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 AP1 - Primeiro Semestre de 2019

Nome -
Assinatura -

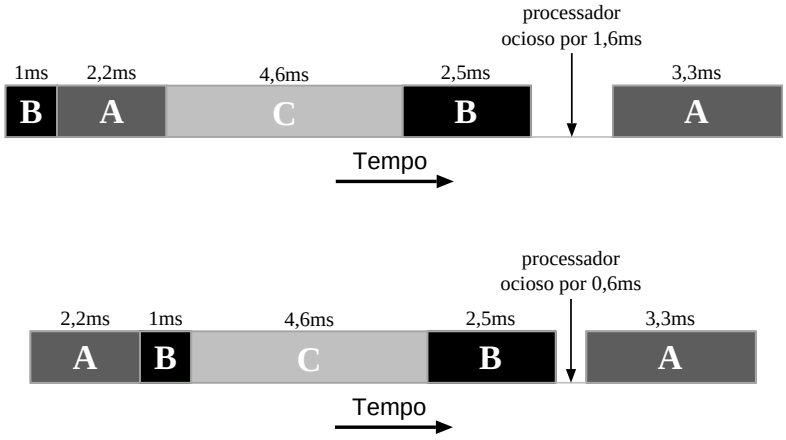
Observações:

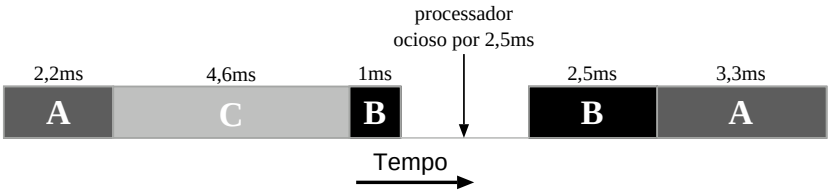
1. Prova sem consulta e sem uso de máquina de calcular.
2. Use caneta para preencher o seu nome e assinar nas folhas de questões e nas folhas de respostas.
3. Você pode usar lápis para responder as questões.
4. Ao final da prova devolva as folhas de questões e as de respostas.
5. Todas as respostas devem ser transcritas nas folhas de respostas. As respostas nas folhas de questões não serão corrigidas.

---

1. (1,5) Considere três programas, A, B e C. Suponha que A precise executar no processador por 5,5ms, fazendo uma operação de E/S, com duração de 8,7ms, após executar no processador por 2,2ms. B precisa executar no processador por 3,5ms e faz uma operação de E/S, com duração de 2,5ms, após executar no processador por 1,0ms. Finalmente, C não faz operações de E/S, precisa executar no processador por 4,6ms, e somente pode executar após A ter executado pela primeira vez. Se a multiprogramação for usada somente para evitar a ociosidade do processador quando operações de E/S são feitas pelos programas, que ordem de execução dos programas gerará a menor ociosidade? Justifique a sua resposta.

Resp.: Pelo enunciado, como o programa C sempre deve iniciar a sua execução após o programa A ter sido suspenso devido a ter feito a sua operação de E/S, então podemos concluir que existem os três modos de intercalarmos as execuções dos programas A, B e C dados nas figuras a seguir. Os diferentes modos dependem de quando B inicia sua execução. A primeira figura mostra B executando antes de A e de C, a segunda figura mostra B executando entre A e C e, finalmente, a última figura mostra B executando depois de A e de C. Pelas figuras, vemos que o menor tempo de ociosidade, de 0,6ms, será obtido quando a ordem de execução dos programas for ABCBA.





2. (2,5) Diga se as seguintes afirmativas são falsas ou verdadeiras. Para responder, escreva apenas F ou V para cada item em seu caderno de respostas.

   (a) (0,5) Uma chamada ao sistema operacional deve ser executada diretamente por um processo porque a sua execução nunca depende do hardware e, portanto, não é difícil implementá-la no código do processo.

   Resp.: F (Falsa), porque uma chamada ao sistema operacional não é, em geral, executada diretamente pelo processo exatamente por poder depender do hardware e, devido a isso, poder ser de difícil implementação no código do processo.

   (b) (0,5) A grande diferença entre o modelo cliente-servidor e os outros modelos de estruturação de um sistema operacional é que o núcleo do sistema é o responsável pelo gerenciamento das mensagens trocadas entre os processos clientes e servidores executando no modo usuário, e também pelo acesso aos dispositivos físicos do hardware.

   Resp.: V (Verdadeira).

   (c) (0,5) O contexto de um processo é onde são armazenadas todas as informações necessárias para retomar sua execução a partir do ponto em que ela parou quando o processo foi suspenso pelo escalonador ou foi bloqueado devido a precisar esperar pela ocorrência de um evento externo.

