**Лабораторная работа 02**

**Использование и синхронизация потоков**

**1. ЦЕЛЬ РАБОТЫ**

Целью работы является приобретение навыков работы с потоками при программировании на языке Java или другом языке.

**2. СОСТАВ РАБОЧЕГО МЕСТА**

2.1. **Оборудование**: IBM-совместимый персональный компьютер (ПК).

2.2. **Программное обеспечение**: операционная система Windows, Java 2 SDK версии 1.2.2 и выше.

**3. КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

**3.1. Реализация потоков в Java**

Язык Java является одним из немногих языков программирования, которые содержат средства поддержки потоков. Потоки можно применить в любой программе при необходимости параллельного выполнения нескольких задач.

Так, например, при создании коллективом программистов большого и сложного программного продукта, как правило, отдельные модули программы разрабатываются параллельно отдельными программистами или группами программистов. В этом случае процесс разработки каждого модуля программы можно представить как отдельный поток.

Реализация использования потоков в программах на языке может выполняться двумя способами:

* расширением класса **Thread**;
* реализацией интерфейса **Runnable**.

При первом способе класс становится поточным, если он создан как расширение класса **Thread**, который определен в пакете **java.lang**, например:

**public class GreatRace extends Thread**

При этом становятся доступными все методы потоков.

Обычно, когда необходимо, чтобы данный класс является расширением некоторого другого класса и в нем необходимо реализовать потоки, предыдущий подход нельзя использовать, поскольку, как уже указывалось, языке Java нет множественного наследования. Для решения этой проблемы для данного класса нужно реализовать интерфейс **Runnable**, например:

**public class GreatRace extends Applet implements Runnable**

Интерфейс **Runnable** имеет только один метод **public void run()**. На самом деле класс **Thread** также реализует этот интерфейс, однако стандартная реализация **run()** в классе **Thread** не выполняет никаких операций. Необходимо либо расширить класс **Thread**, чтобы включить в него новый метод **run()**, либо создать объект **Runnnable** и передать его конструктору потока. Когда создается класс, реализующий интерфейс **Runnable**, этот класс должен переопределить метод **run()**. Именно этот метод выполняет фактическую работу, возложенную на конкретный поток.

Создать поток можно с помощью одного из следующих конструкторов:

**public Thread()**

**public Thread(String name)**

**public Thread(Runnable target)**

**public Thread(Runnable target, String name)**

**public Thread(ThreadGroup group, String name)**

**public Thread(ThreadGroup group, Runnable target)**

**public Thread(ThreadGroup group, Runnable target, String name)**

В первом конструкторе создается поток, который использует самого себя в качестве такого интерфейса **Runnable**. В остальных конструкторах используемые параметры имеют следующий смысл:

* **name** – имя, которое присваивается новому потоку;
* **target** – определение целевого объекта, который будет использоваться новым объектом **Thread** при запуске потоков. Если опустить этот параметр или присвоить ему значение **null**, новый объект **Thread** будет запускать потоки посредством вызова метода **run()** текущего объекта **Thread**. Если при создании нового объекта **Thread** указывается целевой объект, то для запуска новых процессов он будет вызывать метод **run()** указанного целевого объекта;
* **group** – предназначен для помещения нового объекта **Thread** в дерево объектов данного класса. Если опустить данный параметр или присвоить ему значение **null**, новый объект класса **Thread** станет членом текущей группы потоков **ThreadGroup**.

Метод **public String toString()** возвращает строковое представление потока, включая имя потока, приоритет и имя группы, а методы **public final String getName()** и **public final void setName(String name)** позволяют получить имя потока или установить имя потока.

Запуск потока выполняет метод **public void start() throws IllegalThreadStateException** (исключение бросается, если делается попытка запуска уже запущенного потока).

Для остановки потока рекомендуется останавливаемому потоку присвоить значение **null**, например:

**Thread myThread;**

**…**

**myThread.start(); // Запуск потока**

**…**

**myThread = null; // Остановка или завершение потока**

Предположим, что на каком-то этапе работы над программным проектом необходимы два модуля, над которыми работают две группы программистов. Этап может начаться, только если обе группы закончили работу над своими модулями, т.е. одной из групп программистов придется дожидаться окончания работы над модулем другой группы. Для такого согласования действий используется **public final void join() throws InterruptedException**.

