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 2020

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.

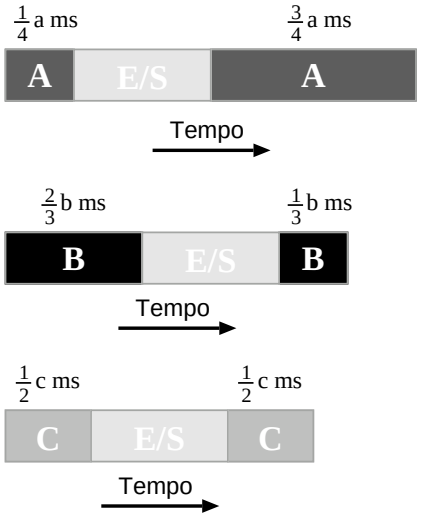
Nome -
Assinatura -

---

1. (2,0) Suponha que três programas, A, B e C, com tempos de execução de, respectivamente, a, b e c ms, estejam prontos para executarem no processador. O programa A faz uma operação de E/S após executar por 1/4 do seu tempo de execução. Já o programa B faz uma operação de E/S após executar por 2/3 do seu tempo de execução. Finalmente, o programa C faz uma operação de E/S após executar por metade do seu tempo de execução. Cada programa faz somente uma operação de E/S.

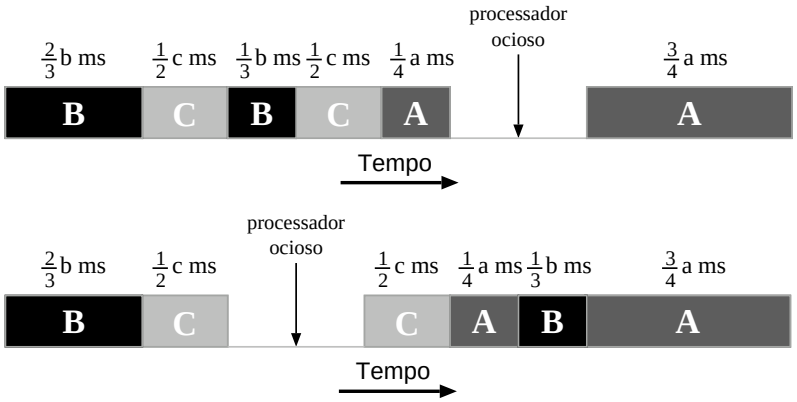
Se a multiprogamação for usada somente para evitar a ociosidade do processador quando operações de E/S são feitas, e se A somente puder executar após C terminar, será possível evitar completamente a ociosidade do processador? Se for possível, quais serão os tempos máximos das operações de E/S feitas pelos programas? Justifique a sua resposta.

Resp.: Pelo enunciado, vemos que o programa A executará por $\frac{a}{4}$ ms antes de fazer a sua operação de E/S, e por mais $\frac{3a}{4}$ ms após o término dessa operação. Já o programa B executará por $\frac{2b}{3}$ ms antes de fazer a sua operação de E/S, e por mais $\frac{b}{3}$ ms depois de fazer essa operação. Finalmente, o programa C executará por $\frac{c}{2}$ ms antes de fazer a sua operação de E/S, e por mais $\frac{c}{2}$ ms após fazer essa operação. A figura a seguir ilustra esse comportamento de A, B e C.



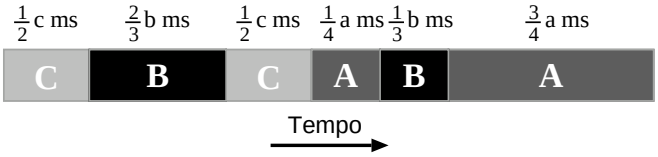
Para tentar evitar a ociosidade do processador, é necessário executar um outro programa quando o programa em execução faz a sua operação de E/S. Se B for o primeiro a executar, poderemos evitar a ociosidade da sua operação de E/S fazendo C executar mas, como A somente pode executar após C terminar, então não poderemos evitar, ao mesmo tempo, a ociosidade quando A e C fizerem as suas operações de E/S. Se a segunda parte da execução de B for usada para evitar a ociosidade durante operação de E/S feita por C, então não será possível

evitar a ociosidade durante a operação de E/S feita por A, já que A somente pode executar após C terminar. Veja a primeira parte da figura a seguir. Agora, se o tempo da operação de E/S de B for grande o suficiente para podermos usar a segunda parte de sua execução para evitar a ociosidade decorrente da operação de E/S feita por A, então o processador ficará ocioso enquanto C fizer a sua operação de E/S. Veja a segunda parte da figura a seguir.



