NetServ JUNOS Port Project

Fall 2011

Krishnan Rajeswar (kr2428)

with

Jae Woo Lee

Table of Contents

1. Juniper M7i - Configuration and Access Info

M7i Router Description

The M7i Multiservice Edge Router is a complete routing system that provides ATM, channelized, Ethernet, IP services, and SONET/SDH interfaces for large networks and network applications, such as those supported by Internet service providers (ISPs).

Hardware Configuration

- 4 Physical Interface Cards (PICs). In addition to the PICs

- 1 Fixed Interface Card (FIC) with 1 Gigabit Ethernet port.

- RE400: routing engine with Celeron 400MHz

- Built-in tunnel interface on the Compact Forwarding Engine Board (CFEB) and Enhanced Compact Forwarding Engine Board (CFEB-E) provides tunneling services.

Software Configuration

The router is currently running on JUNOS 11.2R1.10.

Access Information

CRF has assigned a static IP address 128.59.19.160 with DNS entry netservjun1.cs.columbia.edu

The router can be accessed over SSH using the above IP address.

Management Interface (fxp0) - Configuration

The router's management ethernet interface has the static IP address, 128.59.19.160 assigned to it.

```
kr2428@netservjun1# show interfaces fxp0

kr2428@netservjun1# show interfaces fxp0  unit 0 {     family inet {

address 128.59.19.160/21;

   }

}

kr2428@netservjun1# show interfaces fxp0  unit 0 {
```

```
family inet {

address 128.59.19.160/21;

}

}
```

A default route has been added with the gateway set as 128.59.16.1, this provides access to internet from the router.

```
kr2428@netservjun1# show routing-options

static {

   route 0.0.0.0/0 {

next-hop 128.59.16.1;

resolve;

   }

}
```

2. How to upgrade JUNOS - M7i

Follow the below steps to upgrade the JUNOS software/ install any packages

Step1 : Enable ftp service on router by running following commands:

```
kr2428@netservjun1> configure  Entering configuration mode [edit] kr2428@netservjun1# set
system services ftp  [edit] kr2428@netservjun1# commit  commit complete [edit]
kr2428@netservjun1# exit  Exiting configuration mode kr2428@netservjun1> configure  Entering
configuration mode

[edit] kr2428@netservjun1# set system services ftp

[edit] kr2428@netservjun1# commit  commit complete

[edit] kr2428@netservjun1# exit  Exiting configuration mode

kr2428@netservjun1> configure  Entering configuration mode

[edit] kr2428@netservjun1# set system services ftp

[edit] kr2428@netservjun1# commit  commit complete

[edit] kr2428@netservjun1# exit  Exiting configuration mode

kr2428@netservjun1>  kr2428@netservjun1> configure  Entering configuration mode [edit]
kr2428@netservjun1# set system services ftp  [edit] kr2428@netservjun1# commit  commit
complete [edit] kr2428@netservjun1# exit  Exiting configuration mode kr2428@netservjun1>
```

Step2 : FTP to router and transfer the JUNOS file to /var/tmp as follows :

```
dyn-209-2-215-239:JUNOS SDK krisraj$ ftp netservjun1.cs.columbia.edu

Connected to netservjun1.cs.columbia.edu.

220 netservjun1 FTP server (Version 6.00LS) ready.

Name (netservjun1.cs.columbia.edu:krisraj): kr2428

331 Password required for kr2428.

Password:

230 User kr2428 logged in.

Remote system type is UNIX.
```

```
Using binary mode to transfer files.

ftp> cd /var/tmp

250 CWD command successful.

ftp> mput jinstall-11.2R1.10-domestic-signed.tgz

mput jinstall-11.2R1.10-domestic-signed.tgz [anpqy?]?
```

