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 - Primeiro Semestre de 2014

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.

Nome -
Assinatura -

---

1. (1,5) Suponha que um programa A leve 30s para executar no processador e que, para executar a sua tarefa, ele precise fazer uma operação de E/S por 5s. Se esse programa fosse executado em um sistema anterior ao da terceira geração, qual seria a fração de tempo do processador desperdiçada com a operação de E/S? Esse desperdício ainda ocorreria nos sistemas posteriores ao da segunda geração, se um programa B, que não faz operações de E/S, estivesse pronto para executar antes de A fazer a operação? Justifique a sua resposta.

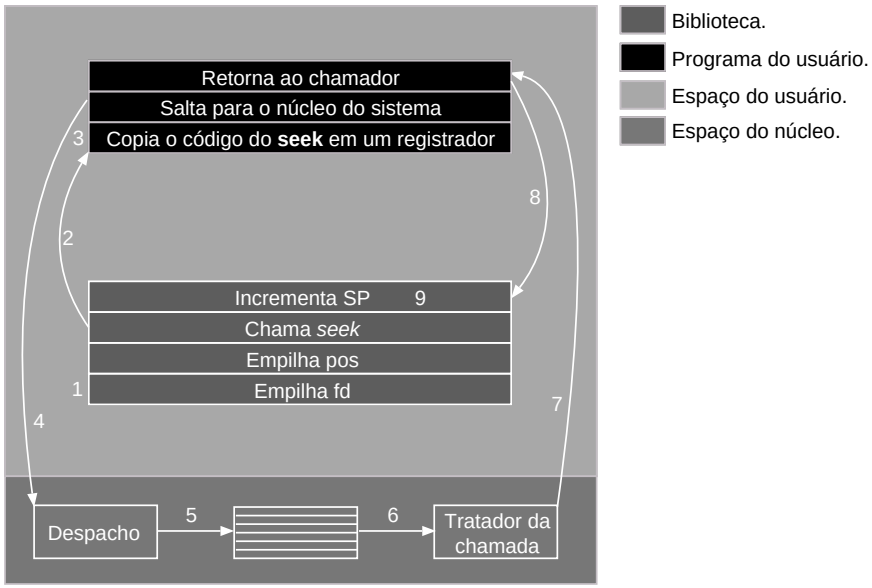
   Resp.: -Note que o programa somente poderia ser executado em um sistema da segunda geração, pois na primeira geração o programador

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 5s de E/S faria parte do tempo de execução do programa. Além disso, como o tempo de execução do programa seria de 35s (o tempo de execução no processador mais o tempo da operação de E/S), e como o processador ficaria ocioso por 5s desses 35s, então a fração de tempo do processador desperdiçada seria de $5/35 = 1/7 \approx 0,143$, ou seja, aproximadamente 14,3% do tempo de execução do programa.
-Agora, o tempo de ociosidade dependeria do tempo de execução do programa B e de B executar depois de A. 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. Caso B executasse antes de A, então o processador ainda ficaria ocioso por 5s, pois não existiria nenhum outro programa além de B para ser executado enquanto A fizesse a sua operação de E/S. Agora, se A executasse antes de B, o processador somente não ficaria ocioso se o tempo de execução de B no processador fosse de pelo menos 5s. Caso fosse menor, o processador ainda ficaria ocioso, mas agora por um tempo igual a 5 menos o tempo de execução de B.

2. (1,5) Na aula 2 vimos os passos executados ao chamarmos a função de biblioteca *read*, a qual implementa a chamada ao sistema operacional **read**. Quais serão agora os passos executados se desejarmos fazer a chamada ao sistema operacional **seek**, usando a função de biblioteca *seek*, para a qual passamos o descritor *fd* do arquivo e a posição do arquivo *pos* para a qual desejamos saltar?

   Resp.: A seguir mostramos a figura obtida, similar à dada na última transparência da aula 2, ao fazermos a chamada ao sistema operacional **seek**. No passo 1, o processo do usuário que executou a função *seek* empilha os parâmetros *fd* e *pos* passados a essa função. Após empilhar os parâmetros o processo, no passo 2, chama a função da biblioteca *seek*. Após ser chamada, esta função então coloca, no passo 3 e em um lugar pré-determinado pelo sistema operacional, o código que identifica a chamada ao sistema operacional **seek**. Depois disso, no passo 4, essa função executa a instrução TRAP do processador, o que mudará

o processador do modo usuário para o modo supervisor e fará com que o controle seja transferido para o endereço do núcleo responsável pelo tratamento das chamadas ao sistema operacional. No passo 5, a parte do núcleo responsável por tratar as chamadas obtém, usando o código (passado pela biblioteca) como um índice em uma tabela com os endereços das funções que executam as chamadas, o endereço da função do núcleo que executa a chamada **seek**. Então, no passo 6, o sistema operacional executa esta função, denominada de **tratador da chamada seek**. Depois de esse tratador executar as tarefas necessárias para saltar para a posição *pos* do arquivo então, no passo 7, o processador será alternado do modo supervisor para o modo usuário, e o controle será passado à instrução, da função *seek* da biblioteca, posterior à instrução TRAP. Após fazer as finalizações necessárias depois de saltar para a posição *pos* arquivo, a função da biblioteca então passa, no passo 8, o controle novamente ao processo do usuário, na instrução seguinte à que chamou a função. Finalmente, no passo 9, o processo do usuário incrementa o ponteiro da pilha SP com o valor necessário para remover os parâmetros *fd* e *pos* colocados na pilha antes de chamarmos a função *seek*.



