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 - Primeiro Semestre de 2010

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.

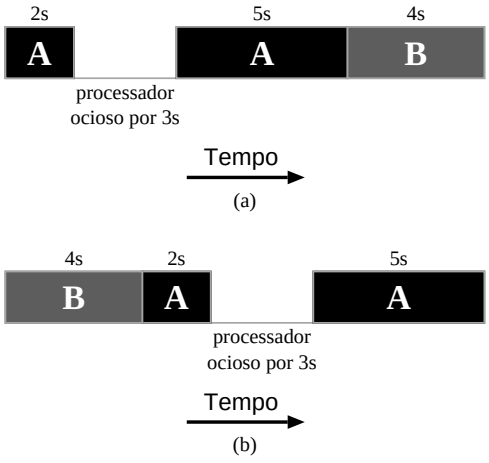
Nome -
Assinatura -

---

1. (1.5) Suponha que um programa A execute por 7s, e que ele faça uma operação de E/S, com duração de 3s, após ter executado por 2s no processador. Suponha ainda que um programa B execute por 4s, e

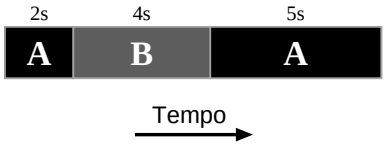
que B não faça nenhuma operação de E/S. Se os programas A e B são os únicos em execução no sistema operacional, após quanto tempo B terminará a sua execução, se o sistema operacional não usar a multiprogramação? E se o sistema operacional usar a multiprogramação, de tal forma que um processo somente seja interrompido para realizar E/S?

Resp.: -Se o sistema operacional não usar a multiprogramação, então o processador ficará ocioso durante a execução das operações de E/S. Se o programa A começar a executar no processador antes do programa B então, como podemos ver pela parte (a) da figura a seguir, o tempo decorrido até o programa B terminar será o tempo de execução de 7s de A (os 2s antes da operação de E/S mais os 5s depois desta operação), mais o tempo de 3s da operação de E/S feita por A, e mais o tempo de 4s de execução de B. Logo, o tempo decorrido até o término de B será de $7 + 3 + 4 = 14$s. Agora, se o programa B começar a executar antes de A, como ocorre na parte (b) da figura a seguir, então o tempo decorrido até B terminar será o tempo de execução de B, ou seja, 4s.



-Se o sistema operacional usar a multiprogramação, o processador não precisará mais ficar ocioso ao executar as operações de E/S. Logo, se o programa A executar antes do programa B então, após A executar por 2s e ser bloqueado por ter feito uma operação de E/S, B poderá executar no processador, e executará por 4s sem interrupções, porque B não faz operações de E/S. Logo, como podemos ver pela figura a

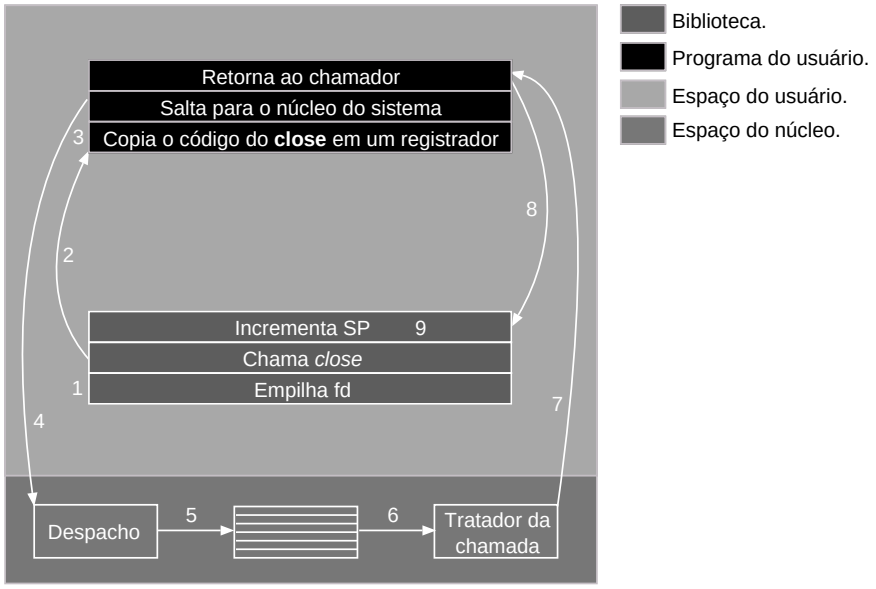
seguir, que o tempo decorrido até B terminar a sua execução será o tempo que A executou antes de fazer a operação de E/S mais o tempo de execução de B, ou seja, será de $2 + 4 = 6s$. Note que A poderá voltar a executar depois de B, porque não existem outros programas em execução no sistema operacional, e porque o tempo da operação de E/S é menor do que o tempo de execução de B. Finalmente, se B executar antes de A então, assim como na parte (b) da figura anterior, o tempo decorrido até B terminar a sua execução também será de 4s.



