NORTHWESTERN
UNIVERSITY

Electrical Engineering and Computer Science Department

Technical Report
EECS-499

Abhinav Kannan
abhinavkannan2013@u.northwestern.edu

Prof. Peter Dinda
pdinda@northwestern.edu

Introduction to the Graphical User Interface for the Palacios VMM

TABLE OF CONTENTS

1 INTRODUCTION 3

2 REQUIREMENTS 3

3 BUILDING THE GUI MODULE 6

4 CODE STRUCTURE 8

5 USING THE GUI 10

6 FUTURE EXTENSIONS 13

ACKNOWLEDGEMENTS 13

REFERENCES 14

1. INTRODUCTION

The V3VEE project (v3vee.org) has created a virtual machine monitor framework for modern architectures (those with hardware virtualization support) that enables the compile-time creation of VMMs with different structures, including those optimized for computer architecture research and use in high performance computing. V3VEE began as a collaborative project between Northwestern University and the University of New Mexico. It currently includes Northwestern University, the University of New Mexico, the University of Pittsburgh, Sandia National Labs, and Oak Ridge National Lab. Research publications on using Palacios can be found on the V3VEE project web site. For a quick start guide to Palacios refer [1]. A discussion of our and others' Palacios-based research is beyond the scope of this document.

This document describes the new Graphical User Interface (GUI) module of Palacios. The GUI module is designed to help users to easily manage the creation and maintenance of Virtual Machine instances. The interface is built on top of the existing Palacios command line tools and serves as a graphical front-end engine for these tools. The following sections provide a detailed description about the module. The current version of the GUI module is 1.0.

2. REQUIREMENTS

In order to run Palacios you will need either suitable hardware or an emulator, as well as a small set of required build software. Almost any recent machine can be used for testing, and if none is available a free emulator can be used. The needed build software is all free and is already a part of typical Linux distributions, or can be readily installed. This document does not describe the setup for Palacios VMM. For a quick startup guide refer [1].

The GUI module is built using the Qt framework [2]. Qt is a C++ framework developed by Nokia. The latest version of Qt can be downloaded from their website. Qt is a very popular framework and many applications including VirtualBox use Qt. A detailed discussion of Qt is beyond the scope of the document but we will discuss components which are relevant to our project. The module is available as part of the Palacios release which can be downloaded from http://www.v3vee.org/palacios. The source is available in the form of a public git repository or source snapshot releases.

To build the GUI module you would need the following libraries:

1 The GUI is built using the Qt framework. Qt is a popular framework for building desktop and embedded applications and is supported by Nokia [2]. It is written in C++ language. For more details in Qt kindly visit the website. Qt is a very well documented software with a lot of examples and community forums for reference. You would need the Qt

open-source linux distribution which is available for free on Qt's website. The GUI module currently uses version 4.8.4 but can be easily upgraded to the latest version.

2. LibVNCServer library which is used to support VNC controls in Palacios [3].
3. GnuTLS and supporting libraries to support VNC controls in Palacios [4].

We now describe the building process for these libraries.

2.1 Building Qt Libraries

Qt can be built either as a static library or a dynamic library. By using static libraries, Qt is built as part of the application executable which frees you from the trouble of linking at compile time. However static linking increases the size of the executable. If you decide to use dynamic linking you must make sure to redistribute Qt libraries along with application executable. This has the advantage of a smaller executable and gives the flexibility of compiling application code separate from Qt.

Currently the GUI module is build using static Qt linking but we aim to support dynamic linking in the next release. We describe the static build process in this document. For more details visit Qt's website.

1. Untar the Qt open source package which you downloaded into some directory on you machine
2. cd to /path/to/Qt
3. Execute the configuration script. The -static option is used to indicate static linking
   ./configure -static -prefix <path where you want to install the Qt libraries> <additional options>.
   This process may take some time
4. Once Qt is configured execute the command,
   make install
   This process may take a long time

After completing the above steps the Qt libraries will be available in the *prefix* directory. If prefix is not given the Qt is installed in */usr/local*. The next release will feature shared library support for Qt libraries. As of this release it is recommended to build Qt libraries once at the beginning of the setup.

