

**Комплект оценочных материалов по дисциплине
«Программирование для параллельных вычислений»**

Задания закрытого типа

Задания закрытого типа на выбор правильного ответа

1. Организация параллелизма за счет нескольких процессов подразумевает:

А) обмен данными средствами операционной системы через каналы межпроцессной коммуникации, процессы можно запускать на разных машинах, объединенных сетью;

Б) запуск нескольких потоков в одном процессе, процессы можно запускать на разных машинах, объединенных сетью;

В) осуществления коммуникации средствами операционной системы через каналы межпроцессной коммуникации, процессы нельзя запускать на разных машинах, объединенных сетью;

Г) все принадлежащие процессу потоки разделяют общее адресное пространство и имеют прямой доступ к большей части данных, процессы нельзя запускать на разных машинах, объединенных сетью.

Правильный ответ: А

Компетенции (индикаторы): ПК-3

2. Параллелизм за счет нескольких потоков подразумевает:

А) разбивка приложения на несколько однопоточных одновременно исполняемых процессов, обмен данными средствами операционной системы через каналы межпроцессной коммуникации;

Б) запуск нескольких потоков в одном процессе, обмен данных через разделяемую память;

В) разбивка приложения на несколько однопоточных одновременно исполняемых процессов, обмен данных через разделяемую память;

Г) разбивка приложения на несколько однопоточных одновременно исполняемых процессов, процессы можно запускать на разных машинах, объединенных сетью.

Правильный ответ: Б

Компетенции (индикаторы): ПК-3

3. При передаче аргументов функции в потоке параметры будут переданы:

```
void threadFunction(int i, double d, const  
std::string &s)
```

```
{
```

```
    std::cout << i << ", " << d << ", " << s
```

```

<< std::endl;
}
int main()
{
    std::thread thr(threadFunction, 1, 2.34,
"example");
    thr.join();
    return 0;
}

```

А) все по значению;

Б) первые два по значению, третий параметр по ссылке;

В) все по ссылке;

Г) неправильный формат передачи аргументов функции потока.

Правильный ответ: А

Компетенции (индикаторы): ПК-3

4. 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;
        }
    }
};

```

А) 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): ПК-3

5. В листинге ниже показан пример:

```

class X {
public:
    X(const X&) = delete;
    X& operator = (const X&) = delete;
// ...
};

```

- А) создание конструктора копирования;
- Б) создание в конструктора объекта;
- В) создание конструктора перемещения;
- Г) запрет копирования.

Правильный ответ: Г

Компетенции (индикаторы): ПК-3

6. В листинге ниже показан пример:

```
class Coord2D {  
public:  
    //  
    void operator= (Coord2 D&& obj) {  
        _x = obj._x;  
        _y= obj._y;  
    }  
private:  
    int  _x, _y;  
};
```

- А) конструктора копирования;
- Б) copy assignment;
- В) конструктора перемещения;
- Г) move assignment.

Правильный ответ: Б

Компетенции (индикаторы): ПК-3

7. В листинге ниже показан пример

```
class MyOperator  
{  
    void operator () ()  
    {  
    }  
};  
sf::Thread thread (MyOperator ());
```

- А) запуска потока с помощью non-member функции;
- Б) запуска потока с помощью метода класса;
- В) запуска потока с помощью функционального объекта;
- Г) запуска потока с помощью лямбда-функции.

Правильный ответ: В

Компетенции (индикаторы): ПК-3

8. Для возвращения результата работы функции в потоке (листинг показан ниже) необходимо:

```
int Sum(int a, int b)
{
    return a + b;
}
```

А) изменить функцию, чтобы она принимала аргументы по ссылке `int Sum(int &a, int &b)` и создать объект `thread th1(Sum, a, b);`

Б) передать аргументы в функцию по ссылке `thread th1(Sum, &a, &b);`

В) `thread th1([&result, a, b]() {result = Sum(a, b); });`

Г) варианты а и б – подходят оба.

Правильный ответ: В

Компетенции (индикаторы): ПК-3

9. Выберите верное утверждение:

А) `mutex`: имеет два метода `try_lock_for()` и `try_lock_until()`;

Б) `mutex`: может войти «сам в себя»;

В) `mutex`: обеспечивает базовые функции `lock()` и `unlock()` и не блокируемый метод `try_lock()`;

Г) `mutex`: это комбинация `timed_mutex` и `recursive_mutex`.

Правильный ответ: В

Компетенции (индикаторы): ПК-3

10. Перед обращением к общим данным, `mutex` должен быть:

А) заблокирован методом `lock()`, а после окончания работы с общими данными – разблокирован методом `unlock()`;

Б) заблокирован методом `try_lock()`, а после окончания работы с общими данными – разблокирован методом `try_unlock()`;

В) заблокирован методом `try_lock_for()`, а после окончания работы с общими данными – разблокирован методом `try_lock_until()`;

Г) присоединен к выполняемому потоку.

Правильный ответ: А

Компетенции (индикаторы): ПК-3

11. Выберите верное утверждение:

А) `lock_guard` поддерживает отложенную блокировку, рекурсивную блокировку и использование условных переменных;

Б) `unique_lock` поддерживает отложенную блокировку, рекурсивную блокировку и использование условных переменных;

В) `unique_lock` не поддерживает отложенную блокировку и рекурсивную блокировку;

Г) в `timed_mutex` не доступен метод `try_lock_for()`.

Правильный ответ: Б

Компетенции (индикаторы): ПК-3

12. Какие мьютексы, допускают несколько захватов в одном потоке:

А) recursive_mutex;

Б) mutex;

В) timed_mutex;

Г) все мьютексы допускают несколько захватов в одном потоке.

Правильный ответ: А

Компетенции (индикаторы): ПК-3

13. Эксклюзивный доступ (только один поток может владеть мьютексом) к ресурсу обеспечивает, только:

А) recursive_mutex;

Б) mutex и timed_mutex;

В) shared_mutex;

Г) все перечисленные выше мьютексы.

Правильный ответ: Г

Компетенции (индикаторы): ПК-3

14. Стандартная библиотека C++ предоставляет следующие реализации условных переменных:

А) только std::condition_variable;

Б) только std::condition_variable_any;

В) std::condition_variable и std::condition_variable_any;

Г) условную переменную можно объявлять без использования класса std::condition_variable.

Правильный ответ: В

Компетенции (индикаторы): ПК-3

15. В стандартной библиотеке C++ есть следующие разновидности будущих результатов, реализованные в шаблоне класса:

А) только std::future<>;

Б) только std::shared_future<>;

В) std::future<> и std::shared_future<>;

Г) std::thread<>.

Правильный ответ: В

Компетенции (индикаторы): ПК-3

16. Какой шаблон класса связывает будущий результат с функцией или объектом, допускающим вызов:

А) только std::future<>;

Б) только std::shared_future<>;

В) std::packaged_task<>;

Г) std::promise<>.

Правильный ответ: В

Компетенции (индикаторы): ПК-3

17. Какой механизм межпоточного взаимодействия позволяет передать результат выполнения потока (или исключение, выброшенное в потоке) в вызывающий поток:

- А) пара `promise-future`;
- Б) только `promise`;
- В) только `future`;
- Г) только `shared_future`.

Правильный ответ: А

Компетенции (индикаторы): ПК-3

18. Шаблон класса, используемый для получения результата асинхронно выполняемой задачи, которая еще не вычислена:

- А) `std::future<>`;
- Б) `std::packaged_task<>`;
- В) `std::thread<>`;
- Г) нет такого механизма.

Правильный ответ: А

Компетенции (индикаторы): ПК-3

19. При неявном преобразовании между типами интервалов:

- А) преобразовать часы в секунды можно, а секунды в часы нельзя;
- Б) преобразовать секунды в часы можно, а часы в секунды нельзя;
- В) можно преобразовывать часы в секунды и секунды в часы;
- Г) различные типы интервалов можно преобразовывать только явно (неявное преобразование использовать нельзя).

Правильный ответ: А

Компетенции (индикаторы): ПК-3

20. Выберите НЕверное утверждение:

- А) если один поток записывает в атомарный объект, а другой читает из него – поведение четко определено;
- Б) стандартные атомарные типы `std::atomic` не допускают копирование и перемещение в обычном смысле;
- В) при использовании атомарных операций необходимую синхронизацию обеспечивает компилятор;
- Г) `std::atomic_flag` можно сконструировать копированием из другого объекта, также разрешается присваивать один `std::atomic_flag` другому.

Правильный ответ: Г

Компетенции (индикаторы): ПК-3

Задания закрытого типа на установление соответствия

1. Сопоставьте термины и их определения:

- | Термины | Определения |
|------------------------|--|
| 1) Поток | А) Специальный объект для синхронизации потоков, который позволяет ограничить доступ к ресурсу. |
| 2) Мьютекс | Б) Поток, который выполняется параллельно с другими потоками. |
| 3) Состояние гонки | В) Ситуация, когда два или более потока пытаются одновременно изменить данные, что приводит к непредсказуемым результатам. |
| 4) Условная переменная | Г) Используется для ожидания определенного условия в многопоточной среде. |

Правильный ответ:

1	2	3	4
Б	А	В	Г

Компетенции (индикаторы): ПК-3

2. Сопоставьте функции библиотеки `<thread>` с их описанием.

- | Функция | Описание |
|-----------------------------|--|
| 1) <code>std::thread</code> | А) Метод, который позволяет потоку завершить свою работу и дождаться его завершения. |
| 2) <code>join()</code> | Б) Метод, который отделяет поток, позволяя ему работать независимо. |
| 3) <code>detach()</code> | В) Функция, которая создает новый поток, связывая его с указанной функцией. |
| 4) <code>joinable()</code> | Г) Метод, который проверяет, связан ли поток с активным потоком |

Правильный ответ:

1	2	3	4
Б	В	А	Г

Компетенции (индикаторы): ПК-3

3. Алгоритмы взаимного исключения не позволяют нескольким потокам одновременно получать доступ к общим ресурсам. Сопоставьте алгоритмы взаимного исключения с их описанием.

Алгоритм	Описание
1) mutex	А) обеспечивает совместное взаимное исключение
2) timed_mutex	Б) обеспечивает возможность взаимного исключения, которая может быть заблокирована рекурсивно тем же потоком
3) recursive_mutex	В) обеспечивает возможность взаимного исключения, которая реализует блокировку с тайм-аутом
4) recursive_timed_mutex	Г) обеспечивает возможность взаимного исключения, которая может быть заблокирована рекурсивно тем же потоком и реализует блокировку с тайм-аутом
5) shared_mutex	Д) обеспечивает базовую возможность взаимного исключения
6) shared_timed_mutex	Е) предоставляет общую возможность взаимного исключения и реализует блокировку с тайм-аутом

Правильный ответ:

1	2	3	4	5	6
Д	В	Б	Г	А	Е

Компетенции (индикаторы): ПК-3

4. 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) <code>unique_lock</code> | В) универсальная оболочка владения мьютексом, предоставляющая отсроченную блокировку, ограниченные по времени попытки блокировки, рекурсивную блокировку, передачу владения блокировкой и использование с <code>condition variables</code> |
| 4) <code>shared_lock</code> | Г) простой класс, конструктор которого вызывает метод <code>lock</code> для заданного объекта, а деструктор вызывает <code>unlock</code> |

Правильный ответ:

1	2	3	4
Г	Б	В	А

Компетенции (индикаторы): ПК-3

5. Сопоставьте условные переменные с их описанием.

- | Переменная | Описание |
|---|--|
| 1) <code>condition_variable</code> | А) предоставляет переменную условия, связанную с любым типом блокировки |
| 2) <code>condition_variable_any</code> | Б) предоставляет переменную условия, связанную с <code>std::unique_lock</code> |
| 3) <code>notify_all_at_thread_exit</code> | В) перечисляет возможные результаты ожиданий по времени для условных переменных |
| 4) <code>cv_status</code> | Г) предоставляет механизм для уведомления других потоков о том, что данный поток полностью завершился, включая уничтожение всех объектов <code>thread_local</code> |

Правильный ответ:

1	2	3	4
Б	А	Г	В

Компетенции (индикаторы): ПК-3

6. Стандартная библиотека предоставляет возможности для получения возвращаемых значений и перехвата исключений, которые выдаются асинхронными задачами. Сопоставьте классы с их описанием.

Класс	Описание