2. (1.5) Na Aula 2 vimos os passos executados ao chamarmos a função de biblioteca *read*, a qual implementa a chamada ao sistema operacional **read**. Quais serão agora os passos executados se desejarmos fazer a chamada ao sistema operacional **close**, usando a função de biblioteca *close*, para a qual passamos o descritor *fd* do arquivo a ser fechado?

   Resp.: A seguir mostramos a figura obtida, similar à dada na última transparência 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 único parâmetro *fd* desta função. Após empilhar o parâmetro *fd* agora o processo, no passo 2, chama a função da biblioteca *close*. Após ser chamada, esta função então, no passo 3, coloca, em um lugar pré-determinado pelo sistema operacional, o código que identifica a chamada ao sistema operacional **close**. Depois disso, no passo 4, esta 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 **close**. Então, no passo 6, o sistema operacional executa esta função, denominada de **tratador da**

chamada close. Depois deste tratador executar as tarefas necessárias para fechar o arquivo então, no passo 7, o processador será alternado do modo supervisor para o modo usuário, e o controle será passado à instrução, da função *close* da biblioteca, posterior à instrução TRAP. Após fazer as finalizações necessárias ao fechamento do arquivo, a função da biblioteca então passa, no passo 8, o controle novamente ao processo do usuário, na instrução seguinte a que chamou a função. Finalmente, no passo 9, o processo do usuário incrementa o ponteiro da pilha SP, para remover o parâmetro *fd* colocado na pilha antes de chamarmos a função *close*.



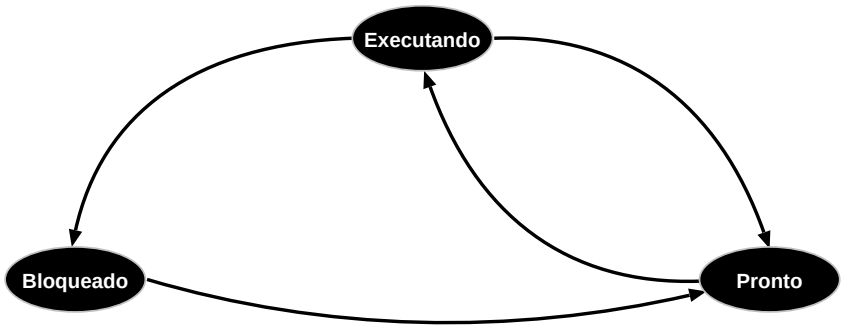
3. (2.0) Suponha que o sistema operacional esteja executando diretamente sobre o hardware do computador, cujas operações de E/S demoram 1ms. Suponha ainda que um processo tenha executado por 6s e que, durante a sua execução, tenha feito 1500 operações de E/S. Se o sistema operacional agora executar sobre uma máquina virtual, que reduz a velocidade do processador em 10% e a velocidade das operações de E/S em 50%, quantas operações de E/S o programa poderá fazer para o seu tempo de execução ainda ser de 6s? Justifique a sua resposta.

Resp.: Como cada operação de E/S demora 1ms e como o processo fez 1500 operações, então 1500ms, ou $1,5$s, do tempo de execução deste processo foram gastos em operações de E/S, quando ele executou no sistema operacional sobre o hardware do computador. Logo, o processo executou no processador do hardware por 6s$-1,5$s$= 4,5$s. Agora, quando o processo executar no sistema operacional sobre a máquina virtual, o tempo de execução dele no processador virtual será de $4,5/0,9 = 5$s, pois o processador da máquina virtual é 10% mais lento do que o processador do hardware. Com isso o processo, para executar no mesmo tempo de 6s, somente poderá executar operações de E/S por 1s ou 1000ms. Agora, como o tempo gasto por cada operação de E/S é de $1/0,5 = 2$ms, pois a máquina virtual reduz a velocidade das operações de E/S em 50%, então o processo deverá executar $1000/2 = 500$ operações de E/S.

4. (1.5) Um aluno de sistemas operacionais fez um estudo sobre os estados de um processo. Ele concluiu que, se o algoritmo de escalonamento for não-preemptivo, somente os estados dados no diagrama da parte (a) da figura a seguir serão necessários. Ele também concluiu que somente os estados dados no diagrama da parte (b) da mesma figura serão necessários se o algoritmo de escalonamento for preemptivo e os dispositivos de E/S não gerarem interrupções. As conclusões do aluno estão corretas? Justifique a sua resposta.

