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
AD1 - Segundo Semestre de 2008

Atenção: Tem havido muita discussão sobre a importância de que cada aluno redija suas próprias respostas às questões da AD1. Os professores da disciplina, após refletirem sobre o assunto, decidiram o seguinte: 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.

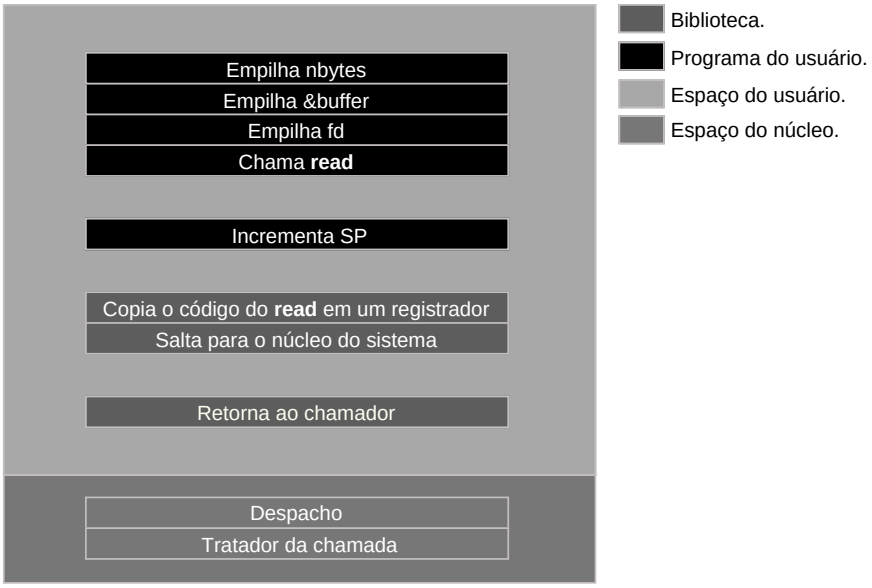
Nome -
Assinatura -

---

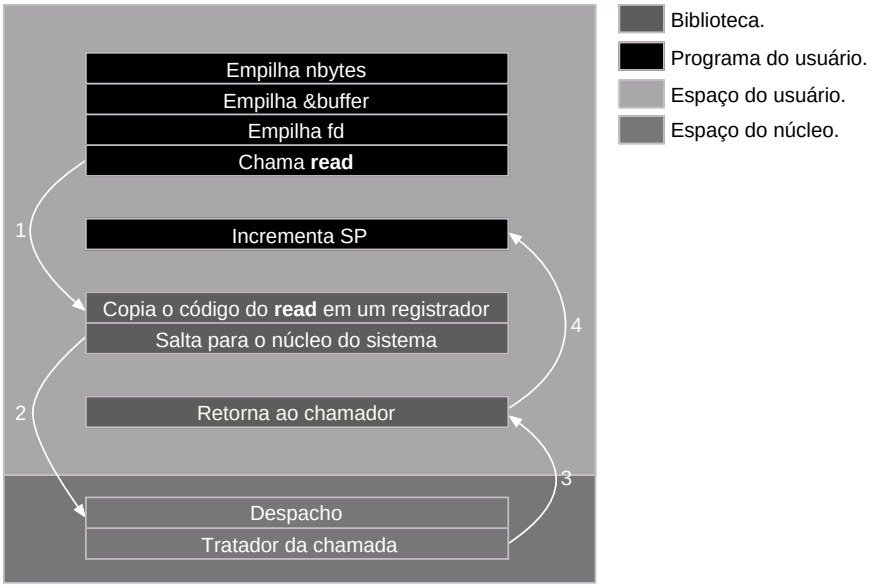
1. (1.0) Suponha que um programa A leve 18s para executar no processador e que, para executar a sua tarefa, ele precise fazer E/S por 4s. Se este programa fosse executado em um sistema anterior ao da terceira geração, qual seria a fração de tempo do processador desperdiçada com operações de E/S? Este desperdício ainda ocorreria nos sistemas posteriores ao da segunda geração?