1) promise	А) сохраняет значение для асинхронного извлечения
2) future	Б) ждет значения, которое устанавливается асинхронно
3) shared_future	В) упаковывает функцию для хранения возвращаемого ею значения для асинхронного извлечения
4) packaged_task	Г) ожидает значения (возможно, на которое ссылаются другие фьючерсы), которое устанавливается асинхронно

Правильный ответ:

1	2	3	4
А	Б	Г	В

Компетенции (индикаторы): ПК-3

7. Сопоставьте тип флага с описанием.

Тип флага и операции	Описание
1) atomic_flag	А) атомарно устанавливает флаг TRUE и возвращает свое предыдущее значение
2) atomic_flag_test_and_set atomic_flag_test_and_set_explicit	Б) неблокируемый логический атомарный тип
3) atomic_flag_clear atomic_flag_clear_explicit	В) атомарно устанавливает значение флага FALSE
4) atomic_flag_wait atomic_flag_wait_explicit	Г) блокирует поток до получения уведомления и изменения флага
5) atomic_flag_notify_one	Д) уведомляет все потоки, заблокированные в atomic_flag_wait
6) atomic_flag_notify_all	Е) уведомляет о блокировке потока в atomic_flag_wait

Правильный ответ:

1	2	3	4	5	6
Б	А	В	Г	Е	Д

Компетенции (индикаторы): ПК-3

Задания закрытого типа на установление правильной последовательности

1. Установите правильную последовательность действий для создания и запуска нового потока с использованием `std::thread`

А) объявить переменную типа `std::thread`

Б) определить функцию, которую будет выполнять поток.

В) присвоить функции, которую будет выполнять поток, созданной переменной `std::thread`

Г) запустить поток

Д) дождаться завершения потока

Правильный ответ: Б, А, В, Г, Д

Компетенции (индикаторы): ПК-3

2. Установите правильную последовательность действий для передачи параметров в поток

А) дождаться завершения потока

Б) объявить переменную типа `std::thread`

В) передать функцию и параметры при создании потока

Г) запустить поток

Д) определить функцию с параметрами

Правильный ответ: Д, Б, В, Г, А

Компетенции (индикаторы): ПК-3

3. 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): ПК-3

4. Установите последовательность действий для корректного завершения работы потоков

А) создать потоки

Б) выполнить работу в потоках

В) дождаться завершения всех потоков с помощью метода `join()`

Г) завершить основную программу

Правильный ответ: А, Б, В, Г

Компетенции (индикаторы): ПК-3

5. 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): ПК-3

Задания открытого типа

Задания открытого типа на дополнение

1. _____ – встроенная функция в C++ `std::thread`. Это функция-наблюдатель, то есть она наблюдает за состоянием, а затем возвращает соответствующий вывод и проверяет, является ли объект потока присоединяемым или нет.

Правильный ответ: `joinable`
Компетенции (индикаторы): ПК-3

2. _____ (взаимное исключение) – это базовый механизм синхронизации. Он предназначен для организации взаимоисключающего доступа к общим данным для нескольких потоков с использованием барьеров памяти

Правильный ответ: Мьютекс
Компетенции (индикаторы): ПК-3

3. Чтобы задать ожидание в течение заданного интервала времени, используется функция _____

Правильный ответ: `std::chrono::duration()`
Компетенции (индикаторы): ПК-3

4. Операции, которые накладывают ограничения на порядок доступа к памяти без модификации данных называются _____.

Правильный ответ: барьерами
Компетенции (индикаторы): ПК-3

5. При использовании условных переменных, для обеспечения синхронизации необходимо взаимодействие с мьютексом. Класс условных переменных _____ может работать только с `std::mutex`.

Правильный ответ: `std::condition_variable`
Компетенции (индикаторы): ПК-3

6. Простейший стандартный атомарный тип _____ представляет булевский флаг. Объекты этого типа могут находиться в одном из двух состояний: установлен или сброшен.

Правильный ответ: `std::atomic_flag`

Компетенции (индикаторы): ПК-3

7. Все стандартные атомарные типы определены в заголовке `< _____ >`

Правильный ответ: `atomic`

Компетенции (индикаторы): ПК-3

8. _____ – это свойство процесса, означающее, что он совершается за один шаг или операцию.

Правильный ответ: Атомарность

Компетенции (индикаторы): ПК-3

Задания открытого типа с кратким свободным ответом

1. Что будет выведено на консоль?

