2023_1 - FUNDAMENTOS DE SISTEMAS PARALELOS E DISTRIBUÍDOS - METATURMA

Primeiro exercício de programação: threads trabalhando em trios (trabalho individual)

Marcar como feito

Exercício individual

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

Introdução

Neste trabalho vamos implementar um sistema de sincronização entre threads que controlará o trabalho das mesmas em grupos de três.

Objetivo

Seu objetivo será criar um programa em C, com Pthreads, que deve criar um certo número de threads, que serão atribuídas cada uma a um grupo, entre três possíveis. Seu mecanismo de sincronização, ao ser chamado, permitirá que threads só possam prosseguir quando houver uma thread de cada tipo tentando formar um trio e só um trio pode existir a cada momento. Se mais threads tentarem formar um trio quando um já exista, elas devem esperar.

O princípio de operação

Ao iniciar, o programa deverá ler da entrada padrão a descrição de um certo número de threads que deverão ser criadas imediatamente, cada uma recebendo um identificador e um tipo/grupo. Identificadores serão inteiros definidos na faixa 1..1000 e haverá apenas três grupos possíveis, identificados pelos inteiros 1, 2 e 3. Cada thread deverá primeiro passar algum tempo executando um trabalho individual (sem sincronização) e depois deverá passar outro tempo trabalhando em um trio. Depois de terminar seu trabalho no trio, cada thread termina. Para isso, além do identificador e do grupo, cada thread receberá dois inteiros que determinarão a quantidade de trabalho/tempo que elas deverão executar, sozinhas (tsolo) e em trio (ttrio), em décimos de segundo.

Seu código deve definir uma abstração de sincronização cujo tipo será denominado trio_t e que executará as seguintes operações:

- void init_trio(trio_t* t) - inicializa a estrutura de dados que você definir para representar o trio;
- void trio_enter(trio_t* t,int my_type) - uma thread pede para entrar em um trio, indicando seu tipo/grupo (my_type);
- void trio_leave(trio_t* t,int my_type) - uma thread que já está no trio sinaliza que completou sua parte do trabalho no trio.

Como mencionado anteriormente, apenas três threads podem entrar em um trio de cada vez e elas devem ser uma de cada tipo. Uma thread que pede para entrar no trio antes que existam outras duas dos dois outros dois tipos esperando para entrar deve esperar até que as demais chamem trio_enter. Se houver mais de uma thread de um certo tipo querendo entrar no trio, ela deve esperar até que o trio atual termine - note que não basta esperar que a thread do mesmo tipo termine! Um novo trio só pode ser formado e liberado para continuar depois que todas as threads do trio anterior deixem o trio.

Os arquivos spend_time.h e spend_time.c são fornecidos. Com aquela função, se uma thread tem seu identificador, tipo, tempo individual e tempo em trio nas variáveis tid, ttype, tsolo e ttrio, respectivamente, o corpo de cada thread deveria conter os comandos:

```
// No início da thread, ela deve preencher tid, ttype, tsolo e ttrio
//    a partir dos parâmetros fornecidos na chamada pthread_create.
  spend_time(tid,ttype,"S",tsolo);
  trio_enter(&trio,ttype);
  spend_time(tid,ttype,"T",ttrio);
  trio_leave(&trio,ttype);
  pthread_exit();
```

Os strings "S" e "T" não devem ser alterados, pois eles serão conferidos na correção automática

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.

Entrada do problema

O formato de entrada (que deve ser lida da entrada padrão, stdin), é bastante simples: para cada thread, serão fornecidos quatro inteiros representando, respectivamente, o identificador da thread (entre 1 e 1000), seu grupo (1, 2 ou 3), o tempo que ela deve executar antes de tentar formar um trio e o tempo que deve executar dentro do trio. A lista de threads termina quando o programa encontrar o fim de arquivo. Haverá sempre o mesmo número de threads criadas de cada tipo. Sendo assim, o número de threads criadas pode variar, mas como não serão criadas duas threads com um mesmo identificador, o limite será 999 threads, formando 333 trios.

A leitura deve ser feita da entrada padrão usando scanf ou uma solução equivalente. O programa não deve receber nenhum parâmetro da linha de comando e não deve abrir nenhum arquivo explicitamente (basta usar a entrada padrão). Certifique-se de entender o comportamento de scanf ao ler um marcador de fim de arquivo e como gerar esse marcador se estiver executando o programa diretamente da linha de comando, sem redirecionamento.

