МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
"Юго-Западный государственный университет"
(ЮЗГУ)

Кафедра биомедицинской инженерии

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г.Локтионова
"___" _____ 2017

МОБИЛЬНЫЕ КОМПЛЕКСЫ ДЛИТЕЛЬНОГО МОНИТОРИРОВАНИЯ БИОФИЗИЧЕСКИХ СИГНАЛОВ ЧЕЛОВЕКА

Методические указания к проведению практических занятий для студентов направления подготовки 12.04.04 - "Биотехнические системы и технологии" (магистр)

Курск 2017

УДК 615.478

Составитель Д.Е.Скопин

Рецензент
Доктор технических наук, профессор *И.Е. Чернецкая*

Мобильные комплексы длительного мониторирования биофизических сигналов человека: методические указания к поведению практических занятий / Юго-Зап. гос. ун-т; сост. Д.Е.Скопин. - Курск, 2017. - 23 с.: ил. 5, табл.1. - Библиогр.: с. 23.

Содержатся сведения, необходимые для выполнения практических занятий по мобильным комплексам длительного мониторирования биофизических сигналов человека.

Методические указания по структуре, содержанию и стилю изложения материала соответствуют методическим и научным требованиям, предъявляемым к учебным и методическим пособиям.

Предназначены для студентов направления подготовки 12.04.04 очной формы обучений.

Текст печатается в авторской редакции

Подписано в печать Формат 60х84 1/16
Усл. печ.л. . Уч. -изд.л. Тираж 100 экз. Заказ. Бесплатно.
Юго-Западный государственный университет
305040, г.Курск, ул. 50 лет Октября, 94

Практическое занятие №1

Android - операционная система мобильных комплексов и основы ее программирования

1.1. Цель работы: изучить основы объектно-ориентированного программирования, классов, объектов, методов, позволяющих создавать программы для операционной системы Android

1.2. Краткие теоретические сведения

В настоящее время наиболее распространенной операционной системой для построения мобильных комплексов, смартфонов, планшетов и прочих устройств является Android. Программирование этой операционной системы базируется на языке программирования JAVA. Элементами данных этого языка являются любые структуры данных внутри класса: переменные, такие как int, float double, short; Массивы некоторых элементов; Объекты некоторых других классов, таких как String, StringBuilder, ArrayList. Методы класса - это члены функции любой функции внутри класса. Итак, просто вы можете сказать, что класс - это набор переменных и набор функций, расположенных вместе в одном месте под некоторым именем. Класс имеет имя; Это похоже на некоторый тип данных. В другой руке у нас есть объект - экземпляр класса, объект выглядит как переменная некоторого типа данных. Пример:

Int a; // a - переменная типа данных int;

String st; // st - объект класса String.

Пример объявления класса в Java:

```
class MyClass {
    // field, constructor, and
    // method declarations
}
```

В примере выше описан новый класс с именем «MyClass». Внутри класса вы можете выделить набор переменных, массивов и объектов других классов. «class» - это ключевое слово Java. Перед этим ключевым словом вы можете написать префикс «public». Слово «public» для классов в Java означает, что этот класс доступен из других классов, из других пакетов и из других программ. Несколько раз вы можете встретить некоторые

другие префиксы перед ключевым словом «class»: «abstract» означает, что некоторые методы класса, объявленные, но не определенные (см. Практическую работу 2, чтобы узнать подробности абстрактных классов), префикс «final» определяет последний класс (без наследования, дочерний класс не может быть создан).

Класс может иметь конструктор. Конструктор - это метод класса (функция внутри класса), который имеет одно и то же имя с классом. Конструктор будет выполняться автоматически в любое время, когда будет создан объект класса. Класс может не иметь конструктора, может иметь один конструктор из нескольких конструкторов. Последний факт, называемый перегрузкой конструктора - ситуация, в которой у вас есть много конструкторов внутри класса с тем же именем (что совпадает с именем класса), но с разными параметрами.