Resp.: Nenhuma das conclusões do aluno está correta. A primeira conclusão diz que o estado pronto não será necessário se o sistema operacional usar um algoritmo de escalonamento não-preemptivo. Esta conclusão não é verdadeira porque, se um processo atualmente em execução for bloqueado (por exemplo, ao fazer uma operação de E/S),

então este processo, quando for desbloqueado, deverá sempre ser colocado no estado pronto (e não no estado executando) se o processador não estiver ocioso. A segunda conclusão também está incorreta, porque um processo não é bloqueado somente ao fazer uma operação de E/S (por exemplo, o processo pode ser bloqueado em um semáforo binário ao esperar para ter acesso exclusivo à sua sessão crítica) e, com isso, o estado bloqueado será necessário. Logo, em ambos os casos estudados pelo aluno, o diagrama de transição de estados deverá ser o dado a seguir, visto em detalhes na Aula 4, com três estados: executando, pronto e bloqueado.



5. (1.5) Suponha que dois processos, A e B, compartilhem uma pilha, usada para armazenar números. Suponha ainda que esta pilha, inicialmente vazia, possa armazenar até n números, e que não exista uma variável para contabilizar a quantidade de números armazenados na pilha. O processo A continuamente insere números na pilha. Já o processo B continuamente remove dois números da pilha, calcula a soma destes números, e depois insere o resultado da soma na pilha. Como os semáforos devem ser usados para garantir a correta execução dos processos A e B? Justifique a sua resposta.

   Resp.: Como não existem variáveis para contabilizar a quantidade de números armazenados na pilha, precisaremos de um semáforo de contagem *cheia*, que contará a quantidade de números na pilha e será usado para bloquear o processo B quando a pilha estiver vazia. Também

precisaremos de um semáforo de contagem *vazia*, que contará a quantidade de números que ainda podem ser inseridos na pilha e será usado para bloquear o processo A quando a pilha estiver cheia. E também precisaremos de um semáforo binário *acesso*, usado para garantir o acesso exclusivo à pilha. Como inicialmente a pilha está vazia e os processos não estão em execução, então o valor inicial de *cheia* será 0, o valor inicial de *vazia* será n e o valor inicial de *acesso* será 1. A seguir mostramos dois possíveis procedimentos que podem ser executados pelos processos A e B. Os procedimentos da sua resposta não precisam ser exatamente iguais aos dados a seguir, mas precisam, para estarem corretos, de: (i) possuírem um laço infinito como os destes procedimentos; (ii) garantirem o acesso exclusivo à pilha; e (iii) garantirem que o processo A não coloque números em uma pilha cheia e que o processo B não retire números de uma pilha vazia.

```
void ProcessoA()
{
  while (1)
  {
    P(vazia);
    P(acesso);
    // Código para colocar um número na pilha.
    V(acesso);
    V(cheia);
  }
}

void ProcessoB()
{
  while (1)
  {
    P(cheia);
    P(cheia);
    P(acesso);
    // Código para remover dois números da pilha, calcular a sua
    // soma, e depois colocar esta soma na pilha.
    V(acesso);
    // Note que somente uma operação V sobre vazia é necessária,
    // porque colocamos a soma dos números na pilha.
```

```
    V(vazia);
    V(cheia);
  }
}
```

6. (2.0) Suponha que um sistema operacional use o algoritmo de escalonamento por *round robin*, sendo que cada quantum equivale a duas unidades de tempo. Suponha ainda que os processos tenham sido escalonados no processador na ordem ABABABABBB, e que os processos usem todos os seus quanta. Qual será a nova sequência de escalonamento se o sistema operacional agora usar o algoritmo por prioridades, sendo que um processo executa até a sua prioridade, que é aumentada de 1 unidade a cada unidade de tempo, deixe de ser a menor, e que as prioridades iniciais dos processos A e B são, respectivamente, de 7 e 5?

   Resp.: Pela ordem de execução, vemos que o processo A executou por 4 quanta e o processo B por 6 quanta. Agora, como cada quanta corresponde a duas unidades de tempo, e como os processos A e B usaram todos os seus quanta, então A e B executaram por, respectivamente, 8 e 12 unidades de tempo no processador. As duas tabelas dadas a seguir mostram a ordem de execução se agora usarmos o algoritmo por prioridades, sendo que cada processo executa até terminar ou até algum outro processo possuir prioridade menor, e supondo que a prioridade é aumentada de uma unidade a cada unidade de tempo em execução. Para cada tabela, a primeira linha mostra a unidade de tempo em que cada processo foi executado. A segunda linha mostra os processos executados em cada unidade de tempo. Finalmente, a terceira linha mostra, para o processo de cada coluna, a prioridade deste processo antes de ele executar na unidade de tempo desta mesma coluna. Como podemos ver pelas tabelas, a sequência de execução agora será BBBAABBAABBAABBAABBB.

0	1	2	3	4	5	6	7	8	9
B	B	B	A	A	B	B	A	A	B
5	6	7	7	8	8	9	9	10	10

10	11	12	13	14	15	16	17	18	19
B	A	A	B	B	A	A	B	B	B
11	11	12	12	13	13	14	14	15	16