Методу **join** можно также передать значение тайм-аута, используя его варианты **public final syncronized void join(long millis) throws InterruptedException** и **public final syncronized void join(long millis, int nanos) throws InterruptedException**. Эти методы ожидают в течение **millis** миллисекунд либо **millis** миллисекунд плюс **nanos** наносекунд.

Если требуется, чтобы перед продолжением работы поток ждал определенное время, можно использовать метод **public static void sleep(long millis) throws InterruptedException** или **public static void sleep(long millis, int nanos) throws InterruptedException**, где параметры **millis** и **nanos** имеют тот же смысл, что и для метода **join**. Метод **sleep()** очень часто используется в циклах, управляющих анимацией.

Если в программе есть поток, который захватывает процессор, производя большое количество вычислений, может появиться необходимость заставлять его время от времени "отпускать" процессор, давая возможность выполняться другим потокам. Это достигается с помощью метода **public static void yield()**.

Метод **public void destroy()** уничтожает поток без всякой очистки относящихся к нему данных, а метод **public final boolean isAlive()** позволяет определить, запущен ли поток и еще «жив».

Потоки в Java могут **прерывать друг друга**. Механизм прерываний реализуется с помощью следующих методов:

* **public void interrupt()** – прерывает данный поток;
* **public static boolean interrupted()** – проверяет, был ли прерван данный поток (этот метод очищает признак прерывания для потока, т.е. при повторном вызове метода для этого же прерванного потока он вернет значение **false**);
* **public boolean isInterrupted()** – аналогичен предыдущему методу, но не очищает признака прерывания для потока.

В классе **Thread** есть ряд статических методов для изучения текущего потока и других потоков из той же группы. Метод **public static Thread currentThread()** возвращает объект, соответствующий выполняемому в момент вызова потоку, метод **public static void dumpStack()** выводит трассировку стека для текущего потока.

Метод **public final void checkAccess()** проверяет, имеет ли право текущий поток модифицировать данный поток.

Обычно программа на Java работает до завершения всех входящих в нее потоков. Иногда, однако, встречаются потоки, работающие в фоновом режиме, выполняя вспомогательные действия, которые никогда не заканчиваются. Можно пометить такой поток как **поток-демон (daemon thread)***,* что говорит JVM о том, что этот поток не надо принимать в расчет при определении, все ли потоки данной программы завершились. Другими словами, приложение на Java выполняется до тех пор, пока не завершится последний поток, не являющийся демоном. Потоки, не помеченные как демоны, называются **пользовательскими потоками (user threads)**.

Чтобы поток считался демоном, надо воспользоваться методом **public final void setDaemon(boolean on) throws IllegalThreadStateException**. Если параметр **on** равен **true**, поток получает статус демона, если **false** — статус пользовательского потока. Статус потока может быть изменен в процессе его выполнения. Метод **public final boolean isDaemon()** возвращает **true**, если поток является демоном, и **false**, если это пользовательский поток.

Метод **public static int enumerate(Thread[] threadArray)** класса **Thread** заполняет массив объектами **Thread**, представляющими потоки в группе, к которой относится текущий поток. Поскольку перед таким вызовом необходимо создать массив **threadArray**, надо знать, сколько элементов будет получено. Метод **public static int activeCount()** класса **Thread** сообщает, сколько активных потоков в группе, к которой относится данный поток.

**3.2. Приоритеты и группы потоков**

Распределения процессорного времени между потоками в Java выполняется по следующим правилам: когда поток блокируется, то есть приостановлен, переходит в состояние ожидания или должен дождаться какого-то события, Java выбирает другой поток из тех, которые готовы к выполнению. Выбирается поток, имеющий наибольший приоритет. Если таких несколько, выбирается любой из них. Приоритет потока можно установить методом **public final void setPriority(int newPriority) throws IllegalArgumentException**.

Приоритет потока должен быть числом в диапазоне от **Thread.MIN\_PRIORITY** до **Thread.MAX\_PRIORITY**. Любое значение вне этих пределов вызывает исключение **IllegalArgumentException**. По умолчанию потоку приписывается приоритет **Thread.NORM\_PRIORITY**. Значение приоритета потока можно выяснить с помощью метода **public final int getPriority()**.

