VPGate on Linux
(aka vpgnix)

Port Design Notes

Steve Yackey

4/15/2022

Table of Contents

1 Revision History

Date	Who	Description
4/15/2022	Yackey	Alpha and Omega

2 Introduction

This paper briefly describes some basic ideas regarding a port of VPGate to Linux.

However, chances for improvement will be taken.

- Move to SOLID principles
- Best practices
  - coding
  - platform

The porting issues discussed below, are categorized.

- critical items
- important items
- separable Items
- mundane items
- other items

The result is a native Linux application(set).

2.1 Objectives

- Provide a native port of VPGate to Linux.
- Aim for improvement where possible.
  - For example:
    - move towards SOLID principles
    - dependency injection
    - composition root (IoC)
- The target platform is Ubuntu 20.04.
- The target development environment is standard Linux (Ubuntu) C++ development environment.
  - g++ -std=c++2a, clang-*
  - gmake (no CMake)
- Use standard C++ types and components – C++20
- Follow Linux best practices, e.g., daemon(7)[1] and systemctl(1)[2]
- Don’t forget - the value is in the logic and functionality – so don’t break it !!

… you end up with a native LinuxVPGate.

2.2 Non-objectives

This is a port of a native Windows application to a native Linux application.

There are no emulators or compatibility layers. These simply add complexity and additional support concerns.

No Boost, logging frameworks etc.

IoC will be part of the improvement. IoC has several benefits such as a move towards open/closed principle (OCP), separation of concerns (SRP) and easier maintainability. A DI container won’t be used, however. Dependency injection will be a manual. Mainly because the most useful containers (Hypodermic, BoostDI) have not had repository activity in years. DI containers are helpful in that they require structure and order. However, there is nothing wrong with a manual composition root.

1 daemon(7) https://man7.org/linux/man-pages/man7/daemon.7.html
2 systemctl(1) https://man7.org/linux/man-pages/man1/systemctl.1.html

3 Critical Items

These are items that are prerequisites for any semblance of success.

3.1 Replace Windows System Facilities

The Unix mantra – from long ago – and now in the Linux world: Everything is a file descriptor.

Window's facilities that provide HANDLE operations include:event operations, waitable timer operations and thread exit.

WaitForMultipleObjects operates on one or more HANDLES.

Message exchange is a per thread Windows facility provided by:

Sinks – PeekMessage, GetMessage

Sources – PostThreadMessage, PostMessage

Thread creation ala OS_ThreadCreate, not a platform facility (wrapper), but is included for completeness.

The objective is to provide porting replacements, using Linux platform facilities that, at their core, are file descriptor based. This collection I call the 'waitables'.

3.2 The Waitables

These are the port replacements for Window's platform facilities. All use standard Linux platform facilities.

Windows Facility	Port Replacement	Underlying Linux Facility	Notes
CreateEvent SetEvent ResetEvent	WaitableEvent	eventfd(2)[3]	
OS_ThreadCreate	WaitableThread	std::thread/ std::function with event(fd)	Not actually platform provided per se.
ThreadExit	WaitableThread	evntfd(2)	
WaitableTimer	PostableTimer WaitableTimer	timer_create(2)[4] timerfd_create(2)[5]	Delivers WM_TIMER Waitable via WaitForMultipleObjects
PeekMessage	WaitableMessageSink	messageQueue(7)[6]	Message queue descriptor is a file descriptor
GetMessage	WaitableMessageSink	messageQueue(7)	
PostThreadMessage	WaitableMessageSource	messageQueue(7)	
PostMessage	WaitableMessageSource	messageQueue(7)	
WaitForMultipleObjects	WaitForMultipleObjects	epoll(7)[7]	Everything is a file descriptor !!

The key underlying idea: Everything is a file descriptor.

3 eventfd(2) https://man7.org/linux/man-pages/man2/eventfd.2.html
4 timer_create(2) https://man7.org/linux/man-pages/man2/timer_create.2.html
5 timerfd_create(2) https://man7.org/linux/man-pages/man2/timer_create.2.html
6 messageQueue(7) https://man7.org/linux/man-pages/man7/mq_overview.7.html
7 epoll(7) https://man7.org/linux/man-pages/man7/epoll.7.html

4 Important Items

These are items that exist in both platform environments. While there may be differences, the basic Windows platform functionality exists in basic Linux platform functionality.

