CSE 660

Operating System Concepts & Theory

June 16, 2011

Lab 3

Remote Procedure Call

Java Remote Method Invocation ( RMI )

Android Application Development

Ahmed Algadi

Ian Jacobs

Table of Contents

[Purpose: 4](#_Toc296002177)

[Part 1 Remote Procedure Call 4](#_Toc296002178)

[Overview 4](#_Toc296002179)

[Application or usage. 4](#_Toc296002180)

[Installation of the RPC and RPCGen? 4](#_Toc296002181)

[Preparation 4](#_Toc296002182)

[Installation of RPC tutorial 5](#_Toc296002183)

[Rpcgen and switches 5](#_Toc296002184)

[Modify Server and client code 5](#_Toc296002185)

[RPC Code Modification assignment for Random numbers. 6](#_Toc296002186)

[Test of RPC 7](#_Toc296002187)

[Installation Issues of RPC 7](#_Toc296002188)

[Part 2 Java Remote Method Invocation ( RMI ) 8](#_Toc296002189)

[What is RMI? 8](#_Toc296002190)

[Application or usage. 8](#_Toc296002191)

[Installation of RMI and Java 8](#_Toc296002192)

[Assignment Part 2 section 1 10](#_Toc296002193)

[Compiling RMI adder 10](#_Toc296002194)

[Test of RMI adder 11](#_Toc296002195)

[Assignment Part 2 section 2 11](#_Toc296002196)

[Compiling random number 12](#_Toc296002197)

[Test of RMI Random number 12](#_Toc296002198)

[Installation Issues of RMI 14](#_Toc296002199)

[Part 3 Android Application Development 15](#_Toc296002200)

[Overview 15](#_Toc296002201)

[How does it work? 16](#_Toc296002202)

[Application or usage. 16](#_Toc296002203)

[Installation of the Android? 16](#_Toc296002204)

[Preparation 16](#_Toc296002205)

[Installation of Android tutorial 16](#_Toc296002206)

[Test of Android 22](#_Toc296002207)

[Conclusions 23](#_Toc296002209)

# Purpose:

The Lab assignment has three parts, Remote Procedure Call, Java Remote Method Invocation (RMI), and Android Application Development. Each part is designed to help us understand communication between different systems using different protocols of communication.

# Part 1 Remote Procedure Call

## Overview

Remote Procedure Calling (RPC) protocol is used to communicate over the internet. Local procedure calls happen all the time with in a computer. But to communicate with other computers a method was created to send a procedure calls to other systems. By standardizing on an agreed communication approach or protocol and Unix system and a windows system can communicate.

## Application or usage.

The ability to communicate between different systems via RPC with different applications has greatly increased the development of applications on different platforms. A database can access on a local system or on a remote system regardless of the platform of application.

## Installation of the RPC and RPCGen?

## Preparation

We used the RPC utility **rpcgen** in this section. We were able to review some the help file using the

$**man rpcgen** command

rpcgen is a compiler. It accepts a remote program interface definition written in a language, called RPC Language, which is similar to C. It produces a C language output that includes stub versions of the client routines, a server skeleton, XDR filter routines for both parameters and results, and a header file that contains common definitions.

## Installation of RPC tutorial

|  |
| --- |
| /\* rand.x \*/  program RADN\_PROG {  version RAND\_VERS {  void INITIALIZE\_RANDOM ( long ) = 1; /\* service #1 \*/  double GET\_NEXT\_RANDOM ( void ) = 2; /\* service #2 \*/  } = 1;  } = 0x30000000; /\* program # \*/ |

As part of the assignment, rpc based code was given to implement.

We created a file named **rand.x** with the above context:

Then we executed the rpcgen command to create the template we will use and modify for the assignment.

### Rpcgen and switches

$**rpcgen -C -a rand.x**

The –C switch tells the rpcgen compiler to generate ANSI C code. The –a switch says to produce all files needed for the client and server.

The above command generated the below files.

Makefile.rand, rand\_clnt.c, rand\_server.c, rand\_client.c, rand.h, rand\_svc.c

The assignment indicates the programs **rand\_client.c** and **rand\_server.c** are the client and server programs that are needed to edit t finsh the assignment.

Now we compile the all them using the make command with makefile.

$**make -f Makefile.rand**

### Modify Server and client code

We now have a complied client and server. But they are really not functional yet. Now we to modify the get\_next\_random\_1\_svc() function in rand\_server.c as follows. (This is given as part of the assignment.)

|  |
| --- |
| Double \*  get\_next\_random\_1\_svc(void \*argp, struct svc\_req \*rqstp)  {  static double result;  result += 0.31;  if ( result >= 1.0 )  result -= 0.713;  return &result;  } |
|  |

Continue by modifying rand\_client.c as follows:

|  |
| --- |
| double  radn\_prog\_1(char \*host)  {  .....  .....  return \*result\_2;  }  int  main (int argc, char \*argv[])  {  ....  ....  double x;  int i;  printf("\n twenty random numbers ");  for ( i = 0; i < 20; ++i ){  x = radn\_prog\_1 (host);  printf(" %f, ", x );  }  exit (0);  } |

The assignment then has us recompile and link the programs with

$**g++ -c rand\_server.c**   
$**g++ -c rand\_client.c**   
$ **make -f Makefile.rand**

Now when we execute the client and server we get twenty numbers printed by the client. The numbers are generated by the server. The numbers generated in the above discussion are not anything random.