Теперь вы знаете, что в Java, как в C ++, есть конструкторы. Но в отличие от C ++ в Java нет деструкторов. Деструктор в C ++ - это метод, который имеет одно и то же имя с префиксом класса плюс ~ перед именем. Деструктор в C ++ - это функция, которая будет выполняться автоматически в любое время, когда объект класса будет уничтожен (используя ключевое слово «удалить»). В Java нет указателей, исключение ключевых слов, отсутствие множества памяти и никаких деструкторов. Все объекты выделены не в множестве памяти, а в некоторой другой памяти виртуальной машины Java, называемой «сбор мусора». Это означает, что программист может создавать объект, но не может его уничтожить. Программист может сообщить о сборе мусора, что какой-то объект больше не нужен, назначив значение «null» объекту, но физически этот объект будет уничтожен последним, может быть через один час, время разрушения непредсказуемо на Java. Вот почему никакие деструкторы в Java не гарантируют, что объект будет уничтожен в будущем. Если вы хотите, то можете использовать деструктор, представленный на Java, с помощью метода «finalize», но ни один орган не может гарантировать, что деструктор будет выполнен в вашей программе, поэтому в практических программах Java нет причин использовать деструкторы.

Пример прояснит ситуацию с классами, конструкторами, деструкторами и объектами.

```
class MyClass //MyClass is class name
{
int a=1,b=2,c=3;//variables a,b,c look like public variables
public int d=4,e;//d and e are public variables
protected float f=5.5f;//f is protected variable, in Java it look like public
private String st="Hello";//This is private variable, like private in C++

public MyClass()//This is constructor. Name equal to the class name.
                //No output of this function!
{
System.out.println("This is default constructor");
}

public MyClass(String s)//This is overloaded constructor
{st=st+s;
 System.out.println("This is String constructor: "+st);
}

public MyClass(float k)//This is also overloaded constructor
{f=k;
 System.out.println("This is float constructor, value of f is "+f);
}

void f1()//this is method of the class
{
    System.out.println("This is method f1 of the class MyClass");
}

protected void finalize()
{
System.out.println("This is destructor, but destructurs in Java are not useful");
}
}//end of the class
```

```
public class Example{//This is public class with main method
    //The code on this page located in the file Example.java
   static public void main(String[] args)
   {MyClass obj1=new MyClass();
    obj1.a=5;
    obj1.f1();
    obj1=null;
    MyClass obj2=new MyClass("Ahmad");
    MyClass obj3=new MyClass(5.5f);
   }
}
```

Результатом этой программы будет:

```
This is default constructor

This is method f1 of the class MyClass

This is String constructor: HelloAhmad

This is float constructor, value of f is 5.5
```

Вы можете видеть, что в основном методе мы пытались уничтожить объект obj1, присвоив значение null obj1 = null ;, но на самом деле этот объект не разрушен, поэтому нет причин использовать деструкторы в реальных Java-программах. Также вы можете видеть, что у нас есть операторы «new» в каждой строке, где созданы объекты, но не «delete» операторов, потому что все объекты, созданные в сборке мусора, а не в памяти кучи и сборке мусора, будут автоматически очищены виртуальной машиной Java после выполнения вашей программы.

Вывод программы показывает, что если объект класса, созданного с помощью пустых скобок, конструктор по умолчанию выполнен автоматически. Перегруженные конструкторы будут выполняться только в том случае, если вы передадите соответствующие параметры. В Java, как в C ++, есть наследование, это означает, что вы можете создать класс на основе некоторого класса. В следующем примере мы создадим Class1 сначала Class2 на основе Class1, чтобы показать наследование, все строки кода Java, пронумерованные, чтобы иметь ссылки в тексте:

```
1   class Class1 //Class1 is superclass of Class2
2   {int a=1;
3    protected float b=2;//protected look like public in Java
    void f1()
```

```
4    {
5        System.out.println("Method f1 of Class1");
6    }
7   }
8   class Class2 extends Class1//Class2 is subclass of Class1
9   {
10    void f2()
11    {
12      System.out.println("Method f2 of Class2:");
13      System.out.println("I can call data members and
    methods of Class1:");
14      System.out.println("a="+a+" b="+b);
15      f1();//We call method f1 of Class1 directly
16      int a=5;//from this moment we have new local variable
    a
17      //now we can access a of Class1 using keyword super:
18      System.out.println("local a="+a+" but a of
    Class1="+super.a);
19    }
20  }
21
22  public class Example{
23       //This is public class with main method
24       //The code on this page located in the file
    Example.java
25      static public void main(String[] args)
26      {Class2 obj=new Class2();
27       obj.b=100;//b is protected but can be accessed
    directly even outside a
28       //class
29       obj.f2();
30      }
31  }
32
33
34
```