Resp.: -Pela questão, vemos que o programa executou em um sistema da terceira ou da quarta geração. Logo, o sistema possui o conceito de multiprogramação e, com isso, o tempo de 4s gasto com E/S não está incluído nos 18s do tempo de execução do programa. Note que o programa somente pode ser executado em um sistema da segunda geração, pois na primeira geração o programador manipulava diretamente o hardware do computador, e não existia um processador que executava os programas. Como na segunda geração não existe o conceito de multiprogramação, agora o tempo de 4s de E/S fará parte do tempo de execução do programa no processador. Isso ocorrerá porque o processador, que está executando este programa, ficará ocioso esperando pelo término da operação de E/S, para depois continuar a executar o programa. Logo, o tempo de execução do programa será agora de 22s. Como o processador ficará ociso por 4s destes 22s, então a fração de tempo desperdiçada do processador será de $4/22 = 2/11 \approx 0.18$, ou seja, aproximadamente 18% do tempo de execução do programa.
-Não, pois devido ao conceito de multiprogramação que surgiu na terceira geração, o processador deixa de ficar ocioso quando o programa em execução faz operações de E/S.

2. (1.0) A figura dada a seguir mostra o que ocorre no programa, na biblioteca e no núcleo do sistema ao fazermos uma chamada ao sistema **read**. Indique a precedência que deve existir entre os blocos da figura a partir do momento em que o programa faz a chamada.



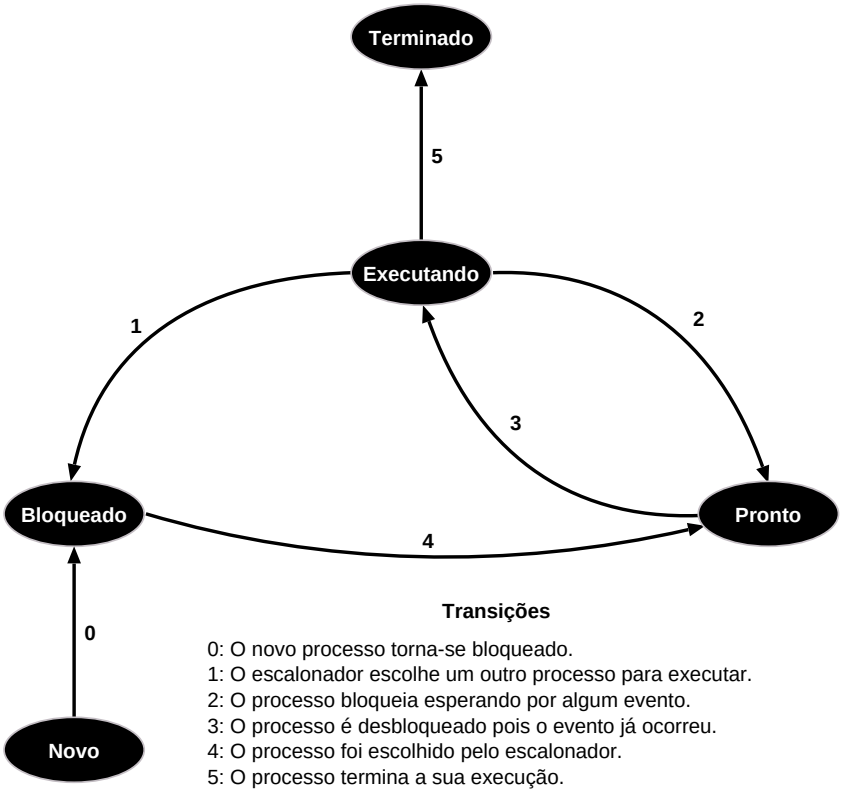
Resp.: Na figura dada a seguir indicamos a precedência entre os blocos. Na transição 1, depois de a chamada **read** ser feita, o controle é passado à biblioteca que implementa uma interface para esta chamada. A transição 2 ocorre após a instrução TRAP ser usada para alternar do modo usuário para o modo supervisor, o que causa a chamada da função de despacho do núcleo do sistema operacional. A transição 3 ocorre após o núcleo terminar de executar a chamada ao sistema operacional, isto é, após o processador ser alternado do modo supervisor para o modo usuário, e o controle ser devolvido à instrução seguinte à instrução TRAP, no código da biblioteca. Finalmente, na transição 4, a biblioteca devolve o controle ao programa do usuário, na instrução incrementa SP seguinte à chamada **read**.