   Resp.: V (Verdadeira).

(d) (0,5) A espera ocupada ocorre quando um processo fica em um *loop* esperando poder acessar a sua seção crítica, ou seja, até ser garantido que o acesso concorrente não gerará condições de corrida.

Resp.: V (Verdadeira).

(e) (0,5) O algoritmo por *round robin* não considera todos os processos como igualmente importantes, porque cada processo pode executar consecutivamente por dois ou mais **quanta**, mesmo quando nem todos os outros processos não bloqueados tenham executado por pelo menos um **quantum** desde a última vez em que foram escalonados.

Resp.: F (Falsa), porque o algoritmo na verdade considera todos os processos como igualmente importantes, já que cada processo, após executar por um **quantum**, somente pode executar novamente por mais um **quantum** após todos os outros processos não bloqueados terem executado também por um **quantum**.

3. (1,5) Diga a quais conceitos vistos em aula se referem as seguintes definições:

(a) (0,5) Nome do método para gerenciar a multiprogramação que, ao invés de criar máquinas virtuais que são cópias exatas do hardware, divide os recursos do hardware entre todas as máquinas virtuais, evitando assim que uma máquina virtual possa indevidamente acessar os recursos alocados a outras máquinas virtuais.

Resp.: Exonúcleo.

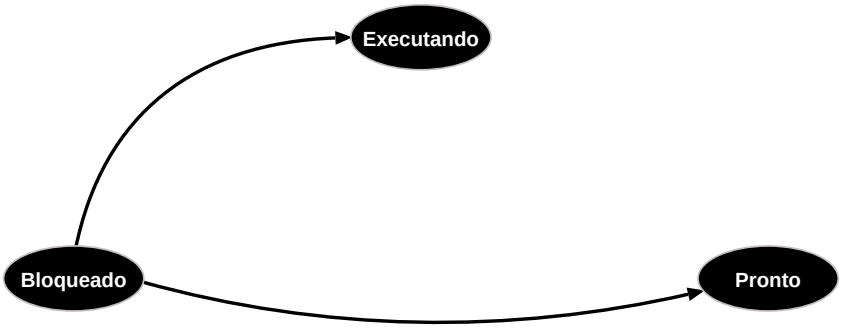
(b) (0,5) Nome do processo que, em geral, precisa interagir com o usuário para poder executar corretamente a sua tarefa.

Resp.: Processo em primeiro plano ou *foreground*.

(c) (0,5) Nome dado ao processo que passa a maior parte do seu tempo de execução computando, ou seja, executando em um processador.

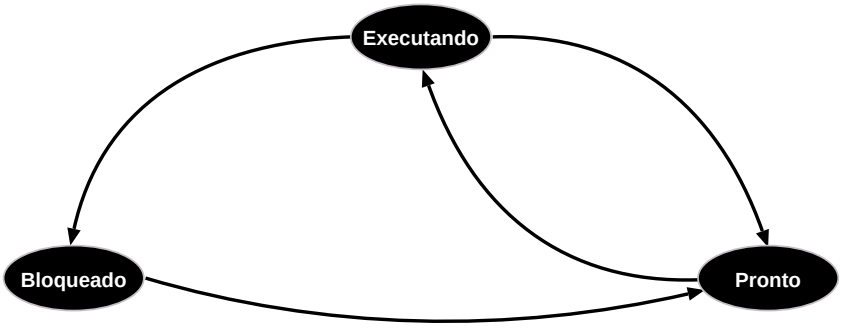
Resp.: *CPU-bound*.

4. (1,5) Um aluno de sistemas operacionais disse que o diagrama a seguir representa todas as transições entre os estados de um processo. O diagrama do aluno está correto? Se você acha que sim, basta dizer isso mas, se você acha que não, indique que transições estão erradas e quais estão ausentes.



Resp.: O diagrama do aluno não está correto porque uma transição está invertida e duas transições estão ausentes. A transição do estado **Bloqueado** para o estado **Executando** está invertida, porque um processo, ao ser desbloqueado, não deve ser imediatamente executado porque não é necessariamente o mais prioritário. A transição correta, do estado **Executando** para o estado **Bloqueado**, indica que o processo em execução foi bloqueado devido a precisar esperar pelo término de um evento externo. As duas transições ausentes relacionam os estados **Executando** e **Pronto**. A primeira delas, do estado **Executando** para o estado **Pronto**, ocorre quando o escalonador decide que o processo em execução precisa ceder o processador a um outro processo escolhido pelo escalonador. Finalmente, a segunda transição, do estado **Pronto** para o estado **Executando**, relacionada à anterior,

ocorre quando o processo escolhido pelo escalonador é efetivamente colocado para executar no processador. A seguir está o diagrama correto.



