Jerry W. Rice

email: jerrywrice@fabnexus.com
12/2/2017 3:17 PM

Advanced Development Tools for Software Development Consultants

This article discusses selecting and configuring a flexible and effective development system which supports overlapping independent software development projects. A thriving software consultancy must be capable of supporting multiple clients over a given period of time. While separate computer systems can be used for each client project, in most cases this is an unnecessary capital expense and encumbers one's practice with excess equipment storage and property tax burden.

Across client projects there are typically a diverse set of operating systems, software tools, target languages and target systems. This mix of projects complicates the consultancy with the required inventory of development system hardware and software configurations.

The development system proposed herein is built around a commercial virtualization tool which executes multiple independent operating systems (Windows, Linux, BSD, etc..) on a single general-purpose computer system.

The focus here is on a laptop development system for mobility, although the concepts apply equally well to desktops.

The Ten-Thousand Foot View

A modern software development computer must be a general-purpose computer. It must support complex language compilers, linkers, loaders, debuggers, translation tools, CAD design tools, source-control tools, data-base intensive applications, web editing tools, a diverse range of hardware peripherals and network protocol stacks, and other assorted software tools and applications depending upon the specific project requirements. Less powerful systems such as mobile phones and hand-held tablets cannot support such a large and complex array of tools. Rather, the relevant computer products available in today's market are either 'Apple' or 'not Apple'. The 'not Apple' development computer products run Microsoft Windows, Linux or other Unix-like systems.

With few exceptions these general-purpose personal computer systems incorporate either Intel or AMD x86 or x64 instruction compatible microprocessor chips. As multicore ARM processor chips evolve and gain processing capacity (become faster), at a future date they may join the ranks of commodity laptop computer processor engines - but not currently.

The majority of contemporary software development projects target (generate code for) one or more of the following run-time computing platforms:

1. Mobile phones and tablets,
2. Microsoft and Apple laptop and desktop computers,
3. Network servers, IOT devices, and dedicated embedded appliances running either a standard OS, RTOS, or direct firmware (bare metal),
4. Cloud-based and web applications, which rely on the aforementioned systems as user access points.

Since 1982 Jerry W. Rice has been self-employed (www.fabnexus.com) while consulting as a software engineer on a broad range of instrumentation, data network, machine control and robotic automation development projects

The majority of newly constructed systems that incorporate software utilize a mix of desktop, rack-mounted, and embedded processor boards. Each such computing element is referred to as a 'target' processor, meaning it's a distinct processor that is integrated into the final delivered system.

Desk-top and rack-mounted computers almost always run a standard Windows, Apple or Linux (or other Unix-like) operating system, and therefore are capable of directly ('natively') executing ('hosting') their own software development and debugging tools.

On the other hand, embedded processor boards usually have minimal RAM and disk storage, constrained processor band-width and often don't run a general-purpose operating system, so they rarely host their own development tools. Instead they rely upon a cross-development tool suite (compiler, linker, remote debugger) which executes (is hosted) on a separate external standard development computer/laptop. The target's cross-compiled binary files are then copied to the target embedded processor in order to execute. The two diagrams below succinctly describe the different development approaches: (a) traditional self-hosted, and (b) cross-development.

Development Computer
running either Windows, Linux (or derivative), or OS-10 OS

Self-hosted software development tools directly generate run-time code for the current (host) environment. Run-time test and debug is directly performed on this host computer system. Historically the vast majority of software development tools are of this type.

Diagram (a) - Self-hosted build environment

Development Computer
running either Windows / Linux (or derivative) / OS-10 OS

Cross-development software development tools run on a standard host computer system, but in-fact emit/generate run-time binary code files for a different computer run-time environment: known as the target system. In most cases the run-time test and debug effort must utilize the external target hardware. Remote cross-debugging may be possible from the host development system.

Diagram (b) - Cross-hosted build environment

The discussion so far has focused exclusively on software development languages that use classic/traditional language compilers which directly generate target hardware machine (or assembly) code. There are in-fact other scenarios, exemplified by .Net, Java (byte code), and LLVM, whose language processing tools generate virtualized/intermediate target object code. Furthermore, there are purely interpretive language processors that execute source code (text) files directly without prior translation. Nevertheless, the relevant development tool hosting considerations apply in each of these cases (with certain variations).

Considering the broad range of potential client software development tools and environments, the use of a virtualization software product such as those provided by VMWare has significant benefits.

Operating System Virtualization Offers Flexibility, Utility and Risk Mitigation

There are various professional-quality virtualization products, including Oracle's VirtualBox and Microsoft's Virtual PC, but I prefer VMWare's for its advanced features and support. Support requires purchasing a license, but the cost is quite minimal (~$300) for a single developer. Based on the specific client requirements, the consultant may find a need to use one or the other of these virtualization products for a given project.

