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 2011

Atenção: ADs enviadas pelo correio devem ser postadas cinco dias antes da data final de entrega estabelecida no calendário de entrega de ADs.

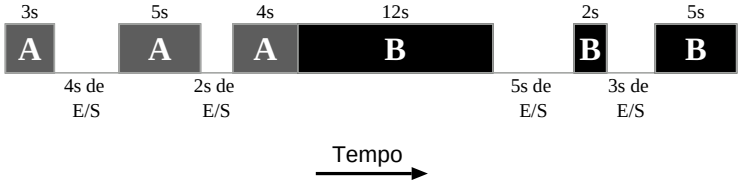
Atenção: Tem havido muita discussão sobre a importância de que cada aluno redija suas próprias respostas às questões da AD1. Os professores da disciplina, após refletirem sobre o assunto, decidiram o seguinte: Cada aluno é responsável por redigir suas próprias respostas. Provas iguais umas às outras 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.

Observação: A questão 6 é opcional. Se você tentar resolvê-la, você poderá ganhar até 2,0 pontos adicionais na nota da prova, mas a sua nota ainda será limitada a 10,0 pontos.

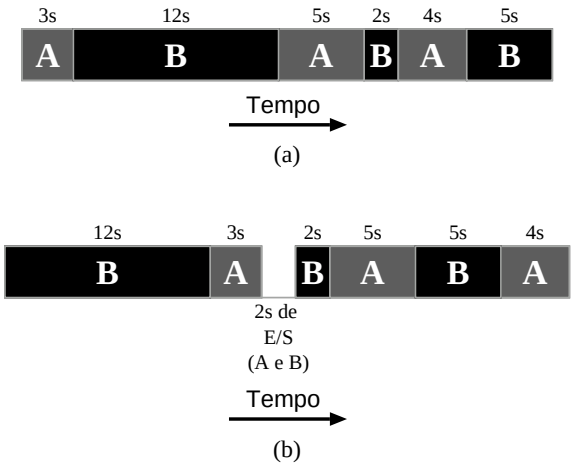
Nome -
Assinatura -

---

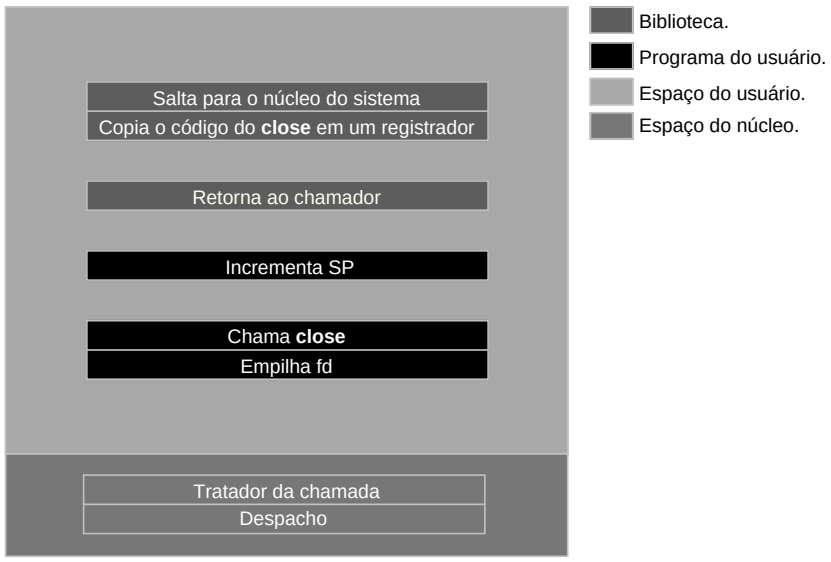
1. (2,0) Considere a ordem de execução dos programas A e B dada na figura a seguir, em um sistema operacional que não use a multiprogramação. Considere que agora uma nova versão do sistema operacional tenha passado a usar a multiprogramação somente para evitar a ociosidade do processador durante a execução de operações de E/S. Qual será a pior ordem de execução nessa nova versão, ou seja, aquela que gerará a maior ociosidade do processador? Justifique a sua resposta.