Step3 : Install the JUNOS using the following command

```
kr2428@netservjun1> request system software add /var/tmp/jinstall-11.2R1.10-domestic-signed.tgz

NOTICE: Validating configuration against jinstall-11.2R1.10-domestic-signed.tgz.

NOTICE: Use the 'no-validate' option to skip this if desired.

Checking compatibility with configuration

Initializing...

Using jbase-10.1R1.8

Verified manifest signed by PackageProduction_10_1_0

Verified jbase-10.1R1.8 signed by PackageProduction_10_1_0

Using /var/tmp/jinstall-11.2R1.10-domestic-signed.tgz

Verified jinstall-11.2R1.10-domestic.tgz signed by PackageProduction_11_2_0

Using jinstall-11.2R1.10-domestic.tgz

Using jbundle-11.2R1.10-domestic.tgz

Checking jbundle requirements on /

Using jbase-11.2R1.10.tgz

Verified manifest signed by PackageProduction_11_2_0

Verified jbase-11.2R1.10 signed by PackageProduction_11_2_0

Using /var/validate/chroot/tmp/jbundle/jboot-11.2R1.10.tgz

Using jruntime-11.2R1.10.tgz
```

```
Verified manifest signed by PackageProduction_11_2_0
Verified jruntime-11.2R1.10 signed by PackageProduction_11_2_0
Using jkernel-11.2R1.10.tgz
Verified manifest signed by PackageProduction_11_2_0
Verified jkernel-11.2R1.10 signed by PackageProduction_11_2_0
Using jcrypto-11.2R1.10.tgz
Verified manifest signed by PackageProduction_11_2_0
Verified jcrypto-11.2R1.10 signed by PackageProduction_11_2_0
Using jpfe-11.2R1.10.tgz
Using jdocs-11.2R1.10.tgz
Verified manifest signed by PackageProduction_11_2_0
Verified jdocs-11.2R1.10 signed by PackageProduction_11_2_0
Using jroute-11.2R1.10.tgz
Verified manifest signed by PackageProduction_11_2_0
Verified jroute-11.2R1.10 signed by PackageProduction_11_2_0
[: /var/validate/chroot/tmp/jbundle/jservices-11.2R1.10.tgz: unexpected operator
Validating against /config/juniper.conf.gz
mgd: commit complete
Validation succeeded
Installing package '/var/tmp/jinstall-11.2R1.10-domestic-signed.tgz' ...
Verified jinstall-11.2R1.10-domestic.tgz signed by PackageProduction_11_2_0
Adding jinstall...
Verified manifest signed by PackageProduction_11_2_0
WARNING:     This package will load JUNOS 11.2R1.10 software.
WARNING:     It will save JUNOS configuration files, and SSH keys
```

```
WARNING:     (if configured), but erase all other files and information
WARNING:     stored on this machine.  It will attempt to preserve dumps
WARNING:     and log files, but this can not be guaranteed.  This is the
WARNING:     pre-installation stage and all the software is loaded when
WARNING:     you reboot the system.
Saving the config files ...
NOTICE: uncommitted changes have been saved in /var/db/config/juniper.conf.pre-install
Installing the bootstrap installer ...
WARNING:     A REBOOT IS REQUIRED TO LOAD THIS SOFTWARE CORRECTLY. Use the
WARNING:     'request system reboot' command when software installation is
WARNING:     complete. To abort the installation, do not reboot your system,
WARNING:     instead use the 'request system software delete jinstall'
WARNING:     command as soon as this operation completes.
Saving package file in /var/sw/pkg/jinstall-11.2R1.10-domestic-signed.tgz ...
Saving state for rollback ...
kr2428@netservjun1>
```

Step 4 : Reboot the router

```
kr2428@netservjun1> request system reboot
Reboot the system ? [yes,no] (no) yes
Shutdown NOW!
Reboot consistency check bypassed - jinstall 11.2R1.10 will complete installation upon reboot
[pid 11168]
kr2428@netservjun1>
*** FINAL System shutdown message from kr2428@netservjun1 ***
System going down IMMEDIATELY
```

Step 6 : Verify the current installed software

```
kr2428@netservjun1> show version
Hostname: netservjun1
Model: m7i
JUNOS Base OS boot [11.2R1.10]
JUNOS Base OS Software Suite [11.2R1.10]
JUNOS Kernel Software Suite [11.2R1.10]
JUNOS Crypto Software Suite [11.2R1.10]
JUNOS Packet Forwarding Engine Support (M/T Common) [11.2R1.10]
JUNOS Packet Forwarding Engine Support (M7i/M10i) [11.2R1.10]
JUNOS Online Documentation [11.2R1.10]
JUNOS Voice Services Container package [11.2R1.10]
JUNOS Border Gateway Function package [11.2R1.10]
JUNOS Services AACL Container package [11.2R1.10]
JUNOS Services LL-PDF Container package [11.2R1.10]
JUNOS Services PTSP Container package [11.2R1.10]
JUNOS Services Stateful Firewall [11.2R1.10]
JUNOS Services NAT [11.2R1.10]
JUNOS Services Application Level Gateways [11.2R1.10]
JUNOS Services Captive Portal and Content Delivery Container package [11.2R1.10]
JUNOS Services RPM [11.2R1.10]
JUNOS Services HTTP Content Management package [11.2R1.10]
JUNOS AppId Services [11.2R1.10]
JUNOS IDP Services [11.2R1.10]
JUNOS Services Crypto [11.2R1.10]
```

