

**Комплект оценочных материалов по дисциплине
«Современные технологии параллельного программирования»**

Задания закрытого типа

Задания закрытого типа на выбор правильного ответа

1. Выберите правильный вариант инициализация потока объект-функцией:

```
class MyOperator
{
public:
    void operator () (int a[], int len) {
        for (int i = 0; i < len; i++) {
            a[i] *= 2;
        }
        for (size_t i = 0; i < len; ++i) {
            std::cout << "Element " << i << ": " << a[i] << std::endl;
        }
    }
};
```

- A) thread th1{ MyOperator::operator , obj, arr, length };
- Б) thread th1{ MyOperator::operator , &obj, &arr, length };
- В) thread th1{ &MyOperator::operator , &obj, arr, length };
- Г) thread th1{ MyOperator(), arr, length };

Правильный ответ: Г

Компетенции (индикаторы): ОПК-5, ПК-2

2. Выберите верный вариант вызова потока с помощью метода класса:

```
class X {
public:
    static void do_lengthy_work(int i, double d);
};
```

- A) X my_x; std::thread t(X::do_lengthy_work, my_x, 1, 2.65);
- Б) X my_x; std::thread t(&X::do_lengthy_work, my_x, 1, 2.65);
- В) X my_x; std::thread t(&X::do_lengthy_work, &my_x, 1, 2.65);
- Г) X my_x; std::thread t(&X::do_lengthy_work, 1, 2.65).

Правильный ответ: Г

Компетенции (индикаторы): ОПК-5, ПК-2

3. Как создать условную переменную в C++ с использованием стандартной библиотеки?

- A) std::condition_variable cond_var;

- Б) `condition_variable cond_var();`
- В) `std::mutex cond_var;`
- Г) `std::lock_guard<std::condition_variable> guard;`

Правильный ответ: А

Компетенции (индикаторы): ОПК-5, ПК-2

4. Какой метод условной переменной используется для блокировки потока и ожидания сигнала от другого потока?

- А) `notify_one()`
- Б) `wait()`
- В) `notify_all()`
- Г) `lock()`

Правильный ответ: Б

Компетенции (индикаторы): ОПК-5, ПК-2

5. Какой из следующих вариантов является правильным способом использования `std::promise` с `std::thread`?

- А) Передать `std::promise` в качестве аргумента функции потоков.
- Б) Возвращать `std::promise` из функции, исполняемой в потоке.
- В) Создать `std::promise` в основной функции и использовать его в потоке.

Правильный ответ: А, Б, В

Компетенции (индикаторы): ОПК-5, ПК-2

6. Что произойдет, если `std::promise` будет уничтожен до того, как его значение будет установлено?

- А) Программа завершится с ошибкой.
- Б) Значение будет установлено автоматически.
- В) Будет выброшено исключение `std::future_error`.
- Г) Ничего не произойдет; значение останется неопределенным.

Правильный ответ: В

Компетенции (индикаторы): ОПК-5, ПК-2

Задания закрытого типа на установление соответствия

1. Сопоставьте алгоритмы взаимного исключения с их описанием.

Алгоритм	Описание
1) <code>mutex</code>	А) обеспечивает базовую возможность взаимного исключения
2) <code>timed_mutex</code>	Б) обеспечивает возможность взаимного исключения, которая может быть заблокирована рекурсивно тем же потоком
3) <code>recursive_mutex</code>	В) обеспечивает возможность взаимного

- 4) recursive_timed_mutex Г) исключения, которая реализует блокировку с тайм-аутом обеспечивает возможность взаимного исключения, которая может быть заблокирована рекурсивно тем же потоком и реализует блокировку с тайм-аутом

Правильный ответ:

1	2	3	4
А	В	Б	Г

Компетенции (индикаторы): ОПК-5, ПК-2

2. Универсальное управление мьютексом. Сопоставьте классы с их описанием.

- | Класс | Описание |
|----------------|---|
| 1) lock_guard | А) представляет собой универсальную обертку владения общим мьютексом, позволяющую отложенную блокировку, временную блокировку и передачу владения блокировкой |
| 2) scoped_lock | Б) обеспечивает возможность блокировки нескольких мьютексов с использованием алгоритма предотвращения взаимоблокировок |
| 3) unique_lock | В) универсальная оболочка владения мьютексом, предоставляющая отсроченную блокировку, ограниченные по времени попытки блокировки, рекурсивную блокировку, передачу владения блокировкой и использование с condition variables |
| 4) shared_lock | Г) простой класс, конструктор которого вызывает метод lock для заданного объекта, а деструктор вызывает unlock |

Правильный ответ:

1	2	3	4
Г	Б	В	А

Компетенции (индикаторы): ОПК-5, ПК-2

3. Сопоставьте условные переменные с их описанием.

