Conceitos Básicos de C++

Programação Multithread em C++

Introdução

A programação multithread em C++ permite a execução de múltiplas threads em paralelo, o que pode melhorar significativamente o desempenho de programas que realizam tarefas simultâneas. Esta seção aborda os conceitos e práticas essenciais para a programação multithread em C++.

1. Criando Threads

A biblioteca padrão C++ fornece a classe std::thread para criar e gerenciar threads.

Exemplo:

```
#include <thread>

void tarefa() {
    // código a ser executado pela thread
}

int main() {
    std::thread t(tarefa); // cria e inicia a thread
    t.join(); // aguarda a conclusão da thread
    return 0;
}
```

Conceitos Básicos de C++

2. Sincronização de Threads

Para evitar condições de corrida e outros problemas de concorrência, C++ oferece várias ferramentas de sincronização.

Mutex

Definição: Um mutex é usado para proteger seções críticas do código, garantindo que apenas uma thread possa acessar a seção crítica por vez.

Exemplo:

```
#include <mutex>
std::mutex mtx;

void tarefa() {
    std::lock_guard<std::mutex> guard(mtx);
    // seção crítica
}
```

Condition Variable

Definição: Usada para bloquear uma thread até que uma condição específica seja satisfeita.

Exemplo:

```
#include <condition_variable>
std::condition_variable cv;
std::mutex mtx;
```

```
bool pronto = false;


void espera() {
    std::unique_lock<std::mutex> lock(mtx);
    cv.wait(lock, []{ return pronto; });
    // continuar execução
}


void sinaliza() {
    std::lock_guard<std::mutex> guard(mtx);
    pronto = true;
    cv.notify_one();
}
```

Future e Promise

Definição: Usados para sincronizar operações assíncronas, permitindo que uma thread espere por um valor resultante de uma operação realizada por outra thread.

Exemplo:

```
#include <future>


int calcula() {
    return 42;
}
```

```
int main() {
    std::future<int> fut = std::async(calcula);
    int resultado = fut.get(); // aguarda o resultado
    return 0;
}
```

3. Gerenciamento de Threads

Join

Definição: Bloqueia a execução até que a thread associada termine.

Exemplo:

```
std::thread t(tarefa);
t.join();
```

Detach

Definição: Separa a thread do objeto std::thread, permitindo que continue a execução independentemente.

Exemplo:

```
std::thread t(tarefa);
t.detach();
```

4. Práticas de Programação Multithread

Evitar Deadlocks: Certifique-se de adquirir e liberar locks na mesma ordem para evitar deadlocks.

Minimizar Seções Críticas: Mantenha as seções críticas tão curtas quanto possível para reduzir a contenção.

Usar Ferramentas de Alto Nível: Prefira std::lock_guard e std::unique_lock para gerenciar mutexes.

Testar Concorretemente: Teste o código em ambientes com diferentes cargas e configurações de hardware para garantir a robustez.

Dicas de Boas Práticas

- Sincronização: Utilize as ferramentas de sincronização apropriadas para garantir a segurança dos dados.
- Performance: Equilibre o uso de threads para melhorar a performance sem causar sobrecarga.
- Legibilidade: Escreva código claro e documentado para facilitar a manutenção e entendimento.
- Análise de Concorrência: Utilize ferramentas de análise para identificar e resolver problemas de concorrência.

Esta seção abrange os conceitos sobre programação multithread em C++. Para mais detalhes, consulte a documentação oficial: https://en.cppreference.com/w/cpp/language/multithread