Результатом этой программы будет:

```
Method f2 of Class2
I can call data members and methods of Class1:
a=1 b=2.0
Method f1 of Class1
local a=5 but a of Class1=1
```

Программа содержит 3 класса: Class1, Class2 и Example. Последний класс является общедоступным, поэтому вся программа находится в файле Example.java. Класс с именем «Class1» является корневым классом, созданным без наследования, пусть «from zero». Класс содержит два элемента данных a и b и один метод f1 (). Члены класса также называются в Java как «fields if the class». Начальные значения элементов данных, которые вы можете назначить непосредственно в классе.

В строке 10 нашего примера вы можете увидеть определение класса 2, созданного с использованием Class1. Вы можете увидеть ключевое слово «extends», что означает, что Class2 унаследовал все от Class1. Внутри Class2 мы имеем только один метод f2 (), но все члены данных и методы, унаследованные от Class1, также доступны в Class2. В строке 16 (method f2 () class 2) мы печатаем значения переменных a и b, объявленных в Class1. Затем в строке 17 мы назвали метод f1 (), расположенный в Class1. В строке 18 мы создали новую локальную переменную a и присвоенное значение 5, начиная с этого момента мы можем получить доступ к переменной «a» class1, используя ключевое слово «super», что означает перейти к суперклассу (родительский класс) и получить определенный член класса после десятичной точки «.», В нашем случае это элемент данных «a» class1, см. Строку 20.

В строке 29 мы создали объект «obj» класса Class2. Затем в строке 30 мы получили доступ к защищенному элементу данных «b», объявленному в Class1. Поскольку вы видите, что «protected» элемент данных выглядит как открытый, его можно получить непосредственно в классе, где он был объявлен, или в любом другом подклассе.

1.3. Задание для лабораторной работы.

Напишите программу для операционной системы Android, которая печатает все реальные решения квадратичного уравнения. Решение квадратичного уравнения должно быть реализовано в классе Quadratic. Класс имеет 3 общих элемента данных **a, b, c.** Вы должны создать методы класса:

Конструктор по умолчанию класса: назначить нулевые значения: a = 0, b = 0, c = 0 для членов данных класса;

Void inputdata (); Этот метод попросит пользователя ввести номера a, b, c с помощью ПК-клавиатуры, все номера должны быть назначены элементам данных a, b и c.

Int checksolution (); Этот метод возвращает число корней (0, если отрицательное значение **Discriminant (D),** 1, if D = 0, 2, if D> 0) или распечатка «Нет входных данных», если метод inputdata () не выполнен. D = b2 + 4ac.

Void printoutresults (); Этот метод печатает корни квадратного уравнения. Если квадратичное уравнение не имеет корней, то распечатка метода «Нет корней», если только один корень квадратного уравнения представляет собой распечатку метода «x = значение корня», если два корня будут напечатаны «x1 = root1 и x2 = root2». Этот метод распечатывает «Нет входных данных», если метод inputdata () не выполнен.

Значения корней, которые вы можете вычислить, используя уравнение:

$x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$, where $D=b^2+4ac$.

Запишите текст класса и основного метода, чтобы создать объект класса Квадратичный и вычислить корни некоторых квадратичных уравнений.

1.4 Содержание отчета

1. Титульный лист
2. Задание на практическую работу
3. Листинг программы
4. Скриншот работы приложения

Практическое занятие №2

Изучение команд операционной системы для контроля функционирования сети

2.1. Цель работы: изучить команды операционной системы, позволяющие оценить наличие сетевого подключения, его свойства, качество связи, пропускную способность сети, список ее вспомогательных узлов и прочие параметры

2.2 Краткие теоретические сведения

Язык Java — это объектно-ориентированный язык программирования, ведущий свою историю от известного языка C++. Но в отличие от последнего Java является языком интерпретируемым, программы, написанные на нем, способны работать в разных местах сети и не зависят от плат-формы, на которой выполняются написанные на нем приложения. Изучая Java, вы будете приятно удивлены тем, что его синтаксис близок к синтаксису языка C++. Унаследовав самое лучшее от языка программирования C++, язык Java при этом избавился от некоторых недостатков C++, в результате чего на нем стало проще программировать. В этом языке нет, например, указателей, которые сложны в использовании и потенциально могут по-служить причиной доступа программы к не принадлежащей ей области памяти. Нет множе-ственного наследования и шаблонов, хотя функциональные возможности языка Java от этого не пострадали. Если вы умеете программировать на C++, для вас не составит особого труда изучить язык Java.

