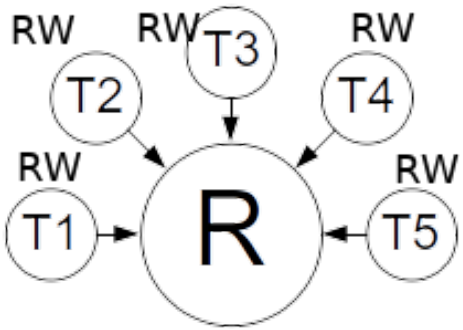
Лекция 25. C++1X: threads - II.

Евгений Линский

Общие переменные

Несколько потоков обращаются (читают и пишут) к общей переменной (объекту)



- Потоки могут читать и писать общую переменную
- Потоки могут работать параллельно на разных процессорах (ядрах)

mutex

```
class Stocks {
  list<int> l;
  std::mutex m;

  void fill(...) {
    m.lock();
    for(...) { l.insert(it);  ... }
    m.unlock();
  }

  void deplete(...) {
    m.lock();
    for(...) { l.erase(it); ... }
    m.unlock();
  }
};
```

main

```
Stocks s;
vector<thread> threads;
for(int i = 0; i < n/2; ++i){
    threads.push_back(thread([&s](){
            s.fill(...); ...
    }));
}
for(int i = 0; i < n/2; ++i){
    threads.push_back(thread([&s](){
            s.deplete(...); ...
    }));
}
```

mutex

Все хорошо?

```
class Stocks {
  list<int> l;
  std::mutex m;

  void calc(...) {
    if(...)  throw std::logic_error(...);
    ...
  }
  void fill(...) {
    m.lock();
    for(...) { l.insert(it);  ... }
    calc(...);
    m.unlock();
  }
  void deplete(...) {
    m.lock();
    for(...) { l.erase(it); ... }
    m.unlock();
  }
};
```

RAII: lock_guard

На предыдущем слайде: если сгенерируется исключение, то потоки будут вечно ждать, пока mutex освободится.

```
class Stocks {
  list<int> l; std::mutex m;

  void calc(...) {
    if(...)  throw std::logic_error(...);
    ...
  }
  void fill(...) {
    std::lock_guard<std::mutex> lg(m);
    for(...) { l.insert(it);  ... }
    calc(...);
  }
  void deplete(...) {
    std::lock_guard<std::mutex> lg(m);
    for(...) { l.erase(it); ... }
  }};
```

Добавить try/catch для обработки ошибки тоже было бы неплохо.

Deadlock (взаимоблокировка)

```
std::mutex a, b;
void f1() {
  a.lock();
  ...
  b.lock(); // < - 1
  ...
  b.unlock();
  ...
  a.unlock();
}
void f2() {
  b.lock();
  ...
  a.lock(); // <- 2
  ....
  a.unlock();
  ...
  b.unlock();
}
```

Поток T1 находится в точке 1. Поток T2 находится в точке 2.

Producers and consumers

- Потоки производители: складывают задания в очередь
- Потоки потребители: забирают задания из очереди и обрабатывают
- Очередь имеет фиксированный размер, чтобы не выесть все ресурсы
  - если очередь пуста — потребители ждут
  - если очередь заполнена — производители ждут

Пример:

- Производители обходят сайты, загружают html страницы и складывают их в очередь
- Потребители забирают страницы из очереди, очищают от html тэгов и строят поисковых индекс (например, выделяя ключевые слова)

BoundedBuffer

Циклический буфер фиксированного размера.

```
struct BoundedBuffer {
    int* buffer;
    int capacity;

    int begin; int end; int count;

    std::mutex lock;

    std::condition_variable not_full;
    std::condition_variable not_empty;

    BoundedBuffer(int capacity) :
        capacity(capacity), begin(0), end(0), count(0) {
        buffer = new int[capacity];
    }

    ~BoundedBuffer(){
        delete[] buffer;
    }
}
```

BoundedBuffer::deposit

```
void BoundedBuffer::deposit(int data){
    std::unique_lock<std::mutex> l(lock);

    // without cond_var:  while(count == capacity) {}
    not_full.wait(l, [this](){return count != capacity; });

    buffer[end] = data;
    end = (end + 1) % capacity;
    ++count;

    l.unlock();
    not_empty.notify_one();
}
```

- wait — освободил mutex и заснул (не тратит процессорные ресурсы), после notify проснулся, проверил условие и взял mutex.
- У std::unique_lock в отлиии от std::lock_guard есть метод unlock, который вызывается внутри wait.
- notify_one — разбудить одного любого (еще есть notify_all)

BoundedBuffer::deposit

```
int BoundedBuffer::fetch(){
    std::unique_lock<std::mutex> l(lock);

    // without cond_var:  while(count = 0) {}
    not_empty.wait(l, [this](){return count != 0; });

    int result = buffer[begin];
    begin = (begin + 1) % capacity;
    --count;

    l.unlock();
    not_full.notify_one();

    return result;
}
```

main

```
int main(){
    BoundedBuffer buffer(200);

    std::thread c1(consumer, 0, std::ref(buffer));
    std::thread c2(consumer, 1, std::ref(buffer));
    std::thread c3(consumer, 2, std::ref(buffer));
    std::thread p1(producer, 0, std::ref(buffer));
    std::thread p2(producer, 1, std::ref(buffer));

    c1.join();
    c2.join();
    c3.join();
    p1.join();
    p2.join();

    return 0;
}
```

producer

```
void producer(int id, BoundedBuffer& buffer){
    for(int i = 0; i < 75; ++i){
        buffer.deposit(i);
        std::cout << "Produced " << id
            << " produced " << i << std::endl;
        std::this_thread::sleep_for(
            std::chrono::milliseconds(100));
    }
}
```

consumer

```
void consumer(int id, BoundedBuffer& buffer){
    for(int i = 0; i < 50; ++i){
        int value = buffer.fetch();
        std::cout << "Consumer " << id
            << " fetched " << value << std::endl;
        std::this_thread::sleep_for(
            std::chrono::milliseconds(250));
    }
}
```