```
#include <iostream>
#include <thread>
void threadCallback(int const& x)
{
    int& y = const_cast<int&>(x);
    y++;
}
int main()
{
    int x = 9;
    std::thread threadObj(threadCallback, x);
    threadObj.join();
    std::cout << x << std::endl;
    return 0;
}
```

Правильный ответ: 9

Компетенции (индикаторы): ПК-3

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

Правильный ответ: `memory_order_consume`

Компетенции (индикаторы): ПК-3

3. Чтобы запустить поток в фоновом режиме, следует вызвать функцию:

Правильный ответ: `detach()`

Компетенции (индикаторы): ПК-3

4. Для получения числа потоков, которые могут работать параллельно, используют функцию:

Правильный ответ: `std::thread::hardware_concurrency()`

Компетенции (индикаторы): ПК-3

Задания открытого типа с развернутым ответом

1. Выполнить возврата результата из потока с использованием механизмов: `std::promise`, `std::future` и лямбда-функцию.

Привести расширенное решение.

Время выполнения – 35 мин.

Ожидаемый результат:

```
#include <iostream>
```

```
#include <thread>
```

```
#include <future>
```

```
int main() {
```

```
    // Создаем promise и future
```

```
    std::promise<int> promise;
```

```
    std::future<int> future = promise.get_future();
```

```
    // Запускаем поток и передаем в него promise
```

```
    std::thread t([&promise]() {
```

```
        // Выполняем некоторую работу
```

```
        std::this_thread::sleep_for(std::chrono::seconds(2));    // имитируем
```

задержку

```
        int result = 42; // результат работы
```

```
        promise.set_value(result); // устанавливаем значение в promise
```

```
    });
```

```
    // По завершении работы потока, получаем результат
```

```
    std::cout << "Ожидание результата потока..." << std::endl;
```

будет готов

```
    int result = future.get(); // блокируем до тех пор, пока результат не
```

```
    std::cout << "Результат: " << result << std::endl;
```

```
    // Завершаем поток
```

```
    t.join();
```

```
    return 0;
```

```
}
```

Пояснение.

– механизм потоков в C++ не позволяет напрямую возвращать значения. Вместо этого, можно использовать такие механизмы, как `std::promise` и `std::future`, чтобы передавать результаты обратно из потока.

– `std::promise<int>` - это объект, который позволяет установить значение, которое будет доступно в будущем через `std::future`.

– `std::future<int>` - это объект, который используется для получения результата, установленного в `std::promise`.

– создаем новый поток, передавая лямбда-функцию, которая имитирует некоторую работу, затем устанавливает значение в promise.

– в основном потоке ожидаем, пока результат не будет готов, с помощью future.get.

– выводим результат и ждем завершения потока с помощью t.join().

Критерии оценивания:

– создание объекта std::promise<int>;

– создание объекта std::future<int> для получения результата;

– создание нового потока и передача лямбда-функции;

– получение результата в основном потоке с помощью future.get;

– вывод результата.

Компетенции (индикаторы): ПК-3

2. Выполнить защиту разделяемого ресурса от одновременного доступа в многопоточной среде с помощью мьютексов из библиотеки <mutex>.

Привести расширенное решение.

Время выполнения – 35 мин.

Ожидаемый результат:

```
#include <iostream>
```

```
#include <thread>
```

```
#include <mutex>
```

```
#include <vector>
```

```
std::mutex mtx; // Мьютекс для защиты разделяемых данных
```

```
int sharedData = 0; // Разделяемые данные
```

```
void incrementSharedData(int threadId) {
```

```
    for (int i = 0; i < 10; ++i) {
```

```
        // Блокируем мьютекс перед доступом к разделяемым данным
```

```
        mtx.lock();
```

```
        ++sharedData; // Увеличиваем общее количество
```

```
        std::cout << "Thread " << threadId << " incremented sharedData to "
```

```
<< sharedData << std::endl;
```

```
        mtx.unlock(); // Освобождаем мьютекс
```

```
        std::this_thread::sleep_for(std::chrono::milliseconds(100)); // Задержка
```

```
        для имитации работы
```

```
    }
```

```
}
```

```
int main() {
```

```
    const int numThreads = 3;
```

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

```
    // Запускаем несколько потоков
```

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

```
        threads.emplace_back(incrementSharedData, i);
```

```
    }
```

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

```

    for (auto& th : threads) {
        th.join();
    }
    std::cout << "Final value of sharedData: " << sharedData << std::endl;
    return 0;
}

```

Пояснение.

Использование мьютекса: Создаем мьютекс `mtx` для защиты переменной `sharedData`.

Функция `incrementSharedData`: Каждый поток будет выполнять эту функцию, увеличивая значение `sharedData`.

Блокировка мьютекса: Перед тем как модифицировать `sharedData`, поток вызывает `mtx.lock()`. Это предотвращает доступ к разделяемым данным от других потоков до тех пор, пока мьютекс не будет разблокирован с помощью `mtx.unlock()`.

Запуск потоков: В `main` создаются несколько потоков, которые запускают функцию `incrementSharedData`.

Ожидание завершения потоков: Используя `join()`, мы ожидаем, пока все потоки завершат свою работу, прежде чем вывести финальное значение `sharedData`.

Критерии оценивания:

- создание мьютекса `mtx` для защиты переменной `sharedData`;
- блокировка мьютекса `mtx.lock()` перед изменением `sharedData`;
- разблокировка мьютекса с помощью `mtx.unlock()`;
- запуск потоков;
- ожидание завершения потоков, `join()`;
- вывод результата.

Компетенции (индикаторы): ПК-3

3. Выполнить синхронизацию доступа на C++ с использованием класса `std::unique_lock`.

Привести расширенное решение.

Время выполнения – 35 мин.

Ожидаемый результат:

```

#include <iostream>
#include <thread>
#include <mutex>
#include <chrono>
#include <condition_variable>

```