Огромное преимущество Java заключается в том, что на этом языке можно создавать прило-жения, способные работать на различных платформах. К сети Internet подключены компьютеры самых разных типов - Pentium PC, Macintosh, рабочие станции Sun и так далее. Даже в рамках компьютеров, созданных на базе процессоров Intel, существует несколько платформ, например, Microsoft Windows 2000, Windows NT, Windows XP, OS/2, Solaris, различные разновидности опе-рационной системы UNIX с графической оболочкой X¬Windows. Приложения Java предназначены для работы на различных платформах и не зависят от конкретного типа процессора и операцион-ной системы.

Программы, составленные на языке программирования Java, можно разделить по своему назначению на две большие группы.

К первой группе относятся приложения Java, предназначенные для автономной работы под управлением специальной интерпретирующей машины Java. Реализации этой машины созданы для всех основных компьютерных платформ.

Вторая группа - это так называемые апплеты (applets). Апплеты представляют собой разно-видность приложений Java, которые интерпретируются виртуальной машиной Java, встроенной практически во все современные браузеры.

Приложения, относящиеся к первой группе (мы будем называть их просто приложениями Java), - это обычные автономные программы. Так как они не содержат машинного кода и рабо-тают под управлением специального интерпретатора, их производительность заметно ниже, чем у обычных программ, составленных, например, на языке программирования C++. Однако не сле-дует забывать, что программы Java без перетрансляции способны работать на любой платформе, что само по себе имеет большое значение в плане разработок для Internet.

Для повышения производительности приложений Java в современных браузерах используется компиляция "на лету"- Just-In-Time compilation (JIT). При первой загрузке апплета его код транс-лируется в обычную исполнимую программу, которая сохраняется на диске и запускается. В ре-зультате общая скорость выполнения апплета Java увеличивается в несколько раз.

Язык Java является объектно-ориентированным и поставляется с достаточно объемной биб-лиотекой классов. Так же как и библиотеки классов систем разработки приложений на языке C++, библиотеки классов Java значительно упрощают разработку приложений, представляя в распоряжение программиста мощные средства решения распространенных задач. Поэтому про-граммист может больше внимания уделить решению прикладных задач, а не таких, как, напри-мер, организация динамических массивов, взаимодействие с операционной системой или реали-зация элементов пользовательского интерфейса.

2.3. Задание на практическую работу

Используя команды операционной системы, такие как ping, tracert, ipconfig, net, команды подсистемы netbios, провести трассировку и изучение параметров сети с использованием узлов, заданных преподавателем.

3.4 Содержание отчета

1. Титульный лист
2. Задание на практическую работу
3. Листинг программы
4. Скриншот работы приложения

Практическое занятие №3

Программирование службы DNS с использованием языка Java

3.1. Цель работы

Изучить классы и их методы, которые позволяют реализовать доменное разрешение имен, в частности преобразование имен в адреса и обратное преобразование

3.2. Краткие теоретические сведения

Если посмотреть большинство более-менее популярных статей об устройстве Интернета, то про DNS там чаще всего будет сказано что-то типа "DNS-сервер обеспечивает трансляцию имен сайтов в IP адреса". В принципе, это действительно является его основной задачей, и для большинства пользователей (и даже компьютерщиков) этих знаний вполне достаточно. Но если вдруг вам придется отлаживать сеть, которой провайдер выделил какой-то блок "честных" адресов, или поднимать в локальной сети свой DNS-сервер, то очень быстро всплывут всякие страшные слова: "зона", "трансфер", "форвардер", "in-addr.arpa" и другие... Поэтому в этой заметке мы попробуем чуть более подробно поговорить о работе DNS.

Очень приблизительно можно сказать, что у каждого компьютера в Интернете есть два основных идентификатора: доменное имя (например, www.listsoft.ru) и IP-адрес (например, 127.0.0.1). Приблизительность заключается в том, что, во-первых, IP-адресов у компьютера может быть несколько (мало того, что у каждого интерфейса свой адрес, так еще и несколько адресов могут "висеть" на одном интерфейсе); во-вторых, имен тоже может быть несколько, причем они могут связываться как с одним, так и с несколькими IP-адресами; в-третьих, у компьютера может и не быть доменного имени... Словом, ясно, что картина уже начинает запутываться.