Класс **ThreadGroup** реализует стратегию обеспечения безопасности, которая позволяет влиять друг на друга только потокам из одной группы. Например, поток может изменить приоритет другого потока из той же группы или перевести его в состояние ожидания. Если бы не было разбиения потоков на группы, один поток мог бы вызвать хаос в среде Java, переведя в состояние ожидания все остальные потоки, или, что еще хуже, завершив их. Группы потоков организованы в иерархическую структуру, где у каждой группы есть родительская группа. Потоки могут воздействовать на потоки из своей группы и из дочерних групп. Группу потоков можно создать, просто задав ее имя, с помощью конструктора **public ThreadGroup(String groupName)**.

Можно также создать группу потоков, дочернюю по отношению к существующей, используя конструктор **public ThreadGroup (ThreadGroup existingGroup, String groupName) throws NullPointerException**.

Метод **public String toString()** возвращает строковое представление данной группы, а методы **public final String getName()** и **public final void setName(String name)** позволяют получить имя группы или установить имя группы.

Другие методы класса **ThreadGroup** являются аналогами соответствующих методов класса **Thread**, но применяются ко всей группе:

* **public int activeCount()** – возвращает число активных потоков в данной группе;
* **public int activeGroupCount()** – возвращает число активных групп в данной группе;
* **public final void checkAccess()** – проверяет, может ли выполняемый в настоящее время поток модифицировать данную группу;
* **public final void destroy() throws IllegalThreadStateException, SecurityException** и **public boolean isDestroyed()** – соответственно уничтожает все потоки данной группы и ее подгрупп (все потоки в группе должны быть предварительно остановлены) или проверяет «жива» ли данная группа;
* **public final void interrupt()** – прерывает все потоки в данной группе;
* **public final boolean isDaemon() и public final void setDaemon(boolean daemon)** – соответственно проверяет и устанавливает признак потока-демона для всей группы.

Можно ограничить приоритет потоков из группы с помощью вызова метода public **final synchronized void setMaxPriority (int priority)**, а определить максимальный приоритет потока в группе можно с помощью метода **public final int getMaxPriority()**.

Родительская группа потоков доступна посредством вызова метода **public final ThreadGroup getParent()**, а с помощью метода **public final boolean parentOf(ThreadGroup g)** можно проверить является ли группа **g** родительской для данной группы.

**3.3. Синхронизация потоков**

Как уже указывалось, основное отличие потоков от процессов состоит в том, что потоки не защищены друг от друга средствами операционной системы. Поэтому любой из потоков может получить доступ и даже внести изменения в данные, которые другой поток считает своими. Решение этой проблемы состоит в **синхронизации** потоков.

Синхронизация потоков состоит в гарантии в каждый момент времени предоставления доступа к данным только к одному потоку. Для этого используется следующий оператор:

**synchronized (***выражение***)**

*оператор*

Взятое в скобки *выражение* должно указывать на блокируемые данные – обычно оно является ссылкой на объект. Чаще всего при блокировке объекта необходимо выполнить сразу несколько операторов, так что *оператор,* как правило, представляет собой блок.

Если оказывается, что критический участок распространяется на весь метод, а разделяемым ресурсом является весь объект в целом, то можно просто указать модификатор **synchronized** в объявлении метода, например:

**synchronized void myMethod()**

**{**

**// Тело метода**

**}**

Более гибкий и эффективный способ координации выполнения потоков обеспечивают методы **wait()** и **notify()** класса **Object**.

Метод **wait()** переводит поток в состояние ожидания выполнения определенного условия и вызывается с помощью одной из следующих форм:

* **public final void wait() throws InterruptedException, IllegalMonitorStateException** – приостановление выполнения текущего потока до получения извещения;
* **public final void wait(long timeout) throws InterruptedException, IllegalMonitorStateException, IllegalArgumentException** – приостановление выполнения текущего потока до получения извещения или до истечения заданного интервала времени timeout (в миллисекундах);
* **public final void wait(long timeout, int nanos) throws InterruptedException, IllegalMonitorStateException, IllegalArgumentException** – приостановление выполнения текущего потока до получения извещения или до истечения заданного интервала времени timeout (в миллисекундах) и, дополнительно, в наносекундах (значение в диапазоне 0-999999).

