INSTITUTO MAUÁ DE TECNOLOGIA

MAUÁ

ECM225 – Sistemas Operacionais
2ª lista de exercícios – Processos e Threads

1. Um computador possui espaço suficiente para manter cinco programas em execução em sua memória. Esses programas ficam ociosos 50% do tempo aguardando operações de E/S. Que fração do tempo da CPU é perdida em operações de E/S? **Dica**: a resposta é a porcentagem de tempo em que o primeiro e o segundo e o terceiro e o quarto e o quinto programa estão simultaneamente ociosos.
2. Um computador possui 4GB de RAM, da qual o sistema operacional ocupa 512MB. Supor, por simplicidade, que todos os processos necessitem de 256 MB e que possuem as mesmas características. Se a meta é obter uma utilização de CPU de 99%, qual é a máxima porcentagem de tempo de E/S que pode ser tolerada? **Dica**: calcule quantos programas podem ser executados na memória. Depois, utilize uma estratégia similar à do exercício 1 para descobrir o valor pedido.
3. Vários *jobs* podem ser executados em paralelo e terminar mais rápido do que se fossem executados sequencialmente. Supor que dois *jobs*, cada um necessitando de 20 minutos de tempo de CPU, são iniciados simultaneamente. Em que tempo o último *job* terminará se: (a) eles são executados sequencialmente; (b) eles são executados em paralelo. Assumir 50% de espera em operações de E/S.
4. Assumir que se está tentando realizar o download de um arquivo de 2GB da Internet. O arquivo está disponível em vários servidores-espelho, cada qual podendo entregar uma fração dos dados (bytes). Para obter dados do arquivo, deve-se informar ao servidor-espelho o número do byte inicial e o número do byte final. Explicar como threads poderiam ser utilizadas para otimizar o tempo de download do arquivo.
5. Em um servidor multi-thread web, se a única forma de ler de um arquivo é utilizar uma chamada `read()` que bloqueia a execução, que tipo de threads seriam mais aconselháveis: threads de nível de usuário ou threads de kernel? Justificar.
6. Por que é necessário se armazenar os valores dos registradores quando se utilizam threads?
7. Por que uma thread voluntariamente deixaria de usar a CPU com uma instrução como `yield()`?
8. Em um servidor de arquivos, considere que leva 12ms para receber, despachar e processar uma requisição, assumindo que os dados necessários já estão em um bloco de cache. Se for necessário executar uma operação no disco (e em 1/3 do tempo será necessário) somam-se 75ms ao tempo, e, neste período, a thread dorme. Quantas requisições por segundo o servidor pode manipular se ele tiver: (a) uma única thread; (b) for multi-thread.
9. Qual é a grande vantagem de se implementar threads no espaço do usuário? Qual é a

grande desvantagem?

10. O algoritmo de exclusão mutual de Peterson funciona quando o agendamento é preemptivo? E se o agendamento for não preemptivo ele funciona?

11. Quando um processo deseja inserir sua região crítica, ele verifica se a entrada é permitida. Se não for, o processo apenas fica em um laço esperando até que seja permitida a entrada. Considere um computador com dois processos, H, com alta prioridade, e L, com baixa prioridade. As regras de agendamento são tais que H é executado sempre que estiver no estado pronto. Em um determinado momento, com L executando em sua região crítica, H fica pronto para ser executado (por exemplo, após a conclusão de uma operação de E/S). H entra em espera ocupada, mas como L nunca é agendado enquanto H está rodando, L nunca tem a chance de deixar sua região crítica, então H fica em espera ocupada para sempre. Esta situação é por vezes referida como o problema de **inversão de prioridade**. **Pergunta**: este problema ocorreria se o agendamento fosse do tipo round-robin? Explique.

12. O problema do produtor-consumidor pode ser estendido para um sistema com diversos produtores e consumidores que respectivamente escrevem em e leem de um buffer compartilhado? Assumir que tanto um produtor quanto um consumidor executa em sua própria thread.

13. Um restaurante fast-food tem quatro tipos de empregados: (1) garçons, que recebem pedidos dos clientes; (2) cozinheiros, que preparam a comida; (3) especialistas em embalagem, que armazenam a comida em embalagens; e (4) caixas, que entregam as encomendas aos clientes e recebem seu dinheiro. Cada funcionário pode ser considerado como um processo sequencial de comunicação. Que forma de comunicação interprocesso eles usam? Relacione este modelo aos processos em um sistema operacional como o Linux.

14. Suponha que se tem um sistema de transmissão de mensagens usando caixas de correio. Ao enviar para uma caixa de correio cheia ou tentar receber de uma caixa vazia, um processo não é bloqueado. Em vez disso, ele recebe um código de erro de volta. O processo responde ao código de erro apenas tentando de novo, repetidamente, até conseguir. Esse esquema leva a condições de disputa? Discutir.

15. Considerar o seguinte código em C:

```
void main( ) {
    fork( );
    fork( );
    exit( );
}
```

    Quantos processos-filho são criados após a execução deste programa?

16. Pode-se medir se um processo é limitado à CPU ou limitado à E/S apenas analisando o seu código-fonte? E como isso pode ser verificado em tempo de execução?

17. Explicar como o valor de *quantum* do tempo e o tempo de comutação de contexto afetam um ao outro, em um algoritmo de agendamento de round-robin.

18. Cinco *jobs* estão esperando para serem executados. Seus tempos de execução esperados são 9, 6, 3, 5 e *X*. Em que ordem eles devem ser executados para minimizar o tempo médio de resposta (a resposta dependerá de X)?

19. Cinco *jobs*, nomeados de A a E, são submetidos a um computador quase ao mesmo tempo. Eles têm tempos de execução estimados de 10, 6, 2, 4 e 8 minutos. Suas prioridades (externamente determinadas) são 3, 5, 2, 1 e 4, respectivamente, sendo 5 a prioridade mais alta. Para cada um dos seguintes algoritmos de agendamento, determinar o tempo médio de retorno do processo (*turnaround time*). Ignorar possíveis sobrecargas resultante da comutação de processos.

    (a) Round-robin.

    (b) Agendamento de prioridade.

    (c) Primeiro a chegar – Primeiro a ser servido (na ordem 10, 6, 2, 4, 8).

    (d) Primeiro o *job* mais curto.

    Para (a), assumir que o sistema é multiprogramado, e que cada *job* tem uma fatia justa da CPU. De (b) até (d), assumir que apenas um *job* executa por vez, até que ele termine. Todos os *jobs* são limitados por CPU.