GABATITO - PROVA FINAL - Sistemas Operacionais I
Professor: Leandro Marzulo
2013-2

Nome:

Instruções: Esta prova é composta de 3 questões totalizando 10 (dez) pontos. Responda as questões de forma sucinta e clara. As questões devem ser respondidas a caneta. BOA PROVA!

1) (1,5) Como são implementadas as chamadas ao sistema operacional em um sistema baseado em micronúcleo?

Em um sistema baseado em micronúcleo, as chamadas ao sistema operacional são implementadas por processos servidores executando no modo usuário, ao invés de usarmos a instrução TRAP para chamar um código dentro do núcleo do sistema. Neste caso, quando um processo executar uma função da biblioteca que precise de algum serviço do sistema operacional, uma mensagem será enviada para o servidor responsável pelo serviço. Depois de receber a mensagem, o servidor fará o que for necessário para executar o serviço. O servidor poderá enviar mensagens aos outros servidores para auxiliá-lo a executar o serviço requisitado. Depois de executar o serviço, o servidor enviará uma mensagem ao código da função da biblioteca com o resultado do serviço requisitado. Finalmente, após fazer alguns processamentos adicionais, a função passará ao processo o resultado do serviço executado pelo servidor.

2) (5,0) Suponha que um processo requer, em média, um tempo **T** de execução antes que se bloqueie por E/S. Suponha também que uma comutação entre processos requer um tempo **S**, o qual é efetivamente desperdiçado (overhead). Para o escalonamento por round robin com quantum **Q**, dê uma equação para a eficiência do processador (fração do tempo usado para executar os processos) para cada um dos seguintes casos:

a) **(1,0)** $Q = \infty$

Como T é menor do que Q (pois Q é infinitamente grande), então o processo executará no processador por um tempo T antes de bloquear, e depois será necessário fazer apenas uma troca de contexto (não teremos nenhuma preempção), o que irá requerer um tempo S. Logo, a eficiência do processador será de T/(T + S).

b) **(1,0)** $Q > T$

Se Q for finito, mas maior do que T, a eficiência será a mesma do item a, isto é, T/(T + S), pois o processo executará, assim como antes, por um tempo T antes de ser bloqueado e não teremos nenhuma preempção.

c) **(1,0)** $S < Q < T$

Se Q for menor do que T, então teremos que fazer T/Q trocas de contexto até que o processo seja finalmente bloqueado. Como cada troca de contexto leva um tempo S, o tempo gasto com as trocas de contexto será de ST/Q. Logo, a eficiência do processador será de T/(T + ST/Q), isto é, Q/(Q + S).

d) **(1,0)** $Q = S$

Assim como no caso anterior, teremos que executar T/Q = T/S trocas de contexto (pois Q=S). Logo, para obter a eficiência do processador, basta substituir, na equação dada no item c, Q por S. Ao fazer isso, vemos que a eficiência do processador será de S/(S + S) = S/2S = 1/2.

e) **(1,0)** Q perto de zero.

Quando Q tender a zero, o número de trocas de contexto tenderá a infinito e o tempo de troca de contexto também. Com isso, a eficiência do processador tenderá a zero, pois quanto mais Q tender a 0, mais tempo será gasto com as trocas de contexto. É fácil observar isso com o limite abaixo, onde X é o número de trocas de contexto:

$$\lim_{X \to \infty} \frac{T}{T + S \times X} = 0$$

2) (2,0) Suponha que dois processos, **A** e **B**, estão executando no sistema operacional. Os dois processos acessam uma mesma fila, e cooperam entre si como descrito a seguir. O processo A continuamente coloca itens no final da fila, e o processo B continuamente retira itens do início da fila. A fila pode armazenar até **n** itens ao mesmo tempo.
Responda:

a) (1,0) Qual dos problemas clássicos de sincronização estudados em sala está sendo abordado nesta questão? Como os semáforos (binários e de contagem) podem ser usados para garantir a exclusão mútua e para garantir o correto

funcionamento dos processos **A** e **B**? Mostre e explique o pseudo-código. Não esqueça de mostrar com qual valor cada semáforo utilizado deve ser inicializado.