Resp.: Como não foi especificado qual programa executa primeiro (A ou B), então precisamos avaliar as duas possibilidades. Na parte (a) da figura a seguir é mostrada a ordem de execução quando A é o primeiro a executar. Como podemos ver pela figura, sempre que um dos programas é suspenso por ter iniciado uma operação de E/S, o outro programa pode ser executado. Esse programa executa até terminar ou fazer E/S, e o seu tempo de execução é maior ou igual ao tempo da operação de E/S que suspendeu o outro programa. Logo, nesse caso o processador não fica ocioso. Já na parte (b) da mesma figura, mostramos o que ocorre quando B executa antes de A. Nesse caso, somente a primeira operação de E/S feita por B, com duração de 5s, gera uma ociosidade do processador de 2s, pois A somente executa por 3s antes de ser suspenso por fazer uma operação de E/S de 4s. Todas as outras operações de E/S não geram, assim como no caso anterior, uma ociosidade do processador. Observe que a segunda execução de B é suficiente para que a primeira operação de E/S feita por A não gere ociosidade do processador, porque durante os 2s nos quais o processador ficou ocioso, foi executado o final da primeira operação de E/S de B e o início da primeira operação de E/S de A. Agora, como não existe nenhuma outra possibilidade além das dadas nas partes (a) e (b), então a pior ordem de execução é a dada em (b), durante a qual o processador fica ocioso por 2s.

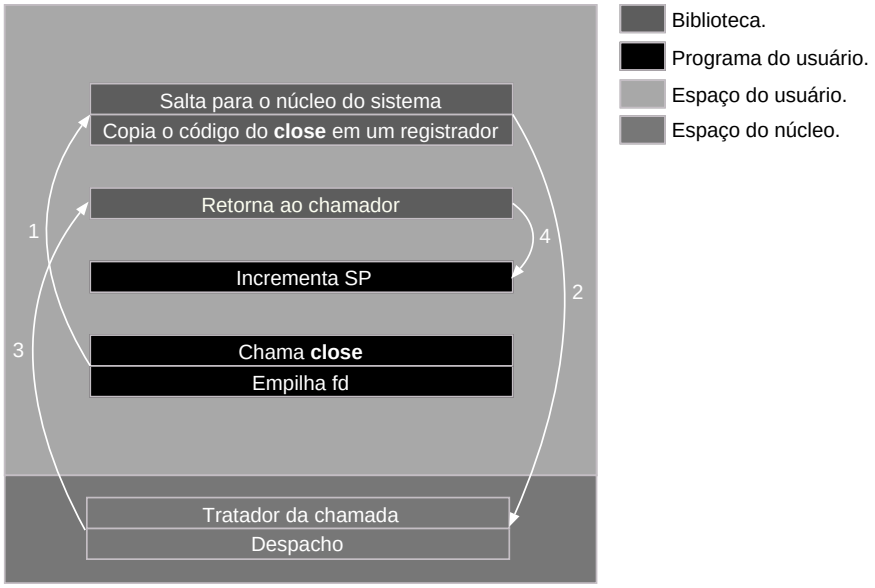


2. (2,0) A figura dada a seguir mostra o que ocorre no programa, na biblioteca e no núcleo do sistema operacional ao fazermos uma chamada ao sistema **close**. Indique a precedência que deve existir entre os blocos da figura a partir do momento em que o programa faz a chamada.



Resp.: Na figura dada a seguir indicamos a precedência entre os blocos. Na transição 1, depois de a chamada **close** ser feita, o controle é passado à biblioteca que implementa uma interface para a 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 **close**.



3. (2,0) Suponha que um programa tenha executado em um sistema operacional sobre uma máquina virtual. Suponha ainda que o tempo decorrido entre o início e o término do programa tenha sido de 6000s e que, durante essa execução, 600 operações de E/S com tempo de 1,5ms cada tenham sido feitas. Se agora o sistema executar diretamente sobre a máquina real, quantas operações de E/S uma nova versão do programa, que também demore 6000s para terminar, poderá fazer, se o processador virtual tiver 70% do poder do processador real, e se as operações de E/S na máquina real durarem 1,2ms cada?

   Resp.: Como foi observado em uma mensagem na plataforma, vamos aceitar duas respostas: usando o tempo total de 6s, ou usando o tempo

total de 6000s. Primeiramente vamos mostrar a seguir a resposta com o tempo de 6s. Como o programa executou 600 operações de E/S de 1,5ms cada, então 900ms do tempo total de 6s ou 6000ms foram gastos com E/S. Isso significa que o tempo de execução do processo no processador foi de $6000 - 900 = 5100$ms. Agora, como o processador virtual tem 70% do poder do processador real, isso significa que o processador virtual pode somente executar 70% do código executado pelo processador real em um dado intervalo de tempo. Como o tempo de 5100ms calculado anteriormente é o tempo de execução no processador virtual, então o tempo de execução no processador real será $0,7 \times 5100ms = 3570$ms. Finalmente, como o tempo total deverá ser de 6000ms, o novo tempo disponível para as operações de E/S será de $6000 - 3570 = 2430$ms. Agora, como cada operação de E/S executa no hardware real em 1,2ms, então o novo número de operações de E/S será $2430/1,2 = 2025$. Para o caso de 6000s ou 6000000ms, o tempo no processador virtual foi de $6000000 - 900 = 5999100$ms. O tempo no processador real será $0,7 \times 5999100 = 4199370$ms. Logo, nesse caso, teremos um tempo de $6000000 - 4199370 = 1800630$ms para executar as operações de E/S, significando que poderemos executar $1800630/1,2 = 1500525$ operações.

4. (2,0) Suponha que os processos A, B, C e D sejam os únicos em execução no computador, e que cada processo precise executar no processador por 6 unidades de tempo. Suponha ainda que o escalonador alterne o processador entre esses processos a cada 2 unidades de tempo, de tal modo que um processo somente volte a executar por 2 novas unidades após todos os outros processos também terem executado por 2 unidades. Quantas unidades de processamento deverão existir no computador para que o tempo decorrido entre o início e o término da execução de cada processo seja de 10 unidades de tempo? E se esse tempo for de 18 unidades de tempo? E se for de 6 unidades de tempo? Justifique a sua resposta.

   Resp.: -Como cada processo precisa necessariamente executar no processador por 6 unidades de tempo, isso significa que as 4 unidades de tempo adicionais no tempo decorrido de 10 unidades de tempo serão o tempo de espera pelo processador. Agora, como cada processo executa por 2 unidades de tempo antes de o escalonador ser chamado, então po-

demos concluir que cada processo alterna a sua execução com um outro processo antes de executar novamente (não necessariamente na mesma unidade de processamento). Finalmente, como 4 processos estão em execução no sistema, e como todos eles terminam após 10 unidades de tempo, então são necessárias 2 ou 3 unidades de processamento. Com 2 unidades de processamento, a cada 2 unidades de tempo, dois processos diferentes executam em paralelo nas unidades. Já com 3 unidades de tempo, a cada 4 unidades de tempo, três processos executam em paralelo por 2 unidades de tempo, seguido pelo processo que ainda não executou (duas unidades ficam ociosas) nas 2 unidades de tempo finais.
-No caso do tempo decorrido de 18 unidades de tempo, lembrando que cada processo precisa executar no processador por 6 unidades de tempo, então temos agora 12 unidades de tempo adicionais, que novamente são o tempo de espera pelo processador. Lembrando também que como cada processo executa ininterruptamente por 2 unidades de tempo no processador, então um processo somente executa novamente quando os outros 3 processos também executaram por 2 unidades de tempo. Logo, como os processos devem executar sequencialmente, um após o outro, então teremos somente 1 unidade de processamento.
-Finalmente, no caso do tempo decorrido de 6 unidades de tempo, como esse tempo é igual ao tempo de execução no processador, então cada processo precisa necessariamente executar ininterruptamente (em alguma unidade de processamento) desde o início até o final da sua execução. Como temos 4 processos em execução no sistema então, neste caso, deverão exisitr pelo menos 4 unidades de processamento.

5. (2,0) Suponha que diversos processos acessem um arquivo compartilhado, sendo que alguns deles acessam o arquivo para leitura, enquanto que outros acessam o arquivo para escrita. Como os semáforos binários podem ser usados para garantir um uso eficiente do arquivo, ou seja, que diversos processos possam ler simultaneamente o arquivo se nenhum outro processo estiver escrevendo nele, mas um processo somente possa escrever se obtiver acesso exclusivo ao arquivo? Justifique a sua resposta.