3. (1,5) Suponha que o sistema operacional esteja executando diretamente

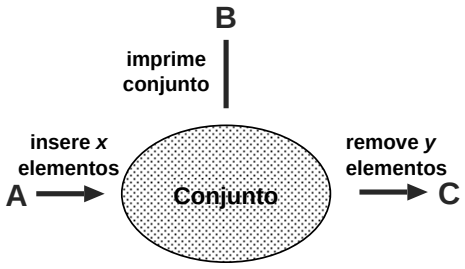
sobre o hardware de um computador cujas operações de E/S demorem 0,25ms. Suponha ainda que um processo tenha executado por 7s e que, durante a sua execução, tenha feito 8000 operações de E/S. Se o sistema operacional agora executar sobre uma máquina virtual que reduza a velocidade do processador em 20% e a velocidade das operações de E/S em 50%, quantas operações de E/S o programa poderá fazer para o seu tempo de execução ainda ser de 7s? Justifique a sua resposta.

Resp.: Como cada operação de E/S demora 0,25ms e como o processo fez 8000 operações, então 2000ms do tempo de execução de 7s ou 7000ms desse processo foram gastos em operações de E/S, quando ele executou no sistema operacional sobre o hardware do computador. Logo, o processo executou no processador do hardware por $7000 - 2000 = 5000$ms. Note que a velocidade do processador ser reduzida em 20% significa que a velocidade do processador virtual é 80% da velocidade do processador real, o que por sua vez significa, durante os 5000ms, que somente 80% das instruções serão executadas. Com isso, quando o processo executar no sistema operacional sobre a máquina virtual, o tempo de execução dele no processador virtual será de $5000/0,8 = 6250$ms. Agora o processo, para executar no mesmo tempo de 7000ms, somente poderá executar operações de E/S por 750ms. Como o tempo gasto por cada operação de E/S na máquina virtual é de $0,25/0,5 = 0,5$ms, pois ela reduz a velocidade das operações de E/S em 50%, então o processo poderá executar $750/0,5 = 1500$ operações de E/S.

4. (1,5) 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 cada um dos outros processos tiver sido executado também por t milissegundos em algum processador. Se existirem x processos no sistema, cada um com um tempo de execução de at, qual será o tempo necessário para executar todos os processos se somente 1 processador estiver disponível? E se existirem agora x/b processadores disponíveis, supondo que b é um divisor de x? Justifique a sua resposta.

Resp.: -Se existir somente 1 processador então, como desejamos determinar o tempo necessário para executar todos os processos, o tempo será igual ao tempo necessário para executar todos os x processos, e não dependerá do algoritmo de escalonamento usado. Logo, o tempo total de execução será de axt milissegundos.
-No caso em que existem x/b processadores, vamos supor, para cada processador, que os mesmos b processos executem alternadamente nele, já que existem x processos e x/b processadores. Agora, como em cada quantum de t milissegundos sempre x/b processos executam em paralelo, então o tempo de término dos processos é o tempo necessário para executar todos os b processos em um dos processadores. Como esse tempo também independe do algoritmo de escalonamento usado, então o tempo de execução é agora de abt milissegundos.

5. (1,5) O conjunto dado na figura a seguir pode armazenar até n elementos e é compartilhado por três processos, A, B e C. O processo A sempre coloca x elementos no conjunto, o processo B sempre imprime o conteúdo atual do conjunto e o processo C sempre remove y elementos do conjunto. Como os semáforos binários podem ser usados para garantir o correto funcionamento dos processos A, B e C, supondo que o conjunto inicialmente possua $e < n$ elementos? Justifique a sua resposta.



Resp.: Para garantir o correto funcionamento dos processos A, B e C, isto é, a sua correta sincronização, vamos precisar de três semáforos binários, *acesso*, *livres* e *ocupadas*. Além disso, vamos precisar de uma variável E, inicializada tal que $E = e$, para armazenar o número de elementos no conjunto. O semáforo *acesso* é usado para garantir o acesso exclusivo ao conjunto e à variável E, e é inicializado com