- | Переменная | Описание |
|------------------------------|---|
| 1) condition_variable | А) предоставляет переменную условия, связанную с любым типом блокировки |
| 2) condition_variable_any | Б) предоставляет переменную условия, связанную с std::unique_lock |
| 3) notify_all_at_thread_exit | В) перечисляет возможные результаты ожиданий по времени для условных переменных |
| 4) cv_status | Г) предоставляет механизм для |

уведомления других потоков о том, что данный поток полностью завершился, включая уничтожение всех объектов `thread_local`

Правильный ответ:

1	2	3	4
Б	А	Г	В

Компетенции (индикаторы): ОПК-5, ПК-2

Задания закрытого типа на установление правильной последовательности

1. Установите последовательность действий для корректного использования мьютекса для защиты общего ресурса от одновременного доступа нескольких потоков.

- А) создать объект `std::mutex`
- Б) в каждом потоке заблокировать мьютекс
- В) запустить потоки, которые используют общий ресурс
- Г) обработать общий ресурс
- Д) освободить мьютекс

Правильный ответ: А, В, Б, Г, Д

Компетенции (индикаторы): ОПК-5, ПК-2

2. Установите последовательность действий при использовании `std::atomic` для синхронизации модификации переменной в многопоточном контексте?

- А) дождаться завершения всех потоков
- Б) запустить потоки, которые будут модифицировать эту переменную
- В) объявить переменную типа `std::atomic`
- Г) обработать значение переменной после завершения потоков

Правильный ответ: В, Б, А, Г

Компетенции (индикаторы): ОПК-5, ПК-2

3. Установите правильный порядок выполнения шагов для создания и использования условной переменной:

- А) определить объект `std::condition_variable`.
- Б) зафиксировать мьютекс с помощью `std::unique_lock<std::mutex>`.
- В) поставить поток в состояние ожидания, вызвав `cv.wait(lock)`.
- Г) изменить состояние, которое проверяет ожидающий поток.
- Д) разбудить один или все потоки, ожидающие на условной переменной, вызвав `cv.notify_one()` или `cv.notify_all()`

Правильный ответ: А, Б, В, Г, Д

Компетенции (индикаторы): ОПК-5, ПК-2

4. Установите правильный порядок выполнения шагов для освобождения ресурсов после работы с условными переменными:

А) освободить мьютекс.

Б) закончить работу с разделяемым ресурсом.

В) вызвать `cv.notify_all()` (если необходимо) для пробуждения ожидающих потоков.

Г) протестировать состояние разделяемого ресурса перед его изменением.

Правильный ответ: Г, Б, В, А

Компетенции (индикаторы): ОПК-5, ПК-2

Задания открытого типа

Задания открытого типа на дополнение

1. Шаблон класса _____ оборачивает любую цель Callable (функцию, лямбда-выражение, связывающее выражение или другой объект функции) таким образом, чтобы её можно было вызывать асинхронно..

Правильный ответ: `std::packaged_task`

Компетенции (индикаторы): ОПК-5, ПК-2

2. Шаблон класса _____ предоставляет механизм доступа к результату асинхронных операций

Правильный ответ: `std::future` или `std::shared_future`

Компетенции (индикаторы): ОПК-5, ПК-2

3. Шаблон класса _____ позволяет сохранять значение или исключение, которые впоследствии извлекаются асинхронно с помощью объекта `std::future`, созданного объектом _____. Обратите внимание, что объект _____ предназначен для использования только один раз.

Правильный ответ: `std::promise`

Компетенции (индикаторы): ОПК-5, ПК-2

4. Защёлки и _____ – это механизмы координации потоков, которые позволяют любому количеству потоков блокироваться до тех пор, пока не прибудет ожидаемое количество потоков. Защёлку нельзя использовать повторно, а _____ можно использовать многократно..