   Resp.: A seguir, vamos chamar os processos que leêm o arquivo de

processos leitores e os processos que escrevem no arquivo de processos escritores. Como precisamos garantir que diversos processos leitores possam acessar o arquivo ao mesmo tempo, e como um processo escritor deve necessariamente ter acesso exclusivo ao arquivo, vamos usar um semáforo binário *acesso_arquivo* para garantir o acesso exclusivo, mas com uma diferença: todos os processos leitores precisam, em essência, compartilhar uma mesma "seção crítica", como será descrito a seguir. Para que todos os processos leitores possam compartilhar a mesma "seção crítica", o primeiro processo leitor a acessar o arquivo deverá ser o único a obter o acesso exclusivo a ele. Depois de obtido o acesso exclusivo, somente o último processo leitor, ou seja, o último a deixar de acessar o arquivo, deverá liberar o acesso exclusivo a ele. Para implementar a idéia anterior, precisaremos de uma variável *numero_leitores*, que contará o número de leitores atualmente acessando o arquivo. Como diversos processos leitores precisarão acessar essa variável, para incrementá-la antes de acessarem o arquivo e decrementá-la após acessarem o arquivo, então precisaremos definir um outro semáforo binário, *acesso_numero*, para garantir o acesso exclusivo à variável. Antes da execução de quaisquer procesos, os semáforos binários deverão ser inicializados com 1, e a variável *numero_leitores* com 0. A seguir estão os pseudocódigos (em C, mas podem ser descritos ou estar em qualquer outra linguagem, desde que estejam corretos) LerArquivo e EscreverArquivo, sendo que cada um dos processos deve usar a função similar ao tipo de acesso que faz ao arquivo. Note que, caso o primeiro processo leitor não consiga acessar exclusivamente o arquivo (porque um processo escritor está acessando ele), todos os outros processos leitores também serão bloqueados, porque o primeiro leitor somente liberará o acesso exclusivo à variável *numero_leitores* após garantir o acesso exclusivo ao arquivo.

```
void LerArquivo(void)
{
  P(acesso_numero);
  numero_leitores = numero_leitores + 1;
  if (numero_leitores == 1)
  {
    // O primeiro leitor deverá obter o acesso exclusivo ao arquivo.
```

```
    P(acesso_arquivo);
  }
  V(acesso_numero);
  // Código para ler o arquivo (provavelmente dependerá do
  // processo leitor).
  P(acesso_numero);
  numero_leitores = numero_leitores - 1;
  if (numero_leitores == 0)
  {
    // O último leitor deverá liberar o acesso exclusivo ao arquivo.
    V(acesso_arquivo);
  }
  V(acesso_numero);
}

void EscreverArquivo(void)
{
  P(acesso_arquivo);
  // Código para escrever no arquivo (provavelmente dependerá do
  // processo escritor).
  V(acesso_arquivo);
}
```

6. (2,0) Suponha que dois processos, A e B, estejam em execução no computador. Suponha ainda que o sistema operacional use o algoritmo por *round robin*, com um quantum de 2 unidades de tempo, ao escalonar os processos. Após quantas unidades de tempo A e B terminarão as suas execuções, supondo que A e B precisem executar no processador por, respectivamente, a e b unidades de tempo, onde $a, b > 0$, e que A seja inicialmente escolhido pelo escalonador?

   Resp.: Vamos supor, sem perda de generalidade, que $a \leq b$. Como o tamanho do quantum é de 2 unidades de tempo, precisamos adicionalmente verificar se a é par ou ímpar, como veremos nos dois casos a seguir.

   - Se a é par, então a sequência de execução é composta por $a/2$ execuções alternadas dos processos A e B (porque cada quantum

tem 2 unidades de tempo), seguida pela execução do final de B por $b-a$ unidades de tempo, pois B já executou por a unidades de tempo quando compartilhou o processador com A. Logo, o tempo decorrido entre o ínicio e o término de A é de $a/2$ (o número de quanta que A executou antes de terminar) multiplicado por 2 (pois cada quantum tem 2 unidades de tempo), novamente multiplicado por 2 (pois durante toda a sua execução A alternou o uso do processador com B), menos 2 unidades de tempo (pois B executa, na sequência inicial, por 2 unidades de tempo após A ter terminado), ou seja, A termina após $2a - 2$ unidades de tempo. Já o tempo decorrido entre o ínicio e o término de B é o tempo $2a$ da sequência inicial quando A e B alternam a execução no processador, mais o tempo $b - a$ que B ainda precisou executar no processador, ou seja, B termina após $a + b$ unidades de tempo.

- Finalmente, se a é impar, então a sequência de execução é composta por $(a - 1)/2$ execuções alternadas dos processos A e B (porque cada quantum tem 2 unidades de tempo e a é ímpar), seguida pela execução final de A por 1 unidade de tempo, e pela execução do final de B por $b - (a - 1)$ unidades de tempo, pois A e B executaram por $a - 1$ unidades de tempo quando compartilharam o processador. Logo, o tempo decorrido entre o ínicio e o término de A é de $(a - 1)/2$ (o número de quanta completos que A executou antes de terminar) multiplicado por 2 (pois cada quantum tem 2 unidades de tempo), novamente multiplicado por 2 (pois durante toda a sua execução A alternou o uso do processador com B), mais 1 unidade de tempo após a qual A termina (pois A executou antes por $(a - 1)/2$ quanta de 2 unidades de tempo), ou seja, A termina após $2a - 1$ unidades de tempo. Já o tempo decorrido entre o ínicio e o término de B é o tempo $2a - 1$ decorrido até A terminar, mais o tempo $b - (a - 1)$ que B ainda precisou executar no processador, ou seja, B termina após $a + b$ unidades de tempo.