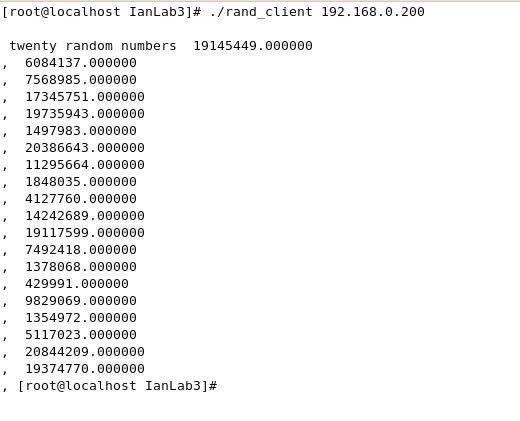
### RPC Code Modification assignment for Random numbers.

Right our own random number generator by modifying the server.c program.

## Test of RPC

We tested this first by trying it between two windows on our centos box. Then we tested it between two different systems.

C:\Users\ijacobs\Desktop\660\lab3\pictures\lab3-1-1 random.JPG



The server and the client communicated across the network to get the result. The clinet requested the information and the server generated the random numbers and sent them back to client to output.

## Installation Issues of RPC

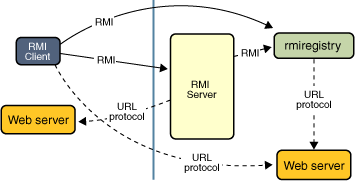
We did not have any installation issues. It was interesting using the rpcgen command to generate code.

# Part 2 Java Remote Method Invocation ( RMI )

## What is RMI?

RMI stands for Remote Method Invocation. It is designed to allow programmers to write code that can be distributed amongst multiple machines across a network. RMI provides a simple and direct model for distributed computation with Java objects.

The following illustration depicts an RMI distributed application that uses the RMI registry to obtain a reference to a remote object. The server calls the registry to associate (or bind) a name with a remote object. The client looks up the remote object by its name in the server's registry and then invokes a method on it. The illustration also shows that the RMI system uses an existing web server to load class definitions, from server to client and from client to server, for objects when needed.

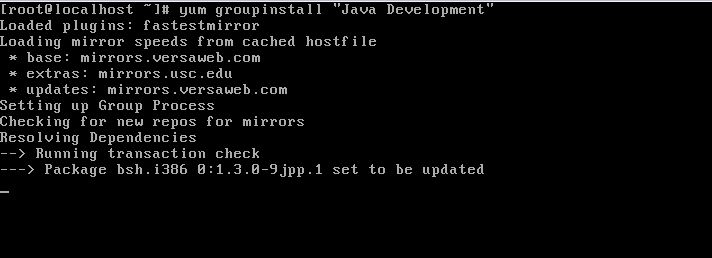
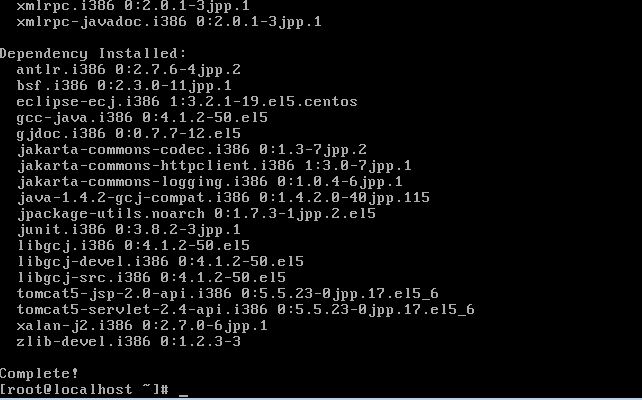


## Application or usage.

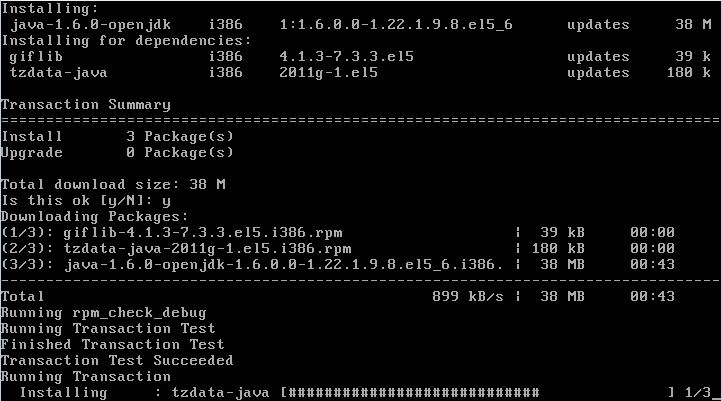
The ability to communicate between different systems via RMI with different applications has greatly increased the development of applications on different platforms. A database can access on a local system or on a remote system regardless of the platform of application.

## Installation of RMI and Java

We started by using an existing installation of Centos from the previous lab (Lab1-2) in CSE 660. The Java development platform had to be installed with Java. We used the yum program to install the needed packages.



After we installed the Development tools we needed to install Java.



Since RMI is included in the latest version of Java, we could start compiling code we had developed.

## Assignment Part 2 section 1

Create a "remote Java object" that performs simple addition of two numbers. Write a program that invokes the addition function on your remote object. (You have to write a client program that sends the server two numbers; the server then adds the two numbers and returns the sum to the client.) Store the classes of your program in a jar file, and invoke your code from the jar file. Submit hardcopy of your code, and a printout of your screen that shows how you register your remote object with the “rmiregistry” running on your machine, how you run the test program that invokes the add method of your remote object.