JUNOS Services SSL [11.2R1.10]

JUNOS Runtime Software Suite [11.2R1.10]

JUNOS Routing Software Suite [11.2R1.10]

3. How to Build and Deploy a JUNOS SDK Application

Contents

This page covers the instructions for the following things -

1. How to setup the Virtual Build Environment(VBE) and the backing sandbox.
2. Build and package a sample application, sync-packetproc-data, a Services SDK application
3. How to install the sample application on the Juniper router
4. Test it

For the complete Online Video Tutorial on JUNOS SDK, check out - JUNOS SDK Online Training

For the complete Documentation on JUNOS SDK, check out JUNOS SDK: Main Page

1. Setup Virtual Build Environment(VBE) and Backing Sandbox

The VBE is the development environment in which the sample applications are developed. It is essentially a FreeBSD version 7. We run this on a VM Player on a local system and install the "backing sandbox"(contains essentially all the JUNOS SDK libraries to which the applications developed in a "application sanbox" links to) in it.

Step 1 : Download the JUNOS SDK latest release from here. Please note the JUNOS SDK version should match the JUNOS version running on the router in which the application will be deployed.

Step 2 : Install a VM Player/VM Fusion to host the VBE on your local development system.

Step 3 : The VBE image (eg. junos-sdk-os-11.2R1.X.tgz) is available along with JUNOS SDK download. Follow the instructions here to complete the VBE setup.

Step 4 : Next we install the backing sandbox and toolchain packages in the VBE. Detailed instructions are available here.

Step 5 : Update the PATH variable (/home/user/.profile) with the "/usr/local/junos-sdk/11.2R1.X/bin", to access the executables in the backing sandbox.

2. Build and package a sample application

Before we can go ahead and build the sample application there is one important step. We require a authorized certificate signed by Juniper which needs to be associated with every application developed before it can be installed on a Juniper router.

Step 0 : Since the signing process with Juniper has already been completed the only thing to do for this is to copy the following files to "/usr/local/junos-sdk/certs" -

```
columbiauniv-netserv-1.pem

columbiauniv-netserv-1_key.pem
```

Packet Processing Services Application - sync-packetproc-data

For the purpose of demonstration we choose a sample packet processing application called sync-packetproc-data which was developed using the Services SDK APIs. This application demonstrates three different ways of creating data loops to handle traffic in a Services SDK daemon.

Step 1 : Login to the VBE.

Step 2 : Run the following command to create a application sandbox with code for the sample app - sync-packetproc-data

```
$mksb -n <sanbox_name> sync-packetproc
```

Step 3 : The application code resides in <sandbox_name>/src/sbin/ . To build the code base -

Step 3.1 : `$ cd <sanbox_name>/src`

Step 3.2 : `$ mk-xlr` (If building an RE application the command will be just 'mk'. The make command is corresponding to the architecture on which the application will be running on. The default is i386 for the RE.).

If the compilation is successful, we build the application package -

Step 3.3 : `$ cd release`

Step 3.4 : `$ mk`

This 'mk' will package the application code base along with the authorization certificate and store the package files(sync-packetproc-data-11.2I20111014_0652_user.tgz) at '<sandbox_name/ship/'.

Alternatively we can combine Steps 3.2-3.4 by running -

```
$ mk-xlr release                    in <sandbox_name>/src
```

For more details on this process the JUNOS SDK documentation can be found here.

3. Deploy the sample application on the Juniper router.

Before the application package can be installed on the router, the following configuration is required to facilitate verifying the signed certificate associated with the package -

Step 0 : Add the following configuration -

```
> conf

# set system extension provider columbiauniv license-type evaluation deployment-scope private

# commit
```

Step 1 : FTP the package file (sync-packetproc-data-11.2I20111014_0652_user.tgz) to the router. The default location on the router is /var/home/<username>.