5. (1,5) Suponha que dois processos, A e B, compartilhem um *buffer* que pode armazenar até n elementos e está inicialmente cheio. O processo A continuamente adiciona três novos elementos ao *buffer*, desde que haja espaço para todos eles. Já o processo B continuamente remove dois elementos do *buffer*, desde que existam pelo menos dois elementos, e depois disso os imprime. Dadas as funções *RemoveBuffer*(), para remover e retornar um elemento aleatório do *buffer*, e *AdicionaBuffer*(e), para adicionar o elemento e ao *buffer*, como dois semáforos de contagem e um semáforo binário podem ser usados para os processos executarem corretamente, sem a ocorrência de condições de corrida ou impasses? 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 *AcessoBuffer*, é usado para garantir o acesso exclusivo ao *buffer*. Em relação aos dois semáforos de contagem, o primeiro deles, chamado *LivresBuffer*, conta a quantidade de elementos que ainda podem ser inseridos no *buffer* e é usado para bloquear A quando o *buffer* não possui espaço para três elementos adicionais. Finalmente, o segundo semáforo, chamado de *UsadosBuffer*, conta a quantidade de elementos no *buffer* e é usado para bloquear B quando o *buffer* não possui pelo menos dois elementos. Como inicialmente o *buffer* tem n elementos (está cheio) e não está sendo acessado, então os semáforos *AcessoBuffer*, *LivresBuffer* e *UsadosBuffer*

são inicializados, respectivamente, com 1, 0 e n. A seguir mostramos os códigos para os processos A e B.

```
void ProcessoA(void)
{
  while(1)
  {
    // Código para gerar três elementos e salvá-los em e1, e2 e e3.
    // Usa a operação P sobre LivresBuffer para garantir que e1, e2 e e3 possam
    // ser inseridos no buffer.
    P(LivresBuffer);
    P(LivresBuffer);
    P(LivresBuffer);
    // Garante o acesso exclusivo ao buffer.
    P(AcessoBuffer);
    // Insere e1, e2 e e3 no buffer.
    AdicionaBuffer(e1);
    AdicionaBuffer(e2);
    AdicionaBuffer(e3);
    // Libera o acesso exclusivo ao buffer.
    V(AcessoBuffer);
    // Usa a operação V sobre UsadosBuffer para indicar que e1, e2 e e3 foram
    // inseridos no buffer.
    V(UsadosBuffer);
    V(UsadosBuffer);
    V(UsadosBuffer);
  }
}
```

```
void ProcessoB(void)
{
  while(1)
  {
    // Usa a operação P sobre UsadosBuffer para garantir que existam pelo
    // menos dois elementos no buffer.
    P(UsadosBuffer);
    P(UsadosBuffer);
    // Garante o acesso exclusivo ao buffer.
    P(AcessoBuffer);
    // Remove dois elementos do buffer e os armazena em e1 e e2.
    e1 = RemoveBuffer();
    e2 = RemoveBuffer();
    // Libera o acesso exclusivo ao buffer.
    V(AcessoBuffer);
    // Usa a operação V sobre LivresBuffer para indicar que dois elementos
    // foram removidos do buffer.
    V(LivresBuffer);
    V(LivresBuffer);
    // Código para imprimir os elementos e1 e e2.
  }
}
```

6. (1,5) Suponha que o algoritmo do trabalho mais curto primeiro tenha sido usado pelo sistema operacional, e que os tempos de término de três processos, A, B e C, que não fazem operações de E/S, tenham sido de, respectivamente, 15ms, 23ms e 7ms. Suponha agora que o sistema operacional passe a usar o algoritmo por prioridades, no qual o processo em execução executa até existir um outro processo com prioridade maior, e que cada prioridade seja decrementada por 2 unidades a cada 2ms. Quais serão os novos tempos de término de A, B e C se as suas prioridades iniciais forem de, respectivamente, 25, 20 e 28? Justifique a sua resposta.

   Resp.: Pelo enunciado, vemos que os tempos totais de execução no processador para os processos A, B e C são de, respectivamente, 8ms, 8ms e 7ms. Usando estes tempos e as prioridades dadas no enunciado para A, B e C, temos a execução dada na tabela a seguir. Nessa tabela mostramos, na primeira linha, o tempo antes de o processo da coluna ter sido executado. 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 a prioridade do processo antes de ele executar no processador no tempo dado na mesma coluna. Pela tabela, vemos que os novos tempos de término de A, B e C serão de, respectivamente, 17ms, 23ms e 11ms.

0	2	4	6	8	10	11	13	15	17	19	21	23
C	C	A	C	A	C	A	B	A	B	B	B	-
28	26	25	24	23	22	21	20	19	18	16	14	-