## Compiling RMI adder

To compile the adder program we used the java compiler

[root@localhost ~]# javac \*.java

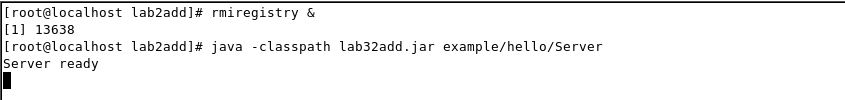
This compiled the code and created the .class files. Since we are using the example code from the internet, we copied the ‘class” files to the example/hello folder for functionality. Then we needed to create the jar file.

[root@localhost ~]# jar cf lab32add.jar \*.java example/hello/\*.class

This creates the jar file we use to test the code.

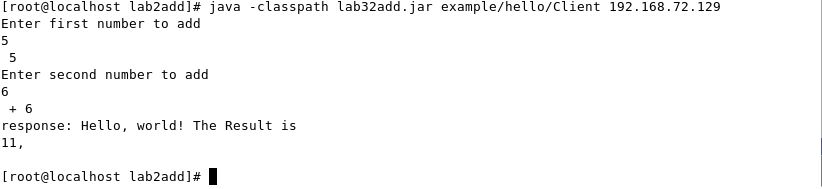
## Test of RMI adder

We first had to disable or stop the firewall. And we changed the hosts file to reflect the IP of the server.

We tested the RMI adder program by executing the RMIREGISTRY command on the server side. Then we executed the server program.

Then on the other computer we executed the client side.

The system asks for two numbers and then those numbers are sent to the server side to be added. The answer is then sent back to the client to be outputted.

The above picture shows the client server application works.

## Assignment Part 2 section 2

Implement and test a remote object that provides an interface to a "random number generator." The system interface should provide two operations: a method that accepts the number of random numbers **n** as the input parameter and returns **n** random numbers, and a method that accepts the number of random numbers **n**, and an upper and a lower bound for the requested random numbers, and returns **n** random numbers within the bounds. Demonstrate that client code running on another machine can invoke the random number generation system operations on your server. (Note: you may need to reconfigure iptables to accept tcp connections on port 1099 in order to access the rmi registry from a remote system.)   
Note that when executing "rmiregistry", you need to point to the correct class path. Easy way is to set CLASSPATH is to set in ".bashrc". You may check class path by "echo $CLASSPATH".   
Also, add your IP to the file "/etc/hosts".

## Compiling random number

To compile the random generator we used the java compiler

[root@localhost ~]# javac \*.java

This compiled the code and created the .class files. Since we are using the example code from the internet, we copied the ‘class” files to the example/hello folder for functionality. Then we needed to create the jar file.

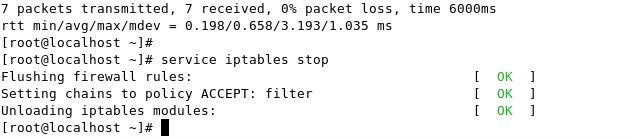
[root@localhost ~]# jar cf lab2A.jar \*.java example/hello/\*.class

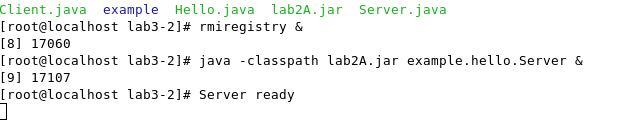
This creates the jar file we use to test the code.

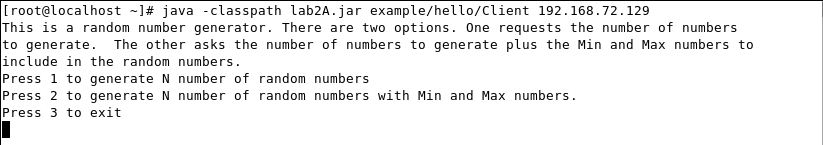
## Test of RMI Random number

On the server and on the client computers we changed the /etc/hosts file to include the ip of the server to reflect the localhost name. We then had to shut down the firewall.

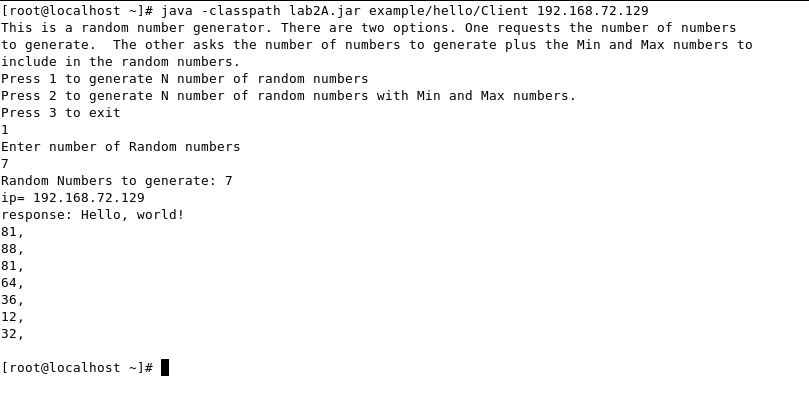
[root@localhost ~]# service iptables stop



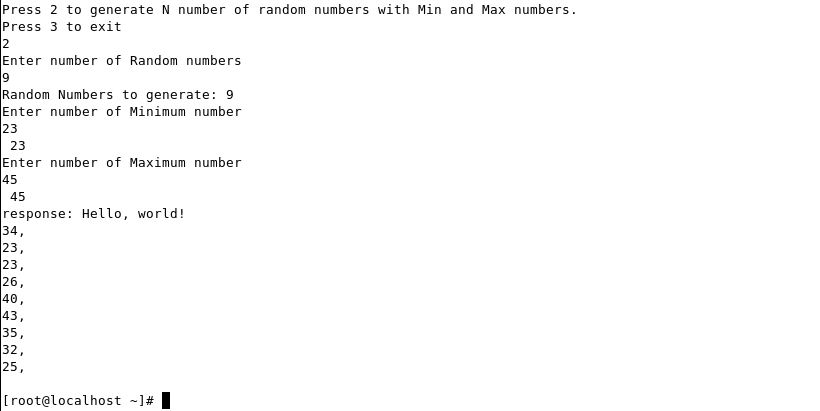
We then started the RMI registry and started the server side of the application.