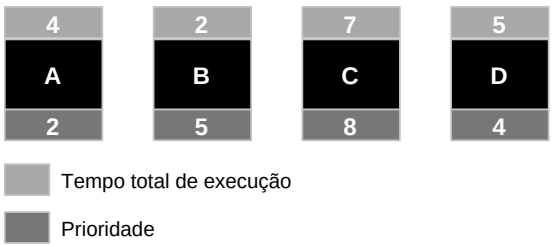
o valor 1, pois nenhum processo ainda está executando. O semáforo *livres* é usado para bloquear A se existirem menos do que x posições livres no conjunto, e o semáforo *ocupadas* é usado para bloquear B se existirem menos do que y elementos no conjunto. Como os semáforos *livres* e *ocupadas* são usados para bloquear, respectivamente, A e B, e como somente executaremos uma operação **P** sobre eles quando for necessário (como veremos nos algoritmos a seguir), então o valor inicial de ambos os semáforos é 0. O processo A deve usar a função $InserirElemento(a_1,a_2,\ldots,a_x)$ dada a seguir para inserir x elementos no conjunto. Já o processo B deve usar a função *ImprimirConjunto*(**void**) a seguir para imprimir os elementos do conjunto. Finalmente, o processo C deve usar a função $RemoverElementos(a_1,a_2,\ldots,a_y)$ dada a seguir para remover y elementos do conjunto. Nos algoritmos estamos também supondo que os valores de x e y são acessíveis por todos os processos. Observe que a função *ImprimirConjunto* somente precisa usar o semáforo *acesso*, pois somente imprime os elementos do conjunto.

```
void InserirElemento(a1, a2, ..., ax)
{
  P(acesso);
  if (n − E < x)
  {
    V(acesso);
    P(livres);
    P(acesso);
  }
  // Código para inserir os elementos a1, a2, ..., ax no conjunto.
  E = E + x
  if (E ≥ y)
    V(ocupadas);
  V(acesso);
}
```

```
void ImprimirConjunto(void)
{
  P(acesso);
  // Código para imprimir todos os elementos no conjunto.
  V(acesso);
}

void RemoverElementos(a_1, a_2, ..., a_y)
{
  P(acesso);
  if (E < y)
  {
    V(acesso);
    P(ocupadas);
    P(acesso);
  }
  // Código para remover y elementos e armazenálos em a_1, a_2, ..., a_y.
  E = E - y
  if (n - E ≥ x)
    V(livres);
  V(acesso);
}
```

6. (2,5) Suponha que os processos da figura dada a seguir tenham acabado de entrar no estado pronto, e que sejam os únicos processos em execução. Quais seriam os tempos de término desses processos se o sistema operacional usasse, ao escalonar os processos:



Nota: Existem várias respostas para os itens (a) e (b) desta questão. Em cada um deles, somente é preciso fornecer uma dessas respostas, a

qual será considerada correta se, dada a sua suposição para o item, ela estiver coerente. Para o item (a), é necessário escolher a duração do quantum e a ordem inicial dos processos. Para o item (b), é necessário escolher como se dá redução das prioridades.

(a) (0,9) o algoritmo de *round robin*.

Resp.: Supondo que cada quantum equivalha a 1 unidade de tempo, e que as quatro escolhas iniciais do escalonador tenham sido, em ordem, A, B, C e D, obtemos a sequência de execução dada nas tabelas a seguir. A primeira linha de cada tabela mostra, da esquerda para a direita, a ordem de execução dos processos no processador. A segunda linha mostra, em cada coluna, o tempo decorrido após a execução do processo. Como podemos ver pelas tabelas, os tempos de término dos processos A, B, C e D são de, respectivamente, 12, 6, 18 e 16 unidades de tempo.

-	A	B	C	D	A	B	C	D	A
0	1	2	3	4	5	6	7	8	9

C	D	A	C	D	C	D	C	C
10	11	12	13	14	15	16	17	18

(b) (0,9) o algoritmo de prioridades.

Resp.: Nesta resposta, vamos supor que cada redução de 1 unidade da prioridade do processo em execução equivalha a 1 unidade de tempo. Vamos supor também que um processo execute até que a sua prioridade deixe de ser a maior de todas. Nas tabelas a seguir mostramos, na primeira linha, a prioridade do processo antes de ele executar no processador. Na segunda linha mostramos, da esquerda para a direita, a ordem de execução dos processos no processador. Finalmente na última linha mostramos, para cada coluna, o tempo após o processo da coluna ter sido executado. Pelas tabelas, vemos que os tempos de término dos processos A,

B, C e D são de, respectivamente, 18, 6, 13 e 17 unidades de tempo.

-	8	7	6	5	5	4	4	4	3
-	C	C	C	C	B	B	D	C	C
0	1	2	3	4	5	6	7	8	9

3	2	2	2	1	1	0	0	-1
D	D	A	C	D	A	A	D	A
10	11	12	13	14	15	16	17	18

(c) (0,7) o algoritmo do trabalho mais curto primeiro.

Resp.: Como vimos na aula 6, no algoritmo do trabalho mais curto primeiro, os processos são executados, em ordem crescente, de acordo com os seus tempos de execução. Além disso, quando um processo começa a executar, ele executa exclusivamente no processador até terminar. Então a única possível ordem de execução é B, A, D e C. O processo B então executaria do tempo 0 até o 2, o processo A do tempo 2 até o 6, o processo D do tempo 6 até o 11 e, finalmente, o processo C do tempo 11 até o 18. Logo, os tempos de término dos processos A, B, C e D seriam de, respectivamente, 6, 2, 18 e 11 unidades de tempo.