Curso de Tecnologia em Sistemas de Computação
Disciplina de Sistemas Operacionais
Professores: Valmir C. Barbosa e Felipe M. G. França
Assistente: Alexandre H. L. Porto

Quarto Período
Gabarito da AD1 - Segundo Semestre de 2012

Atenção: ADs enviadas pelo correio devem ser postadas cinco dias antes da data final de entrega estabelecida no calendário de entrega de ADs.

Atenção: Cada aluno é responsável por redigir suas próprias respostas. Provas iguais umas às outras terão suas notas diminuídas. As diminuições nas notas ocorrerão em proporção à similaridade entre as respostas. Exemplo: Três alunos que respondam identicamente a uma mesma questão terão, cada um, 1/3 dos pontos daquela questão.

Observação: A questão 6 é opcional. Se você tentar resolvê-la, você poderá ganhar até 2,0 pontos adicionais na nota da prova, mas a sua nota ainda será limitada a 10,0 pontos.

Nome -
Assinatura -

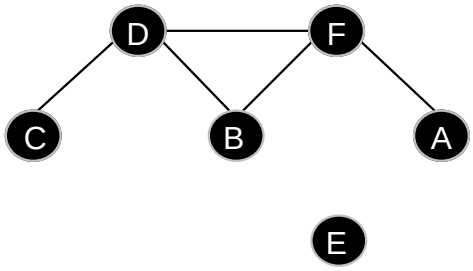
---

1. (2,0) Suponha que um programa A leve 21s para executar no processador e que, para executar a sua tarefa, ele também precise fazer E/S

por 7s. Se esse programa fosse executado em um sistema anterior ao da terceira geração, qual seria a fração de tempo do processador desperdiçada com operações de E/S? Ainda ocorreria algum desperdício nos sistemas posteriores ao da segunda geração, se somente existisse um outro programa B que executasse em 5s sem fazer operações de E/S?

Resp.: -Note que o programa somente poderia ser executado em um sistema da segunda geração, pois na primeira geração o programador manipulava diretamente o hardware do computador, não existindo um processador que executasse os programas. Como na segunda geração não existia o conceito de multiprogramação, o tempo de 7s de E/S faria parte do tempo de execução do programa. Além disso, como o tempo de execução do programa seria de 28s (o tempo de execução no processador mais o tempo da operação de E/S), e como o processador ficaria ocioso por 7s desses 28s, então a fração de tempo do processador desperdiçada seria de $7/28 = 1/4 = 0,25$, ou seja, 25% do tempo de execução do programa.
-Sim. O conceito de multiprogramação, que surgiu na terceira geração, somente tenta evitar que o processador fique ocioso quando o programa em execução faz operações de E/S. Infelizmente, no caso da questão, o processador ainda ficaria ocioso por 2s, porque o único programa B em execução no sistema além do programa A, apesar de não fazer operações de E/S, precisa executar no processador somente por 5s, um tempo menor do que os 7s da operação de E/S feita por A.

2. (2,0) Um aluno de sistemas operacionais alegou que a hierarquia dada a seguir relaciona os processos A, B, C, D, E e F em execução no sistema operacional. A alegação do aluno está correta? Justifique a sua resposta.



Resp.: Como vimos na Aula 2, os processos no primeiro nível da hierarquia não possuem um pai. Além disso, para cada processo da hierarquia em qualquer outro nível diferente do primeiro, deve existir somente um processo, no nível imediatamente anterior, que seja o seu pai. Logo, a alegação do aluno está incorreta, pois existem três erros na sua hierarquia: a conexão entre D e F (porque D e F estão no mesmo nível); uma das conexões envolvendo B (porque B somente pode ter um pai, ou D, indicado pela conexão entre D e B, ou F, indicado pela conexão entre D e F); e o processo E, que está no terceiro nível da hierarquia sem nenhum pai.

3. (2,0) Suponha que um processo execute em 10s, durante os quais sejam geradas, pelo sistema operacional, 800 operações de E/S. Suponha ainda que o sistema operacional esteja executando sobre o hardware real, e que uma operação de E/S execute em 2,5ms. Se o sistema operacional agora executar sobre uma máquina virtual que dobre o tempo de execução das operações de E/S, e cujo processador virtual tenha velocidade de 50% da velocidade do processador real, qual será o novo tempo de execução do processo? Justifique a sua resposta.

