   pthread_join(tid, NULL);
30.   for(i=0; i< NTHREADS; i++)
31.        printf("ring[%d]=%d\n", i, ring[i]);
32.
33.   return 0;
34. }
```

a) É necessária a existência do *array* `arg` em `main()`? Não basta passar simplesmente o endereço da variável `i`, como é feito na linha 24 (código comentado)?

b) Se a linha 5 for comentada, o programa termina ao fim de criar 382 *threads,* apesar de em Linux o número máximo de *threads* ser da ordem das dezenas de milhar. Que faz exactamente `pthread_detach()` para evitar que o programa termine em erro?

c) Fazer `pthread_detach()` a uma *thread* impede que mais tarde se use `pthread_join()`nessa *thread*?

d) Não é necessário proteger o acesso de cada *thread* ao array `ring` com uma *mutex*? Porquê?

e) [1 val] A *thread* `watcher` só deve começar a trabalhar quando todas as outras *threads* tiverem terminado. Mostre como pode cumprir este requisito usando uma variável de condição, sem alterar o código das *thread* `filler`, nem alterar estruturalmente o programa.

JMMC, JFSC