Правильный ответ: барьер

Компетенции (индикаторы): ОПК-5, ПК-2

5. _____ — это примитив синхронизации, который позволяет нескольким потокам взаимодействовать друг с другом. Он позволяет

некоторому количеству потоков ожидать (возможно, с таймаутом) уведомления от другого потока о том, что они могут продолжить работу.
_____ всегда связана с мьютексом.

Правильный ответ: условная переменная
Компетенции (индикаторы): ОПК-5, ПК-2

Задания открытого типа с кратким свободным ответом

1. Какой метод условной переменной используется для пробуждения одного из ожидающих потоков?

Правильный ответ: `notify_one()`
Компетенции (индикаторы): ОПК-5, ПК-2

2. При доступе к памяти, операцию упорядочения захвата-освобождения выполняют:

Правильный ответ: `memory_order_consume`
Компетенции (индикаторы): ОПК-5, ПК-2

3. Какой метод `std::promise` используется для установки значения, которое будет передано в `std::future`?

Правильный ответ: `set_value()`
Компетенции (индикаторы): ОПК-5, ПК-2

4. Какое значение будет выведено в результате выполнения следующего кода?

```
#include <iostream>
#include <thread>
#include <mutex>
std::mutex mtx; // Создаем мьютекс
int shared_resource = 0; // Общий ресурс
void increment() {
    for (int i = 0; i < 1000; ++i) {
        mtx.lock(); // Захватываем мьютекс
        ++shared_resource; // Увеличиваем общий ресурс
        mtx.unlock(); // Освобождаем мьютекс
    }
}
void decrement() {
    for (int i = 0; i < 1000; ++i) {
        mtx.lock();
        --shared_resource;
        mtx.unlock();
    }
}
```

```

}
int main() {
    std::thread t1(increment);
    std::thread t2(decrement);

    t1.join();
    t2.join();

    std::cout << "Final value: " << shared_resource << std::endl;
    return 0;
}

```

Правильный ответ: 0

Компетенции (индикаторы): ОПК-5, ПК-2

Задания открытого типа с развернутым ответом

1. Привести пример использования условной переменной `<condition_variable>` для обеспечения корректной работы между потоками при использовании общих ресурсов.

Привести расширенное решение.

Время выполнения – 40 мин.

Ожидаемый результат:

```

#include <iostream>
#include <thread>
#include <mutex>
#include <condition_variable>
#include <vector>
class SharedData {
public:
    void produce(int value) {
        std::unique_lock<std::mutex> lock(mutex_);
        data_.push_back(value);
        std::cout << "Produced: " << value << std::endl;
        // Уведомляем другой поток о том, что данные доступны
        cond_var_.notify_one();
    }
    void consume() {
        std::unique_lock<std::mutex> lock(mutex_);
        // Ждем, пока не будет доступно хотя бы одно значение
        cond_var_.wait(lock, [this]() { return !data_.empty(); });
        int value = data_.back();
        data_.pop_back();
        std::cout << "Consumed: " << value << std::endl;
    }
};

```

```

    }
private:
    std::vector<int> data_;
    std::mutex mutex_;
    std::condition_variable cond_var_;
};
void producer(SharedData& shared) {
    for (int i = 0; i < 10; ++i) {
        shared.produce(i);
        std::this_thread::sleep_for(std::chrono::milliseconds(100)); //
Имитация задержки
    }
}
void consumer(SharedData& shared) {
    for (int i = 0; i < 10; ++i) {
        shared.consume();
        std::this_thread::sleep_for(std::chrono::milliseconds(150)); //
Имитация задержки
    }
}
int main() {
    SharedData shared;
    // Создаем потоки
    std::thread prod1(producer, std::ref(shared));
    std::thread cons1(consumer, std::ref(shared));
    // Ждем завершения потоков
    prod1.join();
    cons1.join();
    return 0;
}

```

Пояснение.

– SharedData: Это класс, который содержит общий ресурс (вектор данных) и механизмы синхронизации (мьютекс и условную переменную).

– produce(): Метод, который добавляет значение в вектор и уведомляет один из ожидающих потоков, что данные стали доступны.

– consume(): Метод, который ожидает наличия элементов в векторе. Если данных нет, поток блокируется с помощью wait(), пока другой поток не вызовет notify_one(). Этот метод блокирует поток до тех пор, пока условие в лямбда-функции не станет истинным (в данном случае, пока вектор не станет непустым).