Por outro lado, se C for o primeiro a executar, poderemos usar a primeira parte da execução de B, antes de ele fazer a sua operação de E/S, para evitar a ociosidade durante a operação de E/S feita por C. Além disso, poderemos usar a segunda parte da execução de C mais a primeira parte da execução de A para evitar a ociosidade durante a operação de E/S feita por B. Finalmente, a segunda parte da execução de B poderá ser usada para evitar a ociosidade da operação de E/S feita por A. Nesse caso, mostrado na figura a seguir, poderemos portanto evitar a ociosidade do processador durante todas as operações de E/S. O tempo máximo da operação de E/S de A será de $\frac{b}{3}$ ms, pois esse é o tempo de execução de B após ele fazer a sua operação de E/S. Já o tempo máximo da operação de E/S feita por B será a soma dos tempos de execução de C após ele fazer a sua operação de E/S e de A antes de ele fazer a sua operação de E/S, ou seja, será de $\left(\frac{c}{2} + \frac{a}{4}\right)$ ms. Finalmente, o tempo máximo da operação de E/S de C será de $\frac{2b}{3}$ ms, pois esse é o tempo de execução de B antes de ele fazer a sua operação

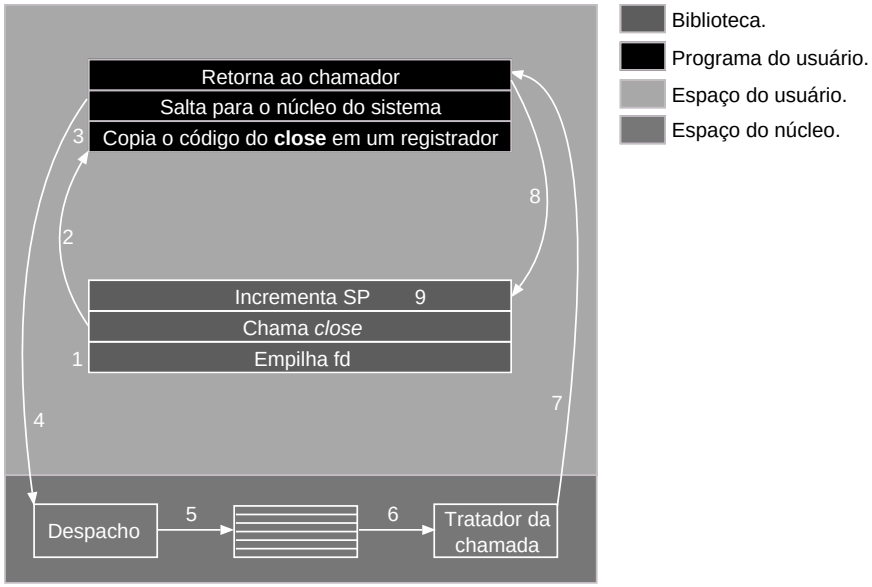
de E/S.



2. (1,0) 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 **close**, usando a função da biblioteca *close*, para a qual passamos somente o descritor do arquivo no sistema de arquivos, dado em *fd*?

