UNIVERSIDADE DA BEIRA INTERIOR
Departamento de Informática

SISTEMAS OPERATIVOS

Exame 1, 20 de Junho de 2018, 15.00 Horas Escala 0:20 Sem Consulta Sala 6.04/6.06 Duração: 2h30m

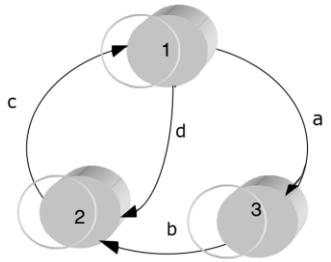
Observações: Não é permitida a consulta de livros ou de apontamentos. Não se esclarecem dúvidas durante a prova. Se tiver dúvidas, indique na folha de teste a sua interpretação. Utilize uma caligrafia legível.

Grupo A: História, Estrutura e Arquitectura dum Sistema Operativo (4 valores)

1. Sistemas operativos podem ter propósitos diferentes! Explique a diferença entre sistemas operativos de tempo real e sistemas operativos populares para desktop.
2. Diga o que entende por virtualização de hardware pelo sistema operativo. Explique e dê um exemplo.
3. Explique quais são os componentes dum sistema operativo que ajudam na proteção do funcionamento da CPU.
4. Dada a variável char *str="ubi\n", explique as diferenças entre printf("%s",str) e write(1,str,4).

Grupo B: Programas, Processos e Threads(4 valores)

5. Considere o seguinte diagrama de estados básico dos processos num sistema operativo (note que dele não consta nenhum estado que podia ser denominado *suspenso*).



- Escreva uma legenda para o diagrama - não se esqueça de indicar o nome dos estados {1,2,3} bem como explicar as transições {a,b,c}.
- No diagrama não existe uma transição entre o estado 3 e o estado 1. Explique por que motivo com base no modelo de filas de espera que suportam a execução do escalonamento.
- O diagrama não inclui o estado para os processos descritos em Linux como em estado *zombie*. Qual o objetivo deste estado?
- Explique de que estado ou estados do diagrama haverá transições para esse estado *zombie* e porquê.

6. Qual é o output do seguinte programa, o output é determinístico? Justifique? (Deverá mostrar o *trace* do funcionamento do programa) **.**

```
{
   int i, pid, x = 2;
   pid = fork();
   if (0 == pid) {
      for (i=0; i<2; i++)
         pid = fork()
      x--;
      if ( 0 != pid )
         execl( "/bin/date", "date", 0 );
   }
   else
         x++;
   printf("x=%d\n",x);
}
```

Grupo C: Gestão de Memória (4 Valores)

7. Explique como é que "código/funcionalidade" pode ser partilhado entre processos em sistemas paginados e em sistemas segmentados.

8. Considere um sistema com um endereçamento virtual de 31 bits e páginas de 4k bytes. Cada entrada na tabela de páginas ocupa 64 bits. (i) Qual a dimensão máxima da tabela de páginas de um processo, 512 Kbytes, 4 Mbytes , 32 Mbytes ou 128Mbytes ? (Deverá indicar qual é o calculo efetuado). (ii) Que problemas levanta esta dimensão da tabela e proponha algumas soluções para as resolver?

9. Considere a noção de "working set" de um processo. Diga o que entende por esta noção. Qual a implicação desta noção ao nível da paginação efetuada pelo sistema operativo?

10. Um sistema de memória virtual tem um tamanho de página de 2000 palavras, 6 páginas virtuais e 4 páginas físicas. Assuma que inicialmente a tabela de páginas está vazia (nenhuma página carregada em memória).
    - Preencha a tabela seguinte com o estado que terá após acessos aos endereços (de palavra) virtuais 7000, 2500, 200, 3500, 1000, 6500, 13500, 10050.
    - Diga quais dos endereços originam "page faults" (falhas), "page hits" (sucessos) ou que são inválidos.