3. (2.0) Suponha que um sistema operacional esteja executando sobre uma máquina virtual. Suponha ainda que o processador virtual possua um poder de processamento 25% menor do que o do processador real, e que o tempo de execução de uma chamada ao sistema aumente de 10ms para 12ms. Se um determinado programa, tendo feito 25 chamadas ao sistema, executou em 25s, qual seria o tempo de execução diretamente sobre o hardware do computador?

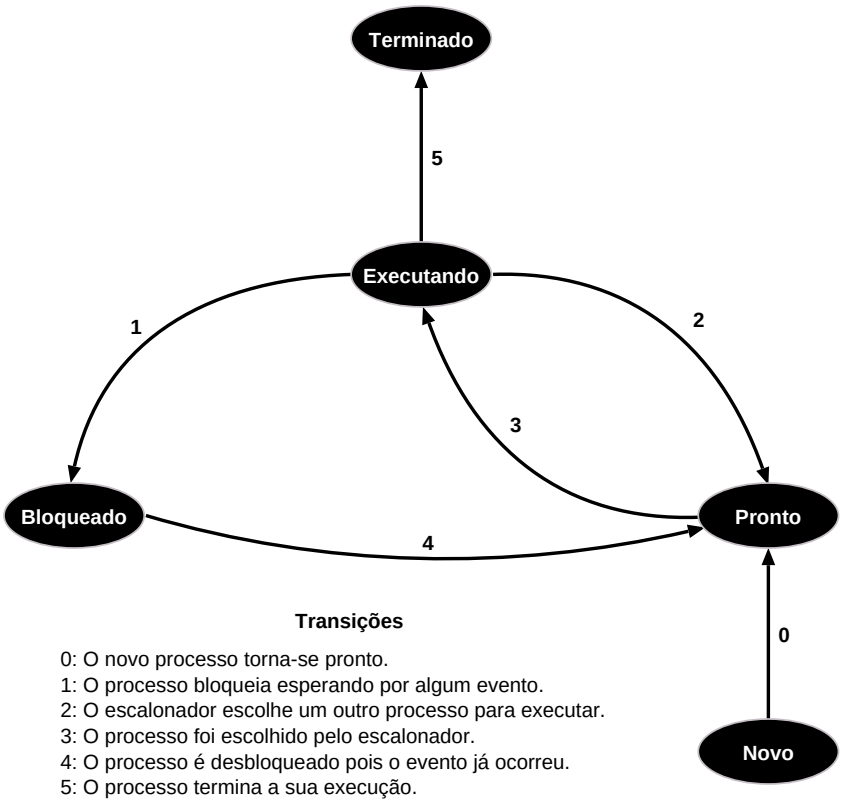
   Resp.: Do tempo de 25s, ou seja, 25000ms, de execução do programa, $25 \times 12 = 300$ms foram gastos com as 25 chamadas ao sistema operacional feitas pelo programa. Logo, o programa executou no processador durante $25000 - 300 = 24700$ms. Agora, como o uso da máquina virtual reduziu o poder de processamento em 25% então, quando o programa executar sobre o processador real, o seu tempo de execução será reduzido em 25%. Com isso, o programa executará agora no processador por $24700 - 0,25 \times 24700 = 24700 - 6175 = 18525$ms. Além disso, como o tempo de execução de uma chamada ao sistema agora será de 10ms, então o programa gastará $25 \times 10 = 250$ms para executar as 25 chamadas ao sistema. Logo, o tempo de execução do programa, se ele fosse executado diretamente no processador, seria de $18525 + 250 = 18775$ms, ou seja, $18,775$s.

4. (1.0) Na figura dada a seguir mostramos uma versão estendida do diagrama de transição dos estados de um processo, com dois novos estados: o estado **Novo**, em que o processo é colocado quando é criado, e o estado **Terminado**, em que o processo é colocado quando termina a sua execução. Este diagrama está correto? Justifique a sua resposta.