   Resp.: A seguir mostramos a figura obtida, similar à dada no último slide da aula 2, ao fazermos a chamada ao sistema operacional **close**. No passo 1, o processo do usuário que executou a função *close* empilha o parâmetro *fd* passado a essa função. Após empilhar o parâmetro o processo, no passo 2, chama a função da biblioteca *close*. 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 **close**. 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 identificador passado pela biblioteca como um índice em uma tabela, o endereço da função do núcleo que executa a chamada **close**. Então, no passo 6, o sistema operacional executa essa função, denominada de **tratador da chamada close**. Depois de esse tratador executar as tarefas necessárias para fechar o arquivo identificado pelo descritor *fd* então, no passo 7, o processador será alternado do modo supervisor de volta ao modo usuário e o controle será passado à instrução, da função *close* da biblioteca, posterior à instrução TRAP. Após fazer as finalizações necessárias depois de o arquivo ter sido fechado, 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 ajusta o ponteiro da pilha SP para remover o parâmetro *fd* colocado na pilha antes de ser chamada a função *close*.

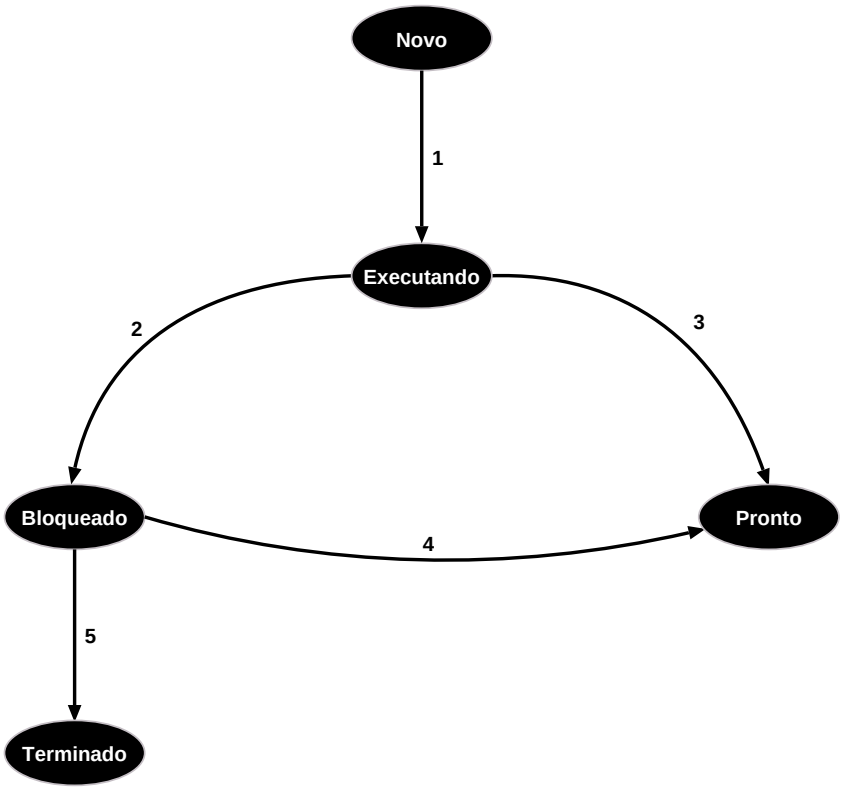


3. (2,0) Suponha que o sistema operacional esteja executando diretamente sobre o hardware de um computador onde cada operação de E/S demore 1,25 ms. Suponha ainda que um processo tenha executado por x ms e que, durante a sua execução, tenha feito 4 500 operações de E/S. Se o sistema operacional agora executar sobre uma máquina virtual que reduza a velocidade do processador em 55% e a velocidade das operações de E/S em 50%, e se além disso forem feitas 1 500 operações de E/S a menos do que sobre o hardware, qual será o valor de x se o tempo de execução do processo na máquina virtual for de 20 000 ms? Justifique a sua resposta.

