MIEIC SISTEMAS OPERATIVOS – 2009/2010 1S FEUP

Tipo de prova: sem consulta **Exame de Recurso da Época Normal**
Duração: 2 horas **12 de Fevereiro de 2010**
Cotação máxima: 20 valores
Estrutura da prova: Parte I (escolha múltipla, 25%); Parte II (teórica, 50%); Parte III (Aplicação, 25%)

Nota: Utilize folhas de resposta independentes para cada uma das partes.

Parte I: Escolha Múltipla [5 valores]

Utilização: para cada pergunta só há uma resposta correcta; indique-a com a letra correspondente na folha de respostas; se não souber a resposta correcta, não preencha ou faça um traço nessa alínea.

Cotação: cada resposta certa vale 1 ponto; cada resposta errada vale –0.3 pontos; cada resposta ambígua, ininteligível ou não assinalada vale 0 ponto. O total é 15 pontos.

1. O que é um processo?
   a. Um programa em execução
   b. Um programa com vários fluxos de execução
   c. Qualquer programa que evita de forma eficiente *deadlocks*

2. Exclusão mútua
   a. Se um processo está na região crítica, os outros não podem entrar
   b. Serve para evitar *deadlocks*
   c. Requer semáforos para ser implementada

3. Qual dos seguintes *schedulers* controla o grau de multiprogramação
   a. *Scheduler* de curto prazo (ou *dispatching*)
   b. *Scheduler* de longo prazo
   c. *Scheduler* de médio prazo

4. Um computador contém 4 *page frames*. O tempo para carregar, a hora do último acesso, os bits R e M das páginas estão na tabela (tempos estão em *clock ticks*). Que página será substituída usando o algoritmo segunda oportunidade?

Page	Loaded	Last ref.	R	M
0	126	280	1	0
1	230	265	0	1
2	140	270	0	0
3	110	285	1	1

   a. 1
   b. 2
   c. 3

5. Em gestão de memória, como se designam os blocos de tamanho fixo que dividem a memória física no *paging*
   a. *Pages*
   b. *Frames*
   c. *Segments*

6. Qual o estado de um processo após uma instrução de I/O
    a. Pronto
    b. Bloqueado/Em espera
    c. A executar

7. Algoritmos de escalonamento de processos são justos se
    a. Nenhum processo sofrer de *starvation*
    b. Forem usados semáforos
    c. For usada uma fila de prioridades para o escalonamento

8. Qual o output produzido pelo seguinte comando "echo $PWD ; cat"?
    a. Imprime o valor da variável $PWD e executa `cat`
    b. O output produzido é igual a "echo $PWD | cat" e "cat < echo $PWD"
    c. O comando `cat` envia para o *stdout* o valor da variável $PWD pois o output do comando `echo` é redireccionado para o input do comando `cat`

9. Suponha que todas as *page frames* estão inicialmente vazias, e que um processo necessita de 3 *page frames* em memória real. A ordem pela qual o processo referencia as suas páginas é a seguinte 1232452323241. Usando o algoritmo FIFO, quantas *page faults* irão ocorrer?
    a. 7
    b. 8
    c. 9

10. Sistemas de multiprogramação
    a. São mais fáceis de desenvolver do que sistemas monoprogramáticos
    b. Executam processos de forma mais célere
    c. Executam mais processos no mesmo período de tempo

11. Em situações na qual dois ou mais processos estão a escrever ou ler informação partilhada, o resultado final está dependente da ordem de execução dos processos devido a
    a. Race conditions
    b. Deadlocks
    c. Critical Sections

12. Considere um sistema em que a memória apresenta os seguintes blocos de memória vazios: 10KB, 4KB, 20KB, 18KB, 7KB e 9KB. Após serem feitos 3 pedidos sequenciais de memória usando o algoritmo *worst fit*, quais os blocos que se encontram vazios?
    a. 4KB, 7KB e 9KB
    b. 4KB, 7KB e 10KB
    c. 4KB, 9KB e 10KB

13. Nas substituições de páginas, qual dos seguintes algoritmos não obedece à condição de Belady
    a. FIFO
    b. LRU
    c. Não existe relação entre a condição de Belady e a substituição de páginas

14. Sistemas de memória virtual
    a. Requerem a utilização de paginação e segmentação
    b. Requerem a utilização de paginação e/ou segmentação
    c. Necessitam de muita memória física para serem eficientemente implementados

15. Escalonamento de pedidos ao disco: que algoritmo reduz o movimento do braço/tempo de resposta, garantindo um serviço consistente de todos os pedidos?
    a. FIFO
    b. *Shortest seek first* (SSF)
    c. Algoritmo do elevador

Parte II: Predominantemente teórica [10 valores]

Utilização: respostas devem ser (brevemente) justificadas.

Cotação: indicada em cada pergunta.

1. **[1 val.]** Descreva sucintamente o processo de *booting* de um PC.

2. **[1 val.]** O que é multiprogramação? Apresente duas razões para preferir um sistema operativo com multiprogramação a um com monoprogramação.

3. **[2 val.]** O que ocorre quando um processo requisita I/O num escalonamento *round-robin*? Contextualize a sua resposta dizendo o que acontece nas diversas camadas de I/O (de um sistema de software).