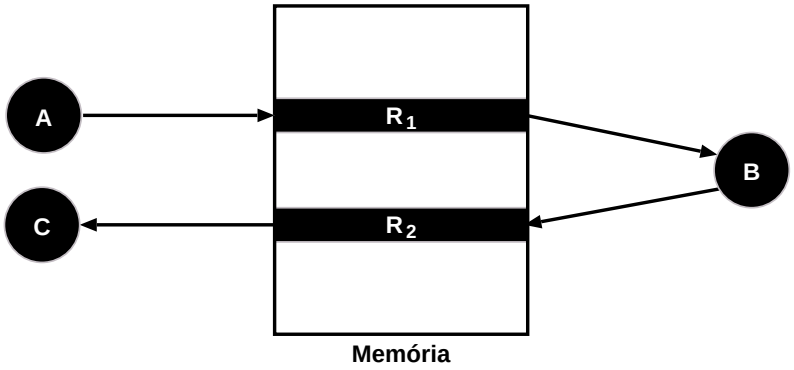


Resp.: Não, existem diversos erros no diagrama dado na figura. Quando um processo é criado, isto é, está no estado **Novo**, ele deve passar ao estado pronto para esperar pela sua vez de executar no processador. Não tem sentido o processo passar diretamente ao estado bloqueado depois de ser criado, pois um processo somente passa ao estado bloqueado quando está no estado executando (ou seja, está executando no processador) e precisa ser suspenso para esperar pela ocorrência de algum evento externo. Logo, a descrição da transição 0 está incorreta, e ela deve sair do estado **Novo** para o estado **Pronto**. Com base no diagrama que vimos na Aula 4, vemos que as descrições das transições

1 e 2 foram trocadas, isto é, a descrição dada na transição 1 é a da 2, e a dada na 2 é a da 1. O mesmo ocorre com as descrições das transições 3 e 4. Na figura dada a seguir temos o diagrama correto, com todas as correções descritas anteriormente.



5. (2.0) Suponha que temos duas regiões de memória, R_1 e R_2, compartilhada por três processos A, B e C, como mostrado na figura dada a seguir. O processo A deposita um valor em R_1, o qual sempre deverá ser consumido pelo processo B. O processo B, por sua vez, lê o valor de R_1, processa-o, e depois coloca um novo valor em R_2, o qual sempre deverá ser consumido pelo processo C. Finalmente, o processo C somente lê o valor de R_2. Como os semáforos podem ser usados para garantir o correto funcionamento dos processos A, B e C, supondo que eles repetem o procedimento acima indefinidamente?



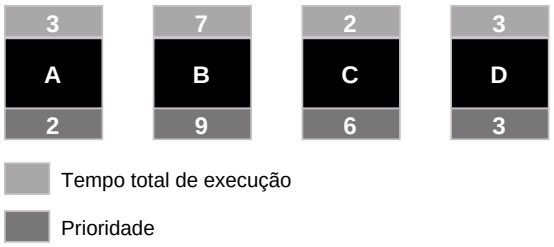
Resp.: Precisamos de dois semáforos binários, $dado\_ja\_lido_1$ e $dado\_disponivel_1$, para a região R_1, e de dois semáforos binários, $dado\_ja\_lido_2$ e $dado\_disponivel_2$ para a região R_2. O semáforo $dado\_disponivel_1$ irá bloquear o processo B se o processo A ainda não colocou um novo valor em R_1 após B ler o valor de R_1. Já o semáforo $dado\_ja\_lido_1$ irá bloquear o processo A caso o processo B ainda não tenha lido o último valor colocado por A em R_1. Similarmente, o semáforo $dado\_disponivel_2$ irá bloquear o processo C se o processo B ainda não colocou um novo valor em R_2 após C ler o valor de R_2, e o semáforo $dado\_ja\_lido_2$ irá bloquear o processo B se o processo C ainda não leu o último valor colocado por B em R_2. Como inicialmente os processos não estão em execução, então não existirão valores válidos em R_1 e R_2 e, com isso, os valores dos semáforos $dado\_disponivel_1$ e $dado\_disponivel_2$ serão ambos 0, e os valores dos semáforos $dado\_ja\_lido_1$ e $dado\_ja\_lido_2$ serão ambos 1. A seguir damos um esboço dos procedimentos (em C) ProcessoA, ProcessoB e ProcessoC que devem ser executados, respectivamente, pelos processos A, B e C. A função GerarDado gera um novo valor para ser colocado em R_1, a função ProcessarDado processa o valor lido de R_1 e gera um novo valor para ser colocado em R_2 e, finalmente, a função UsarDado usa o valor lido de R_2.

```
void ProcessoA(void)
{
  int Valor;
  while (1) {
    Valor = GerarDado();
    P(dado_ja_lido_1);
    R_1 = Valor;
    V(dado_disponivel_1);
  }
}
```

```
void ProcessoB(void)
{
  int Valor, NovoValor;
  while (1) {
    P(dado_disponivel_1);
    Valor = R_1;
    V(dado_ja_lido_1);
    NovoValor = ProcessarDado(Valor);
    P(dado_ja_lido_2);
    R_2 = NovoValor;
    V(dado_disponivel_2);
  }
}
```

```
void ProcessoC(void)
{
  int Valor;
  while (1) {
    P(dado_disponivel_2);
    Valor = R_2;
    V(dado_ja_lido_2);
    UsarDado(Valor);
  }
}
```

