UFMG

Exercício de Programação 1

Threads trabalhando em trios

Arthur Pontes Nader - 2019022294

1) Introdução

Os sistemas paralelos são de grande importância atualmente devido ao crescente aumento da quantidade de dados a serem processados nas mais diversas áreas. Nesses sistemas, tarefas independentes podem ser executadas ao mesmo momento em diferentes processadores, sendo assim possível reduzir o tempo de execução total.

Dessa maneira, o objetivo desse exercício de programação é implementar um sistema de sincronização entre threads que controlará o trabalho das mesmas em grupos de três. Para isso, será utilizado a linguagem C e a biblioteca Pthreads.

2) Implementação

a) Tipos de dados

- **struct thread_params:** esse tipo de dado guarda 4 inteiros relacionados com os parâmetros de execução da thread, sendo eles o id da thread, seu tipo, o seu tempo de execução sozinha e o seu tempo de execução em trio.

- **struct trio_t:** já esse tipo de dado é responsável pela lógica da execução em trio. É composto por um vetor de 3 posições chamado tipos_de_thread. Se o valor guardado em uma posição for 0, indica que ainda não há thread do tipo correspondente ao índice+1 no trio. Se for 1, já há uma thread do tipo no trio.
  O inteiro threads_no_trio guarda o número de threads que já estão no trio.
  Já a mutex é para garantir que as operações realizadas sobre o trio sejam feitas em exclusão mútua, evitando assim condições de corrida.
  Dois tipos de variáveis de condição são usadas. A variável wait é para uma thread esperar pelo trio ser completado antes de continuar a execução, enquanto o vetor chamado cond, composto por 3 variáveis de condição, é usado para que uma thread espere um trio que já contenha uma thread do mesmo tipo que o seu acabar de executar para tentar acessar o trio.

b) Funções

- **init_trio:** a função inicia as variáveis de condição e a mutex. A variável threads_no_trio e o vetor tipos_de_thread são iniciadas com valores 0.

- **trio_enter:** primeiramente, trava-se a mutex. Em seguida, verifica-se se já existe uma thread do mesmo tipo no trio. Se houver, a thread atual fica em espera até que o trio termine de executar. Caso contrário, incrementa o array tipos_de_thread na posição correspondente ao tipo da thread atual e a variável threads_no_trio.
  Depois, a função verifica se o trio está completo, ou seja, se há pelo menos uma thread de cada tipo. Se ainda não estiver completo, a thread atual fica em espera. Quando o trio estiver completo, a última thread a entrar acorda as threads que estavam esperando, de tal forma que o trio está liberado para continuar a execução. Libera-se a mutex.

- **trio_leave:** primeiro, trava-se a mutex. Em seguida, decrementa-se threads_no_trio. Se a thread for a última a sair do trio, ela envia um sinal para todas as três variáveis de condição (uma para cada tipo de thread). Assim, forma-se um novo trio. Ao final da função, a mutex é liberada.

- **iniciar_thread:** já essa função é usada na criação da thread. Ela recebe como parâmetro uma struct com a descrição da thread. Essa struct tem seus elementos usados nas funções spend_time, trio_enter e trio_leave. Ao final pthread_exit(NULL) indica que a thread terminou sua execução.

- **main:** após iniciar a variável global trio, usá-se “scanf” em loop para ler e guardar os parâmetros de execução de todas as threads. Assim, com esses parâmetros devidamente guardados em um array, cria-se cada uma das threads usando a função pthread_create. Em seguida, usa-se pthread_join para esperar que cada uma das threads termine de executar. Ao final, libera-se toda a memória alocada.

3) Resultados

Os testes realizados apresentaram resultados tal qual esperado. Variou-se o número de threads, o tempo de execução sozinha e o tempo de execução em trio para se observar situações mais complexas que envolvessem a sincronização entre as threads e como isso afetaria o desempenho geral do sistema.

4) Conclusão

A implementação da solução para o problema proposto foi uma boa maneira de se fixar os conceitos de sistemas paralelos abordados durante a disciplina. Programar pensando de forma paralela foi uma novidade no curso, que até então possuía trabalhos práticos com lógica totalmente sequencial.