4. **[1 val.]** Considere o programa em baixo. Qual é o output do programa? Descreva a sua evolução, dando ênfase especial aos processos e respectivas estruturas de dados.

```
int main() {
        int x = 0;
        int p = fork();
        if(p == -1) {
                x = -1;
        } else if(p == 0) {
                x++;
        } else {
                x++;
        }
        printf("x = %d\n", x);
}
```

5. **[1 val.]** Num laboratório equipado com computadores pessoais, os utilizadores, quando desejam imprimir documentos, servem-se de um sistema clássico de *spooling*, operado numa máquina própria. Tipicamente, os documentos são enviados para o servidor de impressão, sendo depois impressos, um de cada vez. Supondo que o espaço de disco do servidor é limitado, explique
   - **a.** O que é um *spooler*?
   - **b.** Em que circunstâncias poderá ocorrer uma situação de encravamento?
   - **c.** Indique como essa situação poderia ser evitada?
   - **d.** O que poderia acontecer se fosse permitido comunicar directamente com a impressora? Justifique tendo como base a forma como o sistema operativo comunica com a impressora.

6. **[1 val.]** Indique vantagens e desvantagens (pelo menos duas) do escalonamento com preempção (*preemptive scheduling)*.

7. **[2 val.]** Em gestão de memória, qual é o objectivo da segmentação? E da paginação? Compare sucintamente as duas técnicas**.**

8. **[1 val.]** Um processo pretende indicar a outro informação sobre o estado do tempo numa dada região (sol, chuva, nublado). Explique como poderá tal ser conseguido, e em que circunstâncias, com cada uma das seguintes técnicas de inter-comunicação
   a. Canais com nome (*named pipes)*.
   b. Canais normais (unnamed pipes).
   c. Sinais.

Parte III: Predominantemente de aplicação [5 valores]

Utilização: respostas devem ser (brevemente) justificadas.

Considere o código fonte do programa `xpto` em baixo. Como pode verificar, este programa deve ser invocado com um argumento diferenciando o seu comportamento. Por uma questão de simplificação, omitiu-se o controlo de erros e assumiu-se que os *named pipes* já estão criados.

```
1.  #define size 10
2.  #define chunk 2
3.  int data[size] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
4.  pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
5.
6.  void sendLine(int part) {
7.     int fd = open("source", O_WRONLY);
8.     for(int i = 0; i < chunk; i++)
9.        write(fd, &data[part*chunk+i], sizeof(int));
10.    close(fd);
11. }
12.
13. int processLine() {
14.    int result = 0, tmp;
15.    int fd = open("source", O_RDONLY);
16.    for(int i = 0; i < chunk; i++) {
17.       read(fd, &tmp, sizeof(int));
18.       result += tmp;
19.    }
20.    close(fd);
21.    return result;
22. }
23.
24. void sendResult(int val) {
25.    int fd = open("sink", O_WRONLY | O_NONBLOCK);
26.    write(fd, &val, sizeof(int));
27.    close(fd);
28. }
29.
30. int receiveResult() {
31.    int val, fd = open("sink", O_RDONLY);
32.    read(fd, &val, sizeof(int));
33.    close(fd);
34.    return val;
35. }
36.
37. int main (int argc, const char * argv[]) {
38.    int p; int isClient = atoi(argv[1]);
39.    if (!isClient) {
40.       int result = 0;
41.       for (int i = 0; i < size/chunk; i++) {
42.          sendLine(i);
43.          result += receiveResult();
44.       }
45.       printf("%d", result);
46.    } else {
47.       while (1) sendResult(processLine());
48.    }
49. }
```

1. Analise cuidadosamente o código e diga, de forma sucinta, qual é a funcionalidade implementada pelo mesmo. Centralize a sua resposta em torno dos dois papeis que o processo pode assumir (cliente e servidor), e das respectivas comunicações entre os dois *named pipes*.

2. Considere que o programa é invocado simultaneamente em duas shells (*sh1$> ./xpto 0 ; sh2$> ./xpto 1*)
    a. Qual o resultado produzido para o *stdout* e o estado final em ambas as shells? E se o valor de chunk for modificado de 2 para 5?
    b. Na sua opinião este programa poderia ser implementado com apenas um *named pipe*? Justifique a sua resposta.
    c. Por vezes o programa não funciona correctamente. Explique porquê e descreva uma sequência de execução errónea. Como resolveria esse problema?
    d. A situação anterior pode levar a um *deadlock*? Se sim, demonstre como.
    e. O problema anterior poderia ser resolvido por exclusão mútua? Indique uma técnica possível e que secções de código afectaria.
    f. Existe algum problema em executar múltiplos clientes? Se sim, como resolveria?
3. Escreva uma única linha de código (e diga onde) a qual resulte no servidor actuar simultaneamente como cliente.
4. Considere as seguintes modificações colocadas no final do programa original:

```
46.     } else {
47.         void *status; pthread_t core[2];
48.         pthread_create(&core[0], NULL, job, (void*) 1);
49.         pthread_create(&core[1], NULL, job, (void*) 2);
50.     }
51. }
52.
53. void* job(void* p) {
54.         while (1) {
55.             int result = processLine();
56.             sendResult(result);
57.         }
58. }
```

    a. O programa termina inesperadamente momentos após iniciar. Explique porquê e sugira o código para a solução.
    b. Quando executado correctamente até ao final, os valores que devolve por vezes não fazem sentido. Explique a razão e sugira o respectivo código para uma solução.
    c. Que aconteceria se fossem colocado um "trinco" fora do ciclo while(1) na função job? O comportamento é determinístico (i.e. sempre o mesmo)?

RMA, JVV, HSF