UNIVERSIDADE DA BEIRA INTERIOR
Departamento de Informática

SISTEMAS OPERATIVOS

Frequência II, 2 de Junho de 2014, 14Horas Escala 0:20 Sem Consulta Duração: 1h30m

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: Gestão de Memória (10 Valores)

1. Especifique quatro objectivos dum sistema de Gestão de Memória dum Sistema Operativo.

2. Qual são os mecanismos dum sistema operativo que permite que vários utilizadores usem o mesmo programa carregado em memória? Explique.

3. "Um sistema de memória paginada que disponha duma memória cache para o conjunto de trabalho (*working set*) dum processo é até três vezes mais rápido do que um sistema que utilize apenas processos cujos endereços de instrução e dados são sempre endereços verdadeiros de memória física". Explique porque razão esta frase está incorrecta.

4. Considere um sistema de gestão de memória com locação contígua de memória com partições múltiplas. Dadas partições de memória disponíveis de 50K, 300K, 200K e 100K (nesta ordem) como é que o SO colocará os processos com 170K, 120K, 140K,80K, 50K (nesta ordem) usando os algoritmos de (i) Worst-Fit (Pior Ajuste)e (ii) Best-Fit (Melhor Ajuste).

5. Um sistema de memória virtual tem um tamanho de página de 16 palavras, 5 páginas virtuais e 4 páginas físicas. A tabela de páginas está no estado apresentado em baixo. Um endereço virtual neste sistema tem 7 bits, sendo que os primeiros 3 bits indicam o número de página.

Página virtual	Página física	*Válido/Inválido*
0	2	*1*
1		*0*
2	1	*1*
3	0	*1*
4	3	*1*

   Para os seguintes endereços (i) diga se o resultado é uma page hit/miss (falha/sucesso) ou é uma interrupção/trap para o sistema operativo e (ii) calcule o endereço físico quando apropriado - em valor decimal

   (a) 001 0111
   (b) 100 1001
   (c) 011 0100
   (d) 111 0011
   (e) 000 1001
   (f) 010 0001

6. Num sistema de memória virtual paginada, quantas faltas de página aconteceriam usando os algoritmos de substituição de página "FIFO" (*First in First Out*) e "Optimal" com a seguinte *string* de referência: 1, 2, 3, 4, 1, 2, com apenas três molduras de memória física.

Grupo B: Threads, Concorrência e Sincronização (10 valores)

7. Considere a seguinte situação com 3 processos (P1,P2 e P3) e 3 recursos (R1,R2 e R3) - os recursos R1 e R3 tem apenas uma instancia mas o R2 tem duas instâncias do mesmo recurso.
   Ao processo P1 está atribuído uma das instâncias do recurso R2 e está a espera do R1. R1 está atribuído ao P2. O processo P2 está a espera da atribuição duma instância do R2 e o P3 está à espera do recurso R3.

   a) Desenhe o grafo de atribuição/alocações de recursos para esta situação.
   b) Existe uma situação de bloqueio mútua (*Deadlock*) entre os processos ou não? Justifique.
   c) Considere o seguinte: o sistema evolve da situação descrita em cima e o P3 pede e é atribuído uma instância do recurso R2 e o P2 pede o recurso R3. Desenhe o grafo actualizado de atribuição/alocações de recursos e explicar se agora há ou não bloqueio mútua (*Deadlock*) e/ou em que condição poderá ocorrer.

8. Considere o seguinte programa.

```
int x=0;
void *maisx(void *args)   { int add = *(int*)args ; x=x+add; }
int main() {
   pthread_t  th[3];
   int ids[3]={1,2,3};
   for (int i=0; i<3; i++)
       pthread_create( &th[i], NULL, maisx, &ids[i]);
   for (int i=0; i<3; i++)
        pthread_join( th[i], NULL );
   printf("x=%d\n",x);
}
```

   a) Quais são os outputs possíveis do programa?
   b) Usando a sintaxe de Posix *Threads* explique como é que pode usar um trinco logico para garantir que o resultado deste programa seja "x=6"

9. Segue-se uma possível solução para garantir a exclusão mútua, progressão e espera limitada no acesso a uma secção crítica. Note que a variável int flag[2]={FALSE,FALSE}; é partilhada entre os dois processos.

Processo 0	*Processo 1*

while (TRUE)
{
flag[0]=TRUE;
while (flag[1]); // não faz nada
<seccao critica>
flag[0]=FALSE;
<restante codigo>
}
	while (TRUE)
	{
	flag[1]=TRUE;
	while (flag[0]) ; // não faz nada
	<seccao critica>
	flag[1]=FALSE;
	<restante codigo>
	}

   (a) Explique onde pode ocorrer espera activa (*busy-waiting*) na solução apresentada em cima.
   (b) Mostre que a solução <u>não é</u> satisfatória.
   (c) Escreva o pseudo-código para os dois processos anteriores usando um semáforo por forma a garantir o acesso exclusivo à secção crítica.
   (d) Explique porque não ocorre espera activa (*busy-waiting*) com a solução dum semáforo.

10. Considere o código para três *threads* a executar concorrentemente

void *a(){
puts("um\n");
puts("dois\n");
}
	void *b(){
	puts("três\n");
	puts("quatro"\n");
	}
		void *c(){
		puts("cinco\n");
		puts("seis\n");
		}

    Tomando em conta que é necessário imprimir "três" antes de "seis" e "dois" antes de três (e isto são as únicas restrições) reescreva as funções usando um máximo de dois semáforos para sincronizar o output. Deverá usar a sintaxe POSIX e explicar como os semáforos são inicializados.