Lista de Exercícios de Threads 2012.2

Professor: Eduardo Tavares e Carlos Ferraz
Monitores:
Erico Moutinho Medeiros
Guilherme Praciano Karst Caminha
Kleyton Lourenco Bezerra Leacarla
Luis Victor Ferreira de Carvalho
Marina Maria Austregesilo Saraiva
Pedro Henrique Dias da Silva
Pedro Louback Castilho
Rafael Teixeira Mendes de Carvalho
Rebeca Vasconcelos de Sa Alencar
Rodrigo Lopes de Carvalho
Thais Alves de Souza Melo
Thalles Cezar Rodrigues de Lima
Walter Sobral Andrade

A lista deverá ser respondida em dupla. Considera-se que uma lista na qual **menos que 8 das respostas** estão corretas não foi entregue. No caso, se forem entregues menos de 8 questões, a lista não valerá. **A questão 10 é obrigatória!** A entrega da lista fará parte da composição da terceira nota para os membros da dupla. Se qualquer situação de cópia de respostas for identificada, os membros de todas as duplas envolvidas **perderão 50% da terceira nota.** O mesmo vale para respostas obtídas a partir da Internet. As respostas deverão ser entregues exclusivamente em formato texto ASCII (nada de .pdf, .doc, .docx ou .odt) e deverão ser enviadas através do site da disciplina (cin.ufpe.br/~if677ec) na seção “Listas”, **até o dia 16/02/2013.** Devem ser organizadas em arquivos separados, um por questão, com o nome no formato “Q[número da questão].[c ou cpp]”, sem aspas. **Questões com o nome diferente deste formato não serão aceitas**.

1) Escreva um programa que realize o cálculo das somas dos valores das linhas de uma matriz qualquer de números inteiros e imprima o resultado na tela. Faça com que o cálculo do somatório de cada linha seja realizado em paralelo por uma thread.

2) Cinco lebres disputarão uma corrida. Cada lebre pode dar um salto que varia de 1 a 3 metros de distância. A distancia percorrida é de 20 metros. Na corrida, cada lebre dará um salto de comprimento aleatorio (dentro do intervalo permitido) e informará quantos metros ela pulou a cada salto realizado. Em seguida, a lebre pára para descansar, ficando parada enquanto as outras lebres saltam. Escreva um programa, utilizando threads (uma para cada lebre), que informe a lebre vencedora e a colocação de cada uma delas no final da corrida. Informar também quantos pulos cada uma delas deu.
Obs.: Só é permitido o uso de join() para thread main.

3) Implemente um programa que deve gerenciar a troca dos dados de um vetor de tamanho n (n<50) para uma lista encadeada feita por várias threads, preservando a ordem dos dados. Essa passagem

deve ser feita por (n = tamanho do vetor escolhido) threads, ressaltando que a ordem dos dados deve ser preservada.

Ex: [0][1][2][3][4] (vetor) ----> 0 - > 1 - > 2 - > 3 - > 4 (lista)

Obs: Para a lógica da questão os valores do vetor não precisam ser lidos do usuário, logo o vetor pode ser preenchido de qualquer forma.

4) Suponha que em uma cozinha exista apenas uma boca de fogão para
ser usado tanto pelas frigideiras quanto para panelas de barro. Para que não hajam
problemas, foram impostas algumas regras:

- Não podem haver panelas de barro e frigideiras no na boca de fogão ao mesmo tempo
- Não podem haver mais de três panelas ao mesmo tempo na boca do fogão

Utilizando semáforos, crie um programa que simule o uso da boca do fogão em um
ambiente com N frigideiras e M panelas de barro (N e M podem ser iguais ou diferentes),
de maneira que não haja starvation e/ou deadlock.
Imprima a quantidade de frigideiras ou panelas na boca do fogão cada vez que o valor das mesmas seja atualizado (cada vez que a quantidade mudar)

Exemplo:
A boca do fogão está com 1 frigideiras.
A boca do fogão está com 2 frigideiras.
A boca do fogão está com 1 panela.
A boca do fogão está com 2 panelas.

5) Implemente um programa utilizando Mutex, que:

* existem 3x reservas de um certo recurso com 10000 unidades(cada reserva) e elas são independentes, ou seja, uma thread que bloquear uma reserva nao pode bloquear as outras.
* Existem 10 threads(consumidoras desse recurso).

Seu programa deve mostrar quanto cada thread consumiu a cada instante que ela conseguir acessar a região critica.
ex:
Consumidor: X / Utilizou: Y recursos
Reserva B Possui C recursos.

