UNIVERSIDADE DE SÃO PAULO - ICMC
SISTEMAS OPERACIONAIS I
Gabarito – Lista 3

1) Na sua opinião, qual foi a principal evolução de hardware e software que possibilitou o desenvolvimento dos Sistemas Operacionais que existem atualmente? Justifique sua resposta.

Resposta livre, desde que bem justifica. Conceitos importantes: multiprogramação, *timesharing*, circuitos integrados.

2) Descreva quais são as principais tarefas de um Sistema Operacional.

Aula 3 – slides 05 e 10

3) Defina e diferencie: programa, processo e *thread*.

Capítulos 1 e 2 do livro texto
Aula 3 – slide 14; Aula 4 – slide 2; Aula 4 (complemento *threads*) – slide 3

4) No que diz respeito à implementação, qual a diferença entre *threads* de usuário (*user-level threads*) e *threads* de *kernel* (*kernel-level threads*)?

Aula 4 (complemento *threads*) – slides 13-19
Capítulo 2, seção 2.2

5) O custo (e conseqüentemente o tempo) de criação de uma *thread* (seja ela uma *thread* de usuário ou uma *thread* de *kernel*) é menor que o custo de criação de um processo. Por que?

Aula 4 (complemento *threads*) – slides 5-7
Capítulo 2, seção 2.2

6) Um processo é caracterizado por ser um "programa em execução". Durante sua existência no sistema, um processo pode assumir basicamente três estados. Cite quais são esses estados e descreve os eventos que fazem com que um processo mude de um estado para o outro.

Aula 3 – slide 16; Aula 4 – slides 2, 8-11;
Capítulo 2, seção 2.1.5

7) Qual a diferença entre processos *CPU-bound* e *I/O-bound*? Quais são os problemas que podem ocorrer caso o sistema tenha muitos processos *CPU-bound* ou muitos processos *I/O-bound*?

Aula 4 – slide 12;

8) O que são chamadas de sistema? Descreva o que o Sistema Operacional deve fazer quando uma chamada de sistema ocorre simulando uma chamada de sistema.

Aula 3 – slides 22-26
Capítulo 1, seção 1.6

9) Para que servem as *chamadas de sistema*? Como elas podem ser implementadas? (sugestão de leitura: "How System Calls Are Implemented on i386 Architecture", http://www.tldp.org/LDP/lki/lki-2.html#ss2.11).

Aula 3 – slides 22-26
Leitura sugerida acima!

10) [POSCOMP] No que diz respeito às vantagens da arquitetura de micro-núcleo (micro-kernel) para sistemas operacionais em relação à arquitetura de núcleo monolítico, quais das seguintes afirmações são verdadeiras?

I. A arquitetura de micro-núcleo facilita a depuração do S.O.
II. A arquitetura de micro-núcleo permite um número menor de mudanças de contexto.
III. A arquitetura de micro-núcleo facilita a reconfiguração de serviços do SO, pois a maioria deles reside em espaço de usuário.

(a) apenas I; (b) II e III; (c) I e III; (d) I e II; (e) todas são verdadeiras.

11) Cite pelo menos duas vantagens do uso de sistemas operacionais baseados na estrutura em Camadas.

Aula 3 – slides 34-42
Capítulo 1, seção 1.7.2

12) Explique por que a tarefa do escalonador de processos é importante para o desempenho da CPU.

Aula 4 – slides 13-17;
Capítulo 2, seção 2.1

13) Descreva a execução dos seguintes algoritmos de escalonamento: *Round-robin*, Prioridades e Múltiplas Filas.

Aula 4 – slides 36-45
Capítulo 2, seção 2.5.3

14) A maioria dos escalonadores *Round-robin* usa um *quantum* de tamanho fixo. O que pode acontecer se o *quantum* for muito pequeno? E se ele for muito grande?

Aula 4 – slides 39-40

15) Diferencie *Race Condition* de Exclusão Mútua.

Aula 5 – slides 3-8
Capítulo 2, seções 2.3.1, 2.3.2

16) Uma boa solução de Exclusão Mútua deve satisfazer quais condições? Por que?

Aula 5 – slide 08
Capítulo 2, seção 2.3.2

17) Cinco processos (A, B, C, D, E) estão no estado de pronto para serem executados pela CPU. O tempo estimado de execução de cada processo é: 10, 6, 2, 4 e 8 minutos, respectivamente. A ordem de prioridade é a seguinte: 3, 5, 2, 1, e 4, respectivamente, sendo 5 a mais alta prioridade. Simule a execução dos seguintes algoritmos de escalonamento com esses processos e determine o *turnaround time* (tempo de retorno) de cada processo em cada algoritmo:

a) *Round-Robin* (defina o *quantum* para cada processo);
b) Prioridades;
c) FIFO;

18) Cinco processos estão prontos para serem executados. Os tempos de execução são: 9, 6, 3, 5 e X minutos. Qual deve ser a ordem de execução dos processos para que o tempo médio de resposta (*turnaround time*) de cada processo seja minimizado?

Neste caso, o algoritmo SJF é o mais eficiente. Supondo valores para o processo X, ordene a execução dos processos.
Se X<=3, ordem: X, 3, 5, 6, 9 {tempo médio de espera: (5*X+4*3+3*5+2*6+1*9)/5}
Se 3<=X<=5, ordem: 3, X, 5, 6, 9
E assim por diante!

19) Quais das instruções a seguir devem ser realizadas somente pelo *Kernel* do Sistema Operacional? Justifique sua resposta.

a) Desabilitar interrupções; b) Ler a hora do relógio; c) Modificar a hora do relógio; d) Modificar o mapeamento da memória.

20) Descreva sobre espera ociosa e como evitá-la. Em que situação ela é aceitável.

21) Quais as condições para a ocorrência de *deadlock*?

22) Cite e descreva quatro estratégias para tratamento de *deadlocks*.

23) Observe a figura 3.4 do livro Sistemas Operacionais Modernos – Tanenbaum. Suponha que no passo (o) *C* requisite *S* em vez de requisitar *R*. Isso levaria a uma situação de *deadlock*? E se ele requisitasse ambos, *S* e *R*?

24) Observe a Figura 3.11(b) do livro Sistemas Operacionais Modernos – Tanenbaum. Se *D* requisitar uma unidade a mais, isso levará a um estão seguro ou inseguro? O que acontecerá se a requisição chegar de *C* em vez de *D*?

25) Um sistema tem dois processos e três recursos idênticos. Cada processo precisa de no máximo dois recursos. É possível ocorrer *deadlock*? Justifique a sua resposta.

26) Um sistema tem quatro processos e cinco recursos alocáveis. A alocação atual e as necessidades máximas são as seguintes:

	Alocado	*Máximo*	*Disponível*
Processo A	1 0 2 1 1	1 1 2 1 3	0 0 ***x*** 1 1
Processo B	2 0 1 1 0	2 2 2 1 0	
Processo C	1 1 0 1 0	2 1 3 1 0	
Processo D	1 1 1 1 0	1 1 2 2 1	

Qual é o menor valor de ***x*** para que esse estado seja seguro?

If *x* is 0, we have a deadlock immediately. If *x* is 1, process *D* can run to completion. When it is finished, the available vector is 1 1 2 2 1. Unfortunately we are now deadlocked. If *x* is 2, after *D* runs, the available vector is 1 1 3 2 1 and *C* can run. After it finishes and returns its resources the available vector is 2 2 3 3 1, which will allow *B* to run and complete, and then *A* to run and complete. Therefore, the smallest value of *x* that avoids a deadlock is **2**.