While competing virtualization engines may be installed together on a given development system, in practice there is rarely a need to execute different competing virtualization products simulataneously.

If the need arises, I advise initially testing this under the precise configuration desired, as the results (success or failure) will depend upon the development system processor's hypervisor support, BIOS settings, guest operating system, and such. Again, the vast majority of use cases for virtualization incorporate a single virtualization product: VMWare, Oracle, Microsoft, etc.

These virtualization tools support simultaneously running one or more 'guest' operating system images (Windows and/or Linux) in addition to the native boot-time host OS, primarily at the expense of additional disk storage space, RAM and processor bandwidth usage. In my practice the host OS is typically Windows, but could be Linux or even Apple OS-10. Virtualization enables the consultant to easily support a mix of overlapping (simultaneous) operating systems executing on the same physical laptop.

For the perfectionists, the actual degree of run-time concurrency depends upon the quantity of physical cores the host laptop processor incorporates, and the total number of actively executing guest VMs and application threads/processes, etc.. With only a single core physical host processor we're in-fact only achieving pre-emptive time multiplexing of a single processor core among many threads and processes. In theory this should work correctly, but in-practice results in annoying graphic video screen freezes, or application and network specific time-out errors due to resource starvation. This is why in a later section we describe the need for a development laptop based on a high-end multicore microprocessor such as an Intel I7, and at least 16 (preferably 32) Gb of RAM.

One should understand basic virtualization terminology: the 'host' operating system is the primary image that boots when the physical computer hardware powers-up and under which the 'hosted' virtualization tool runs. For an Apple laptop, this would be OS-10 and VMWare Fusion. For non-Apple laptops either Windows or Linux are the host operating system and VMWare WorkStation is the virtualization tool.

One should distinguish between the term 'hosted' in the context of operating system virtualization and the term 'hosted' in the context of cross-development tools, as they're slightly different concepts.

With respect to virtualization the 'guest' operating system is a secondary operating system (disk-based) image executing within the 'hosted' virtualization tool (VMWare Workstation or Fusion); all of which execute on the same physical computer. Typical guest operating systems are Windows, Linux, or OS-10. Note that multiple copies of the same or different guest OS images may be launched and executed simultaneously, only limited by the available physical memory and processor cores and bandwidth. The laptop configuration described later can easily and efficiently support up to four simultaneous executing guest images plus the primary host operating system (five active operating system images). Finally, note that the 'host' and 'guest' operating systems may be the same (i.e. 'Windows' and 'Windows', or 'Linux' and 'Linux'), or different ('Windows' and 'Linux', or vice versa).

For perspective on the power and flexibility operating system virtualization provides, the guest and host operating system run-times may optionally be configured to share common file-systems. The host and guest system environments can also be configured with virtual ethernet connections so they may utilize standard network protocols and interact (via sockets for example) as if they were running on separate physical computers which are connected by network links (even though they're in-fact running in the same physical computer and utilizing ‘under the covers’ shared memory network connections.

Reiterating, the ‘host’ operating system is the non-virtual environment which physically boots and starts every time the computer powers-up. It’s normally set up when the system is initially configured by the computer manufacturer prior to shipment to the computer purchaser, and typically never changes. The one or more ‘guest’ operating systems may be easily added or removed at any time (on demand) by means of installation tools supplied with the virtualization software.

As needed the guest operating system licenses as well as other guest environment (installed) software product licenses should be acquired from their relevant vendors separate from the virtualization product itself (VMWare in my case).

VMWare Workstation supports configuring each separate ‘guest’ virtual machine with a specified quantity of RAM space, disk space, network options and other device resources: RS-232 ports, various USB devices, etc..

Prior to the availability of advanced virtualization software, developers needed separate hardware computers for each operating system and target platform supported. While one could potentially use a special boot manager and choose to boot a specific operating system image at computer power-up, this was inconvenient, technically risky (boot managers are notorious for corrupting one’s disk image) and supported running only a single operating system image at a time.

For reasonable performance the hosting laptop should have at least 16 Gb of physical RAM memory installed. I typically recommend 32 Gb. I understand current Apple laptops only support 16 Gb of physical RAM, but the upcoming generation will apparently support more. Note that this quantity of physical RAM typically requires a 64-bit host operating system. In addition, each guest (Windows) operating system should be the 64-bit edition to allow usage of more than 4 Gb of VM allocated RAM.

The laptop’s physical processor should be a high-end multi-core processor (an Intel I7 or better) in order to achieve adequate run-time performance during development and test.

At least 1 TB (preferably 2 TB) of high performance disk storage file storage is needed. Some of this is preferably SSD, or hybrid disk storage.