2.2 Building LibVNCServer

LibVNCServer is available freely [3]. The details of installation and various configuration options are available in the README and INSTALL files of LibVNCServer. The basic setup for building is similar to Qt,

```
1   cd /path/to/LibVNCServer
2   ./configure -prefix=<your path>
3   make
4   make install
```

After the 4th step the *libvncserver* library will be available in the prefix path. This will be used in the GUI module to support the VNC interactions.

2.3 Building GnuTLS

GnuTLS is used by the GUI application to enable secure VNC communication across the network. GnuTLS requires additional support libraries *libgcrypt* and *libgpg-error*. The source of these libraries are available from [4]. Some of the versions of *libgnutls* works with only certain other versions of *libgcrypt* and *libgpg-error*. It is important to check the version dependencies before building the libraries. The build process for the libraries is again similar to the previous procedures.

```
1   cd /path/to/gpg-error
2   ./configure -prefix <path>
3   cd /path/to/gcrypt
4   ./configure --with-gpg-error-prefix=<path of gpg-error> -prefix=<path>
5   cd /path/to/gnutls
6   ./configure --with-libgcrypt-prefix=<path of gcrypt> -prefix=<path>
7   make
```

Once the make is complete the library will be available in the prefix path. It would be better if you put all the required libraries in the same path as it would be easier to find the libraries at compile time.

The next section describes how to build the GUI module and how to link the above libraries.

3. BUILDING THE GUI MODULE

After successfully compiling all the required libraries, you can move on to building the GUI module. Qt provides a utility called *qmake* which helps to automatically generate the Makefile for the project. Instead Qt requires a *.pro* file which will be used by qmake to create the proper Makefile for the project. We will describe how to create a .pro file which will be needed for the project.

A simple structure of the .pro file is given below

```
TEMPLATE = app
TARGET = <app name>
QT += core gui xml <other qt modules if required>
CONFIG = debug_and_release
INCLUDEPATH += <include paths for header files if required>
LIBS += <list of libraries>
HEADERS += <list of header files>
SOURCES += <list of source files>
RESOURCES  += <list of resources, ex. images>
```

For the purpose of this project it is sufficient to understand the above .pro structure. There are other extensive options which can be added to the .pro file ex. if you need to support multiple platforms and need to include platform specific information you can use *conditional scoping* of Qt. For more details visit the website.

Most of the above labels are self-explanatory. The QT label specifies which Qt modules are required by the application. In most of the cases the QtCore (core) and QtGui (gui) modules are needed. Other modules can be included by adding it to the QT label. The CONFIG label specifies the mode of building Qt. It is useful to have separate Debug and Release configurations to easily manage the development process. The RESOURCES label is also interesting because it is used to include resources such as images into the application. The resource file is define with a .qrc extension and the format is similar to an Xml file, ex.

```
<RCC>
        <qresource prefix=”/images”>
                <file></file>
        </qresource>
</RCC>
```

The LIBS label should be used to include any shared libraries which might be used for the application. Since we are using *libvncserver* and *libgnutls*, they will be added to the label.

To generate the Makefile you need to run qmake and pass in the .pro file as argument

qmake Palacios.pro

Depending on the configuration specified in the CONFIG label, qmake will generate the required files ex. if the configuration is set as above qmake generates three Makefiles (Makefile, Makefile.Debug and Makefile.Release). It is suggested to use the *debug_and_release* mode. Once the makefiles are generated the GUI module can be built using

make <-f Makefile name>

To run the application type the following command

./Palacios

This will launch the GUI application.

3.1 Additional build instructions

In case you do not want to build Qt source we ship pre-compiled libraries for all the Qt libraries. However, these are compiled for a particular Linux distribution and may not work. It is recommended to download the Qt source code and statically compile the libraries following the above steps in case GUI module fails to compile with the pre-compiled libraries. The precompiled libraries are available in the *libraries* folder of the distribution. We also provide the qmake utility. When using precompiled libraries you would need to make the following changes. You need to make these changes to both Makefile.Debug and Makefile.Release files.

