Condition Variable

План на сегодня

Научимся пользоваться `std::condition_variable`

Попробуем написать Semaphore

```
class Semaphore {
public:
  Semaphore(int n) : counter_(n) {}
  void Enter();
  void Leave();

private:
  int counter_ = 0;
};
```

- Корзина в которой лежит n токенов
- Перед входом в критическую секцию поток забирает токен
- После выхода -- возвращает токен
- Внутри секции может находиться не больше чем n потоков

Leave()

```
class Semaphore {
  std::mutex mutex_;
  int counter_ = 0;
};

void Semaphore::Leave() {
  std::unique_lock<std::mutex> guard(mutex_);
  counter_++;
}
```

Enter()

```
class Semaphore {
  std::mutex mutex_;
  int counter_ = 0;
};

void Semaphore::Enter() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ > 0) {
    counter_--;
    return;
  } else {
    // ??
  }
}
```

Встречайте `std::condition_variable`

```
class condition_variable {
public:
  void wait(); // не настоящая сигнатура
  void notify_one();
  void notify_all();
};
```

- Очередь потоков (не FIFO)
- Поток, делающий `wait()`, атомарно отпускает мьютекс, встаёт в очередь и засыпает
- Поток, делающий `notify_one()`, пробуждает один поток из очереди

Семафор - Попытка №1

```
class Semaphore {
  std::mutex mutex_;
  std::condition_variable not_zero_;
  int counter_ = 0;
};

void Semaphore::Leave() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ == 0) not_zero_.notify_one();
  counter_++;
}

void Semaphore::Enter() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ == 0) {
    not_zero_.wait(); // ???
  } else {
    // ...
  }
}
```

Семафор - Попытка №1

```
void Semaphore::Leave() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ == 0) not_zero_.notify_one();
  counter_++;
}

void Semaphore::Enter() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ == 0) {
    not_zero_.wait(); // <= DEADLOCK
  } else {
    // ...
  }
}
```

Семафор - Попытка №2

```
void Semaphore::Leave() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ == 0) not_zero_.notify_one();
  counter_++;
}

void Semaphore::Enter() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ == 0) {
    guard.unlock();
    not_zero_.wait(); // ???
    guard.lock();
  } else {
    // ...
  }
}
```

Семафор - Попытка №2

```
void Semaphore::Leave() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ == 0) not_zero_.notify_one();
  counter_++;
}

void Semaphore::Enter() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ == 0) {
    guard.unlock();
    not_zero_.wait(); // Lost Wakeup problem
    guard.lock();
  } else {
    // ...
  }
}
```

Что делать?

```
void deadlock() {
  std::unique_lock<std::mutex> guard(mutex_);
  not_zero_.wait();
}

void lost_wakeup() {
  std::unique_lock<std::mutex> guard(mutex_);
  guard.unlock();
  not_zero_.wait();
  guard.lock();
}
```

Семафор - Попытка №3

```
class condition_variable {
public:
  // атомарно отпускает mutex и встаёт в очередь
  void wait(std::unique_lock<std::mutex>&);
};

void Semaphore::Leave() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ == 0) not_zero_.notify_one();
  counter_++;
}

void Semaphore::Enter() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ == 0) {
    not_zero_.wait(guard);
  }

  counter_--;
}
```

Семафор - Попытка №3

```
void Semaphore::Leave() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ == 0) not_zero_.notify_one();
  counter_++;
}

void Semaphore::Enter() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ == 0) {
    not_zero_.wait(guard); // spurious wakeup
  }

  counter_--;
}
```

Семафор - Попытка №4

```
void Semaphore::Leave() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ == 0) not_zero_.notify_one();
  counter_++;
}

void Semaphore::Enter() {
  std::unique_lock<std::mutex> guard(mutex_);
  while (counter_ == 0) {
    not_zero_.wait(guard);
  }

  counter_--;
}
```

Семафор - Попытка №4

```
void Semaphore::Leave() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ == 0) not_zero_.notify_one();
  counter_++;
}

void Semaphore::Enter() {
  std::unique_lock<std::mutex> guard(mutex_);
  // even better
  not_zero_.wait(guard, [this] {
    return counter_ > 0;
  });

  counter_--;
}
```

Семафор - Попытка №5

```
void Semaphore::Leave() {
  std::unique_lock<std::mutex> guard(mutex_);
  if (counter_ == 0) not_zero_.notify_all();
  counter_++;
}

void Semaphore::Enter() {
  std::unique_lock<std::mutex> guard(mutex_);
  // even better
  not_zero_.wait(guard, [this] {
    return counter_ > 0;
  });

  counter_--;
}
```

Семафор - Попытка №5

```
void Semaphore::Leave() {
  std::unique_lock<std::mutex> guard(mutex_);
  // thundering herd problem
  if (counter_ == 0) not_zero_.notify_all();
  counter_++;
}

void Semaphore::Enter() {
  std::unique_lock<std::mutex> guard(mutex_);
  not_zero_.wait(guard, [this] {
    return counter_ > 0;
  });

  counter_--;
}
```

Семафор - Попытка №6

```
void Semaphore::Leave() {
  std::unique_lock<std::mutex> guard(mutex_);
  not_zero_.notify_one();
  counter_++;
}

void Semaphore::Enter() {
  std::unique_lock<std::mutex> guard(mutex_);
  not_zero_.wait(guard, [this] -> bool {
    return counter_ > 0;
  });

  counter_--;
}
```

Очень просто использовать `condition_variable` неправильно

Разблокируем слишком мало потоков

- Lost Wakeup

Разблокируем слишком много потоков

- Spurious Wakeup
- Thundering Herd

Доказательство корректности

Хотим доказать свойства

- В любой момент времени внутри критической секции находится не более n потоков (свойство безопасности — safety)
- Если начиная с некоторого момента `counter_ > 0`, то любой поток, пытающийся зайти в критическую секцию, рано и ли поздно в неё зайдёт (свойство живости — liveness)

- В любой момент времени внутри критической секции находится не более n потоков
- Если начиная с некоторого момента `counter_ > 0`, то любой поток, пытающийся зайти в критическую секцию, рано и ли поздно в неё зайдёт

```
void Semaphore::Leave() {
  std::unique_lock<std::mutex> guard(mutex_);
  not_zero_.notify_one();
  counter_++;
}
void Semaphore::Enter() {
  std::unique_lock<std::mutex> guard(mutex_);
  not_zero_.wait(guard, [this] -> bool {
    return counter_ > 0;
  });
  counter_--;
}
```

Как правильно использовать `std::condition_variable`

1. Понять что входит в состояние класса. Какие нужны очереди.
2. Для каждого метода определить предусловие на это состояние.
3. В начале каждого метода сделать `wait()` на CV, соответствующую предусловию.
4. В теле метода после каждого изменения состояния, которое могло привести к изменению предусловия, делать `notify_*`.
5. Стараться не делать лишние `notify_*`.

Разберём пример

```
class FixedQueue {
public:
  FixedQueue(size_t max_size);
  void Push(int elem);
  int Pop();
};
```