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 2016

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 -

---

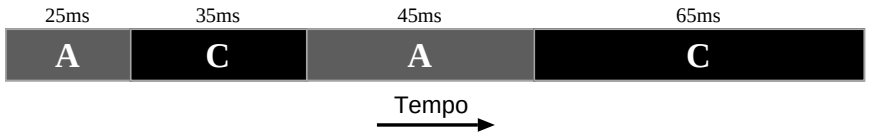
1. (1,5) Suponha que um programa A tenha precisado executar no processador por 70ms e que, durante a sua execução, tenha precisado fazer uma operação de E/S, com duração de 30ms, após executar por 25ms no processador. Suponha ainda que o sistema operacional use a multiprogramação somente para evitar a ociosidade do processador durante a execução de operações de E/S. Responda, justificando a sua resposta:

   (a) (0,7) Como um programa B, que não faz operações de E/S, pode evitar totalmente a ociosidade do processador ao executar o programa A?

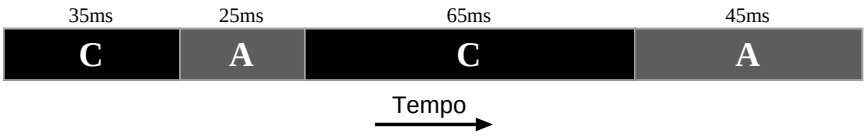
Resp.: Como o programa B não faz operações de E/S então, para evitar totalmente a ociosidade do processador quando o programa A faz E/S, basta B executar por pelo menos 30ms, o tempo de duração da única operação de E/S de A, logo após A ter feito essa operação de E/S.

(b) (0,8) Se um programa C, que precise executar por 100ms no processador, fizer uma operação de E/S após executar por 35ms no processador, é possível evitar totalmente a ociosidade do processador ao executar o programa A?

Resp.: Temos duas possibilidades, dependendo de C executar antes ou depois de A. Se A começar a executar antes de C, a ociosidade do processador ao executar a operação de E/S de A será evitada, porque C executa por 35ms antes de fazer a sua operação de E/S, tempo maior do que o tempo de 30ms da operação de E/S de A. Agora, como A precisa executar por mais 45ms antes de terminar então, para evitar a ociosidade do processador, o tempo da operação de E/S de C não pode ser maior do que 45ms. Este caso é ilustrado na figura dada a seguir:



Agora, se C começar a executar antes de A, não teremos ociosidade do processador quando C executar a sua operação de E/S, se o tempo desta operação for menor ou igual do que o tempo de execução de A antes de ele fazer a sua operação de E/S, ou seja, se for menor ou igual a 25ms. Como o tempo de 65ms que C precisa executar para terminar é maior do que o tempo de 30ms da operação de E/S de A, então o processador não ficará ocioso quando A executar a sua operação de E/S. Este caso é ilustrado na figura dada a seguir:

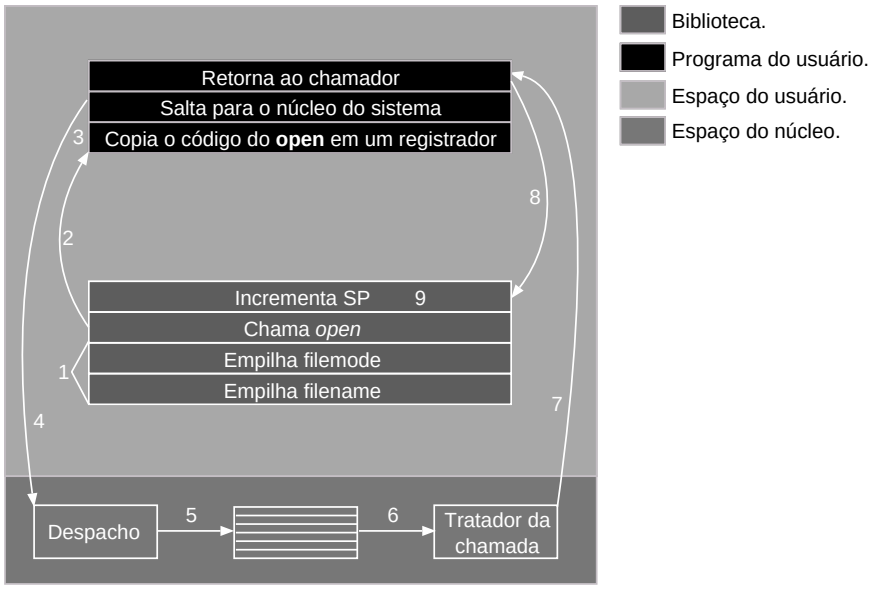


Logo, concluímos que o processador não ficará ocioso, independentemente de A executar antes ou depois de C, se o tempo da operação de E/S de C for menor ou igual a 25ms.