O problema da questão é o problema do produtor e do consumidor, em que os processos usam um buffer compartilhado para a troca de elementos. Então precisaremos de três semáforos: o semáforo cheia, usado para bloquear o processo A quando a fila possuir n elementos; o semáforo vazia, usado para bloquear o processo B quando a fila estiver vazia; e o semáforo binário acesso, usado para garantir o acesso exclusivo à fila. O semáforo acesso será inicializado com 1, pois inicialmente nenhum processo está acessando a fila. Os valores dos outros semáforos dependerão do nu´mero de elementos da fila antes da execução dos processos A e B. Se a fila possuir a ≤ n elementos, então o semáforo cheia dever´a ser inicializado com n − a, e o semáforo vazia com a. A seguir mostramos os fragmentos dos pseudo-códigos dos processos A (funcão ProcessoA) e B (funcão ProcessoB):

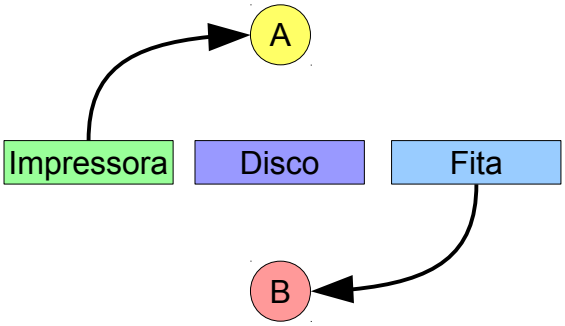
```
void ProcessoA()
{
	// Gera um elemento.
	// Bloqueia se a fila estiver cheia.
	P(cheia);
	// Obtém acesso exclusivo à fila.
	P(acesso);
	// Coloca o elemento no final da fila.
	// Libera o acesso exclusivo à fila.
	V(acesso);
	// Desbloqueia B se a fila estava vazia.
	V(vazia);
}

void ProcessoB()
{
	// Bloqueia se a fila estiver vazia.
	P(vazia);
	// Obtém acesso exclusivo à fila.
	P(acesso);
	// Remove o elemento da fila.
	// Libera o acesso exclusivo à fila.
	V(acesso);
	// Desbloqueia A se a fila estava cheia.
	V(cheia);
	// Processa o elemento lido
}
```

b) (1,0) Suponha que a fila está inicialmente vazia. Suponha ainda que usamos o algoritmo por round robin ao escalonar os processos do sistema. Quantos elementos estarão na fila depois de **L** execuções de cada um dos processos, se **A** colocar exatamente **a** elementos em cada quantum, e **B** retirar exatamente **b** elementos em cada quantum, sendo que **a > b e n > (L * a)**?

Temos dois casos dependendo do processo que executar primeiro. Se A executar antes de B, então cada processo executará alternadamente no processador por L vezes, sem bloquear. Com isso, teremos La−Lb = L(a−b) elementos na fila depois destas L execucões, pois em cada quantum A coloca a elementos na fila e B retira b elementos da fila. Agora, se B executar antes de A, ele será bloqueado na primeira execução, pois a fila está inicialmente vazia. Logo, em uma das L execuções, B não tirou elementos da fila. Portanto, em relação ao cenário anterior, teremos mais b elementos na fila, ao final das L execuções alternadas de A e B, ou seja, a fila terá agora La – (L-1) b = L(a−b)+b elementos.

3) (1,5) Suponha que dois processos, **A** e **B**, compartilham três recursos: uma impressora, um disco e uma unidade de fita magnética. Suponha ainda que o processo **A** obteve a impressora e o processo **B** obteve a unidade de fita magnética. Se os processos **A** e **B** desejarem obter um dos recursos que não possuem, sem liberar os recursos alocados a eles, que escolhas de **A** e **B** gerariam um *deadlock*?

Temos inicialmente a seguinte situação:



Para ocorrer um deadlock, precisamos ter as seguintes condições: exclusão mútua, posse e espera, não preempção de recursos e espera circular. Assumiremos que os recursos não são preemptivos. A exclusão mútua no acesso existe. A e B já estão de posse de recursos e vão solicitar outros, caracterizando posse e espera. Precisamos agora causar a espera circular. Para isso, A precisa solicitar a Fita e B a Impressora. Com na imagem abaixo: