2024_2 - FUNDAMENTOS DE SISTEMAS PARALELOS E DISTRIBUÍDOS - METATURMA

Primeiro exercício de programação: threads movendo-se em trios (v3)

Feito: Ver

Exercício individual

Prazo de entrega: confira o moodle, no link de entrega

Introdução

Neste trabalho vamos implementar um sistema de sincronização entre threads que se assemelha a um sistema de controle de movimentação para um jogo digital, ou para o controle de robôs.

Objetivo

Seu objetivo será criar um programa em C (ou C++ simples), com Pthreads, que deve criar um certo número de threads que seguirão, cada uma, um trajeto definido por um conjunto de salas. Cada thread visitará um certo número de salas e deverá seguir certas regras no seu movimento.

O princípio de operação

A especificação das salas, do número de threads e dos seus trajetos será fornecida através da entrada padrão, como linhas de texto. Cada thread representará uma entidade se movendo com certas restrições sobre quando sua entrada em uma sala é permitida. Ao conseguir entrar, cada thread deve passar um certo tempo ali (definido nos dados do trajeto) antes de se mover para a próxima sala. Depois de ficar o tempo determinado na última posição do seu trajeto, cada thread deve sair da última sala e terminar.

O tempo mínimo a ser gasto em cada sala será fornecido em décimos de segundo; ao se estabelecer em uma certa posição, a thread deve executar a função **`passa_tempo()`** que deve ser incluída no código exatamente como a seguir (código disponível no moodle):

```
-------------
#include <time.h>

void passa_tempo(int tid, int sala, int decimos)
{
    struct timespec zzz, agora;
    static struct timespec inicio = {0,0};
    int tstamp;

    if ((inicio.tv_sec == 0)&&(inicio.tv_nsec == 0)) {
        clock_gettime(CLOCK_REALTIME,&inicio);
    }

    zzz.tv_sec  = decimos/10;
    zzz.tv_nsec = (decimos%10) * 100L * 1000000L;

    if (sala==0) {
        nanosleep(&zzz,NULL);
        return;
    }

    clock_gettime(CLOCK_REALTIME,&agora);
    tstamp = ( 10 * agora.tv_sec  +  agora.tv_nsec / 100000000L )
            -( 10 * inicio.tv_sec + inicio.tv_nsec / 100000000L );

    printf("%3d [ %2d @%2d z%4d\n",tstamp,tid,sala,decimos);

    nanosleep(&zzz,NULL);

    clock_gettime(CLOCK_REALTIME,&agora);
    tstamp = ( 10 * agora.tv_sec  +  agora.tv_nsec / 100000000L )
            -( 10 * inicio.tv_sec + inicio.tv_nsec / 100000000L );

    printf("%3d ) %2d @%2d\n",tstamp,tid,sala);
}
-------------
```

O código apresentado para a função passa_tempo fará com que a thread seja suspensa pelo tempo indicado. Para todos os efeitos de sincronização, seria o mesmo que se ela estivesse

executando qualquer tipo de operação computacionalmente intensiva, porém sem ocupar a CPU durante esse período. Isso permite que os testes sejam independentes do desempenho da CPU utilizada.

Especificação do problema

O formato da entrada (que deve ser lida da entrada padrão), será o seguinte:

- dois inteiros, representando o número de salas, S, e o número de threads, T;
- para cada thread:
  - um inteiro com o identificador da thread;
  - um inteiro indicando um tempo inicial, em décimos de segundos, que a thread deve esperar antes de iniciar o trajeto;
  - um inteiro com o número de salas que devem ser visitadas pela thread em seu trajeto;
  - para cada posição:
    - um inteiro identificador de uma sala;
    - um inteiro com o tempo mínimo, em décimos de segundo, a permanecer naquela posição.

Sobre o formato da entrada, valem as seguintes regras:

- o espaçamento entre valores pode ser qualquer: podem haver vários valores em uma linha, um por linha, etc.;
- o número de espaços entre valores também pode variar;
- o número máximo de salas será 10 e o número máximo de threads será o triplo do número de salas;
- os identificadores de threads estarão dentro do intervalo [0..1000];
- as salas serão numeradas sequencialmente de 1 a N;
- o comprimento máximo de um trajeto será 2 x S;
- cada sala pode ser visitada inúmeras vezes, mas nunca duas vezes em seguida (isto é, ao sair da sala x a thread não pode visitar x imediatamente);
- o tempo de permanência estará sempre no intervalo [1..1000].