*A thread deve consumir uma quantidade de recursos aleatória, sendo no máximo esse valor = 99.(caso o valor aleatorio seja maior que o existente na reserva deve-se pegar o valor que há na reserva).

O programa encerrará quando não existir mais recursos em nenhuma das reservas.
Ao Termino do programa deverá ser impresso o quanto cada consumidor utilizou no total.
Ex.:
Consumidor: X / Utilizou: Y recursos no total.

6) Campeonato de Carros

* Programa deve criar 10 threads (sendo 10 o número de competidores numerados de 1 a 10);
* Haverão M corridas(M>10);
* Apos cada corrida deve-se esperar o termino de todos os competidores;
* A competição se daria pela concorrencia entre as threads por acesso a regiao crítica. Assim que a thread conseguisse acessar a regiao critica receberia o ponto correspondente a sua posição;
* pontuação seria:
    1º = 10,
    2º = 9 ,
    3º = 8 ,
    ...
* ao temino do campeonato haveria 1 competidor campeão;
* Campeão o competidor que juntou o maior numero de pontos.

Obs.: no caso de empates o campeao seria decidido por ordem numérica (crescente)

7) [Teórica]
José, programador, estava desenvolvendo um servidor de chat para sua turma da faculdade e depois de dias de implementação, chegou o dia de liberar o programa para seus colegas. Mas, para sua surpresa, os seus amigos o informaram que as mensagens estavam chegando desorganizadas, sem fazer nenhum sentido. Ele achou estranho, porque quando testou o programa **sozinho** ele estava funcionando sem problemas.
Depois de muito pensar, ele suspeita que os problemas estão relacionados ao uso de concorrencia no processamento das mensagens no núcleo do servidor, usando uma thread para cada usuário do chat.
O comportamento do servidor e dos clientes está descrito no pseudo-código seguinte:

Clientes:
    espera um caractere do teclado
    envia o caractere para o servidor
    repete.

Threads de cada cliente no servidor:
    espera um caractere do cliente pela rede.
    imprime esse caractere no buffer principal.
    espera outro caractere.

Thread principal do servidor:
    Se há caracteres no buffer principal, mande ele para todos os usuarios conectados
    repita.

assumindo que todo caractere consegue ser escrito no buffer, responda:
    1 - Porque as mensagens estão chegando fora de ordem aos amigos de José?
    2 - Qual seria um modo de resolver este problema?

8) Triópolis é uma cidade pequena onde a única maneira de acessar a internet é na lan house Trinet. Essa lan house possui apenas 3 computadores e uma grande sala de espera que suporta 15 pessoas.

Cada computador fica suspenso se não houver nenhum cliente utilizando-o. Quando aparecer alguém, o computador nao utilizado a mais tempo é pego por esse cliente. Após o uso do computador, o cliente libera o computador e vai embora. Depois que um computador é utilizado, o próximo cliente na sala de espera é chamado para acessar. Se não houver clientes na sala de espera, o computador permanece desligado. Quando o cliente chega na lan house e houver computadores livre, ele deve utilizar o que está desligado a mais tempo. Se não houver computadores livres, o cliente sentará na sala de espera, aguardando sua vez. Se a sala de espera estiver cheia, o cliente desiste e vai embora. Os clientes são educados e respeitam a ordem de chegada, portanto quem estiver sentado a mais tempo é o próximo a utilizar um computador.

Você deve escrever um programa para simular a atividade na lan house. Cada computador e cada cliente deve ser simulado como uma thread separada. A entrada será o número de clientes que irão entrar na lan house. A chegada de cada cliente deve ocorrer aleatoriamente a cada 10, 20, 30, 40 ou 50 ms. Da mesma forma, cada computador só pode ser utilizado por 80 ou 90 ms.

Deverá ser informado: a chegada ou saída de um cliente; a saída imediata de um cliente (se a sala de espera estiver lotada); o ínicio e o término do uso de cada computador (e qual cliente irá utilizar cada computador); o término das atividades da lan house (quando o último cliente for embora).

*Proibido o uso de espera ocupada nas threads dos clientes.