Метод **public final void notify()** переводит в активное состояние один из потоков, установленных в состояние ожидания с помощью метода **wait()**. Критерий выбора потока является произвольным и зависит от конкретной операционной системы и реализации виртуальной машины Java. Если необходимо перевести все ожидающие потоки в активное состояние, можно воспользоваться методом **public final void notifyAll()**.

Методы могут вызываться только потоком, который является владельцем монитора данного объекта. Если до вызова методов **wait()** или **notify()** не выполнить захват монитора данного объекта, генерируется исключение **IllegalMonitorStateException**. Поток становится владельцем данного объекта одним из трех способов:

* вызовом метода, объявленного как **synchronized**, который осуществляет доступ к требуемому экземпляру объекта класса **Object**;
* выполнением тела оператора **synchronized**, предназначенного для синхронизации доступ к требуемому экземпляру объекта класса **Object**;
* выполнением, для объектов типа **Class**, синхронизированного статического метода этого класса.

**4. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ**

Напишите программу моделирования с использованием на языке Java по одному из приведенных ниже вариантов.

В программах класс **CPUQueue** описывает очередь, класс **CPUProcess** моделирует поток процессов, а класс **CPU** – поток обслуживания процесса центральным процессором. Очереди, потоки процессов и обслуживание процесса моделируются с помощью объектов соответствующего класса. Параметры очереди моделируются с помощью алгоритмов вставки и извлечения процесса из очереди. Параметром процесса является интервал времени между двумя последовательными генерациями процессов. Параметром процессора является время обслуживания процесса.

Случайные времена для интервалов между моментами генерации процессов и для времен обслуживания распределены по равномерному закону с заданными верхней и нижней границами (см. метод **random()** в классе **Math**). Исходными данными для моделирования являются количество процессов, которые должны быть сгенерированы (для каждого потока процессов), а также нижние и верхние границы для потоков.

**Вариант 1**

Программа моделирует обслуживание двух потоков процессов с разными параметрами одним центральным процессором компьютера. Для каждого потока задается своя очередь. Очередь для первого потока имеет фиксированный размер, и, если процесс сгенерирован в тот момент, когда первая очередь заполнена, процесс уничтожается. Размер очереди для второго потока неограничен. На каждые n запросов из первой очереди (n задается как исходное данное), процессор берет на обработку один запрос из второй очереди. Определить процент уничтоженных процессов первого потока и максимальную длину второй очереди.

**Вариант 2**

Программа моделирует обслуживание одного потока процессов несколькими центральными процессорами компьютера с несколькими очередями одинакового размера. Если процесс сгенерирован в тот момент, когда первая очередь заполнена, процесс помещается в следующую очередь, для обработки которой запускается второй процессор. Если и эта очередь заполнена, создается третья очередь и запускается третий процессор и т.д. Определить количество задействованных процессоров, максимальную длину последней очереди и процент от общего количества процессов, которые были обработаны каждым процессором.

**Вариант 3**

Программа моделирует обслуживание одного потока процессов несколькими центральными процессорами компьютера без очередей. Если процесс сгенерирован в тот момент, когда первый процессор занят, процесс берется на обработку вторым процессором, если и этот процессор занят, запускается третий процессор и т.д. Определить количество задействованных процессоров и процент от общего количества процессов, которые были обработаны каждым процессором.

**Вариант 4**

Программа моделирует обслуживание одного потока процессов одним центральным процессором компьютера с двумя очередями. Сгенерированный процесс помещается в ту очередь, размер которой меньше. В свою очередь, процессор берет на обслуживание запрос из той очереди, размер которой больше. Определить процент запросов, которые были выбраны на обработку из первой очереди.

**Вариант 5**

Программа моделирует обслуживание двух потоков процессов с разными параметрами двумя центральными процессорами компьютера с одной очередью. Если сгенерирован процесс второго потока и второй процессор занят, процесс поступает в очередь. Если сгенерирован процесс первого потока, то, если первый процессор занят обработкой первого потока, то процесс поступает на обработку на второй процессор. Если и второй процессор занят, то процесс уничтожается. Если в момент генерации процесса первого потока на первом процессоре обрабатывается процесс второго потока, то процесс прерывается и возвращается в очередь. Определите максимальную длину очереди, процент уничтоженных процессов для первого потока и процент прерванных запросов для второго потока.