O formato da entrada será garantido sem erros (não é preciso incluir código para verificar se os valores seguem a especificação). Não há nenhuma restrição quanto à relação entre uma sala e a próxima no trajeto: cada posição no trajeto pode ser qualquer. Além disso, é garantido que todo arquivo usado no processo de avaliação terá uma combinação de threads e trajetos que fará com que todas as threads terminem o trajeto durante a execução (nenhum arquivo fará com que threads fiquem paradas indefinidamente em uma posição no meio do trajeto por não conseguirem atender as restrições de movimento para continuar).

Um exemplo de arquivo de entrada será apresentado ao final desta página.

Movimento pelo trajeto

Cada thread, ao ser criada, deve receber seu identificador, o tempo de espera inicial e o trajeto a ser seguido. Ela inicia "fora" do tabuleiro e deve, após o tempo de espera, se mover pelo trajeto, pelas salas indicadas. Ao terminar seu tempo de permanência na última casa do trajeto, a thread deve sair do tabuleiro (deixar a última casa visitada). Além da garantia já fornecida de

que todos os movimentos na descrição do trajeto são válidos, a única regra a ser observada é:

> *Estando em uma sala S1 e devendo se mover para a sala S2, uma thread só pode realmente se entrar em S2 quando a sala estiver vazia e a thread fizer parte de um trio de threads, as quais entrarão na sala "juntas" (ao mesmo tempo).* Em outras palavras, se a posição P2 estiver ocupada por qualquer thread de um trio anterior, a thread que deseja se mover deve permanecer em S1 até que o movimento seja possível.
>
> Se várias threads estiverem bloqueadas tentando se mover para uma certa sala já ocupada, não há uma ordem garantida para elas se moverem, exceto pela restrição acima. Isto é, se uma sala fica vazia e há quatro threads querendo entrar, quaisquer três (das quatro) pode compor um trio e entrar na sala).
>
> Obviamente, a entrada na primeira sala não está associada à saída de outra sala; da mesma forma, a saída da última sala não implica na entrada de outra sala.

Sugestão: cada thread, ao ser criada, deve receber um espaço onde os demais dados da thread (lidos da entrada) serão armazenados. Isso pode ser, por exemplo, um espaço em um vetor global com uma posição para cada thread possível (para simplificar a implementação).

Cada thread, depois de criada e de obter seus dados de identificação e trajeto, deve executar a seguinte sequência de operações:

1. obtém o tempo de espera inicial antes de iniciar o trajeto;
2. executa a função passa_tempo com aquele tempo passando 0 para a sala;
3. **enquanto** existe próxima sala S' que pode ser retirada do trajeto
4. **quando** as regras o permitirem (forma trio, sala está vazia),
5. entra na próxima sala;
6. libera a posição anterior (se existir);
7. executa a função passa_tempo com o tempo associado à posição S';
8. fim quando
9. fim enquanto
10. libera a posição da última sala.

Note que uma thread só sai realmente de uma posição quando puder entrar na próxima, exceto se estiver na última sala.

Detalhamento da sincronização

Em síntese, o objetivo principal deste exercício, do ponto de vista da disciplina, é a criação e controle de um grupo de threads, que deverão acessar algumas das salas disponíveis de forma sincronizada, segundo a regra de movimentação apresentada. Essa sincronização deve ser implementada usando variáveis de exclusão mútua e de condição (**uma solução com semáforos não será aceita nesse caso**).

Uma possível solução poderia usar operações denominadas **entra**/**sai** que receberiam como parâmetros o número de uma sala e outros parâmetros que sejam necessários. No seu movimento de S1 para S2, bastaria a uma thread executar a sequência `entra(S2,...); sai(S1,...);` sendo que as primitivas de sincronização estariam dento das funções `entra()` e `sai()`. Ao entrar na primeira posição, obviamente, a função **sai()** não pode ser chamada, pois não há posição anterior nesse caso.

A função **passa_tempo** deve ser chamada assim que a thread já tiver entrado na posição e liberado a posição anterior (se for o caso).

Sobre a execução do programa:

Seu programa deve ler da entrada padrão (stdin em C, cin em C++) e escrever na saída padrão (stdout, cout). Ele não deve receber parâmetros de linha de comando. Não é preciso testar por erros na entrada, mas seu programa deve funcionar com qualquer combinação válida.

A única saída esperada para seu programa será gerada pelos comandos printf dentro da função passa_tempo. Caso você inclua mensagens de depuração ou outras informações que sejam exibidas durante a execução, certifique-se de removê-las da saída na versão final a ser submetida para avaliação.

O código deve usar apenas C ou C++ padrão, sem bibliotecas além das consideradas padrão. O paralelismo de threads deve ser implementado usando POSIX threads (Pthreads) apenas (em C++, não é permitido usar os recursos de threads específicos da linguagem).