- Change INCPATH to point to the *libraries* folder for all the include paths
    - –Ilibraries/mkspecs/linux-g++-64
    - –Ilibraries/include, –Ilibraries/include/QtCore, -Ilibraries/include/QtGui and –Ilibraries/include/QtXml
- Change LFLAGS = -m64, -Wl, -rpath, /path/to/libraries
- Change LIBS to point to the *libraries* path for all the shared libraries
    - Change –L/usr/X11R6/lib64 to –L/usr/lib64

Run make. If the application fails to compile, it is suggested to use the previous method.

4. CODE STRUCTURE

We now consider the directory structure used by the GUI module.

4.1 Directory structure

The GUI code is included in the *gui* directory of the Palacios source code. The structure is given below

```
palacios/
    gui/
        palacios/
            All the main source files are located here

            vnc_module/
                All the source files necessary for VNC communication

        libs/
            Used to store all the libraries used by Palacios
```

The main source files are described below:

- gui/palacios/newpalacios {.h, .cpp}

  This is the main source and header file for the application. This file defines and implements the main window of the application including UI creation and event handling.

- gui/palacios/vm_console_widget {.h, .cpp}

  Define the UI widget to show the stream, console and vnc views of Palacios

- gui/palacios/vm_info_widget {.h, .cpp}

  Define the main widget of the application. This widget hosts the console widget and also shows information about the VM created by parsing the configuration file.

- gui/palacios/vm_threads.cpp

  Defines background threads used for loading, adding and deleting VM instances

- gui/palacios/defs.h

  Defines string constants for all UI labels used by the application

- gui/palacios/vnc_module

  This directory hosts files which are used to support VNC interaction in Palacios. We reuse an existing Qt VNC solution provided by KDE in their KRDC application. The source was freely available without any license agreement [5].

The next section describes how to use the GUI module.

5. USING THE GUI

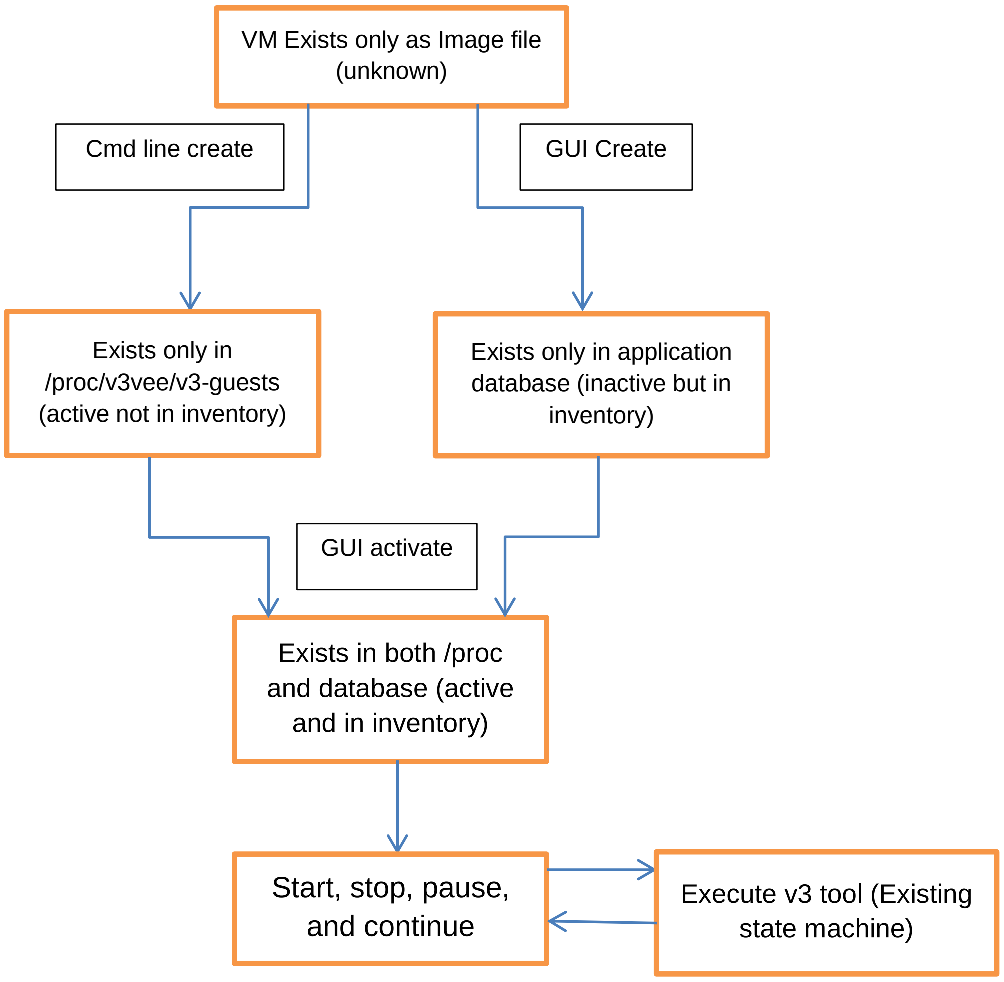
The application is built on top of the existing command line tools provided by Palacios. This means that the application is effectively a frontend and all the background processing is still done by the command line tools (or v3 tools). The goal of the application therefore is not to replace the existing tools but provide a simple visual interface to the users to easily manage their virtual machines. We describe the important features of the application.

