Simple Multithreading Using C++

计53 唐适之

1. POSIX thread

1. POSIX thread

- Fork / Join threads

```
#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);

int pthread_join(pthread_t thread, void **retval);
```

Compile and link with -pthread.

1. POSIX thread

- Mutual Exclusion Lock

```
#include <pthread.h>

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

int pthread_mutex_init(
            pthread_mutex_t* mutex,
            const pthread_mutexattr_t* attr );

int pthread_mutex_lock( pthread_mutex_t* mutex );

int pthread_mutex_unlock( pthread_mutex_t* mutex );
```

Compile and link with -pthread.

1. POSIX thread

- Example:
- Calculate (1+2+…+100) and (100+101+…+200) parallel.

- Output should be concurrent.
- Or use return value.

```cpp
#include <iostream>
#include <pthread.h>

struct Param
{
    int a, b;
    Param(int _a, int _b)
        : a(_a), b(_b) {}
};

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void *plus(void *param)
{
    int a = ((Param*)param)->a;
    int b = ((Param*)param)->b;
    int ret(0);
    for (int i=a; i<=b; i++)
        ret += i;
    pthread_mutex_lock(&lock);
    std::cout << ret << std::endl;
    pthread_mutex_unlock(&lock);
    return 0;
}

int main()
{
    pthread_t thread1, thread2;
    Param *p1 = new Param(1,100);
    Param *p2 = new Param(100,200);
    pthread_create(&thread1, NULL, plus, p1);
    pthread_create(&thread2, NULL, plus, p2);
    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);
    delete p1;
    delete p2;
    return 0;
}
```

1. POSIX thread

- Low level
- C style API
- Only work on POSIX platforms (including Linux, OSX)

How to encapsulate it?

2. C++11 thread

2. C++11 thread

- Fork / Join threads

```
class thread
{
    template <class Fn, class... Args>
    explicit thread (Fn&& fn, Args&&... args);

    void join();

    // and more...
};
```

Still need –pthread on POSIX platform.

2. C++11 thread

- Mutual Exclusion Lock

```
class mutex
{
    void lock();

    void unlock();

    // and more...
};
```

Still need –pthread on POSIX platform.

2. C++11 thread

- Mutual Exclusion Lock

```
template <class Mutex> class lock_guard;
```

Still need –pthread on POSIX platform.

2. C++11 thread

- The same example

```cpp
#include <thread>
#include <mutex>
#include <iostream>

std::mutex lock;

void plus(int a, int b)
{
    int ret(0);
    for (int i=a; i<=b; i++)
        ret += i;
    lock.lock();
    std::cout << ret << std::endl;
    lock.unlock();
}

int main()
{
    std::thread thread1(plus, 1, 100),
                thread2(plus, 100, 200);
    thread1.join();
    thread2.join();
    return 0;
}
```

2. C++11 thread

- Well encapsulated, but still need to manually control threads.
- Cross platform, but need different makefile.

3. OpenMP multithreading

3. OpenMP multithreading

- Example first

```
#include <iostream>
#include <omp.h>

omp_lock_t lock;

void plus(int a, int b)
{
    int ret(0);
    for (int i=a; i<=b; i++)
        ret += i;
    omp_set_lock(&lock);
    std::cout << ret << std::endl;
    omp_unset_lock(&lock);
}

int main()
{
    omp_init_lock(&lock);
#pragma omp parallel
    {
#pragma omp sections
        {
#pragma omp section
            plus(1, 100);
#pragma omp section
            plus(100, 200);
        }
    }
    return 0;
}

```

3. OpenMP multithreading

- Or simpler…

```cpp
#include <iostream>
#include <omp.h>

omp_lock_t lock;

void plus(int a, int b)
{
    int ret(0);
    for (int i=a; i<=b; i++)
        ret += i;
    omp_set_lock(&lock);
    std::cout << ret << std::endl;
    omp_unset_lock(&lock);
}

int main()
{
    omp_init_lock(&lock);
    int st[] = {1, 10000};
    int en[] = {10000, 20000};
#pragma omp parallel for
    for (int i=0; i<2; i++)
        plus(st[i], en[i]);
    return 0;
}
```

3. OpenMP multithreading

```
#pragma omp parallel

#pragma omp sections

#pragma omp section


#pragma omp parallel for
```

Compile and link with -fopenmp.

3. OpenMP multithreading

```c
#include <omp.h>

void omp_init_lock(omp_lock_t *lock);

void omp_set_lock(omp_lock_t *lock);

void omp_unset_lock(omp_lock_t *lock);
```

Compile and link with -fopenmp.

3. OpenMP multithreading

- Cross platform
- Support by most of C++ compilers
- If not using functions in “omp.h”, it will automatically compiled to be a synchronized program when not supported.
- Dynamically alter the number of threads.
- Easy to create many parallel tasks using for-loop.

Thanks