Resp.: Como o tempo total de execução é de x ms, e como o processo faz 4 500 operações de E/S com duração de $1,25$ ms cada, então 5 625 ms dos x ms são gastos com operações de E/S quando a execução ocorre sobre o hardware do computador. Logo, o processo executa no

processador desse hardware por $(x - 5\,625)$ ms. Note que a velocidade do processador ser reduzida em 55% significa que a velocidade do processador virtual é 45% da velocidade do processador real, o que por sua vez significa que, durante os $(x - 5\,625)$ ms, somente 45% das instruções poderão ser executadas. O tempo necessário para executar 100% das instruções sobre a máquina virtual será de $\left(\frac{x-5\,625}{0,45}\right)$ ms. Agora, como o processo faz $4\,500 - 1\,500 = 3\,000$ operações de E/S na máquina virtual, e como o novo tempo de cada operação de E/S é de $\frac{1,25}{0,5} = 2,5$ ms (já que, similarmente à redução da velocidade do processador, a redução da velocidade de cada operação de E/S em 50% significa que no mesmo tempo podemos, na máquina virtual, executar somente 50% das operações de E/S originais), então 7 500 ms dos 20 000 ms de execução do processo na máquina virtual serão gastos com E/S. Logo, o tempo de execução do processo no processador virtual será de $20\,000 - 7\,500 = 12\,500$ ms. Usando $\frac{x-5\,625}{0,45} = 12\,500$, temos $x = 11\,250$. Logo, o tempo de execução do processo no hardware será de 11 250 ms.

4. (1,0) Na figura a seguir mostramos uma versão estendida do diagrama de transição de estados de um processo, com dois novos estados, **Novo** e **Terminado**. Um processo é colocado no estado **Novo** quando ele é criado e passa ao estado **Terminado** quando termina a sua execução. O diagrama está correto? Se você acha que sim, basta responder isso mas, em caso contrário, indique os erros no diagrama. Justifique a sua resposta.

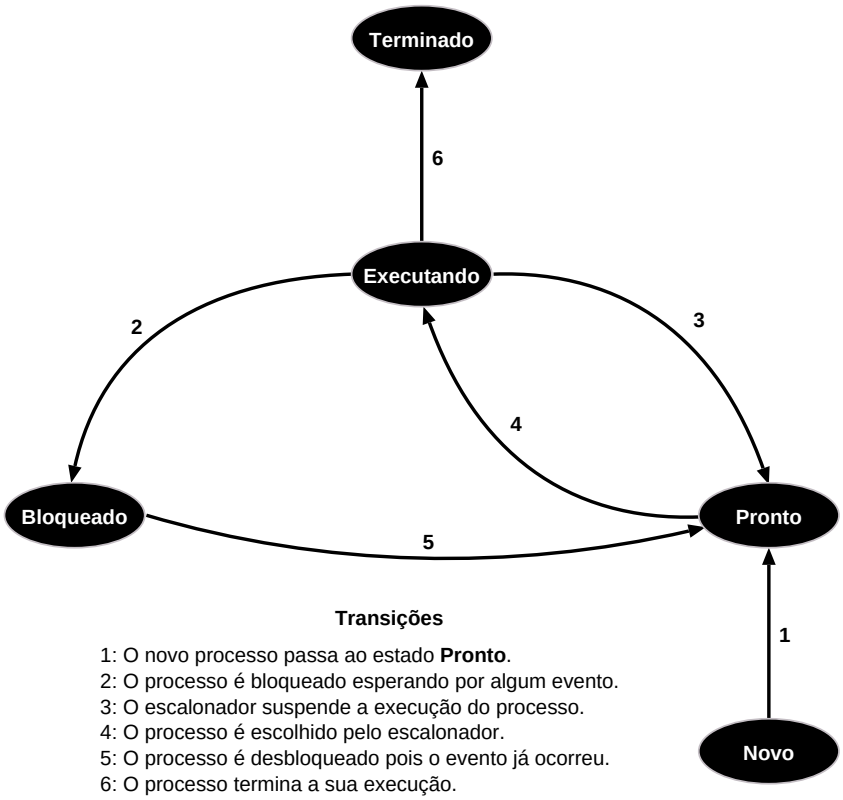


Transições

1: O novo processo é executado.
2: O processo é bloqueado, esperando por algum evento.
3: O escalonador suspende a execução do processo.
4: O processo é desbloqueado pois o evento já ocorreu.
5: O processo bloqueado termina a sua execução.