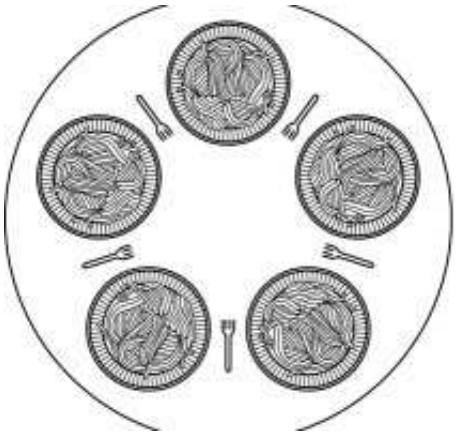
   Resp.: Pelo enunciado, o programa executa por 10s, ou seja, 10000ms. Como durante a execução são feitas 800 operações de E/S, e como cada operação de E/S demora 2,5ms, então 2000ms dos 10000ms são gastos com elas. Logo, o programa executa no processador do hardware por 8000ms. Considere agora que o programa execute sobre a máquina virtual. Note que a velocidade do processador virtual ser 50% da velocidade do processador real significa que, durante os 8000ms, somente 50% das instruções serão executadas. Com isso, no processador virtual,

o programa executará em 8000/0,5=16000ms. Como o tempo de cada operação de E/S dobrou e é agora de 5ms, então o tempo total de E/S passará para $5 \times 800 = 4000$ms. Logo, o tempo total de execução do programa na máquina virtual será de $16000 + 4000 = 20000$ms, ou seja, 20s.

4. (2,0) Suponha que o escalonador substitua o processo em execução em um processador a cada t milissegundos, e que um processo somente possa executar novamente quando todos os outros processos tiverem sido executados também por t milissegundos em algum processador. Suponha ainda que existam p processadores no computador no qual o sistema operacional está sendo executando. Se existirem xp processos em execução com tempo total de execução de yt milissegundos, qual será o tempo decorrido entre o início e o término de cada um dos processos? Justifique a sua resposta.

   Resp.: Vamos supor, para cada processador, que os mesmos x processos executem alternadamente nele, já que existem xp processos e p processadores. Além disso, como cada processo executa em yt milissegundos, e como os mesmos x processos executam em cada um dos processadores, todos os tempos decorridos são iguais e, portanto, podemos avaliar o tempo pedido somente para o primeiro processo a executar em um dos processadores. Esse processo executa por um total de yt milissegundos, divididos em y quanta, cada um de t milissegundos. Durante cada um dos $y - 1$ intervalos entre esses quanta, cada um dos outros $x - 1$ processos executa também por 1 quantum. Assim, cada um desses intervalos tem duração de $(x - 1)t$ milissegundos. O tempo total é então de $yt + (y-1)(x-1)t = [x(y-1)+1]t$ milissegundos.

5. (2,0) Um problema clássico da comunicação de processos é o problema dos filósofos. Neste problema, cada filósofo passa por períodos alternados de comer e de pensar, estando os filósofos dispostos em torno de uma mesa circular. Nessa mesa existe um prato de espaguete à frente de cada filósofo e um garfo entre dois pratos consecutivos, como podemos ver na figura a seguir, em que temos 5 filósofos (e 5 garfos e pratos de espaguete):



Quando um filósofo está com fome, ele tenta usar os garfos à sua esquerda e à sua direita, pois o espaguete está muito escorregadio e por isso são necessários dois garfos para comê-lo. Se conseguir obter os garfos, o filósofo comerá por um tempo um pouco do espaguete e depois recolocará os garfos na mesa, voltando a pensar até ficar novamente com fome.

A seguir é dada uma solução para o problema com n filósofos baseada em n semáforos binários, sendo que o semáforo $fork[i]$ está associado ao garfo i, $1 \leq i \leq n$. Os semáforos são todos inicializados com o valor 1. Esta solução funcionará se os processos que implementam os filósofos executarem a função *philosopher* dada a seguir, sendo que cada um deles está associado a um identificador i, $1 \leq i \leq n$? Caso a solução dada não funcione, dê uma solução correta para o problema proposto no enunciado.

```
void philosopher(int i)
{
  while (1)
  {
    pensar();
    P(fork[i]);
    if (i > 0)
      P(fork[i − 1]);
    else
      P(fork[n]);
    comer();
    V(fork[i]);
    if (i > 0)
      V(fork[i − 1]);
    else
      V(fork[n]);
  }
}
```

Resp.: A solução não funcionará, pois podemos ter um impasse, como visto a seguir. Suponha que nenhum processo é bloqueado ao executar a primeira operação **P** do procedimento *philosopher*, isto é, a segunda operação **P** somente é executada após todos os processos executarem a primeira operação **P**. Nesse caso, depois de cada processo executar a primeira operação **P**, o valor de cada um dos semáforos binários $fork[i]$, $1 \leq i \leq n$, é igual a 0. Com isso, todos os processos são bloqueados ao executarem a segunda operação **P**, e nunca mais serão desbloqueados, pois nenhum deles poderá executar a operação **V** sobre um dos semáforos binários, a qual é necessária para desbloquear o processo correspondente. A seguir descrevemos um algoritmo correto para resolver o problema, em uma sintaxe similar à da linguagem C. Nesse algoritmo, $estado[i]$, $1 \leq i \leq n$, define o estado atual do filófoso i, que pode estar comendo ($estado[i] = COMENDO$), pensando ($estado[i] = PENSANDO$) ou com fome ($estado[i] = COMFOME$). O último estado ocorre quando o filófoso i deseja comer mas ainda não obteve os dois garfos necessários. O semáforo binário *acesso* é para garantir o acesso exclusivo ao vetor *estado* por cada filósofo. Finalmente, um semáforo binário $f[i]$, $1 \leq i \leq n$, é usado para bloquear o