2. (1,5) Na aula 2 vimos os passos executados ao chamarmos a função da biblioteca *read*, a qual implementa a chamada ao sistema operacional **read**. Quais serão os passos executados se desejarmos fazer a chamada ao sistema operacional **open**, usando a função da biblioteca *open*, para a qual passamos o caminho do arquivo no sistema de arquivos, dado em *filename*, e o modo de abertura do arquivo, dado em *filemode*?

   Resp.: A seguir mostramos a figura obtida, similar à dada na última transparência da aula 2, ao fazermos a chamada ao sistema operacional **open**. No passo 1, o processo do usuário que executou a função *open* empilha os parâmetros *filename* e *filemode* passados a essa função. Após empilhar os parâmetros o processo, no passo 2, chama a função da biblioteca *open*. 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 **open**. 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 **open**. Então, no passo 6, o sistema operacional executa esta função, denominada de **tratador da chamada open**. Depois de esse tratador executar as tarefas necessárias para abrir o arquivo *filename* com o modo *filemode* então, no passo 7, o processador será alternado do modo supervisor para o modo usuário, e o controle será passado à instrução, da função *open* da biblioteca, posterior à instrução TRAP. Após fazer as finalizações

necessárias depois de abrir o arquivo *filename*, 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 *filename* e *filemode* colocados na pilha antes de ser chamada a função *open*.

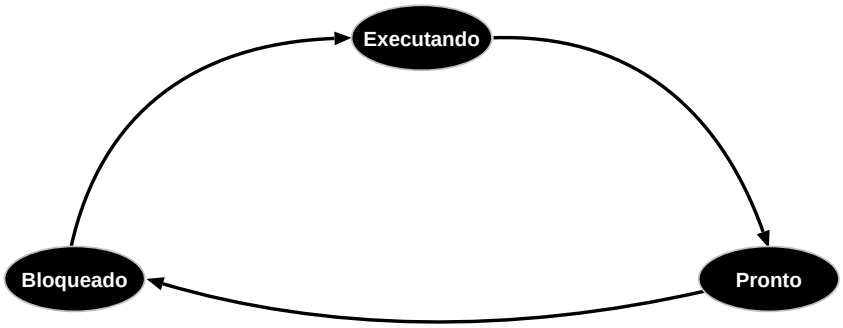


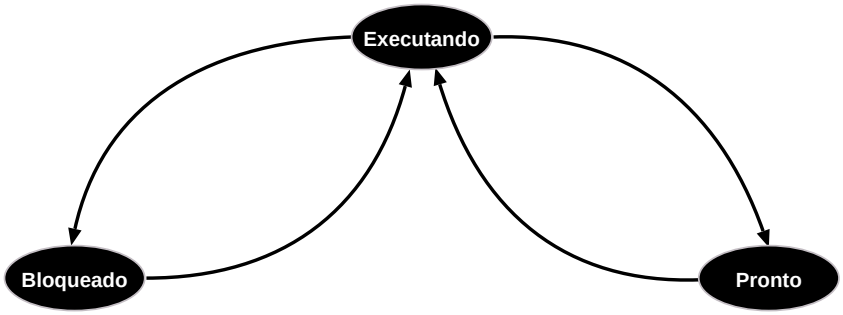
3. (2,0) Suponha que o sistema operacional esteja executando diretamente sobre o hardware de um computador cujas operações de E/S demorem $1,5$ms. Suponha ainda que um processo tenha executado por 7s e que, durante a sua execução, tenha feito 1800 operações de E/S. Se o sistema operacional agora executar sobre uma máquina virtual que reduza a velocidade do processador em 60% e para a qual cada operação de E/S demore $2,0$ms, quantas operações de E/S deverão ser executadas, quando o processo executar sobre a máquina virtual, para que o tempo de execução do processo seja o dobro do tempo de execução sobre a máquina real? Justifique a sua resposta.

Resp.: Como o tempo total de execução é de 7s ou 7 000ms, e como o processo faz 1 800 operações de E/S com tempo de $1,5$ms, então