On the client side we started the application in client mode.

There are three options. The first 1 is for generating a requested number of random numbers between 1 -100. The client accepts the amount of numbers to generate and then passes this information to the server to calculate the random numbers and return them to the client.



The second option has two addition bits of information. The first is the same as above, but then it requests the upper and lower bound of the random numbers to generate. The last option is just for exiting.

The above figure shows the random generator using RMI works.

## Installation Issues of RMI

We had two figure out about disabling the firewall to communicate. We also added the server IP to the Hosts file on both the client and server.

# Part 3 Android Application Development

## Overview

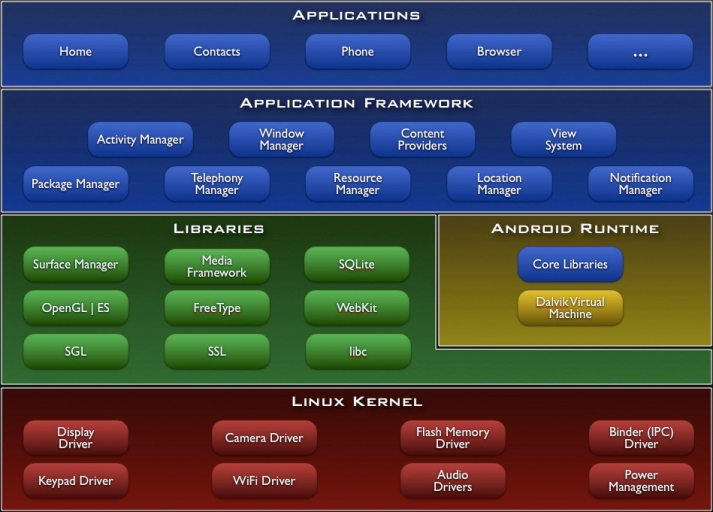
**Android** is a software stack for mobile devices that includes an operating system, middleware and key applications. Android's mobile operating system is based on the Linux kernel. The Android SDK is Open Source Project (AOSP) provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.

**Java JDK** The Java JDK is the Java Development Kit. It contains everything that a software developer needs to write programs in the Java Programming Language.

**Android SDK:** The Android SDK is similar to the Java JDK in that it is a bundle of tools such as a debugger, a phone emulator, software libraries, and other required components for software development on the Android Platform

**Eclipse:** Eclipse was written in Java, and it is open source software. Eclipse is an extremely popular IDE (Integrated Development Environment) that is used by developers across many different languages including Java, C++, PHP, and many more.