Página virtual	Página física	*Valido/Invalido*
0		*0*
1		*0*
2		*0*
3		*0*
4		*0*
5		*0*

Grupo C: Concorrência, Sincronização e Bloqueio (4 valores)

11. Considere um sistema que gere contas bancárias e que serve clientes, identificados através dum número inteiro único, que efetuam pedidos de transferência de dinheiro entre contas do mesmo sistema. Tais pedidos podem ocorrer concorrentemente, sendo cada pedido servido por uma ***thread*** diferente.

    Assuma os seguintes requisitos fundamentais deste sistema:

    Requisito A: Transferir uma dada quantia duma conta origem, A, para uma conta destino, B, consiste em debitar essa quantia de A, e creditar B com a mesma quantia, e mais nada.

    Requisito B: Uma dada transferência entre as conta A e B deve parecer **atómica** do ponto de vista de outras transferências que observem o saldo de A ou de B.

    Requisito C: Se, concorrentemente, um cliente pedir para transferir de A para B, e outro cliente pedir para transferir de C para D (em que A, B, C e D são contas diferentes), as duas transferências devem correr completamente em paralelo (i.e. sem que nenhuma seja obrigada a esperar pela outra).

    Requisito D: O sistema não deve chegar a situações de deadlock.

Considere as seguintes implementações da função ***transfere*** (que cada thread executa para servir cada pedido de transferência):

```
transfere1(int A, int B, int quantia) {
   saldo[A] -= quantia;
   saldo[B] += quantia;
}
```

```
pthread_mutex_t trinco;

transfere2(int A, int B, int quantia) {
   pthread_mutex_lock( &trinco )
    saldo[A] -= quantia;
    saldo[B] += quantia;
   pthread_mutex_lock( &trinco )
}
```

```
pthread_mutex_t trincos[MAX_NUMERO_CONTAS];

transfere3(int A, int B, int quantia) {
   pthread_mutex_lock( &trincos[A] )
   pthread_mutex_lock( &trincos[B] )
     saldo[A] -= quantia;
     saldo[B] += quantia;
   pthread_mutex_unlock( &trincos[A] )
   pthread_mutex_unlock( &trincos[B] )
}
```

Assuma que a instrução alto-nível "uMsaldo -= quantia" ("uMsaldo += quantia")
corresponde à seguinte sequência de instruções máquina em escritas em assembler NASM:
mov AX, uMsaldo
mov BX, quantia
sub AX, BX (ou add AX, BX)
mov uMsaldo, AX

- Para cada implementação, qual ou quais dos requisitos não são satisfeitos? Ilustre com um exemplo cada requisito que afirmar não ser satisfeito. No caso de haver DeadLock deve ser feito um diagrama de alocação de recursos.

- Altere a solução ***transfere3*** para que, no caso da conta de origem não ter saldo suficiente, a transferência se **bloqueie** até que tal aconteça. Mas, não deve utilizar nem semáforos nem mutexes.

  Comente a sua solução em termos de desempenho e funcionamento global do sistema.

Grupo E: Exercício Prático (4 valores)

12. Utilizando pipetas de baixo nível (*pipes*) é possível criar uma comunicação bidirecional entre dois processos.

    Escreva um programa no qual o processo cliente leia (a partir do teclado) o nome dum ficheiro já existente, uma *string* de (máximo) 30 caracteres, e envie esta *string* para o processo servidor. O processo servidor deverá abrir o ficheiro. Este ficheiro contém cem números reais em formato binário. O processo servidor enviará estes valores para o cliente. O processo cliente depois de receber estes valores deverá imprimir o valor médio destes valores.

Notas: O processo servidor deverá utilizar I/O de baixo nível para ler o ficheiro.
O processo "progénito" (filho) será o cliente e o processo "progenitor" (pai) será o servidor.
O seu programa não necessita de verificar erros (input errado) mas deverá ser o mais eficaz possível.
A sintaxe das funções necessárias são dadas numa folha anexa ao teste.