6. (3.0) Suponha que os processos da figura dada a seguir acabaram de entrar no estado pronto, e que são os únicos processos em execução. Quais seriam os tempos de término destes processos, se o sistema operacional usasse, ao escalonar os processos:



Resp.: Existem várias respostas para cada um dos itens desta questão. Em cada um dos itens, vocês somente precisam fornecer uma destas respostas, que será considerada como correta se, dada a sua suposição para o item, a sua resposta estiver coerente com o algoritmo especificado no item e a sua suposição. Supondo que os tempos dos processos são dados na questão em uma certa unidade de tempo, vocês precisarão supor, no item (a) da questão, quantas unidades de tempo equivalem a um quantum. Já no item (b), vocês precisarão supor quantas unidades de tempo equivalem à redução, em 1 unidade, da prioridade do processo em execução no processador. Além disso, no item (a), vocês precisam escolher uma ordem em que os processos começam a sua execução.

(a) (1.0) o algoritmo de *round robin*.

Resp.: Na nossa resposta, vamos supor que cada quantum equivale a 1 unidade de tempo, e que as quatro escolhas iniciais do escalonador foram, em ordem, A, B, C e D. Lembrando que no algoritmo de *round robin* cada processo executa alternadamente no processador até terminar, obteremos a seqüência de execução dada na figura a seguir. A primeira linha 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 desta coluna na primeira linha. Como podemos ver pela figura, os tempos de término dos processos A, B, C e D serão de, respectivamente, 9, 15, 7 e 11 unidades de tempo.

-	A	B	C	D	A	B	C	D	A	B	D	B	B	B	B
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

(b) (1.0) o algoritmo de prioridades.

Resp.: Nesta resposta, vamos supor que cada redução de 1 unidade da prioridade do processo em execução equivale a 1 unidade de tempo. Na Aula 6 vimos que, no algoritmo de prioridades, um processo executa até que a sua prioridade deixe de ser a maior de todas. Na figura 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. Pela figura, vemos que os tempos de término dos processos A, B, C e D são de, respectivamente, 15, 9, 6 e 14 unidades de tempo.

-	9	8	7	6	6	5	5	4	3	3	2	2	1	1	0
-	B	B	B	B	C	C	B	B	B	D	D	A	A	D	A
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

(c) (1.0) 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, uma das possíveis ordens de execução seria C, A, D e B (a outra seria C, D, A e B). O processo C então executaria do tempo 0 até o 2, o processo A do tempo 2 até o 5, o processo D do tempo 5 até o 8 e, finalmente, o processo B do tempo 8 até o 15. Logo, os tempos de término dos processos A, B, C e D seriam de, respectivamente, 5, 15, 2 e 8 unidades de tempo.