**Вариант 6**

Программа моделирует обслуживание двух потоков процессов с разными параметрами двумя центральными процессорами компьютера с одной очередью. Если сгенерирован процесс второго потока и второй процессор занят, процесс поступает в очередь. Если сгенерирован процесс первого потока, то, если первый процессор занят обработкой первого потока, то процесс поступает на обработку на второй процессор. Если и второй процессор занят, то процесс уничтожается. Если в момент генерации процесса первого потока на первом процессоре обрабатывается процесс второго потока, процесс считается потерянным. Определите максимальную длину очереди и проценты уничтоженных процессов для первого и второго потока.

**Вариант 7**

Программа моделирует обслуживание одного потока процессов одним центральным процессором компьютера с одной очередью. Если очередной процесс генерируется в момент, когда процессор свободен, процесс поступает на обработку в процессор, иначе процесс поступает в очередь. Если процессор свободен и в очереди есть процессы, процесс удаляется из очереди. Определите максимальную длину очереди.

**Вариант 8**

Программа моделирует обслуживание одного потока процессов двумя центральными процессорами компьютера без очереди. Если очередной процесс генерируется в момент, когда любой из процессоров свободен, процесс поступает на обработку в данный процессор, иначе процесс уничтожается. Определите процент уничтоженных процессов для двух одинаковых процессоров.

**Вариант 9**

Программа моделирует обслуживание одного потока процессов двумя центральными процессорами компьютера с общей очередью. Если очередной процесс генерируется в момент, когда любой из процессоров свободен, процесс поступает на обработку, иначе процесс становится в очередь. Если один из процессоров освобождается, и в очереди есть процессы, процесс удаляется из очереди. Определите максимальный размер очереди для двух одинаковых процессоров.

**Вариант 10**

Программа моделирует обслуживание одного потока процессов двумя центральными процессорами компьютера с двумя очередями. Если очередной процесс генерируется в момент, когда любой из процессоров свободен, процесс поступает на обработку в свободный процессор, иначе процесс помещается в ту из двух очередей, размер которой меньше. Определите максимальный размер для каждой из очередей для двух одинаковых процессоров.

**Вариант 11**

Программа моделирует обслуживание двух потоков процессов с разными параметрами центральным процессором компьютера. Для каждого потока задается своя очередь. Если очередной процесс генерируется в момент, когда процессор свободен, процесс поступает на обработку на процессор, иначе процесс становится в свою очередь. Если процессор освобождается, он берет на обработку процесс из первой очереди и, только в том случае, если первая очередь свободна, процессор берет на обработку процесс из второй очереди. Определите максимальный размер для каждой из очередей.

**Вариант 12**

Программа моделирует обслуживание одного потока процессов одним центральным процессором компьютера с одной очередью заданного размера. Если очередной процесс генерируется в момент, когда процессор свободен, процесс поступает на обработку в процессор, иначе процесс поступает в очередь. Если процессор свободен и в очереди есть процессы, процесс удаляется из очереди. Если размер очереди превышает заданный размер, процесс уничтожается. Определите процент уничтоженных процессов.

**Вариант 13**

Программа моделирует обслуживание одного потока процессов двумя центральными процессорами компьютера с общей очередью заданного размера. Если очередной процесс генерируется в момент, когда любой из процессоров свободен, процесс поступает на обработку, иначе процесс становится в очередь. Если размер очереди превышает заданное значение, процесс уничтожается. Определите максимальный размер очереди и процент уничтоженных процессов для двух одинаковых процессоров.

**Вариант 14**

Программа моделирует обслуживание двух потоков процессов с разными параметрами двумя центральными процессором компьютера. Для каждого потока задается своя очередь. Процесс из каждого потока поступает на свой процессор, и только в том случае, когда в своей очереди нет процесса, процессор берет на обработку чужой процесс. Определите максимальный размер для каждой из очередей.