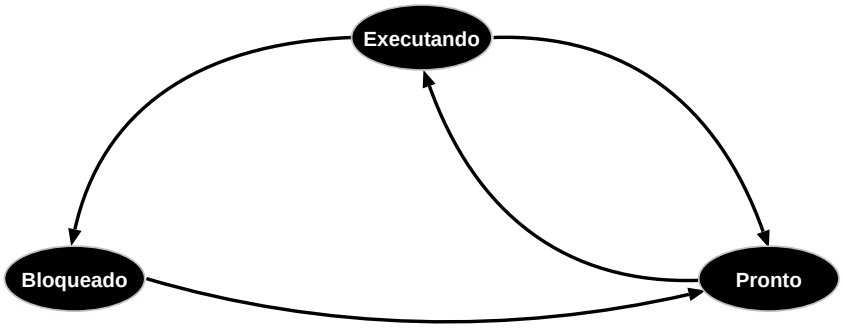
2 700ms do tempo de execução de 7 000ms desse processo são gastos com operações de E/S, quando ele executa no sistema operacional sobre o hardware do computador. Logo, o processo executa no processador do hardware por $7\,000 - 2\,700 = 4\,300$ms. Note que a velocidade do processador ser reduzida em 60% significa que a velocidade do processador virtual é 40% da velocidade do processador real, o que por sua vez significa que, durante os 4 300ms, somente 40% das instruções são executadas. Com isso, quando o processo executa no sistema operacional sobre a máquina virtual, o tempo de execução dele no processador virtual é de $\frac{4\,300}{0{,}4} = 10\,750$ms. Para que o tempo de execução sobre a máquina virtual seja o dobro do tempo sobre a máquina real, ou seja, um tempo igual a 14s ou 14 000ms então, como o tempo de execução no processador virtual é de 10 750ms, $14\,000 - 10\,750 = 3\,250$ms precisam ser gastos com operações de E/S quando o processo executar sobre a máquina virtual. Agora, como o tempo de cada operação de E/S na máquina virtual é de $2,0$ms, então podemos concluir que, para o tempo de execução ser o dobro, o processo precisa executar $\frac{3\,250}{2{,}0} = 1\,625$ operações de E/S.

4. (1,0) Suponha que dois alunos tenham fornecido os diagramas de transições, entre os possíveis estados de um processo, dados na figura a seguir. Algum dos diagramas dados está correto? Se ambos estiverem errados, qual deles possui menos erros? Justifique a sua resposta.



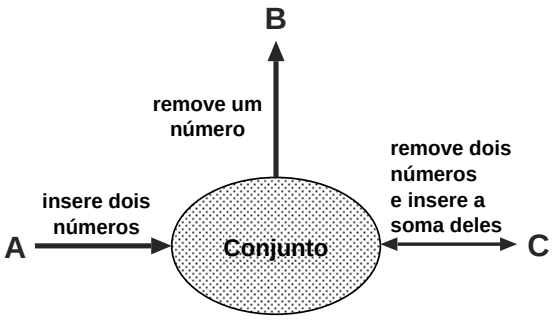


Resp.: Pelo diagrama de transições correto, dado na figura a seguir, percebemos que os dois diagramas estão errados. No caso do primeiro diagrama, notamos que duas transições estão invertidas, a do estado *Bloqueado* para o estado *Executando* e a do estado *Pronto* para o estado *Bloqueado*. A transição de *Bloqueado* para *Executando* está incorreta porque o processo em execução, ou algum processo no estado *Pronto*, pode ser mais prioritário do que o processo que foi desbloqueado. Finalmente, a transição de *Pronto* para *Bloqueado* está incorreta porque um processo no estado *Pronto*, por não estar executando no processador, não pode esperar pelo término de um evento externo. Além disso, falta a transição de *Pronto* para *Executando*, totalizando três erros. Já o segundo diagrama tem dois erros, a aresta incorreta de *Bloqueado* para *Executando*, e a aresta ausente de *Bloqueado* para *Pronto*. Logo, ambos os diagramas estão errados e o segundo tem menos erros do que o primeiro.



5. (2,0) O conjunto dado na figura a seguir pode armazenar até $n > 2$ números e é compartilhado por três processos, A, B e C. O processo A

sempre insere dois números no conjunto, o processo B sempre remove um número do conjunto, e o processo C sempre remove dois números do conjunto e depois insere a soma deles no conjunto. Como os semáforos podem ser usados para garantir o correto funcionamento de A, B e C, supondo que o conjunto inicialmente possua 2 números? Justifique a sua resposta.