Основной задачей DNS-сервера является трансляция доменных имен в IP-адреса и обратноКак было сказано выше, основной задачей DNS-сервера является трансляция доменных имен в IP адреса и обратно. На заре становления Интернета (когда он еще был ARPANET'ом) это решалось ведением длинных списков, включающих все компьютеры сети, причем копия такого списка должна была присутствовать на каждом компьютере. Понятно, что с ростом сети эта технология перестала

удовлетворять кого бы то ни было — ведь эти файлы надо было еще и синхронизировать, не говоря уж об их размере... Некоторые "пережитки" этого метода можно обнаружить и сейчас: существует файл HOSTS (и в UNIX, и в Windows), в котором можно прописывать адреса серверов, с которыми вы регулярно работаете (кстати, именно его использование лежит в основе многих "ускорителей Интернета" — такие программы просто записывают адреса серверов, к которым вы обращаетесь, в файл HOSTS и при следующем обращении берут данные из него, не тратя время на запрос к DNS-серверу).

На смену "однофайловой" схеме пришел DNS — иерархическая структура имен. Существует "корень дерева" с именем "." (точка). Так как корень един для всех доменов, то точка в конце имени обычно не ставится (но она используется в описаниях DNS — тут надо быть очень внимательным!). Ниже корня лежат домены первого уровня. Их немного — com, net, edu, org, mil, int, biz, info, gov (есть еще несколько) и домены государств, например, ru. Еще ниже находятся домены второго уровня, например, listsoft.ru. Еще ниже — третьего и т.д.

DNS-сервер работает как хороший компьютерщик: он всегда либо знает ответ, либо знает у кого спросить...Иерархичность DNS-серверов — штука довольно интересная, если проследить прохождение запроса. При установке (точнее, при настройке) клиенту указывается как минимум один DNS-сервер (как правило, их два) — его адрес выдается провайдером. Клиент посылает запрос этому серверу. Сервер, получив запрос, либо отвечает (если ответ ему известен), либо пересылает запрос на "вышестоящий" сервер (если он известен) или на корневой (каждому DNS-серверу известны адреса корневых DNS-серверов). Так выглядит "восходящая иерархия". Затем запрос начинает спускаться вниз — корневой сервер пересылает запрос серверу первого уровня, тот — серверу второго уровня и т.д. Таким образом, каждый DNS-сервер работает как хороший компьютерщик: он всегда либо знает ответ, либо знает, у кого спросить...

Помимо "вертикальных связей", у серверов есть еще и "горизонтальные" отношения — "первичный — вторичный". Действительно, если предположить, что сервер, обслуживающий какой-то домен и работающий "без страховки" вдруг перестанет быть

доступным, то все машины, расположенные в этом домене, окажутся недоступны! Именно поэтому при регистрации домена второго уровня выдвигается требование указать минимум два сервера DNS, которые будут этот домен обслуживать.

Рекурсивные сервера удобно использовать в локальных сетяхDNS-сервера бывают рекурсивные и нерекурсивные. Первые всегда возвращают клиенту ответ — они самостоятельно отслеживают отсылки к другим DNS-серверам и опрашивают их. Нерекурсивные сервера возвращают клиенту эти отсылки, так что клиент должен самостоятельно опрашивать указанный сервер. Рекурсивные сервера удобно использовать на низких уровнях, в частности, в локальных сетях. Дело в том, что они кэшируют все промежуточные ответы, и при последующих запросах ответы будут возвращаться намного быстрее. Нерекурсивные сервера обычно стоят на верхних ступенях иерархии — поскольку они получают очень много запросов, то для кэширования ответов никаких ресурсов не хватит.

Использование "пересыльщиков" ускоряет разрешение именПолезным свойством DNS является умение использовать "пересыльщиков" (forwarders). "Честный" DNS-сервер самостоятельно опрашивает другие сервера и находит нужный ответ, но если ваша сеть подключена к Интернету по медленной (например, dial-up) линии, то этот процесс может занимать довольно много времени. Вместо этого можно перенаправлять все запросы, скажем, на сервер провайдера, а затем принимать его ответ. Использование "пересыльщиков" может оказаться интересным и для больших компаний с несколькими сетями: в каждой сети можно поставить относительно слабый DNS-сервер, указав в качестве "пересыльщика" более мощную машину, подключенную по быстрой линии. При этом все ответы будут кэшироваться на этом мощном сервере, что ускорит разрешение имен для целой сети.