9) Crie um programa que simule o funcionamento do processo de construção de carros. O processo é dividido em três partes: a construção das peças, a montagem do carro, e o teste dos carros. Os robôs construtores enviam as peças prontas para a área de montagem em uma esteira que suportam 30 conjuntos de peças no total. A montadora recebe as peças, os robôs montadores montam o carro (lembrem-se que um carro não pode ser montado se não houver peças na esteira) e o enviam pronto para o centro de teste, que libera o carro após ser testado. A garagem do centro de testes suporta apenas 20 carros e existem vários pilotos de teste. Todos os robôs construtores constroem a mesma quantidade de conjuntos de peças, a menos que a quantidade de conjuntos a serem construídos dividida pela quantidade de robôs construtores dê resto diferente de zero. Nesse caso o primeiro robô vai construir (qtd_de_conjuntos/qtd_de_construtores + qtd_de_conjuntos%qtd_de_construtores) e todos os outros vão construir apenas (qtd_de_conjuntos/qtd_de_construtores). O mesmo acontece com os robôs montadores e os pilotos de teste. Lembre-se que os robôs montadores, os robôs construtores e os pilotos são independentes entre si, mas apenas um robô pode acessar a esteira num determinado momento assim como apenas um robô montador ou um piloto pode acessar a garagem num determinado momento.

As entradas do programa serão a quantidade de conjuntos de peças que devem ser produzidos, a quantidade de robôs construtores, a quantidade de robôs montadores e a quantidade de pilotos de teste.

O seu programa deve imprimir na tela toda vez que um robô construtor, robô montador ou piloto de testes tentar exercer sua função. Deve imprimir também a quantidade de conjunto de peças/carros que estão na esteira e na garagem toda vez que essas quantidades são alteradas. E também deve imprimir quando um construtor, montador ou piloto terminar o trabalho.

Anexo: Exemplo de IN/OUT

10) Para receber o passaporte, um requerente deve ir a um posto da Polícia Federal

com os documentos necessários. Os requerentes devem entrar no posto, entregar os documentos, esperar para receber o passaporte e sair. Em algum momento,o encarregado entra no posto, autoriza a emissão de todos os passaportes, entrega-os aos requerentes e sai. Entretanto, o encarregado não passa o tempo inteiro no posto e por isso existem algumas regras.

Uma vez que o encarregado entre, nenhum novo requerente pode entrar. O oficial precisa esperar até que todos os requerentes entreguem os documentos para autorizar a emissão. Todos os passaportes são autorizados de uma única vez. Os requerentes saem apenas quando recebem o passaporte. O encarregado apenas pode sair ao entregar os passaportes de todos os requerentes que estão no posto e todos os que receberam passaporte saírem. Quando o encarregado sai, podem entrar novos requerentes.

PS: O programa deve rodar indefinidamente.

11) Há um processador com 2 núcleos e 7 processos a serem executados. Todos os processos tem mesma prioridade e basta haver um núcleo livre para que execute. Ao término de uma execução, os outros processos em espera são avisados e podem utilizar o núcleo vago. Cada processo leva uma quantidade de tempo X para executar por completo e o processador possui um quantum Q, que é o tempo que cada processo pode utilizar o processador. A cada execução do programa, é subtraído de seu tempo total de execução o valor do quantum e, caso o tempo restante para execução seja menor que esse quantum, o processo executa apenas pelo tempo que lhe resta.
Desenvolva um programa que utilize threads para simular o comportamento desse sistema. O seu programa deve encerrar quando todos os processos terminarem suas execuções por completo e imprimir na tela o tempo que cada processo passou em espera e e também o tempo que cada núcleo passou processando, ambos em ms.
A entrada consistirá de um inteiro X que representará o quantum (em ms), seguido de 7 inteiros que serão os tempos de execução relativos a cada processo (também em ms). Ao executar em determinado núcleo, o processo deve imprimir na tela “Processo P executando no núcleo N” e permanecer no processador durante o seu tempo de execução.

12) Está acontecendo uma competição de programação (para programadores extremamente rápidos e que nunca erram), em que N (>=2) times, cada um com 3 integrantes, devem resolver uma quantidade Y (>0) de questões. Os valores de N e Y ficam a critério do aluno.