```

std::mutex mtx; // Мьютекс для синхронизации доступа к ресурсу

```

```

std::condition_variable cv; // Условная переменная для условного
ожидания

```

```

bool ready = false; // Флаг готовности

```

```

void worker(int id) {
    std::this_thread::sleep_for(std::chrono::milliseconds(100 * id)); //
Имитируем работу
    std::unique_lock<std::mutex> lock(mtx); // Получаем уникальную
блокировку
    std::cout << "Worker " << id << " is ready" << std::endl;
    ready = true; // Устанавливаем флаг готовности
    cv.notify_all(); // Уведомляем всех ожидающих потоков
}
void waiter() {
    std::unique_lock<std::mutex> lock(mtx); // Получаем уникальную
блокировку
    cv.wait(lock, []{ return ready; }); // Ожидаем, пока готовность не
станет true
    std::cout << "Waiter has been notified!" << std::endl;
}
int main() {
    std::thread t1(worker, 1);
    std::thread t2(worker, 2);
    std::thread t3(waiter);
    t1.join();
    t2.join();
    t3.join();
    return 0;
}

```

Пояснение.

Класс `std::unique_lock` позволяет реализовать гибкую блокировку, что полезно в различных ситуациях, таких как временные блокировки или блокировки с условием.

Подключение библиотек:

- `#include <iostream>` – для работы с вводом и выводом.
- `#include <thread>` – для работы с потоками.
- `#include <mutex>` – для работы с мьютексами.
- `#include <chrono>` – для работы с временными задержками.
- `#include <condition_variable>` – для работы с условными

переменными.

Глобальные переменные:

– `std::mutex mtx` – мьютекс, который будет использоваться для защиты общей переменной.

– `std::condition_variable cv` – условная переменная для синхронизации потоков.

– `bool ready` – флаг, который указывает, готов ли поток.

Функция `worker`:

- Каждому рабочему потоку (worker) назначен идентификатор, и он выполняет некоторую работу в течение времени.

- После завершения работы, поток захватывает мьютекс с помощью `std::unique_lock`.

- Устанавливает флаг `ready` в `true` и уведомляет все потоки, ожидающие на условной переменной.

Функция `waiter`:

- Этот поток ожидает, пока флаг `ready` не станет `true`.

- Он использует `cv.wait(lock, [])`, чтобы заблокировать поток и ожидать уведомлений от других потоков.

Главная функция `main`:

- Создается три потока: два для функции `worker` и один для функции `waiter`.

- Используется `join()` для того, чтобы дождаться завершения всех потоков.

- Преимущества использования `std::unique_lock`:

- Гибкость: `std::unique_lock` позволяет временно отпустить мьютекс, а затем запрашивать его снова.

- Удобство: Автоматическое освобождение мьютекса при выходе из области видимости, что снижает вероятность возникновения ошибок.

- Работа с условными переменными: Легкая интеграция с `std::condition_variable`, что позволяет удобно реализовывать сложные механизмы синхронизации.

Критерии оценивания:

- создание мьютекса `mtx` для защиты переменной;

- создание условной переменной для синхронизации потоков;

- создание потоков;

- запуск потоков;

- ожидание завершения потоков, `join()`;

- вывод результата.

Компетенции (индикаторы): ПК-3

4. Создать пример использования атомарных операций для безопасного инкремента значения из нескольких потоков.

Привести расширенное решение.

Время выполнения – 35 мин.

Ожидаемый результат:

```
#include <iostream>
#include <thread>
#include <atomic>
#include <vector>
std::atomic<int> counter(0);
void increment(int times) {
    for (int i = 0; i < times; ++i) {
```

```

        counter.fetch_add(1, std::memory_order_relaxed);
    }
}
int main() {
    const int numThreads = 10;
    const int incrementsPerThread = 1000;
    std::vector<std::thread> threads;
    for (int i = 0; i < numThreads; ++i) {
        threads.emplace_back(increment, incrementsPerThread);
    }
    for (auto& t : threads) {
        t.join();
    }
    std::cout << "Final counter value: " <<
counter.load(std::memory_order_relaxed) << std::endl;
    return 0;
}

```

Пояснение.

- Создаем атомарную переменную counter, инициализируя ее значением 0.
- Функция increment увеличивает значение counter определенное число раз. Каждый вызов fetch_add выполняется атомарно.
- В main создаются и запускаются 10 потоков, каждый из которых инкрементирует счетчик 1000 раз.
- После завершения всех потоков выводится окончательное значение counter.

Критерии оценивания:

- создание атомарной переменной;
- создание функции для работы в потоках;
- создание потоков;
- запуск потоков;
- ожидание завершения потоков, join();
- вывод результата.

Компетенции (индикаторы): ПК-3

Экспертное заключение

Представленный комплект оценочных материалов по дисциплине «Программирование для параллельных вычислений» соответствует требованиям ФГОС ВО.

Предлагаемые оценочные материалы адекватны целям и задачам реализации основной профессиональной образовательной программы по направлению подготовки 09.03.04 Программная инженерия.

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