Step 2 : To install the package run the following command -

```
> request system software add sync-packetproc-data-11.2I20111014_0652_user.tgz
```

To verify that the package has been installed use the following command -

```
 > show version     .     .     .     SDK Your Net Corp. PacketProc Dataplane Component
[11.2I20111009_2324_user]    >
```

Now the package is only installed. The application daemon is still not running on the MS-PIC. The following steps does not pertain to an RE application.

Step 3 : To download the package to the MS-PIC(currently residing in FPC 1 and PIC 2) and start the daemon add the following configuration under 'chassis'-

```
   chassis {      fpc 1 {        pic 2 {          adaptive-services {             service-package {
  extension-provider {                  control-cores 1;//The number of control-cores and data-cores
can configured as per requirement. But there can be a maximum of only 8 of them together with
atleast 1 control-core.                 data-cores 6;                  package sync-packetproc-data;
        syslog {                    daemon any;                    pfe any;                    external any;
            kernel any;                 }             }            }         }      }    }
```

Step 4 : To verify that the daemon is running on the MS-PIC use the following command -

```
> show extension-provider system processes wide | grep packetproc

  2816  ??  R    56108:51.54 columbia /opt/sdk/sbin/packetproc-data -N
```

Alternatively we can login to the MS-PIC and check processes running there -

```
> show route forwarding-table all | grep pc-1/2/0

  10.0.0.34/32       user     0                  ucst   580     4 pc-1/2/0.16383

  128.0.3.17/32      user     1                  ucst   580     4 pc-1/2/0.16383

  20.0.0.34/32       intf     1 20.0.0.34         ucst   582     2 pc-1/2/0.16384

> telnet routing-instances _juniper_private1_ 128.0.3.17
```

```
Trying 128.0.3.17...

Connected to 128.0.3.17.

Escape character is '^]'.

ms12 (ttyp0)

login: root

root@ms12% ps ax | grep packetproc

2816 ?? R 56227:30.20 /opt/sdk/sbin/packetproc-data -N 14588 p0 S+ 0:00.05
```

Step 3 is required only for Services SDK application in order to facilitate its download to the MS-PIC. Also some applications might require additional configurations to startup. But this is specific to each application based on how they are developed and these configuration will be part of the application documentation.

4. Test sync-packetproc-data

This is a simple services application running on a MS-PIC. In addition to showing the different ways to spawn data loops, this application does some simple packet processing. If a packet is received on unit 1 of the MS interface, it will be forwarded to unit 2 of the same MS interface and vice versa. For testing, we will send packets to unit 1 and verify that it comes back through unit 2.

Step 1: Configure the MS interface. The following is a sample config -

```
interface ?{

  ms-1/2/0 {

    unit 0 {

      family inet {

        address 7.7.7.7/32;

      }

    }

    unit 1 {

      family inet {

        address 8.8.8.8/32;

      }
```

```
   }

   unit 2 {

     family inet {

       address 9.9.9.9/32;

     }

   }

 }

}
```

In order to demonstrate the working of this application we will create a virtual routing (VR) instance on the router and associate it with one of the logical interface of the MS-PIC (unit 1 in this case). This way all the units configured are part of the Master VR except unit 1 which belongs to the new created VR instance.

Step 2 : Create a virtual routing instance named 'vr1' and associate it with unit 1 of ms-1/2/0 -

```
   # set routing-instances vr1 instance-type virtual-router

   # set routing-instances vr1 interface ms-1/2/0.1
```

Step 3 : Add a next-table route to forward all traffic destined to unit 1 of ms-1/2/0 to 'vr1' routing table for lookup.

```
   #  set routing-options static route 8.8.8.8/32 next-table vr1.inet.0
```

Step 4 : Also provision the 'vr1' to direct traffic it receives to unit 1 of ms-1/2/0

```
   # set routing-instances vr1 routing-options static route 0/0 next-hop ms-1/2/0.1
```

Now that the traffic routes to the MS interface have been setup, we send packets to ms-1/2/0.1 using ping.