Considerações:
- Cada time só possui 1 computador disponível. Ou seja, os integrantes devem revezar-se para usá-lo.
- A resolução de cada questão ocorre INDIVIDUALMENTE em 2 etapas. Entendimento e implementação.(Ou seja, um unico integrante deve entender e implementar uma questão INDIVIDUALMENTE).
//- O entendimento de uma questão dura o tempo necessário para que um contador, o qual representa a compreensão que o programador tem da questão, seja incrementado de 0 a 1000. Quando um programador começa a pensar, deve ser impressa a frase “Equipe X, integrante K, questão Z: Pensando”. Quando um programador compreende a questão, deve ser impressa a frase “Equipe X, integrante K, questão Z: Compreendida”.
//- Cada membro do time quando estiver ocupando o computador, programa a ques-

//tão que foi lida/entendida no tempo necessário para que um contador, o qual representa o progresso da implementação, seja incrementado de 0 a 10000. Quando um programador começa a implementar, deve ser impressa a frase “Equipe X, Integrante K , Questão Z: Implementando” , e, ao final da implementação, deve ser impressa a frase “Equipe X, Integrante K, Questão Z: Implementada”. Nesse momento, o programador vai ler outra questão, liberando o computador para outro membro do time.
- Assim que um time resolver as Y questões a competição será interrompida. Imprima o ranking dos times em ordem decrescente do número de questões que cada um resolveu.

13) Analise o código em anexo e identifique possíveis deadlocks. Corrija-os e descreva (em forma de comentário) como o programa sem correção poderia entrar em deadlock. (a resposta deve ser entregue em um arquivo .c)

14) Futebol de Sabão

Joãzinho e seus amigos adoram Futebol de Sabão. Cada amigo de Joãozinho é uma thread. A bola sempre começa com Joãozinho e cada jogador só pode dar 1 toque na bola. É sabido que se 5 jogadores do mesmo time tocam na bola consecutivamente um gol acontecerá. Também é sabido que se 2 jogadores do mesmo time tocam na bola 3 vezes alternadamente, também é gol. A partida só acaba quando um time faz 500 gols a mais do que outro. Imprima placar do jogo a cada gol indicando o tipo do mesmo, se por 5 toques ou tabelinha.

Obs.: 2 jogadores não tocam na bola ao mesmo tempo.
O número total de jogadores é variável, porém tem que ser par e maior que 4. (Solicitar esse valor ao usuário)

15)Iterações do Matlab

Um aluno do CIn, cansado de procurar computadores livres nos grads e se frustrar ao ver que os mesmos tinham sempre o aviso “Rodando iterações do MATLAB, favor não deslogar”, decidiu tentar otimizar os cálculos do MATLAB, trocando as funções iterativas(somatório, produtório, etc) do código em C do MATLAB por uma própria sua.

Identificar o que deveria ser mudado foi fácil:
implementar uma função chamada Fold, com assinatura:
float Fold(int **i**, int **n**, float (***f**)(int) , float (***op**)(float,float), float **first**);

onde, dado uma sequencia A = (a_1,a_2,a_3,...,a_n):
i -> valor inicial do iterador
n -> valor final do iterador
f -> ponteiro para função que, dado um numero x, retorna o valor a_x da sequencia A.
op -> ponteiro para função que dado dois valores *a* e *b* , retornem um unico valor. Essa função será aplicada a todos os elementos de A.
first -> primeiro valor a ser usado na função *op* com o primeiro elemento da sequencia A. Normalmente é a identidade da operação *op* (0 para soma, 1 para multiplicação)

como exemplo

$\sum_{i=1}^{5} i^2$
 	
seria escrita:
float soma(float a,float b){
return a+b;
}
 	
float seq(int x){
return pow((float)x,2);
}
 	
Fold(1,5,&seq,&soma,0)
	$\prod_{i=2}^{20} i! * 2^i$
	
	seria escrita:
	float mult(float a,float b){
	return a*b;
	}
	
	float seq2(int x){
	return fact(x)*pow(2.0,x);
	}
	
	Fold(2,20,&seq2,&mult,1)

Mas após testar sua implementação, o aluno viu que não houve mudança de eficiência, e ainda iria demorar horas esperando os experimentos rodarem.
Foi quando ele resolveu pesquisar, e descobriu que ele poderia acelerar o processo usando threads.
Como ele era um simples calouro, passou a tarefa para os alunos de Infra de Software, que estão mais aptos a essa tarefa.

A sua tarefa é implementar a mesma função Fold, com mesma assinatura, mas que rode as chamadas à função *f* em paralelo.

Ponteiros para função são declarados na forma:
tipo_de_retorno *(nome_da_variavel) (lista de tipos dos parametros);

para atribuir uma função ao ponteiro, se faz:
ponteiro = &funçao; OU **ponteiro = funçao;**
(tanto faz, pois o endereço de uma função é o proprio endereço do começo de seu código)

para usar um ponteiro de função, chamamos ele como uma função:
ponteiro(parametros);

referência:
Learn C++ - 7.8 - Function Pointers
C Programming Tutorial - Function Pointers