filósofo i quando ele está com fome mas seus vizinhos estão comendo. De acordo com o enunciado do problema, como inicialmente todos os filósofos estão pensando, então $estado[i] = PENSANDO$, $1 \leq i \leq n$, o semáforo *acesso* é inicializado com 1, e os semáforos $f[i]$, $1 \leq i \leq n$, são inicializados com 0, para garantir que o processo que executa o filósofo i será bloqueado quando a operação **P** sobre esse semáforo for executada.

```
void philosopher(int i)
{
  while (1)
  {
    pensar();
    pegargarfos(i);
    comer();
    soltargarfos(i);
  }
}

void pegargarfos(int i)
{
  P(acesso);
  estado[i] = COMFOME;
  verificargarfos(i);
  V(acesso);
  P(f[i]);
}

void soltargarfos(int i)
{
  P(acesso);
  estado[i] = PENSANDO;
  verificargarfos((i + n - 1)%n);
  verificargarfos((i + 1)%n);
  V(acesso);
}
```

```
void verificargarfos(int i)
{
  garfoesquerdo = (i + n − 1)%n;
  garfodireito = (i + 1)%n;
  if ( (estado[i] == COMFOME) &&
       (estado[garfoesquerdo] != COMENDO ) &&
       (estado[garfodireito] != COMENDO ) )
  {
    estado[i] = COMENDO;
    V(f[i]);
  }
}
```

6. (2,0) Cinco trabalhos em lote, A até E, chegam a um centro de computação quase ao mesmo tempo. Eles têm tempos de execução estimados em 12, 4, 6, 16 e 2 minutos, respectivamente. Suas prioridades (externamente determinadas) são 3, 7, 9, 1 e 5, respectivamente, com 9 sendo a do trabalho mais prioritário. Para cada um dos seguintes algoritmos de escalonamento, determine o tempo de término médio dos processos. Ignore o acréscimo (*overhead*) da comutação de processos.

   (a) (0,5) *Round robin* com um quantum de 2 minutos de duração, e com os processos inicialmente executando na ordem A, C, E, D e B.

   (b) (0,5) Escalonamento por prioridade.

   (c) (0,5) O primeiro a chegar é o primeiro a ser servido (execução na ordem dada no enunciado).

   (d) (0,5) Trabalho mais curto primeiro.

   Para (a), suponha que o sistema é multiprogramado e que cada trabalho receba a sua justa parte da CPU. Para (b) a (d), suponha que um único trabalho execute por vez, até terminar ou ser suspenso. Nenhum dos trabalhos faz E/S.

   Resp.: A seguir damos o tempo de término médio para cada um dos algoritmos de escalonamento.

(a) Pelo enunciado, vemos que a ordem de execução dos processos é como a dada na tabela a seguir. Nesta tabela mostramos como os processos são escolhidos pelo algoritmo, sendo que cada coluna refere-se à execução de um processo dando o tempo de início de cada quantum e o processo correspondente. Pela tabela, vemos que os tempos de término dos processos A, B, C, D e E, são de, respectivamente, 34, 18, 22, 40 e 6 unidades de tempo, o que nos dá um tempo médio de 24 minutos.

0	2	4	6	8	10	12	14	16	18
A	C	E	D	B	A	C	D	B	A

20	22	24	26	28	30	32	34	36	38
C	D	A	D	A	D	A	D	D	D

(b) Como cada trabalho executa no processador até o seu término, pois nenhum trabalho é suspenso (por não fazer operações de E/S) então, de acordo com as prioridades, C é o primeiro a executar, por 6 minutos. Depois de C, os trabalhos B, E, A e D executam, nessa ordem, por respectivamente 4, 2, 12 e 16 minutos. Logo, os tempos de término dos trabalhos de A a E são, respectivamente, de 24, 10, 6, 40 e 12. Com isso, o tempo médio é de 18,4 minutos (isto é, 18 minutos e 24 segundos).

(c) Como todos os trabalhos executam exclusivamente no processador até o seu término, o trabalho A executa no processador por 12 minutos, seguido pelos trabalhos B, C, D e E, que executam no processador por, respectivamente, 4, 6, 16 e 2 minutos. Logo, os tempos de término dos trabalhos de A a E são, respectivamente, de 12, 16, 22, 38 e 40. Com isso, o tempo médio é de 25,6 minutos (isto é, 25 minutos e 36 segundos).

(d) Neste caso, a ordem de execução dos trabalhos é a seguinte: E, B, C, A e D. Como mais uma vez cada processo executa exclusivamente no processador até o seu término, E executa por 2 minutos,

seguido pelos trabalhos B, C, A e D, que executam por, respectivamente, 4, 6, 12 e 16 minutos. Logo, os tempos de término dos trabalhos de A a E são, respectivamente, de 24, 6, 12, 40 e 2, o que nos dá um tempo médio de 16,8 minutos (isto é, 16 minutos e 48 segundos).