**Вариант 15**

Программа моделирует обслуживание двух потоков процессов с разными параметрами двумя центральными процессором компьютера. Для каждого потока задается своя очередь. Процесс из каждого потока поступает на свой процессор, и только в том случае, когда длина своей очереди не превышает заданного для этой очереди размера, а длина другой очереди превышает заданный для нее размер, процессор берет на обработку чужой процесс. Определите максимальный размер для каждой из очередей.

**Вариант 16**

Программа моделирует обслуживание одного потока процессов двумя центральными процессорами компьютера с общей очередью. Если очередной процесс генерируется в момент, когда первый процессор свободен, процесс поступает на обработку, иначе процесс становится в очередь. Если размер очереди превышает заданное значение, процесс поступает на обработку во второй процессор. Определите общее количество процессов и отношения (в процентах) обработанных процессов к общему количеству сгенерированных процессов для обоих процессоров.

**Вариант 17**

Программа моделирует обслуживание двух потоков процессов с разными параметрами двумя центральными процессором компьютера. Для каждого потока задается своя очередь. Процесс из каждого потока поступает на первый процессор, и только в том случае, когда длина одной из очередей превышает заданную для нее величину, поступает на второй процессор. Определите общее количество процессов и отношения (в процентах) обработанных процессов к общему количеству сгенерированных процессов для обоих процессоров.

**Вариант 18**

Программа моделирует обслуживание двух потоков процессов с разными параметрами одним центральным процессором компьютера без очереди. Процесс из каждого потока поступает на процессор, но если при обработке процесса генерируется процесс из того же потока, обрабатываемый процесс считается потерянным. Однако, если при обслуживании процесса из второго потока генерируется процесс первого потока, то процесс уничтожается и начинается обработка процесса из первого потока. Определите процент уничтоженных процессов для первого и второго потоков.

**Вариант 19**

Программа моделирует обслуживание одного потока процессов двумя центральными процессорами с одной очередью. Когда процесс удаляется из очереди, он сначала поступает на обработку в первый процессор. Как только процесс обработки заканчивается, процесс обрабатывается на втором процессоре, а на первый процессор поступает очередной процесс из очереди (если очередь не пустая). Процесс на первом процессоре дожидается окончания обработки предыдущего процесса на втором процессоре (если обработка предыдущего процесса еще не закончилась). Определите максимальную длину очереди.

**Вариант 20**

Программа моделирует обслуживание двух потоков процессов с разными параметрами одним центральным процессором и одной очередью. Если сгенерирован процесс второго потока и процессор занят, процесс поступает в очередь. Если сгенерирован процесс первого потока, то, если процессор обрабатывает процесс первого потока, то процесс уничтожается. Если же процессор обрабатывает процесс второго потока, то процесс прерывается и возвращается в очередь. Определите максимальную длину очереди, процент уничтоженных процессов первого потока и количество прерванных процессов второго потока.

**Вариант 21**

Программа моделирует обслуживание двух потоков процессов с разными параметрами одним центральным процессором и тремя очередями. Если сгенерирован процесс второго потока и процессор занят, процесс поступает во вторую очередь. Если сгенерирован процесс первого потока, то, если процессор обрабатывает процесс второго потока, процесс второго потока приостанавливается и помещается в третью очередь (с оставшимся временем обработки). Если процессор же обрабатывает процесс первого потока, то процесс поступает в первую очередь. При освобождении процессора очереди просматриваются в следующем порядке: первая очередь, третья очередь, вторая очередь. Определите максимальные длины очередей и количество прерванных процессов второго потока.

**Вариант 22**

Программа моделирует обслуживание двух потоков процессов с разными параметрами одним центральным процессором и двумя очередями. Каждый процесс поступает в свою очередь. При освобождении процессора обрабатывается запрос из первой очереди (если она не пуста) или из второй очереди, если время ожидания первого запроса из второй очереди превышает заданную величину. Определите максимальные длины очередей и средние времена ожидания запросов в очередях.

