Curso: C++ Moderno
Período: 1º semestre/2017
Aluno: Arthur Nunes de Paiva Santos Queiroz

Lista de exercícios Módulo 11: Concorrência e multithreading

Questões

Questão 1

Implemente uma versão paralela do quicksort que ordena as subdivisões (dos dois lados do pivot) da entrada via std::async(). Sua assinatura é std::list<int>par_quicksort(std::list<int>& in).
Lembre-se que você pode utilizar std::partition( ) para a partição do pivot.

main.cpp

```
#include <iostream>
#include <vector>

#include <algorithm>
#include <future>

// https://stackoverflow.com/a/28682886/702828
template <class RandomIt>
void quicksort(RandomIt b, RandomIt e) {
        size_t d = std::distance(b, e);

   if (d > 2) {
        auto mid_value = *(b + d/2);

        RandomIt pivot = b + (d)/2;

           std::iter_swap(pivot, e-1); // save pivot.
           pivot = std::partition(b, e-1,  [e] (const typename RandomIt::value_type& x) { return x < *(e-1); });
           std::iter_swap(pivot, e-1); // restore pivot.

     quicksort(b, pivot);
     quicksort(pivot+1, e);
   }
}

template <class RandomIt>
void par_quicksort(RandomIt b, RandomIt e) {
        size_t d = std::distance(b, e);

   if (d > 2) {
        auto mid_value = *(b + d/2);

        RandomIt pivot = b + (d)/2;

           std::iter_swap(pivot, e-1); // save pivot.
           pivot = std::partition(b, e-1,  [e] (const typename RandomIt::value_type& x) { return x < *(e-1); });
           std::iter_swap(pivot, e-1); // restore pivot.
```

```
        auto t1 = std::async([b, pivot] () { par_quicksort(b, pivot); } );
        auto t2 = std::async([e, pivot] () { par_quicksort(pivot+1, e); } );

        t1.get();
        t2.get();
    }
}

template< class ForwardIteratorType >
void print_it( ForwardIteratorType begin, ForwardIteratorType end ){
    while(begin != end)
    {
            std::cout << *(begin) << "\t";
        ++begin;
    }
}

int main()
{
    std::vector<int> vData = { 5, 7, 8, 9, 1, 2 };

    par_quicksort(vData.begin(), vData.end());
    std::cout << "Sorted vector: \t"; print_it(vData.begin(), vData.end()); std::cout << std::endl;

    return 0;
}
```

Questão 2

Implemente um processador de mensagens simples.
a) A função main() cria uma std::thread que invoca a post( ).
b) post( ) cria várias "mensagens", em um loop infinito que dorme por 1 segundo a cada iteração, e as posta em um std::deque<std::packaged_task<void(int)>>.
c) Cada task é uma função que retorna void e recebe um id. Essa função deve dormir por valor aleatório (use rand( )) entre 1 e 10 segundos, e, em seguida, imprimir 0 id na tela.
d) Apos ter criado a thread post( ), main( ) cria agora uma std::thread que invoca process( ).
e) process( ) realiza um loop, enquanto houver mensagens, retirando-as da fila e invocando-as.
Nota 1: Proteja a std::queue com um std::mutex e utilize std::lock_guards internamente em post() e process( ).
Nota 2: 0s temporizadores utilizados nesse exercício tem objetivo de simular o ambiente concorrente.

main.cpp

```
#include <iostream>
#include <vector>
#include <deque>

#include <algorithm>
#include <future>
#include <thread>
#include <chrono>
#include <random>
#include <mutex>

#include <signal.h>

volatile sig_atomic_t continue_flag = 1;
void termination_handler(int sig){ // can be called asynchronously
    continue_flag = 0; // set flag
    std::cout << "Termination requested." << std::endl;
}

std::deque<std::packaged_task<void()>> message_queue;
std::mutex m;
```

```
float randf(float a, float b) {
    float random = ((float) rand()) / (float) RAND_MAX;
    float diff = b - a;
    float r = random * diff;
    return a + r;
}

void post(){
    int i = 0;
    while(continue_flag){
        std::packaged_task<void()> task( [i] () {
            auto duration = std::chrono::duration<double>(randf(0.5, 5.0));
            std::cout << "Id " << i << ": sleeping for " << duration.count() << std::endl;
            std::this_thread::sleep_for(duration);
            std::cout << "Id: " << i << std::endl;
        });

        {
            std::lock_guard<std::mutex> lock(m);
            std::cout << "Adding message..." << std::endl;
            message_queue.push_back(std::move(task));
        }

        i++;

        std::cout << "Post sleeping..." << std::endl;
        std::this_thread::sleep_for(std::chrono::seconds(1));
        std::cout << "Post woke!" << std::endl;
    }
}

void proccess(){
    int i = 0;
    while(continue_flag){
        {
            std::lock_guard<std::mutex> lock(m);
            if(!message_queue.empty()){
                std::cout << "Processing message..." << std::endl;
                auto& task = message_queue.front();
                auto future = task.get_future();
                task(); // This blocks!!!
                        // https://stackoverflow.com/questions/18143661/what-is-the-difference-between-packaged-task-and-
async
                message_queue.pop_front();
                std::cout << "Processed message!" << std::endl;
            }
        }

        std::this_thread::sleep_for(std::chrono::milliseconds(10));

    }
}

int main()
{
    // Register signals
    signal(SIGINT, termination_handler);
    signal(SIGTERM, termination_handler);

    // post();
    std::thread t1(&post);
    std::thread t2(&proccess);
    t1.join();
    t2.join();

    return 0;
}
```

Questão 3

Quando lidamos com variaveis de condição, o mutex protege duas entidades. Quais são elas?

Questão 4

Por quê std::condition_variable::wait( ) recebe uma função de condição como parâmetro?

Porque, em certos sistemas, a forma como o mecanismo de *wake up* é implementada permite a ocorrência de *spurious wake up*, onde a thread pode ser acordada por um motivo qualquer, e não só ao ser notificada.

Questão 5

Reimplemente o exemplo produtor/consumidor mostrado nos slides, substituindo o bool ready por um std: :atomic<bool>. (Apenas o sincronismo é relevante, não se preocupe as partes // . . .).

main.cpp

```
std::queue<int> q;
std::atomic<bool> done(false);

void writer()
{
   // ...
   q.push(10);
   done = true;
}

void reader(){
   while(!done){
      // ...
   }
   int v = q.front();
   q.pop();
}
```