The main thought here is:

- Network stuff
- Loading plug-ins

5 Separable Items

VPGate contains a lot of functionality. The port offers the opportunity to separate some of the ‘embedded’ functionality into **external** cooperating processes.

5.1 (Re-)Config

Typical Linux services (re)evaluate configuration changes in response to a signal, typically SIGHUP (kill -1)[8].

That will be the pattern here.

A ‘front-end’ to the configuration information, e.g. web page or whatever, will be a totally separate application/service. Obviously, it can be developed independently. When the front-end is happy with configuration changes, the vpgate daemon can be signaled. The daemon will wait for signal delivery through signalfd(2).[9]

Note: I also prototyped and observer/subscriber based on inotify(7)[10]. The signal method seemed preferable, allowing independent configuration mechanisms. Observer/subscriber will likely watch the signal.

5.2 Managed Host

Similar to (re)config managed host communications are an independent external service. Developed totally independently. This would also be a signal(7) operation.

8 signal(7) https://man7.org/linux/man-pages/man7/signal.7.html
9 signalfd(2) https://man7.org/linux/man-pages/man2/signalfd.2.html
10 inotify(7) https://man7.org/linux/man-pages/man7/inotify.7.html

5.3 Alarm Interface

Really a subset of the managed host described above in terms of an independent externalized development. The plumbing is reversed however, with vpgate having information for the managed host external application. For this it makes sense to use either a message queue or a Unix domain socket.[11] A message queue seems appropriate, but either is fine. Note: the choice depends somewhat on the variability in the size of the messages. I have no code to look at and I don't recall. Either is fine.

5.4 Redundancy

My shakiest ground, no code to look at + feeble memory :) Redundancy would seem to be able to be a separate, external service. Notification to VPGate would be a signal(7) operation.

6 Mundane Items

This is the grunge work in porting. Its not that its easy or fast, just not rocket surgery.

6.1 Type Transformation

Windows types will be changed to standard C++ types. Not by some typedef based redefinition mechanism, but actually changed in the code.

The most obvious ways include a) a shell script with a bunch of sed(1)[12] directives 2) an awk(1p)[13] script using a file of transformation definitions. Awk is built for pattern matching, so it is the natural choice. Additionally, the awk script will be far more readable than the convoluted sed syntax. And when done, will provide a good form of documentation.

The transformation will include simple types, e.g., DWORD as well as complex types, e.g., CString. The entire 'transformation' process will be a makefile target.

7 Other Items

7.1 Licensing

I only include this 'separately' because I do not recall the code details (and obviously can't look :)) Therefore, it is unclear if there are any Windows platform specific items that need attention. However, it would seem that licensing would remain 'the same'.

11 Unix(7) https://man7.org/linux/man-pages/man7/unix.7.html
12 sed(1) https://man7.org/linux/man-pages/man1/sed.1.html
13 awk(1p) https://man7.org/linux/man-pages/man1/awk.1p.html

7.2 Plugins

Plug-ins are not discussed here, obviously there would need to be a device plug-in or two :)

Without having code to look at, its hard to be precise.

But that is do-able, too.

7.3 Installation/Update

Follow Linux packaging best practices utilizing the facilities of dpkg(1).[14]

8 Conclusion

- The ‘waitables’ provide a foundational structure to allow a port of VPGate to Linux, vpgnix.
- They are straightforward and not overly complex. Additionally, they could be the basis for a port of all of Scout; or other ‘components’, such as **ASP consumer.**
- Vpgnix can take advantage of the chance to change by allowing existing ‘monolithic’ portions of VPGate to become independent external components.
  - Architecturally desirable
  - Allows parallel development if desired
- Code transformation can be automated to a large extent. Allowing ‘fast follow’ updates.
  - New features, bug fixes, etc.
- There is no emulation or compatibility layers. These simply add complexity to the code base as well as additional support concerns.

Totally do-able !!

14 dpkg(1) https://man7.org/linux/man-pages/man1/dpkg.1.html