5.1 Startup

When the application is launched all the VMs currently available on the machine are displayed in a side pane. We separate the VMs on the machine into three categories

- Active VM in inventory
- Inactive VM in inventory
- Active VM but not in inventory

The state machine for Palacios is augmented with new states to support the GUI operations.


VM Exists only as Image file (unknown)
Cmd line create
GUI Create
Exists only in /proc/v3vee/v3-guests (active not in inventory)
Exists only in application database (inactive but in inventory)
GUI activate
Exists in both /proc and database (active and in inventory)
Start, stop, pause, and continue
Execute v3 tool (Existing state machine)

There are two main sources for loading VMs, the /proc file and a text file (future versions will support an SQL database to store information on VMs). It is important to understand the purpose of these files.

The /proc/v3vee/v3-guests file is used by Palacios to hold information of the VMs created on the machine using the command line tool v3_create. Any VM created using v3_create will update the /proc file with the name of the VM and the device file (/dev/v3-vm#).

<Vm name1> /dev/v3-vm0
<Vm name2> /dev/v3-vm1
...

The text file (virtual_machines_list.txt) also stores information about VMs created from the application. An entry in the text file is shown below

<Vm name>,<path of config file>,<dev file name>,<current state of VM>,<path of image file>

- Vm name - name of the VM
- Config file - path of the config file used to create the VM
- Device file - /dev/v3-vm# used to control the VM
- State - stopped, paused, running
- Image file - path of the image file used to create the VM

On startup, both the /proc file and the text file is read and the VMs are loaded into the application.

5.2 Creating a VM

There are two ways a user can create a VM, using the GUI or using the v3_create utility.

5.2.1 Creating a VM using GUI

To create a new VM, users can either press the “New VM” icon on the toolbar or go to File and click “New”. On clicking the button a wizard is launched which will guide you through the VM creation process. The wizard prompts you to enter the name of the VM (unique name for easy identification), the path of the config file and the path of the image file. On completing the wizard a new VM entry is added to the *inactive inventory* of the application. At this point the VM entry is added to the application database (currently a text file). All VMs created using the wizard are initially inactive. To use the VM it must be activated. Activation can be done from the menu or toolbar. Once the VM is activated it is added to the *active inventory*. This stage corresponds to launching v3_create to create the VM. When the VM is created an entry is added to the /proc file and the database is updated with the device file information.