Step 5 : Ping ms-1/2/0.1

```
    > ping count 3 8.8.8.8

    PING 8.8.8.8 (8.8.8.8): 56 data bytes

    64 bytes from 8.8.8.8: icmp_seq=0 ttl=65 time=5.251 ms

    64 bytes from 8.8.8.8: icmp_seq=1 ttl=65 time=4.828 ms

    64 bytes from 8.8.8.8: icmp_seq=2 ttl=65 time=4.320 ms
```

```
3 packets transmitted, 3 packets received, 0% packet loss

round-trip min/avg/max/stddev = 4.320/4.800/5.251/0.381 ms
```

Step 6 : To verify that the packets did return through ms-1/2/0.2, we check the interface counters -

```
> show interfaces ms-1/2/0.1          Logical interface ms-1/2/0.1 (Index 73) (SNMP ifIndex 511)
    Flags: Point-To-Point SNMP-Traps Encapsulation: Adaptive-Services          Input packets : 0
    Output packets: 3          Protocol inet, MTU: 9192               Flags: Sendbcast-pkt-to-re, Is-
Primary, Receive-options, Receive-TTL-Exceeded             Addresses, Flags: Is-Default Is-Primary
     Local: 8.8.8.8

> show interfaces ms-1/2/0.2          Logical interface ms-1/2/0.2 (Index 74) (SNMP ifIndex 512)
    Flags: Point-To-Point SNMP-Traps Encapsulation: Adaptive-Services          Input packets : 3
    Output packets: 0          Protocol inet, MTU: 9192             Flags: Sendbcast-pkt-to-re, Receive-
options, Receive-TTL-Exceeded             Addresses, Flags: Is-Primary             Local: 9.9.9.9
```

4. JVM on JUNOS

Introduction

Inorder to port the NetServ Architecture to JUNOS, the essential feature is to able to run JVM on either the RE (x86) or the Services place(MIPS). To test, this a simple C program called 'invoke.c' is used, that launches JVM and displays "Hello World" from the main method of a Java program. This document explains the approach taken to achieve this goal.

Using the existing libjvm.so

In the FreeBSD VBE, this can be done by linking with the precompiled library libjvm.so. The JDK version installed is Diablo JDK, which is a Sun certified JDK supported by FreeBSD foundation and installed from precompiled binaries. The following are the steps to run a C program (invoke.c) that loads a Java virtual machine and calls the Prog.main method defined as follows -

```
public class Prog {      public static void main(String[] args) {           System.out.println("Hello World "
+ args[0]);      }  } public class Prog {

    public static void main(String[] args) {

        System.out.println("Hello World " + args[0]);

    }

 }
```

```
[root@junos-sdk-vm /usr/home/user]# export LD_LIBRARY_PATH=/usr/local/java/jre/lib/i386/server

[root@junos-sdk-vm /usr/home/]# kldload sem

[root@junos-sdk-vm /usr/home/user]# javac Prog.java

[root@junos-sdk-vm /usr/home/user]# gcc -I/usr/local/java/include
-I/usr/local/openjdk6/include/freebsd/ -L/usr/local/diablo-jdk1.6.0/jre/lib/i386/server -ljvm invoke.c

[root@junos-sdk-vm /usr/home/user]# ./a.out

Hello World  from C!
```

On adding this library to a sample JUNOS SDK application using the instructions provided here, did not work as JUNOS compiler was unable to detect the precompiled library (as expected).