## How does it work?

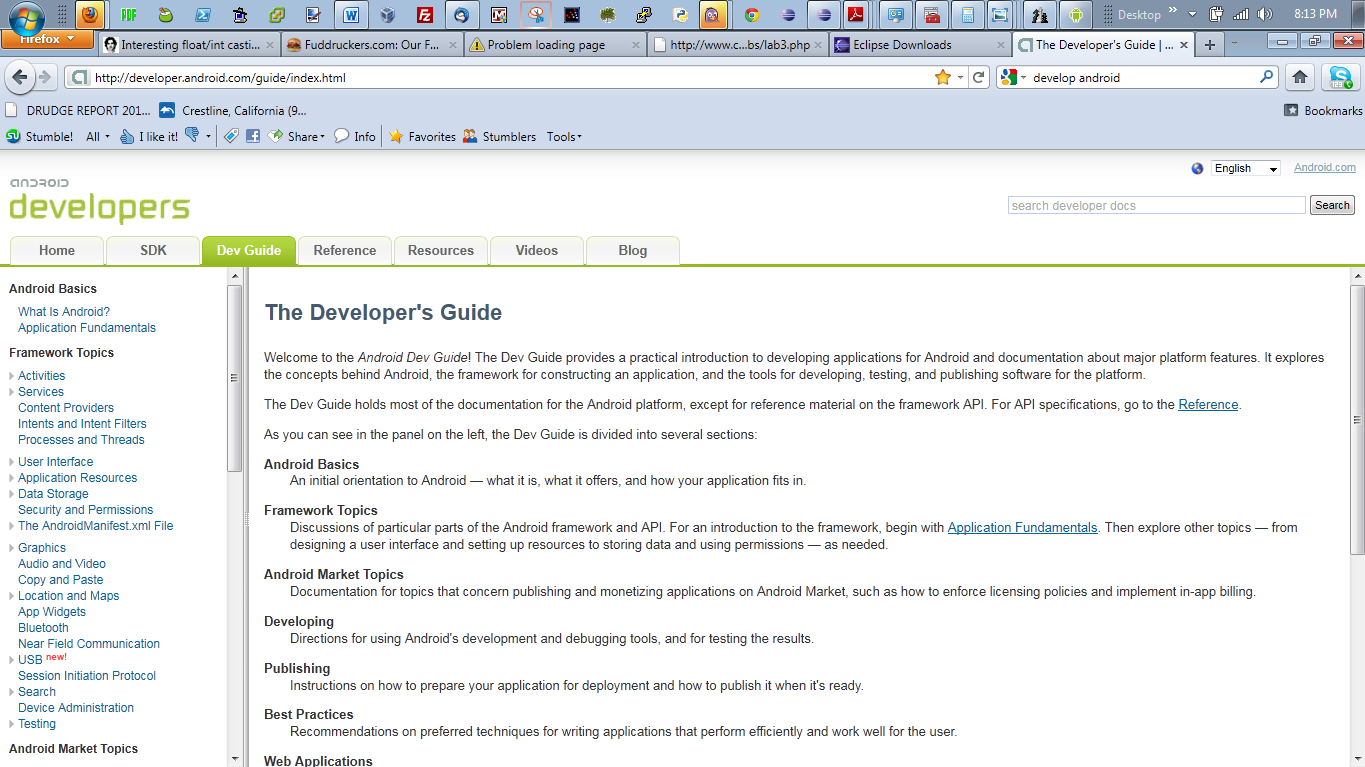


## Application or usage.

The ability to communicate between different systems via Android OS with different applications has greatly increased the development of applications on different platforms. A database can access on a local system or on a remote system regardless of the platform of application.

## Installation of Android tutorial

1. Here's an overview of the steps you must follow to set up the Android SDK:
2. 1. Prepare your development computer and ensure it meets the system requirements.
3. 2. Install the SDK starter package from [android-sdk\_r10-linux\_x86.tgz](http://cse.csusb.edu/tongyu/courses/cs660/labs/android-sdk_r10-linux_x86.tgz).
4. 3. Install the ADT Plugin for Eclipse (if you'll be developing in Eclipse).
5. 4. Add Android platforms and other components to your SDK.
6. 5. Explore the contents of the Android SDK (optional).
7. See [Android Developer's Guide](http://developer.android.com/guide/index.html)



1. What is Android?

Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.

Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework.

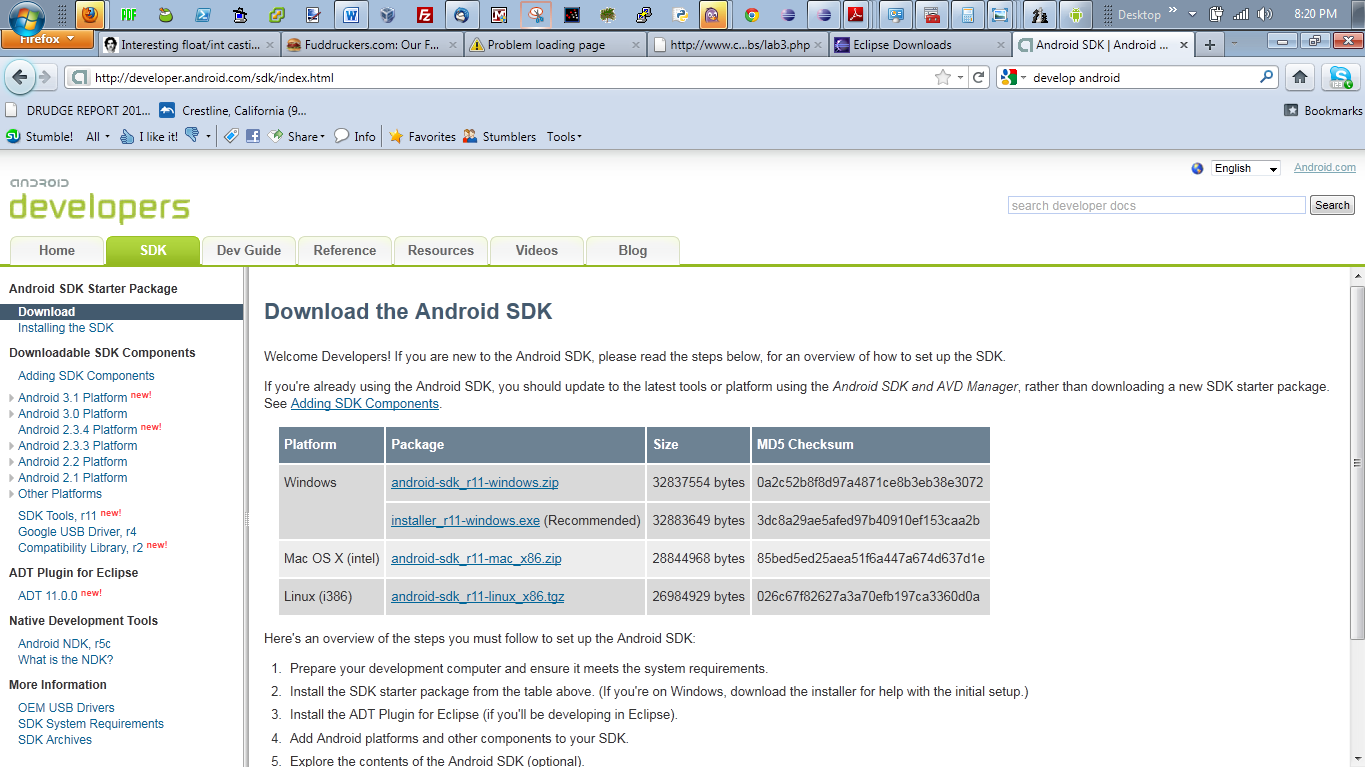
Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool.

The Dalvik VM relies on the **Linux kernel** for underlying functionality such as threading and low-level memory management. Dalvik, like the rest of Android, is open-source software. It was originally written by Dan Bornstein, who named it after the fishing village of Dalvíin Eyjafjör, Iceland, where some of his ancestors lived.