O formato da entrada será garantido sem erros (não é preciso incluir código para verificar se os valores seguem a especificação). Também será garantido que haverá um número igual de threads de cada tipo/grupo.

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

Detalhamento da sincronização

Abstração adotada: 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 o trio de forma sincronizada, segundo a regra de apresentada. Essa sincronização deve ser implementada usando variáveis de exclusão mútua e de condição (uma solução com semáforos nesse caso seria mais complexa e será penalizada se for usada).

Garantia de ordem: uma thread de um determinado tipo que chame trio_enter quando não há outras threads do mesmo tipo já esperando para formar um trio deve estar no primeiro trio que for formado. Outras threads do mesmo tipo que chamem trio_enter não podem passar na frente da primeira, mas não há garantia de ordem entre elas (isto é, se três threads do tipo 1 chegam antes que o primeiro trio seja formado, a primeira thread a chegar estará obrigatoriamente no primeiro trio, mas a terceira pode estar no segundo trio e a segunda, no terceiro, ou o contrário).

Uso controlado de travas: existem diversas soluções possíveis para esse problema. Entretanto, a **sua solução só deve manter variáveis de exclusão mútua travadas dentro das funções de entrada e saída** - isto é, toda mutex travada dentro de uma função enter/leave deve ser destravada antes do retorno daquela função. Soluções que não obedeçam esse requisito serão penalizadas.

Sobre a execução do programa:

Como mencionado, seu programa deve ler da entrada padrão e escrever na saída padrão. 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 spend_time. Caso você inclua mensagens de depuração ou outras informações que sejam exibidas durante a execução, certifique-se de removê-las da saída na execução da versão final.

O código deve usar apenas C padrão (não C++), sem bibliotecas além das consideradas padrão. O paralelismo de threads deve ser implementado usando POSIX threads (Pthreads) apenas.

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 formato da entrada e da saída, ou apresentarem erros durante a execução, serão desconsiderados.

O que deve ser entregue:

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

- código fonte do programa final produzido em C, devidamente comentado para destacar as decisões de implementação;
- makefile com as seguintes regras: `clean` (remove todos os executáveis, .o e outros temporários) e `build` (compila e gera o executável com nome **`ex1`**) - o comportamento default deve ser `build`;
- relatório em PDF.

Os arquivos spend_time.[ch] não podem ser alterados e não precisam ser incluídos na entrega (eles serão de qualquer forma substituídos pelos originais antes da correção). Novamente, o material desenvolvido por você deve executar sem erros nas máquinas linux do laboratório de graduação.

O relatório não precisa ser longo, mas deve documentar as principais decisões de projeto consideradas - em particular, como foi implementada a sincronização entre as threads.

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.

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 definido, as linhas a seguir definem 6 threads, duas de cada tipo.

```
-------------
11 1 10 20
12 2 10 10
13 3 10 20
21 1 15 10
22 2 15 20
23 3 15 10
-------------
```

Não executei o programa ainda, mas a saída para essa entrada, em uma máquina que não esteja sobrecarregada, deve ser a como a lista a seguir. As linhas com um mesmo timestamp (o primeiro inteiro) podem aparecer em ordens diferentes. O tempo total de execução deveria ser da casa de 5 segundos. Mudanças nesse tempo total podem indicar erros na sincronização (mas pequenas variações são possíveis).

```
-------------
0:11:1:S+(10)
0:12:2:S+(10)
0:13:3:S+(10)
0:21:1:S+(15)
0:22:2:S+(15)
0:23:3:S+(15)
10:11:1:S-
10:12:2:S-
10:13:3:S-
10:11:1:T+(20)
10:12:2:T+(10)
10:13:3:T+(20)
15:21:1:S-
15:22:2:S-
15:23:3:S-
20:12:2:T-
30:11:1:T-
30:13:3:T-
30:21:1:T+(10)
30:22:2:T+(20)
30:23:3:T+(10)
40:21:1:T-
40:23:3:T-
50:22:2:T-

------------
```

(No momento, essa saída foi gerada manualmente, sem a execução do programa - isto é, podem haver erros.)

◄ Dúvidas e curiosidades sobre a disciplina

Seguir para...

Código auxiliar para o primeiro exercício de programação (spend_time.[ch]) ►