Building libjvm.so from source

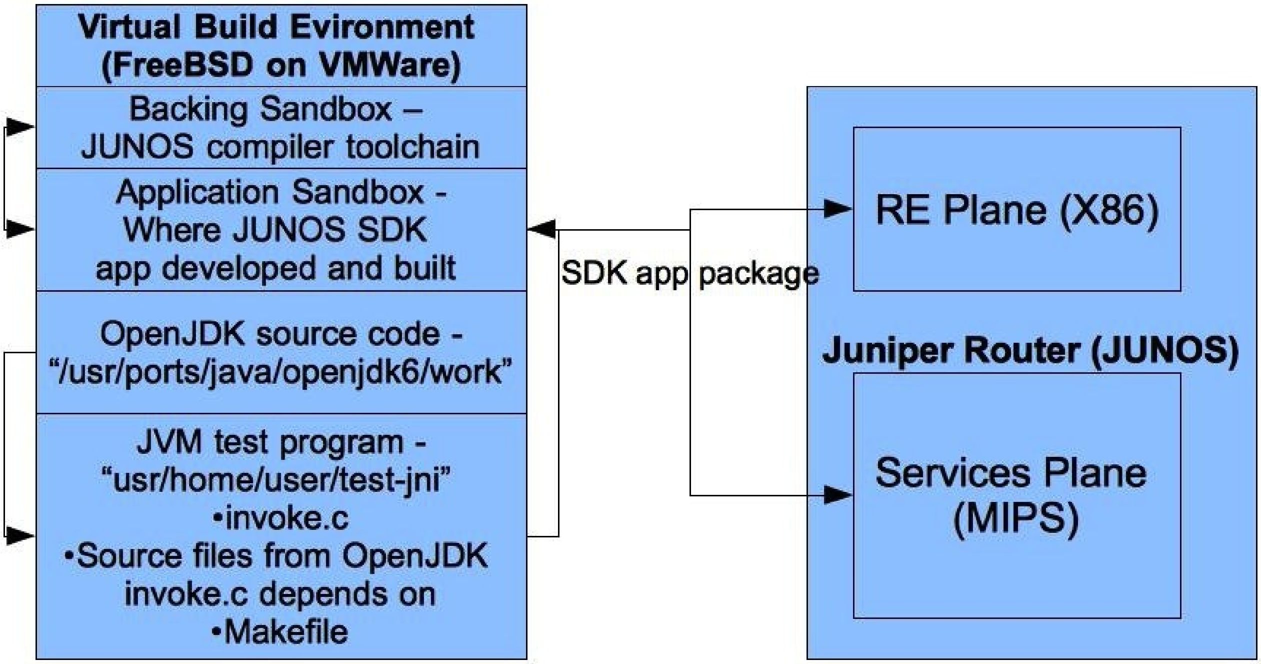
So inorder to run JVM on JUNOS, the libjvm.so has to be compiled from the source files using the JUNOS tool chain. Therefore we need the source code of JDK, which is available from OpenJDK. As most other software packages, OpenJDK can be downloaded and installed using "ports" in FreeBSD (Use 'portsnap' to update the ports tree). This link explains the working of 'ports' in detail.

```
[root@junos-sdk-vm /usr/ports/java/openjdk6]# make
```

This initiates the download and extraction of the entire source code of OpenJDK onto "/usr/ports/java/openjdk6/work" and starts to build it.This includes compiling different bundles required to install Java in a system, one among which is libjvm.so.

The versions of JDKs available in FreeBSD are the linux-jdk (JDK built for linux that can be run on FreeBSD) and native jdk(Oracle released JDK source code). Both these versions do not serve our requirements as the former introduces an unnecessary layer of abstraction(Linux on FreeBSD) and the latter cannot be used as it is Oracle released, and requires prior certification to use.
This thread has a detailed discussion on different types of JDKs available on FreeBSD.

Invoke JVM on JUNOS


Virtual Build Evironment (FreeBSD on VMWare)
Backing Sandbox – JUNOS compiler toolchain
Application Sandbox - Where JUNOS SDK app developed and built
OpenJDK source code - “/usr/ports/java/openjdk6/work”
JVM test program - “usr/home/user/test-jni”
•invoke.c
•Source files from OpenJDK invoke.c depends on
•Makefile
SDK app package
RE Plane (X86)
Juniper Router (JUNOS)
Services Plane (MIPS)

Objective

In order to test that JVM can be invoked using a native code on JUNOS, make invoke.c part of a SDK application and successfully invoke it.

Approach

Both JUNOS and OpenJDK compilation processes are complicated. Incorporating one build process to another is not straight forward as there are many external dependencies. Taking a top-down approach, since the objective is to run invoke.c, we can find the source files on which invoke.c depends on. Include those files (.cpp and .hpp) into the SDK application along with invoke.c and compile the whole thing using JUNOS tool chain along with the required flags.

Result

After resolving thousands of dependencies, all the source files (500+) from the OpenJDK source code required to successfully compile and get an executable for invoke.c, was found. On running this executable, it was seen that it tries to load libjava.so. The source code required to compile libjava.so was also narrowed down (31 files). The build log for OpenJDK source code which was helpful in resolving the dependencies and the Makefile I wrote to build the executable is attached.

Future Work

Given the Makefile used to build invoke.c and its dependencies, it will be easy to incorporate these files onto a JUNOS SDK application and try to compile and run the same. Due to the time constraint this part could not be completed.