– producer() и consumer(): Эти функции запускаются в отдельных потоках. Пример продюсера создает числа от 0 до 9 с задержкой, а консюмер ожидает и потребляет эти числа.

– main(): Создает и запускает потоки для производит и потребляет, затем ждет их завершения с помощью join().

Критерии оценивания:

– создание класса, который содержит общий ресурс (вектор данных) и механизмы синхронизации;

– создание метода, который добавляет значение в вектор и уведомляет один из ожидающих потоков, что данные стали доступны;

– создание метода, который ожидает наличия элементов в векторе;

– создание функций, которые запускаются в отдельных потоках;

– запуск потоков;

– вывод результата.

Компетенции (индикаторы): ОПК-5, ПК-2

2. Создать пример демонстрирующий основной принцип использования атомарных операций, для параллельного увеличения значение переменной без риска возникновения гонок данных.

Привести расширенное решение.

Время выполнения – 40 мин.

Ожидаемый результат:

```
#include <iostream>
```

```
#include <thread>
```

```
#include <atomic>
```

```
#include <vector>
```

```
std::atomic<int> counter(0); // Атомарная переменная для счетчика
```

```
void increment(int times) {
```

```
    for (int i = 0; i < times; ++i) {
```

```
        counter++; // Атомарное увеличение счетчика
```

```
    }
```

```
}
```

```
int main() {
```

```
    const int num_threads = 10; // Количество потоков
```

```
    const int increments_per_thread = 1000; // Количество инкрементов от
```

каждого потока

```
    std::vector<std::thread> threads;
```

```
    // Запуск потоков
```

```
    for (int i = 0; i < num_threads; ++i) {
```

```
        threads.emplace_back(increment, increments_per_thread);
```

```
    }
```

```
    // Ожидаем завершения всех потоков
```

```
    for (auto& thread : threads) {
```

```
        thread.join();
```

```
    }
```

```
    // Печать результата
```

```
std::cout << "Final counter value: " << counter.load() << std::endl; //  
Используем load для получения значения  
return 0;  
}
```

Пояснение.

Подключение библиотек:

`#include <iostream>` - для ввода и вывода.

`#include <thread>` - для работы с потоками.

`#include <atomic>` - для работы с атомарными типами.

`#include <vector>` - для хранения потоков.

Атомарная переменная:

– `std::atomic<int> counter(0);` – объявляем атомарный счетчик, который инициализируется значением 0.

Функция `increment`:

– эта функция увеличивает счетчик атомарным способом `counter++`; в цикле. Операция `++` для атомарных переменных не требует дополнительных механизмов синхронизации.

Создание потоков:

– в `main` создается вектор потоков `threads`.

– запускается 10 потоков, каждый из которых выполняет функцию `increment`, увеличивая счетчик 1000 раз.

Ожидание завершения потоков:

– `thread.join()` - позволяет основной программе дождаться завершения выполнения потока перед тем, как продолжить.

Вывод финального значения счетчика:

– используем `counter.load()` для безопасного получения значения счетчика.

Критерии оценивания:

– объявление атомарного счетчика;

– создание функции увеличения счетчика атомарным способом;

– создание функций, которые запускаются в отдельных потоках;

– запуск потоков;

– вывод результата.

Компетенции (индикаторы): ОПК-5, ПК-2

Экспертное заключение

Представленный комплект оценочных материалов по дисциплине «Современные технологии параллельного программирования» соответствует требованиям ФГОС ВО.

Предлагаемые оценочные материалы адекватны целям и задачам реализации основной профессиональной образовательной программы по направлению подготовки 09.04.03 Прикладная информатика.

Виды оценочных средств, включенные в представленный фонд, отвечают основным принципам формирования ФОС.


Разработанные и представленные для экспертизы оценочные материалы рекомендуются к использованию в процессе подготовки обучающихся по указанному направлению.

Председатель учебно-методической комиссии
института компьютерных систем
и информационных технологий



Ветрова Н.Н.

Лист изменений и дополнений

№ п/п	Виды дополнений и изменений	Дата и номер протокола заседания кафедры (кафедр), на котором были рассмотрены и одобрены изменения и дополнения	Подпись (с расшифровкой) заведующего кафедрой (заведующих кафедрами)
1.	Дополнен комплектом оценочных материалов	протокол заседания кафедры компьютерных систем и сетей № <u>8</u> от <u>10.03.2025</u>	 С.В. Попов