Для каждого домена администратор ведет базу данных DNS. Эта база данных представляет собой набор простых текстовых файлов, расположенных на основном (первичном) сервере DNS (вторичные сервера периодически копируют к себе эти файлы). В файлах конфигурации сервера указывается, в каком именно файле содержатся

описания каких зон, и является ли сервер первичным или вторичным для этой зоны.

Элементы базы DNS часто называют RR (сокращение от Resource Record). Базовый формат записи выглядит так:

[имя] [время] [класс] тип данные

Имя может быть относительным или абсолютным (FQDN — Fully Qualified Domain Name). Если имя относительное (не заканчивается точкой — помните про корневой домен?), то к нему автоматически добавляется имя текущего домена. Например, если в домене listsoft.ru я опишу имя «www», то полное имя будет интерпретироваться как "www.listsoft.ru." Если же это имя указать как "www.listsoft.ru" (без последней точки), то оно будет считаться относительным и будет интерпретировано как "www.listsoft.ru.listsoft.ru."

Время задает интервал времени в секундах, в течение которого данные могут сохраняться в кэше сервера.

Класс определяет класс сети. Практически всегда это будет IN, обозначающее INternet.

Тип может быть одним из следующих:

SOA — определяет DNS зону

NS — сервер имен для зоны

A — преобразование имени в IP-адрес

PTR — преобразование IP-адреса в имя

MX — почтовая станция

CNAME — имена машины

HINFO — описание "железа" компьютера

TXT — комментарии или какая-то другая информация

Есть также некоторые другие типы, но они намного менее распространены.

В записях можно использовать символы # и ; для комментариев, @ для обозначения текущего домена, () — скобки — для написания данных на нескольких строках. Кроме того, можно использовать метасимвол * в имени. Порядок записей не имеет значения за одним исключением: запись SOA должна идти первой. Дальнейшие записи считаются относящимися к той же зоне, пока не встретится новая запись SOA. Как

правило, после записи зоны указывают записи DNS-серверов, а остальные записи располагают по алфавиту, но это не обязательно.

SOA — описание зоны Теперь попробуем рассмотреть записи. Первой описываем зону:

mycompany.ru. IN SOA ns.mycompany.ru. admin.mycompany.ru. (1001 ; serial

21600 ; Refresh — 6 часов

1800 ; Retry — 30 мин

1209600 ; Expire — 2 недели

432000) ; Minimum — 5 дней

Сначала идет имя домена: mycompany.ru. (обратите внимание на точку в конце имени). Вместо имени можно было (и чаще всего так и делают) поставить знак @.

ns.mycompany.ru. — основной сервер имен

admin.mycompany.ru. — почтовый адрес администратора в формате имя(точка)машина

затем в круглых скобках идут поля, необходимые для правильного "восприятия" вашей зоны другими серверами. Первое число — serial — является "версией" файла зоны. При внесении изменений это число надо увеличить — если вторичный сервер увидит, что его версия зоны меньше, чем у первичного сервера, то он перечитает данные. Типичной ошибкой является обновление зоны без обновления этого числа. Очень удобно в качестве serial использовать текущую дату, например, 2003040401 — 4 апреля 2003 года, первое обновление.

Refresh говорит вторичным серверам, как часто они должны проверять значение serial.

Retry говорит о том, как часто вторичный сервер должен пытаться прочитать данные, если первичный сервер не отвечает.

Expire говорит вторичным серверам, в течение какого времени они должны обслуживать домен, если первичный сервер не отвечает. По истечении этого времени вторичные сервера будут считать свои данные устаревшими

3.3. Задание на практическую работу

Создайте программу, реализующие следующие функции:

1. Пользователь вводит IP адрес, а программа выдает ему DNS имя, которое относится к этому адресу

2. Пользователь вводит DNS имя сети, а программа выдает ему значение IP адреса, соответствующее этому имени

3. Пользователь вводит DNS имя требуемого узла. Программа производит поиск и печать всего списка IP адресов, принадлежащих заданому узлу.