Resp.: O diagrama não está correto. Um processo que acabou de ser criado e que foi colocado no estado **Novo** não pode passar ao estado **Executando** enquanto está esperando para ser executado, porque algum outro processo no estado **Pronto** pode ser mais prioritário do que o processo criado. Logo estão erradas a transição 1 do estado **Novo** para o estado **Executando** e a descrição dessa transição. Um outro erro está no fato de que, para terminar, um processo precisa estar executando em uma unidade de processamento. Logo, não somente a transição 5 do estado **Bloqueado** para o estado **Terminado** está errada, mas também está errada a descrição dessa transição. Finalmente, foi omitida a transição do estado **Pronto** para o estado **Executando**, que ocorre quando um processo é escolhido pelo escalonador para ser

executado em uma unidade de processamento. A seguir está o diagrama correto.



5. (2,0) Suponha que uma pilha, com tamanho ilimitado, e um vetor, com n entradas, numeradas de 1 até n, sejam compartilhados por três processos, A, B, C, sendo que inicialmente a pilha está vazia e que todas as entradas do vetor possuem o valor 0. O processo A continuamente coloca na pilha a valores escolhidos de modo aleatório. Já o processo B continuamente espera a pilha possiur pelo menos um valor para, depois disso, remover o valor no topo da pilha e atualizar a entrada x do vetor, escolhida de modo aleatório, com a soma do valor removido da pilha e o valor armazenado nessa entrada. Finalmente, o processo C continuamente varre o vetor para imprimir todas as entradas com valores diferentes de 0. Como um semáforo de contagem e $n + 1$ semáforos binários podem ser usados para garantir que os processos executem sem

condições de corrida ou impasses, de tal modo que as entradas do vetor possam ser independentemente acessadas? Justifique a sua resposta.

Resp.: Vamos usar dois semáforos para a pilha, um binário, chamado de *acessopilha*, para garantir o acesso exclusivo à pilha, e um de contagem, chamado *contapilha*, para contar o número de valores na pilha e bloquear o processo B caso a pilha esteja vazia. Além disso, para cada entrada i do vetor, $1 \leq i \leq n$, vamos usar um semáforo binário $acessovetor_i$ para garantir o acesso exclusivo à entrada i do vetor. Como inicialmente a pilha está vazia e não está sendo usada, então os semáforos *acessopilha* e *contapilha* serão inicializados com, respectivamente, 1 e 0. Além disso, como nenhuma entrada do vetor está sendo inicialmente usada, então todos os semáforos $acessovetor_i$ serão inicializados com 1. A seguir mostramos os pseudocódigos para os processos A, B e C, usando esses $n+2$ semáforos. Nestes pseudocódigos, supomos que a função $InserePilha(v)$ insere o valor v no topo da pilha, que a função $RemovePilha()$ remove e retorna o valor no topo da pilha, e que $vetor[i]$ acessa a entrada i do vetor.

```
void ProcessoA(void)
{
  while(1)
  {
    // Obtém o acesso exclusivo à pilha.
    P(acessopilha);
    for(j = 0; j < a; j++)
    {
      // Código para retornar um novo número aleatório em v.
      // Insere v no topo da pilha.
      InserePilha(v);
      // Usa a operação V sobre contapilha para registrar que v foi inserido na
      // pilha.
      V(contapilha);
    }
    // Libera o acesso exclusivo à pilha.
    V(acessopilha);
  }
}
```

```
void ProcessoB(void)
{
  while(1)
  {
    // Garante que existe pelo menos um valor na pilha.
    P(contapilha);
    // Obtém o acesso exclusivo à pilha.
    P(acessopilha);
    // Remove o valor do topo da pilha e o armazena em v.
    v = RemovePilha();
    // Libera o acesso exclusivo à pilha.
    V(acessopilha);
    // Código para escolher, de modo aleatório, uma entrada x do vetor.
    // Obtém o acesso exclusivo à entrada x do vetor.
    P(acessovetor_x);
    // Atualiza a entrada x do vetor com a soma de v ao valor desta entrada.
    vetor[x] = vetor[x] + v;
    // Libera o acesso exclusivo à entrada x do vetor.
    V(acessovetor_x);
  }
}

void ProcessoC(void)
{
  while(1)
  {
    for(i = 1; i ≤ n; i++)
    {
      // Obtém o acesso exclusivo à entrada i do vetor.
      P(acessovetor_i);
      // Código para imprimir o valor de vetor[i] se ele for diferente de 0.
      // Libera o acesso exclusivo à entrada i do vetor.
      V(acessovetor_i);
    }
  }
}
```

