UFES - DEPARTAMENTO DE INFORMÁTICA

1ª. Prova de Sistemas Operacionais / Sistemas de Programação II

Período: 2009/1 – Data: 02/06/2009
Profa. Roberta Lima Gomes

1) Considere os seguintes métodos de escalonamento: FIFO, Round-Robin, Prioridades, Multinível. O sistema executa muitos tipos de processos (curtos, longos, interativos, em lotes, CPU bound, I/O bound, real-time, etc.). Para cada item da lista de requisitos a seguir, de *(a)* a *(d)*, : **(2,0)**

- diga qual a estratégia que **melhor** satisfaz considerando todos eles (ou a maior parte deles) simultaneamente;
- para cada uma das demais estratégias, explique porque pelo menos um requisito que não foi satisfeito.

a) O número de processos interativos é relativamente pequeno. Eles deverão ter alta prioridade, mas são aceitáveis pequenos atrasos.
b) Processos I/O bound devem ter prioridade alta para manter ativos os processadores de I/O.
c) Processos CPU bound devem retardar processos I/O bound, mesmo que esses já estejam esperando há muito tempo.
d) Quando os processos CPU bound são os únicos que estão prontos, a sobrecarga do sistema operacional deve ser minimizada.

2) Em algumas implementações do UNIX, o kernel é não-preemptivo. O que isto significa? Quais as vantagens e desvantagens desta abordagem? **(1,0)**

3) Entre o emprego de instrução TSL (*Test and Set Lock*) e desabilitar interrupções para controle de exclusão mútua, qual dos dois é menos perigoso? Justifique e diga se a opção menos perigosa apresenta alguma desvantagem. **(1,0)**

4) Considere um sistema operacional cuja máquina de estados inclui os estados *Ready* e *Ready- Suspended*. Suponha que seja hora do S.O. despachar um processo e que existam nesse momento processos tanto no estado *Ready* como no estado *Ready-Suspended*, e que pelo menos um processo no estado *Ready-Suspended* possui prioridade maior do que qualquer processo no estado *Ready*. Duas políticas extremas seriam: (a) sempre despachar um processo no estado *Ready*, de forma a minimizar swapping; e (b) sempre dar preferência ao processo de mais alta prioridade, mesmo que isso possa significar a ocorrência de swapping quando este não é necessário. Sugira uma política intermediária (explique e crie um algoritmo) que tente balancear prioridade e desempenho. **(2,0)**

5) Utilizando as primitivas de semáforos, explique como implementar um monitor. Considere que o Monitor segue a disciplina proposta por Hoare (Signal and Wait), onde o processo que sinaliza fica bloqueado até que o processo sinalizado saia do monitor ou execute outro wait. **(2,0)**

6) Observe os trechos de código abaixo implementados com base em semáforos binários e três alternativas para a parte2, e responda as perguntas: **(2,0)**

```
int n;
binary_semaphore a = 1;
binary_semaphore b = 1;
void parte1 ( )
{
   while (true)
   {
      produz();
      P(a);
      acrescenta();
      n++;
      if (n = 1) V(b);
      V(a);
                          }
}
```

(a)

```
void parte2 ( )
{
   P(b);
     while (true)
     {
        P(a)
        toma();
        n--;
        V(a);
        consome();
        if (n = 0)
          P(b);
     }
}
```

(b)

```
void parte2 ( )
{
   P(b);
   while (true)
   {
     P(a)
     toma();
     n--;
     if (n = 0) {
        consome();
        P(b); }
     V(a);
     }
}
```

(c)

```
void parte2 ( )
{
   P(b);
   while (true)
   {
     P(a)
     toma();
     n--;
     if (n = 0) {
        V(a);
        P(b); }
     else  V(a);
     consome();
     }
}
```

```
void main( )
{
   n = 0;
   parbegin(parte1, parte2);
}
```

a) Apresente a finalidade dos semáforos “a” e “b”. Verifique se ambos os semáforos estão iniciados com valores adequados com a lógica do código. Justifique sua resposta. **(0,5)**

c) Dentre as alternativas da parte2, aponte as que estiverem erradas e as que estiverem corretas. Para estiverem erradas, mostre onde está o erro. **(1,0)**

d) Apresente uma outra alternativa correta para a parte2. **(0,5)**

Boa Prova!