O que deve ser entregue:

Você deve entregar um arquivo .zip contendo os seguintes elementos:

- código fonte do programa final produzido em C/C++, devidamente comentado para destacar as decisões de implementação;
- makefile com as seguintes regras: **`clean`** (remove todos os executáveis, .o e outros temporários), **`build`** (compila e gera o executável), e **`run`** (executa o comando criado) - o comportamento default deve ser **`build`**.

Note que não é preciso incluir um relatório em PDF, mas você deve documentar as principais decisões de projeto com um código bem comentado - em particular, explique como foi implementada a sincronização entre as threads.

O material desenvolvido por você deve executar sem erros nas máquinas linux do laboratório de graduação. A correção será feita naquelas máquinas e programas que não compilarem, não seguirem as determinações quanto ao makefile, formato da entrada e da saída, ou apresentarem erros durante a execução, serão desconsiderados. Em particular, não confie que seu código testado em windows com MinGW ou WSL vai funcionar exatamente da mesma forma naquelas máquinas. **TESTE LÁ**.

Preste atenção nos prazos: entregas com atraso serão aceitas por um ou dois dias, mas serão penalizadas.

Sugestões de depuração

Faz parte do exercício desenvolver os casos de teste para o mesmo. Não serão fornecidos arquivos de teste além da entrada descrita a seguir. Vocês devem criar os seus próprios arquivos e podem compartilhá-los no fórum do exercício. Por outro lado, obviamente não é permitido discutir o princípio de sincronização da solução.

Referências úteis

- **Pthreads**

A referência básica sobre pthreads é o tutorial do Lawrence Livermore Labs: https://hpc-tutorials.llnl.gov/posix/. O material necessário para o exercício vai até a seção 8 do turorial.

- **Como construir um makefile corretamente**

Para minha enorme surpresa, pelo visto o comando make não é mais conhecido por todos os alunos de computação. Eu sei, pode parecer pré-histórico, mas é uma ferramenta simples e eficiente para automação de processos de construção de executáveis (build). Grandes ambientes de desenvolvimento têm suas próprias soluções, mas para projetos menores make ainda resolve!

A entrega de um make que facilite minha vida na avaliação automática é obrigatória. Entregas que não sigam as recomendações fornecidas, gerem mensagens ou comandos extras, etc., serão (ligeiramente) penalizadas. Sendo assim, melhor aprender a usar o make corretamente! :-)

Esta é uma boa referência com todos os recursos do make. Para este exercício, vocês devem precisar apenas das seções 2, 2.1 e 2.2. Não vão além disso por enquanto, para ganhar tempo.

Dúvidas?

Usem o fórum criado especialmente para esse exercício de programação para enviar suas dúvidas. **Não é permitido publicar código no fórum**! Se você tem uma dúvida que envolve explicitamente um trecho de código, envie mensagem por e-mail diretamente para o professor.

Exemplo de entrada:

Segundo o formato descrito, as linhas a seguir definiriam um conjunto de três salas e cinco threads, sendo as threads se encontram em momentos diferentes na entrada das três salas. Os trajetos são feitos de forma que ao final, todas as threads terminem seus trajetos, saindo da última sala que visitarem.

```
------------
3 5
10 10 4
    1 40
    2 10
    1 20
    3 20
20 20 3
    1 30
    2 20
    3 40
30 30 1
    1 80
40 40 2
    1 10
    3 30
50 50 2
    2 30
    1 30
------------
```

Exemplo de saída:

Não executei o programa, mas a saída para essa entrada, em uma máquina com utilização baixa, deve ser algo mais ou menos como a lista a seguir. O tempo total de execução deveria ser na casa de 18 segundos. Mudanças nesse tempo total podem indicar erros na sincronização (mas pequenas variações nos valores dos timestamps são possíveis).

As linhas consecutivas marcadas com asteriscos (*) podem aparecer em ordens diferentes.

```
------------
 30 [ 10 @ 1 z 40  *
 30 [ 20 @ 1 z 30  *
 30 [ 30 @ 1 z 80  *
 60 ) 20 @ 1
 70 ) 10 @ 1
 70 [ 10 @ 2 z 10  *
 70 [ 20 @ 2 z 20  *
 70 [ 50 @ 2 z 30  *
 80 ) 10 @ 2
 90 ) 20 @ 2
100 ) 50 @ 2
110 ) 30 @ 1
110 [ 10 @ 1 z 20  *
110 [ 40 @ 1 z 10  *
110 [ 50 @ 1 z 30  *
120 ) 40 @ 1
130 ) 10 @ 1
130 [ 10 @ 3 z 20  *
130 [ 20 @ 3 z 40  *
130 [ 40 @ 3 z 30  *
140 ) 50 @ 1
150 ) 10 @ 3
160 ) 40 @ 3
170 ) 20 @ 3
------------
```

◄ Programação do curso

Seguir para...

Arquivo com o código da função passa_tempo (v2) ►