6. (2,0) Quatro processos, A, B, C e D, foram inicializados e têm tempos de execução no processador de, respectivamente, 8, 13, 7 e 16 unidades de tempo. Para cada um dos seguintes algoritmos de escalonamento, determine a média dos tempos decorridos do início ao término dos processos. Ignore o acréscimo (*overhead*) da comutação de processos e suponha que nenhum processo faça operações de E/S. Justifique a sua resposta.

(a) (0,8) *Round robin*, com um **quantum** de 2 unidades de tempo de duração e com os processos inicialmente executando na ordem A, D, C e B.

Resp.: Pelo enunciado, vemos que a ordem de execução dos processos é como 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, indicando o tempo de início de cada **quantum** e o processo correspondente. Devido a os tempos dos processos B e C não serem múltiplos do tamanho do **quantum**, de 2 unidades de tempo, eles usam somente 1 unidade de tempo dos seus últimos **quanta**. Pela tabela, vemos que os tempos decorridos entre o início e o término dos processos A, B, C e D, são de, respectivamente, 26, 36, 25 e 42 unidades de tempo, o que nos dá um tempo médio de 32,25 unidades de tempo.

0	2	4	6	8	10	12	14	16	18	20	22
A	D	C	B	A	D	C	B	A	D	C	B

24	26	28	29	31	33	35	37	39	41	42	44
A	D	C	B	D	B	D	B	D	B	D	-

(b) (0,8) Escalonamento por prioridades, supondo que a prioridade do processo em execução seja reduzida por 4 unidades a cada 3 unidades de tempo, que um processo em execução somente seja suspenso quando um outro processo passa a ter a maior prioridade, e que as prioridades iniciais dos processos A, B, C e D sejam de, respectivamente, 18, 23, 17 e 28.

Resp.: As tabelas a seguir são similares à tabela do item anterior e possuem somente mais uma linha, a terceira, que mostra a prioridade de cada processo antes de ele executar no intervalo de tempo correspondente à mesma coluna. Novamente, devido a os tempos de todos os processos A, B, C e D não serem múltiplos das 3 unidades de tempo usadas ao reduzir as prioridades, então

A usará somente 2 unidades de tempo logo antes de terminar, e B, C e D usarão somente 1 unidade de tempo logo antes de terminarem. Agora, pelas tabelas, vemos que os tempos decorridos entre o início e o término dos processos A, B, C e D, são de, respectivamente, 26, 38, 24 e 43 unidades de tempo, o que nos dá um tempo médio de 32,75 unidades de tempo.

0	3	6	9	12	15	18	21	24
D	D	B	D	B	A	C	D	B
28	24	23	20	19	18	17	16	15

27	30	33	36	39	41	42	43	44
A	C	D	B	A	C	D	B	-
14	13	12	11	10	9	8	7	-

(c) (0,4) Trabalho mais curto primeiro.

Resp.: Como vimos na aula 6, no algoritmo do trabalho mais curto primeiro os processos são executados segundo a ordem crescente de seus tempos de execução. Além disso, quando um processo começa a executar, ele executa exclusivamente no processador até terminar. Logo, os tempos decorridos entre o início e o término dos processos A, B, C e D são iguais aos seus tempos de execução, ou seja, são iguais a, respectivamente, 8, 13, 7 e 16 unidades de tempo. Logo, o tempo médio é de 11 unidades de tempo.