INE5645 - Programação Paralela e Distribuída - Prova 1B – 24/04/2017

Nome _____

1B – Conceitos Básicos: Indique Verdade/Falso

a) (**Verdade**/**Falso**) Threads distintas em um processo não são tão independentes quanto processos distintos.

Todas as threads no processo tem exatamente o mesmo espaço de endereçamento, o que significa que elas compartilham as mesmas variáveis globais do processo. Além de compartilharem o mesmo espaço de endereçamento, todas as threads compartilham o mesmo conjunto de recursos do processo (arquivos abertos, processos filhos, alarmes pendentes, tratadores de eventos, informação sobre contabilidade de execução, entre outros).

b) (**Verdade**/**Falso**) Não há proteção entre threads porque é impossível e desnecessário.

No caso de processos diversos, que podem ser de usuários diferentes e até mutuamente hostis, um processo sendo criado por um usuário, presume-se que este tenha criado múltiplas threads no processo para que essas possam cooperar concorrentemente, e não competir. Do ponto de vista de threads em processos diferentes, e ainda mais se esses processos forem de usuários diferentes, proteção entre threads é importante.

c) (**Verdade**/**Falso**) Itens propriedade de **threads** são: (a) Espaço de endereçamento, (b) Variáveis globais, (c) Contador de programa lógico, (d) Registradores, (e) Pilha, (f) Estado, (g) Recursos.

Espaço de endereçamento e recursos são propriedades de processos.

d) (**Verdade**/**Falso**) Itens propriedade de **processos** são: (a) Variáveis globais, (b) Contador de programa lógico, (c) Registradores, (d) Pilha, (e) Estado.

Deve ser incluído: Espaço de endereçamento e recursos como propriedades de processos.

e) (**Verdade**/**Falso**) Thread é uma unidade de processamento concorrente.

Antigamente, a única unidade de processamento concorrente era o processo (antes de haver thread), mas do jeito que programamos atualmente, com threads sendo cidadãos de primeira classe, essas passam a ser, no caso de estarmos no espaço do usuário, as unidades de processamento concorrente. Um SO pode ter no nível de kernel, processos com as unidades de processamento concorrente.

f) (**Verdade**/**Falso**) O esquema de escalonamento (scheduling) fundamental para

threads é preemptivo (o ato de forçar uma thread parar sua execução), baseado em prioridade. Um algoritmo **não** baseado em fatias de tempo (time-slicing é preemptivo, mas preempção não implica em time-slicing), que permitem threads de mais alta prioridade executarem tanto tempo elas necessitem.

O que é verdade: Um esquema de escalonamento (scheduling) é o ato de **selecionar** processos /threads prontas **para o estado executável.** Um tal esquema pode ser **preemptivo**, ou seja, o ato de forçar uma thread parar sua execução baseado em prioridades das threads, que permitem threads de mais alta prioridade executarem **tanto tempo elas necessitem**. Ou pode ser **não-preemptivo**, o qual força thread parar sua execução num certo tempo de processamento, ou seja, é baseado em **fatias de tempo** (time slicing) de processamento. O que é verdade é: **escalonamento *time-slicing* é preemptivo, mas preempção não implica em *time-slicing*.**

g) (**Verdade**/**Falso**) Uma aplicação que é melhor construída considerando-se múltiplas threads compartilhando o mesmo espaço de endereçamento, representa a construção de um processo que implementa essa aplicação.

Neste caso, a aplicação é construída como um processo contendo diversas threads internas a ele.

2B – Inconsistência na concorrência entre threads

Considerando as intercalações de concorrência entre duas transações T e U, como apresentadas no quadro abaixo, elas permitem um problema de inconsistência entre essas transações.

Transaction : *T*		**Transaction : *U***	
balance= b.getBalance(); *b.setBalance(balance*1.1);* *a.withdraw(balance/10)*		*balance= b.getBalance();* *b.setBalance(balance*1.1);* *c.withdraw(balance/10)*	
balance=b.getBalance();	$200		
		balance= b.getBalance()	$200
		*b.setBalance(balance*1.1)*	$220
*b.setBalance(balance*1.1)*	$220		
a.withdraw(balance/10)	$80		
		c.withdraw(balance/10)	$280

Que problema é este ?

3B – Controle de Concorrência – Semáforo

Considere a seguinte a definição de semáforo binário **S**. Um semáforo **S** é uma variável de valor inteiro, a qual pode tomar somente valores 0 e 1, e duas operações são definidas sobre **S**: ***wait(S)***, se **S > 0** então ***S = S-1*** e executa o processo/thread senão suspende a execução do processo/thread sobre **S**. O processo/thread é dito estar suspenso sobre **S,** aguardando numa fila em **S**. Na primitiva ***signal(S)***, se existem processos/threads que tenham sido suspensas sobre ***S***, então acorde um deles, senão ***S=S+1***.

Seja o pseudo-código seguinte:

```
S:  Semaphore := 1;

Thread  T1  is

begin
    loop
        Non_Critical_Section;
        wait(S);
        Critical_Section_1;
        signal(S);
```

```
        Non_Critical_Section;
    end loop;
End T1;


Thread T2 is
begin
    loop
        Non_Critical_Section;
        wait(S);
        Critical_Section_2;
        signal(S);
        Non_Critical_Section;
    end loop;
End                                                    T2;
```

Considerando níveis de prioridade de execução de 1 a 10 (1 é de menor prioridade e 10 a maior prioridade) para as threads T1 e T2 (Veja a figura em Deitel Java, Cap23). Em que cenário haverá ***starvation*** na concorrência entre as threads. Explique sua resposta.

4B Controle de Concorrência com Monitor

Usando uma pseudo-linguagem, construir um monitor chamado **emulação-semaforo**, que emule um semáforo representado por uma variável ***S***. O semáforo ***S*** é inicializado a um valor inteiro não-negativo. Deve existir uma variável do monitor chamada **semáforo_not_zero**, a qual mantém a fila de processos ou threads esperando para o semáforo ser não-zero. Os procedimentos do monitor são chamados **semaforo-wait** e **semaforo-signal**. Para emular um monitor usando semáforo, preencha os espaços com a especificação pesudo-código da definição de semáforo.

Monitor emulacao-semaforo

procedure semaforo-wait

end semaforo-wait

procedure semaforo-signal

end semaforo-signal

end emulacao-semaforo