3.4. Содержание отчета

1. Титульный лист
2. Задание на практическую работу
3. Листинг программы
4. Скриншот работы приложения

Практическое занятие №4

Многопоточность в сетевых приложениях

4.1. Цель работы

Изучить практические аспекты создания программ, реализующих свое функционирование с использованием нескольких потоков выполнения

4.2. Краткие теоретические сведения

Наиболее очевидная область применения многопоточности – это программирование интерфейсов. Многопоточность незаменима тогда, когда необходимо, чтобы графический интерфейс продолжал отзываться на действия пользователя во время выполнения некоторой обработки информации. Например, поток, отвечающий за интерфейс, может ждать завершения другого потока, загружающего файл из интернета, и в это время выводить некоторую анимацию или обновлять прогресс-бар. Кроме того он может остановить поток загружающий файл, если была нажата кнопка «отмена».

Еще одна популярная и, пожалуй, одна из самых хардкорных областей применения многопоточности – игры. В играх различные потоки могут отвечать за работу с сетью, анимацию, расчет физики и т.п.

Процесс — это совокупность кода и данных, разделяющих общее виртуальное адресное пространство. Чаще всего одна программа состоит из одного процесса, но бывают и исключения (например, браузер Chrome создает отдельный процесс для каждой вкладки, что дает ему некоторые преимущества, вроде независимости вкладок друг от друга). Процессы изолированы друг от друга, поэтому прямой доступ к памяти чужого процесса невозможен (взаимодействие между процессами осуществляется с помощью специальных средств).

Для каждого процесса ОС создает так называемое «виртуальное адресное пространство», к которому процесс имеет прямой доступ. Это пространство принадлежит процессу, содержит только его данные и находится в полном его распоряжении. Операционная система же

отвечает за то, как виртуальное пространство процесса проецируется на физическую память.

Схема этого взаимодействия представлена на картинке. Операционная система оперирует так называемыми страницами памяти, которые представляют собой просто область определенного фиксированного размера. Если процессу становится недостаточно памяти, система выделяет ему дополнительные страницы из физической памяти. Страницы виртуальной памяти могут проецироваться на физическую память в произвольном порядке.

При запуске программы операционная система создает процесс, загружая в его адресное пространство код и данные программы, а затем запускает главный поток созданного процесса.

Один поток – это одна единица исполнения кода. Каждый поток последовательно выполняет инструкции процесса, которому он принадлежит, параллельно с другими потоками этого процесса.

Следует отдельно обговорить фразу «параллельно с другими потоками». Известно, что на одно ядро процессора, в каждый момент времени, приходится одна единица исполнения. То есть одноядерный процессор может обрабатывать команды только последовательно, по одной за раз (в упрощенном случае). Однако запуск нескольких параллельных потоков возможен и в системах с одноядерными процессорами. В этом случае система будет периодически переключаться между потоками, поочередно давая выполняться то одному, то другому потоку. Такая схема называется псевдо-параллелизмом. Система запоминает состояние (контекст) каждого потока, перед тем как переключиться на другой поток, и восстанавливает его по возвращению к выполнению потока. В контекст потока входят такие параметры, как стек, набор значений регистров процессора, адрес исполняемой команды и прочее. Проще говоря, при псевдопараллельном выполнении потоков процессор мечется между выполнением нескольких потоков, выполняя по очереди часть каждого из них.

После запуска побочного потока его инструкции начинают выполняться вперемешку с инструкциями главного потока. Кол-во выполняемых инструкций за каждый подход не определено.

То, что инструкции параллельных потоков выполняются вперемешку, в некоторых случаях может привести к конфликтам доступа к данным.

4.3. Задание на практическую работу

Получите у преподавателя математическую функцию. Создайте программу, которая запрашивает у пользователя пределы интегрирования a и b. Далее программа запрашивает требуемое количество потоков, после чего запускает параллельный поиск интеграла заданной функции на требуемом интервале. Интервал приращения dx принять равным 0.00000001. Количество потоков не должно превышать 20.