5.2.2 Creating VM using command line

A VM can also be created without the GUI by directly running v3_create. This would update the /proc file but the application will not be aware of such a VM. When the application is launched, it queries the /proc file and compares with the internal database. Any VM not found in the database is added to the *active but not in inventory* list. In order to use these VMs the users must activate them. In this case the user will be prompted to provide information on the configuration file and the image file used during v3_create. VMs which are available in both the /proc and database are designated *active* and can be used. VMs available only in the database are *inactive* but part of the inventory.

The purpose of providing the inventory structure for the VMs is to properly manage the state of the VM. In case the system is restarted all the information in the /proc file is lost and all the active VMs not in the inventory are lost. If the VM was activated it will be available in the database and recreated. In the case the database is lost then all the VM instances are lost.

5.3 Running a VM

VMs which belong to the active inventory list can be run, other VMs need to be activated in order to be used. To start a VM you can press the play button in the toolbar, right click and select start or select start from the menu. On pressing play a dialog is launched which will ask you for the mode of operation. Palacios supports three modes stream, console and VNC. If stream mode is selected you will additionally asked to provide a stream name for communication. The VM is launched as a tab in the UI. The UI supports VM pause, continue and stop as well which can be triggered by selecting the appropriate options. All these tasks are handled by the command line tools at the backend. Some of the command line tools support only one VM at a time, therefore we cannot run two VMs with the same mode. The stream mode is an exception as multiple streams can be opened at the same time. Closing the application while VMs are running is not allowed. All VMs must either be paused or stopped before exiting the application. The application will reload the VMs using the state information stored in the database.

5.4 Deleting a VM

Any VM can be deleted irrespective of the inventory. Right click on a VM and select “Delete VM” to remove a VM from the system. If the inventory is active then it is removed from the /proc file as well as the database. An inactive VM is deleted only from the database and a non-active VM can be removed only from /proc. If a VM is running, it cannot be deleted.

5.5 Telemetry Information

The application displays telemetry information at the bottom. This is done my tailing into /var/log/messages. In the next release we plan to display dmesg output at given time intervals.

5.6 Reloading VMs state

It is also possible to use the GUI and the command line tools simultaneously. We provide a reload option which updates the state of the VMs in the application.

5.7 Assumptions

It is assumed that the following steps are completed before starting the application:

- Install Palacios kernel module
- Setup memory for Palacios
- Set PATH variable to point to the location of the v3 tools (/path/to/linux_usr)

6. FUTURE EXTENSIONS

Since Palacios is continuously evolving we can expect new features to be added to the GUI module in future releases. The current planned upgrades are:

1 Support for SQL database backend in place of text file database.
2 Support for VM checkpointing
3 Support for periodic telemetry information
4 UI improvements

As Palacios continues to evolve the GUI module will expand to support the new VM features.

ACKNOWLEDGMENT

I would like to thank Prof. Peter Dinda for his continued guidance throughout the duration of the project providing valuable ideas and support. I would also like to thank Kyle Hale, Maciej Swiech and Lei Xia for providing feedback during the implementation phase of the project.

This project is made possible by support from the National Science Foundation (NSF) via grants CNS-0709168 (Northwestern), CNS-0707365 (UNM), and the Department of Energy (DOE) via grant DE-SC0005343 (Northwestern, UNM, U.Pittsburgh, Sandia, and ORNL).

REFERENCES

[1] “An Introduction to the Palacios Virtual Machine Monitor---Version 1.3
http://www.v3vee.org/palacios/palacios-1.3-tr.pdf

[2] http://qt-project.org/

[3] “Qt implementation of LibVNCClient”, http://libvncserver.sourceforge.net/success.html

[4] “GnuTLS download link” ftp://ftp.gnutls.org/gcrypt

[5] “KRDC Qt Vnc”,
http://websvn.kde.org/trunk/KDE/kdenetwork/krdc/vnc/qtonly/README?view=markup