Dalvik is said to be a clean-room implementation rather than a development on top of a standard Java runtime, which would mean it does not inherit copyright-based license restrictions from either the standard-edition or open-source-edition Java runtimes. Dalvik is published under the terms of the Apache License 2.0

1. Install the Android SDK by following the instructions of the link:

<http://developer.android.com/sdk/installing.html>



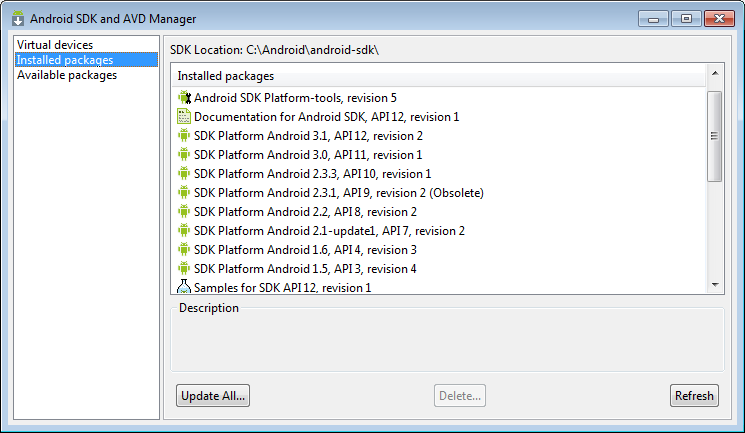
You may unpack the downloaded package into a directory with the command:

$ gunzip -c android-sdk\_r10-linux\_x86.tgz | tar xvf -

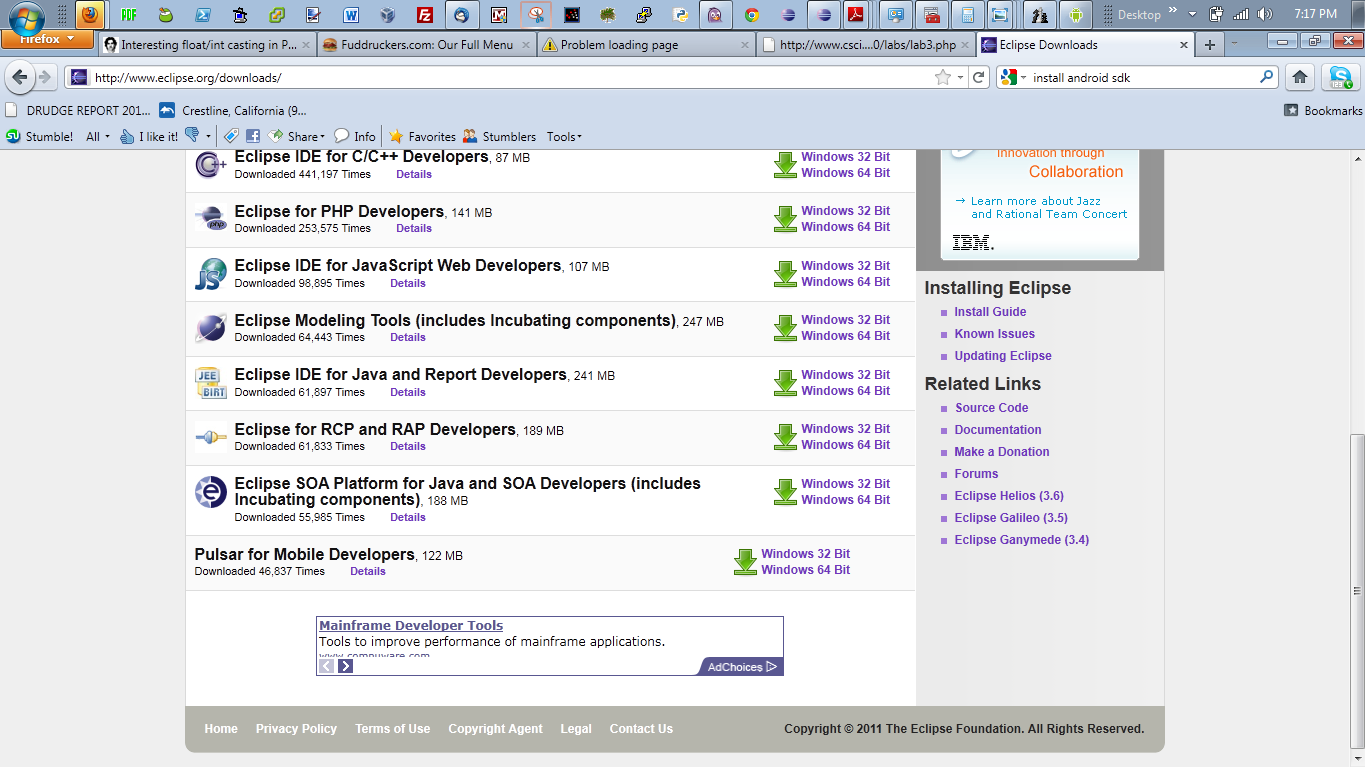
If necessary, install java by:

# yum install java

1. Install other android tools:   
   You may start android by going to the "tools" directory and execute "./android".   
   After starting android, check the "Installed packages". Then click on "Available packages" and install all of them. After the installation, check your "Installed packages" again. Record the packages in your report.