Resp.: A seguir mostramos como três semáforos, um binário e dois de contagem, podem ser usados para implementar os códigos dos processos. O semáforo binário, chamado *acesso*, é usado para garantir o acesso exclusivo ao conjunto. O primeiro semáforo de contagem, chamado *vazias*, conta o número de entradas não usadas no conjunto, e é usado para bloquear o processo A quando o conjunto estiver cheio. Finalmente, o segundo semáforo de contagem, chamado *cheias*, conta o número de entradas usadas no conjunto, e é usado para bloquear os processos B e C quando o conjunto estiver vazio. Como inicialmente o conjunto tem 2 números e não está sendo usado, então os semáforos *vazias*, *cheias* e *acesso* são inicializados, respectivamente, com $n-2$, 2 e 1. A seguir mostramos os códigos para os processos A, B e C, sendo que a função $removenumero()$ remove e retorna um dos números do conjunto, e a função $inserenumero(a)$ insere o número a no conjunto:

```
void ProcessoA(void)
{
  while (1);
  {
    // Usa a operação P sobre vazias para garantir que pelo menos
    // duas entradas estejam livres no conjunto.
    P(vazias);
    P(vazias);
    // Código para gerar dois números e salvá-los em a1 e a2.
    // Garante o acesso exclusivo ao conjunto.
    P(acesso);
    // Insere a1 e a2 no conjunto.
    inserenumero(a1);
    inserenumero(a2);
    // Libera o acesso exclusivo ao conjunto.
    V(acesso);
    // Usa a operação V sobre cheias para registrar que dois
    // números foram inseridos no conjunto.
    V(cheias);
    V(cheias);
  }
}
```

```
void ProcessoB(void)
{
  while (1);
  {
    // Garante que existe pelo menos um número no conjunto.
    P(cheias);
    // Garante o acesso exclusivo ao conjunto.
    P(acesso);
    // Remove um número do conjunto e o armazena em a.
    a = removenumero();
    // Libera o acesso exclusivo ao conjunto.
    V(acesso);
    // Código para usar o número a.
    // Usa a operação V sobre vazias para registrar que um
    // número foi removido do conjunto.
    V(vazias);
  }
}
```

```
void ProcessoC(void)
{
  while (1);
  {
    // Garante que existe pelo menos dois números no conjunto.
    P(cheias);
    P(cheias);
    // Garante o acesso exclusivo ao conjunto.
    P(acesso);
    // Remove dois números do conjunto e os armazena em a1 e a2.
    a1 = removenumero();
    a2 = removenumero();
    // Insere a soma a1 + a2 dos números no conjunto.
    inserenumero(a1 + a2);
    // Libera o acesso exclusivo ao conjunto.
    V(acesso);
    // Usa a operação V sobre vazias para registrar que um número
    // foi removido do conjunto.
    V(vazias);
    // Usa a operação V sobre cheias para registrar a inserção da
    // soma.
    V(cheias);
  }
}
```

6. (2,0) Suponha que dois processos, A e B, compartilhem uma pilha com tamanho ilimitado sendo que, a cada unidade de tempo, A adiciona um elemento na pilha, e B, se a pilha tiver pelo menos três elementos, remove três elementos da pilha. Suponha ainda que A precise executar no processador por 15 unidades de tempo e que B precise executar no processador por 8 unidades de tempo. Para cada um dos algoritmos de escalonamento dados nos itens a seguir, quantos elementos estarão na pilha após o término dos processos A e B, sabendo que A e B nunca são bloqueados, e que a pilha está inicialmente vazia? Para cada item, justifique a sua resposta usando o algoritmo em questão.

   (a) (1,0) *Round robin*, com um quantum de 2 unidades de tempo, e supondo que B seja o primeiro processo a executar no processador.

   Resp.: Pelo enunciado, vemos que a ordem de execução dos processos é como dado 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, o processo correspondente, e a quantidade de elementos na pilha depois que o processo executou. Devido ao tempo do processo A não ser múltiplo do tamanho do quantum de 2 unidades de tempo, A somente usará 1 unidade de tempo do seu último quantum. Note que se existirem menos do que 3 elementos na pilha, B não removerá nenhum elemento no seu quantum e que, se existirem menos do que 6 elementos, B removerá somente 3 elementos da pilha. Como podemos ver pela tabela, a pilha terá 9 elementos após o término dos processos.

0	2	4	6	8	10	12	14	16	18	20	22
B	A	B	A	B	A	B	A	A	A	A	A
0	2	2	4	1	3	0	2	4	6	8	9

(b) (1,0) Prioridades, supondo que a prioridade do processo em execução seja reduzida em 2 unidades a cada 3 unidades de tempo, que um dos processos somente deixe de executar no processador se o outro processo passa a ter a maior prioridade, e que as prioridades iniciais de A e B sejam, respectivamente, 12 e 7.

Resp.: A seguir mostramos a ordem de execução dos processos obtida ao usar o algoritmo por prioridades. A tabela é similar à anterior, mas agora tem mais uma linha, a última, que indica a prioridade de cada processo antes de ele executar na unidade de tempo dada em cada coluna. Note que ao executar por 3 unidades de tempo, adicionalmente ao que já descrevemos para o item anterior, B somente removerá 9 elementos se a pilha possuir pelo menos 9 elementos. Como podemos ver pela tabela, a pilha estará vazia após o término dos processos.

0	3	6	9	12	15	18	21
A	A	A	B	A	B	A	B
3	6	9	0	3	0	3	0
12	10	8	7	6	5	4	3