Alternative Solution

Juniper is developing a Virtual Engine Environment with an SDK for it.Virtual Engine is a module which sits on a Juniper box, and acts as an interface between any guest OS and JUNOS. And the SDK for it allows development in Java also. This means any application that is implemented in linux or any OS can be deployed using VE SDKs. This provides an easy solution for porting NetServ to JUNOS as NetServ is already implemented in linux. Unfortunately this solution is expected to be released only late 2012 or early 2013.

5. Makefile

```
CC = /usr/bin/gcc
CXX = /usr/bin/g++

INCLUDES = -I. -I/usr/home/user/test-jni/hotspot/src/share/vm \
                    -I/usr/home/user/test-jni/hotspot/src/share/vm/prims \
                    -I/usr/home/user/test-jni/hotspot/src/cpu/x86/vm \
                    -I/usr/home/user/test-jni/hotspot/src/os/bsd/vm \
                    -I/usr/home/user/test-jni/hotspot/src/os_cpu/bsd_x86/vm \
                    -I../test-jni \
                    -I/usr/home/user/test-jni/hotspot/src/share/vm/runtime \
                    -I/usr/ports/java/openjdk6/work/hotspot/src/share/vm/opto/


CXXFLAGS = -D_ALLBSD_SOURCE -D_GNU_SOURCE -DIA32 -DPRODUCT
-DHOTSPOT_RELEASE_VERSION="\"20.0-b11\"" -DHOTSPOT_BUILD_TARGET="\"product\""
-DHOTSPOT_BUILD_USER="\"root\"" -DHOTSPOT_LIB_ARCH=\"i386\"
-DJRE_RELEASE_VERSION="\"1.6.0-b23\"" -DHOTSPOT_VM_DISTRO="\"OpenJDK\"" -O2
-fno-strict-aliasing -pipe -DTARGET_OS_FAMILY_bsd -DTARGET_ARCH_x86
-DTARGET_ARCH_MODEL_x86_32 -DTARGET_OS_ARCH_bsd_x86
-DTARGET_OS_ARCH_MODEL_bsd_x86_32 -DTARGET_COMPILER_gcc -DCOMPILER2 -DCOMPILER1
-fno-rtti -fno-exceptions -pthread -fcheck-new -m32 -march=i586 -pipe
-DTARGET_OS_FAMILY_bsd -DTARGET_ARCH_x86 -DTARGET_ARCH_MODEL_x86_32
-DTARGET_OS_ARCH_bsd_x86 -DTARGET_OS_ARCH_MODEL_bsd_x86_32
-DTARGET_COMPILER_gcc -DCOMPILER2 -DCOMPILER1 -fPIC -fno-rtti -fno-exceptions
-pthread -fcheck-new -m32 -march=i586 -pipe -O3 -fno-strict-aliasing
-DVM_LITTLE_ENDIAN -Werror -Wpointer-arith -Wconversion -Wsign-compare

BASEDIR = /usr/home/user/test-jni
SRCDIR = $(BASEDIR)/src
OBJDIR = $(BASEDIR)/objdir
DEPSDIR = $(BASEDIR)/dependencies

vpath %.cpp ../test-jni/src


src = accessFlags.cpp \
          gcLocker.cpp \
          jniHandles.cpp \
          ….

objects = $(patsubst %.cpp, $(OBJDIR)/%.o,$(src))
```

```
invoke : $(OBJDIR)/invoke.o $(objects) $(OBJDIR)/bsd_x86_32.o
	$(CC) -o invoke $? -pthread -lstdc++
	$(CXX) -m32 -march=i586 -Xlinker -O1  -shared -fPIC -Xlinker --version-
script=mapfile_reorder -Xlinker -soname=libjvm.so -o libjvm.so $(objects)

$(OBJDIR)/invoke.o : $(SRCDIR)/invoke.c
	$(CC) $(INCLUDES) -c $? -o $@

$(OBJDIR)/%.o : $(SRCDIR)/%.cpp
	@echo $<
	@$(CXX) $(INCLUDES) $(CXXFLAGS) -c $< -o $@

$(OBJDIR)/bsd_x86_32.o : $(SRCDIR)/bsd_x86_32.s
	$(CC) -x assembler-with-cpp -m32 -march=i586 -c $< -o $@
```