1. If necessary, install eclipse:
   * Go to "http://www.eclipse.org/", click "Download Eclipse", choose "Eclipse for RCP and RAP Developers: Linux 32 Bit", and download the package into a local directory, say, "/apps/downloads".



* + Unpack the downloaded package into "/apps" by:

$ cd /apps   
$ gunzip -c /arnold/download/eclipse-rcp-helios-SR2-linux-gtk.tar.gz | tar xvf -

Then start **eclipse** by:

$ cd eclipse   
$ ./eclipse

* + From the eclipse IDE, install the **Android Development Tools (ADT)**:
    - Click **Help**,--> **Install New Software**
    - In the "Work with" box, type "http://dl-ssl.google.com/android/eclipse/", hit "Enter", select all the "Development Tools", click "Next", click "Next", accept the license agreement to, and click "Finish" to install ADT.
  + After the ADT installation, restart **eclipse**. Click **Window** and you should see the entry "Android SDK and AVD Manager". Add the Android SDK directory by clicking **Preferences**, selecting **Android** and entering the location of your Android SDK. Now click on **Android SDK and AVD Manager** to proceed.
  + If you are new to **eclipse**, click on "Tutorials" and follow the instructions to create a **Hello World** application.

1. Learn Android development basics:   
   Go to the android development site at <http://developer.android.com/guide/index.html>. Read each topic to learn the basics of writing applications in the Android environment. In particular, read the "Graphics" topic to learn how to write simple **graphics** applications in Android.
2. Create **Hello World** in Android emulator:   
   Follow the instructions at

<http://developer.android.com/resources/tutorials/hello-world.html>

to create a "Hello World" application. ( Choose "Android 3.0 - API Level 11" as your target. )   
When you run your Android application, you may encounter an error saying that you need glibc-2.7 or later. If this occurs, you have to upgrade your glibc. I recommend you upgrade it to version 2.9 using rpm; you need to download the following packages from the Internet:

* + glibc-2.9-3.i686.rpm
  + glibc-devel-2.9-3.i386.rpm
  + glibc-utils-2.9-3.i386.rpm
  + glibc-common-2.9-3.i386.rpm
  + glibc-headers-2.9-3.i386.rpm

For example, you may get or search the packages from the site <http://rpm.pbone.net/>; when you make the search, check the options **Fedora 8, 9, 11, 12, 13**. After you have obtained all the packages, issue the following command to install glibc-2.9:

# rpm -Uvh glibc-\*

Check whether the correct version has been installed by the command,

# rpm -qa|grep glibc

Now run your "Hello World" Android Application again. What do you see ?

1. Develop an Android application. Do either **A** or **B** below, **extra credit** for both.   
   Whether you do A) or B), you need to create a simple graphical user interface. You may refer to the following link for Android UI:   
   <http://developer.android.com/guide/practices/ui_guidelines/index.html>

A) **Simple Animation of the Solar System using OpenGL**

* + You may refer to the following link for information about using OpenGL in the Android platform:

<http://developer.android.com/reference/android/opengl/package-summary.html>

* + Write a simple Android application that animates the solar system with nine planets revolving around the sun in 3D view, and the moon revolving around the earth. The scene should be in 3D view and has lighting effects. There should be two buttons that allow a user to click on, one to pause the system and one to resume it.

B) **Simple Remote Calculator**

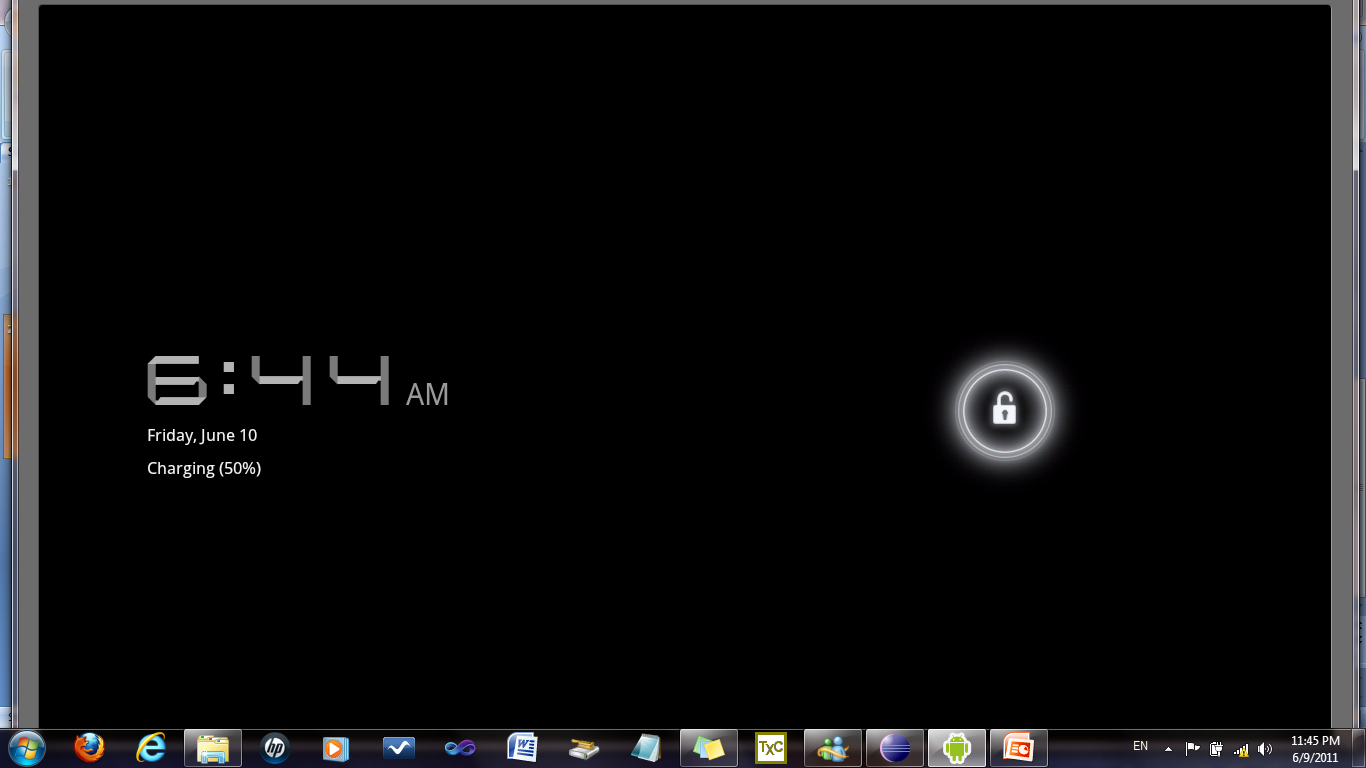
* + First create the graphical interface of a simple calculator that can handle integer arithmetic’s, including addition, subtraction, multiplication and division.
  + Make your Android calculator a client; a user clicks on the UI to input two integers and an operation (+, -, \*, / ). It sends the two integers and the operator to a remote java server program through RMI or other means. (The server can be on any platform. e.g. an Android, Windows, or Linux) The remote server program sends the result back to the Android client, which displays the result on the UI.   
    The following link shows a simple example of Android communication via UDP:

[i http://www.helloandroid.com/tutorials/simple-udp-communication-example](http://www.helloandroid.com/tutorials/simple-udp-communication-example)

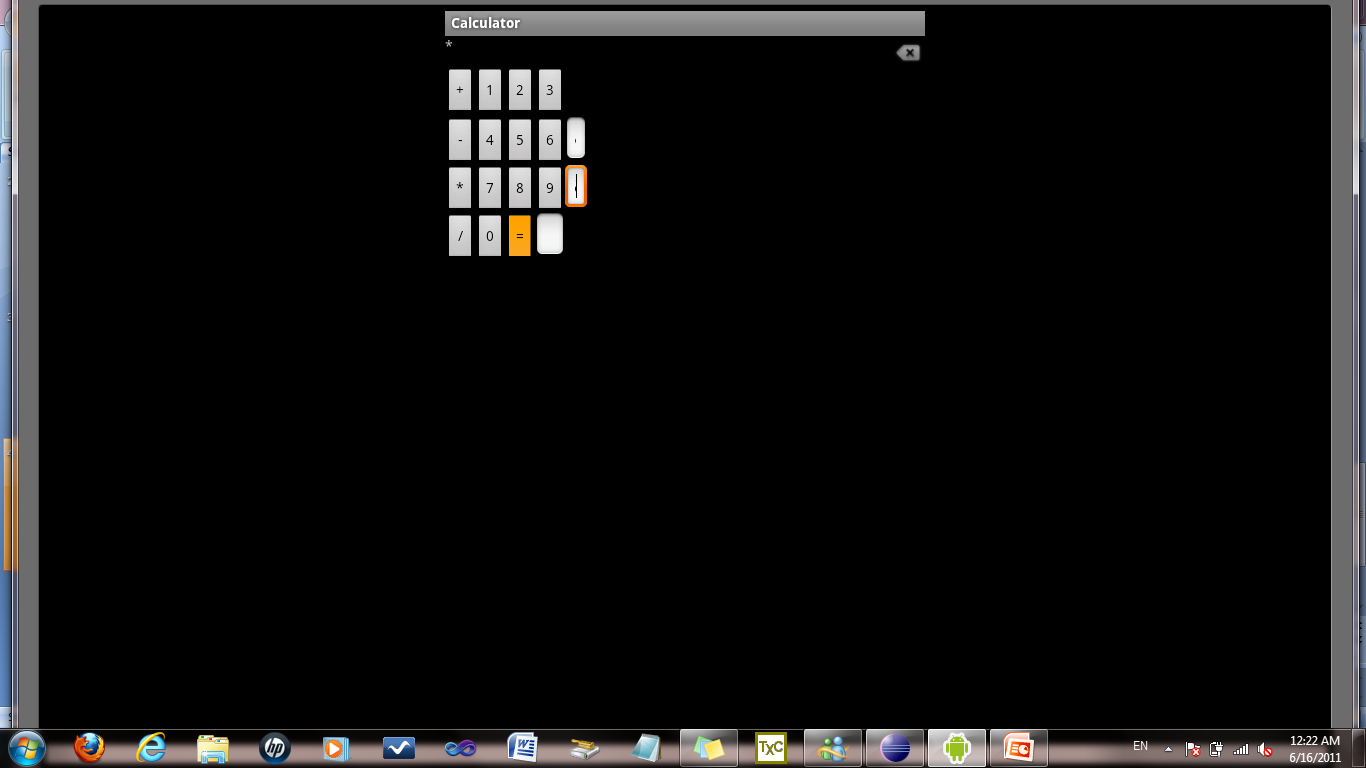
or click [here](http://cse.csusb.edu/tongyu/courses/cs660/labs/udp.html).   
The following link shows a simple example of Android communication via TCP:

<http://www.helloandroid.com/tutorials/simple-connection-example-part-ii-tcp-communication>

## Test of Android







# Conclusions

The ability to communicate between different systems over a common network or via the Internet has greatly allowed the world to develop so many different applications that share information.