**Вариант 23**

Программа моделирует обслуживание двух потоков процессов с разными параметрами двумя центральными процессорами и двумя очередями. Каждый процесс поступает в свою очередь и обслуживается своим процессором. Если первый процессор свободен и очередь первого потока процессов пуста, процессор выбирает процесс из второй очереди, однако, если во время обработки процесса, генерируется процесс первого потока, обработка процесса прерывается и он возвращается в свою очередь как первый в очереди. Определите максимальные длины очередей и количество прерванных процессов второго потока.

**Вариант 24**

Программа моделирует обслуживание двух потоков процессов с разными параметрами тремя центральными процессорами и двумя очередями. Каждый процесс поступает в свою очередь и обслуживается своим процессором. Третий процессор берет запрос на обслуживание сначала из первой очереди или (если первая очередь пустая) из второй. Определите максимальные длины очередей и проценты процессов первого и второго потока, обслуженные третьим процессором.

**Вариант 25**

Программа моделирует обслуживание трех потоков процессов с разными параметрами двумя центральными процессорами и тремя очередями. Каждый процесс поступает в свою очередь. Процессы первого и второго потоков обслуживаются каждый своим процессором. Если первый или второй процессор освобождаются, они сначала просматривают очередь процессоров третьего потока и, если она пуста, очереди

**Вариант 26**

Программа моделирует обслуживание трех потоков процессов с разными параметрами тремя центральными процессорами компьютера без очередей. Первый процессор обрабатывает процессы любого потока. Второй процессор обрабатывает процессы только второго потока, а третий – только третьего. Если сгенерированный процесс не может быть обработан из-за занятости процессора, он уничтожается. Определить процент уничтоженных процессов в каждом потоке.

процессов своего потока. Определите максимальные длины очередей и проценты процессов третьего потока, обслуженные первым и вторым процессором.

**Вариант 27**

Программа моделирует обслуживание одного потока процессов одним центральным процессором компьютера с очередями фиксированного размера (буферными областями). Если размер очереди превышает заданный размер, содержимое очереди поступает на обработку в процессор. Если буферная область заполняется до окончания обработки процессором содержимого предыдущей буферной области, создается новая буферная область и т.д. Определить максимальное и среднее количество буферных областей.

**Вариант 28**

Программа моделирует обслуживание одного потока процессов одним центральным процессором компьютера с двумя очередями. Если размер первой очереди превышает заданный, процесс помещается во вторую очередь (буфер переполнения), размер которой не ограничен. Процессор берет на обработку процесс из второй очереди только в том случае, если размер этой очереди превысит некоторую заданную величину. Определить максимальный размер второй очереди и процент процессов, которые были выбраны на обработку из второй очереди.

**Вариант 29**

Программа моделирует обслуживание одного потока процессов тремя центральными процессорами компьютера без очереди. Первый процессор обрабатывает процессы самостоятельно, второй и третий – последовательно, причем, если второй процессор освобождается (передает процесс третьему), он может взять на обработку следующий процесс; второй процессор должен ждать, пока третий процессор не освободится, чтобы передать обработанный процесс ему. Сгенерированный процесс одновременно пытается начать обработку на первом и на втором процессоре. Если процессор занят, то процесс уничтожается. Определить процент уничтоженных процессов отдельно на первом и отдельно на втором процессоре.

**Вариант 30**

Программа моделирует обслуживание одного потока процессов одним центральным процессором компьютера без очереди. Если очередной процесс генерируется в момент, когда процессор свободен, процесс поступает на обработку в процессор, иначе процесс уничтожается. Определите процент уничтоженных процессов.

**5. СОДЕРЖАНИЕ ОТЧЕТА**

В отчете должен быть представлены тексты программы, а также результаты выполнения программы при различных исходных данных.

**6. ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ**

1. Какие два способа используются для реализации потоков в Java?
2. Как выполняется остановка или завершение потока?
3. Какие операции над потоками определены в Java?
4. Как выполняется прерывание потока и проверка состояния прерывания?
5. Что такое потоки-демоны и чем они отличаются от обычных потоков?
6. Как задать и получить значение приоритета для потока?
7. Как можно создать группу потоков, и какие операции определены для группы потоков в Java?
8. Какие средства синхронизация потоков имеются в Java?
9. Как выполняется синхронизация потоков с помощью оператора **synchronized**?
10. Как выполняется синхронизация потоков с помощью методов **wait()** и **notify()**?