4.4. Содержание отчета

1. Титульный лист
2. Задание на практическую работу
3. Листинг программы
4. Скриншот работы приложения

Практическое занятие №5
Обзор технологий объектно-ориентированного программирования для построения систем диагностики

5.1. Цель работы: изучить основы работы с микроконтроллером Arduino, его аналого-цифровым преобразователем для практического овладения навыками измерений в реальном масштабе времени

5.2. Краткие теоретические сведения

Известно, что все величины в физическом мире носят аналоговый характер. Для измерения этих величин, существует множество различных приборов. Так, например, термометр позволяет узнать температуру вещества, барометр - давление газа, гигрометр - влажность воздуха. Все эти устройства имеют шкалу, которую мы используем для фиксации их показаний. Рассмотрим простой пример — определение температуры с помощью обычного градусника. Человек решает эту задачу очень просто: мы смотрим, к какому из делений ближе всего приблизился уровень жидкости в градуснике. Полученное таким образом значение и будет измеренной температурой. Иными словами, мы осуществляем преобразование аналоговой непрерывной величины в дискретную, которую можно записать на бумаге с помощью цифр.

Чтобы автоматизировать процесс измерения аналоговых величин, и возложить эту задачу на электронные приборы, инженеры создали особое устройство, называемое аналого-цифровым преобразователем (АЦП). Это устройство позволяет превращать аналоговый сигнал в цифровой код, пригодный для использования в ЭВМ.

В приборостроении АЦП являются важной составляющей системы датчиков машины. Акселерометр, гироскоп (гиротахометр), барометр, магнитометр, и даже видеокамера — все эти приборы соединяются с центральным процессором с помощью АЦП.

Конструктивно, АЦП может находиться в одном корпусе с микропроцессором или микроконтроллером, как в случае Arduino Uno. В противном случае, как и все современные электронные устройства, АЦП может быть оформлен в виде отдельной микросхемы, например MCP3008.

Следует отметить, что существует и устройство с обратной функцией, называемое цифро-аналоговым преобразователем (ЦАП, DAC). Оно позволяет переводить цифровой сигнал в аналоговый. Например, во время проигрывания мелодии на мобильном телефоне происходит преобразование цифрового кода из MP3 файла в звук, который вы слышите у себя в наушниках.

На карте Arduino Uno аналоговые входы имеют буквенно-цифровые обозначения A0, A1, …, A5 (снизу слева). Во время работы всё с теми же кнопками, мы познакомились с функцией digitalRead, которая умеет считывать цифровой сигнал с определенного входа контроллера. У этой функции существует аналоговая версия analogRead, которая может делать то же самое, но только для аналогового сигнала.

результат = analogRead(номер_контакта);

после вызова этой функции, микроконтроллер измерит уровень аналогового сигнала на заданном контакте, и сохранит результат работы АЦП в переменную «результат». При этом результатом функции analogRead будет число от 0 до 1023.

5.3. Порядок выполнения практической работы

1. Подключить к микроконтроллеру Arduino Pro Mini следующий набор узлов и элементов: датчик температуры и влажности, сенсор газов
2. Написать программу, для проведения опросов указанных узлов программным способом по двум разным каналам
3. Написать программу, осуществляющую представления полученной с АЦП информации в графической форме
4. Изменяя температуру датчика провести построение графика изменения
5. Напишите программу реакции на превышение предельно допустимой концентрации газов.

5.4 Содержание отчета

1. Титульный лист
2. Задание на практическую работу
3. Листинг программы
4. Результаты измерений

Практическое занятие №6

Основы построения систем дистанционной беспроводной диагностики организма

6.1. Цель работы: получение практических навыков построения системы дистанционной диагностики организма на основе современных микроконтроллеров

6.2. Используемые приборы, аппараты и элементы

6.2.1. Микроконтроллер ATMEGA 328P

6.2.2. Регулятор напряжения питания

6.2.3. Кварцевый резонатор

6.2.4. Модуль GSM SIM800L

6.2.5. Действующая сим-карта

6.2.6. Модуль пульсометра

6.3. Задание на практическую работу

Разработайте принципиальную схему устройства беспроводной регистрации функционального состояния человека. Устройство должно обеспечивать считывание ЧСС человека, его анализ и подачу тревожного сигнала по GSM сети при превышении заранее заданного порога диапазона ЧСС человека. Также предусмотрите автоматизированную отправку СМС с информацией о ЧСС объекта наблюдения на заранее заданные телефонные номера

6.4. Содержание отчета

1. Титульный лист
2. Задание на практическую работу
3. Листинг программы
4. Скриншот работы приложения