Another benefit of using a tool like VMWare Workstation is that each client project can be set up in a separate guest (virtual) operating system environment, exclusively for that customer’s project. While for Windows this means investing an additional $100+ dollars for the Window’s OS license, the benefits are worth it. Your development project will be safely isolated from other customer projects, as well as the (primary) host operating system. This is beneficial since a given customer project typically requires installation of various collections of specialized software: software tools and other support applications. Using a separate dedicated VM avoids potential software installation conflicts that would arise when installing multiple (possibly incompatible) versions of tool-chains and other utilities in a monolithic host environment. Other benefits include assurance to the client that their project artifacts will be isolated in a separate VM and not co-mingled with other client files.

Virtualization technology has improved and become easier to use and significantly more reliable over the past fifteen or so years. Although one may (rarely) encounter a limitation related to a specific hardware peripheral device driver, operating system virtualization provides significant benefits. In practically all scenarios it performs quite reliably and effectively. With this stated, there are specific use case limitations one should be aware of. While VMWare Workstation is quite effective when using standard host system devices (device drivers), if your project uses unusual or otherwise exotic custom device drivers it's prudent to initially verify the virtualized guest operating system can properly recognize and utilize those drivers under virtualization. If you encounter issues, certainly check with VMWare's support prior to assuming that you can't proceed under virtualization.

Unfortunately, when developing, testing, or debugging a device driver for a standard operating system, in most cases you'll need to utilize a dedicated computer without virtualization - working at the host native operating system level.

My experience with VMWare is that it is relatively easy to install/setup a new project operating system image. On occasion the network configuration required for specialized environments can be a bit tricky. Fortunately there is good email and phone support for licensees. A single seat license is currently priced around $300.

Development System Hardware Features and Options

Almost all modern high-end laptops provide USB 3 ports and an external HDMI port, but confirm how many of each is included.

Verify the laptop's keyboard is well designed and ergonomic. Online customer reviews will often provide clues of unusual or unpleasant features of a particular laptop's keyboard. Acquiring a new laptop and afterwards finding it has a strange or unusual keyboard layout/functionality is painful.

Although copper ethernet ports can be attained using external USB adapters, you may prefer a built-in 1-GBit (or faster) RJ-45 ethernet port.

External SD style flash card slots are provided on some laptops, but not all. You may utilize an external USB SD card adapter, but consider whether you prefer or need the built-in SD card slot when selecting your laptop.

If Thunderbolt or other ports are required, confirm your selected laptop provides these.

The laptop's size and weight should be considered based upon your personal preferences and expected travel patterns. I tend to give preference to feature-set and performance over reduced size/weight (within limits). When travelling by air I normally bring a smaller more compact (and less capable) laptop, unless I specifically need the larger primary development laptop during my trip.

Total usage time between battery recharges should be examined. There are trade-offs between battery discharge rate and system performance. Typically I keep the laptop plugged into the power grid when working in a lab or other debug situation, running on battery only when it's necessary. Another consideration is expected battery life before the battery must be discarded, and the effort and complexity required to replace it. Also consider the battery replacement cost. Newer laptops typically require disassembly of the laptop case to replace the battery, and may require the service of a trained product technician. Confirm this to avoid violating any remaining system manufacturer's warranty.

The laptop's display size, quality, and built-in graphics adapter model should be considered based on your particular preferences, your development style and visual health and your expected mix of customer projects. Should you need guidance on particular display or GPU hardware engine features, discuss this with a knowledgeable colleague, and/or refer to recent (reputable) on-line technical reviews and blogs. I personally prefer a larger (17 to 18") laptop display and give this higher priority than reduced system weight. On the other hand I'm not a game developer, nor do I a specialize in high-end graphics software development, so the particular GPU isn't particularly a focal point in my laptop selection process. Others may differ.

The host development laptop should have 32 (or more) Gb of RAM, and plenty (2 Tb) of file system disk storage space, preferably SSD and/or fast hybrid disk drives.

I commonly use RS-232 connections on client instrumentation projects, and find that high-quality USB-based RS-232 adapters (such as FTDI) work perfectly fine.

If I2C or SPI host connections are required, a USB based adapter such as that provided by Total Phase works perfectly well.

Most non-Apple laptops by default come pre-installed with Windows 10 as their host operating system, although one could potentially replace this with an older version of Windows or a version (distro) of Linux. This will almost certainly limit your ability to later use your laptop manufacturer's warranty support, but this is your choice. Of course, installing an older version of Windows as a guest operating system under virtualization is a viable option.

If you choose to replace the system's pre-installed Window's host OS with another (such as Linux), I suggest purchasing a new replacement drive (whatever applicable media is involved) and then remove and keep the original drive intact/unchanged, so in the future you can reinstall it for possible warranty purposes. You can then physically install/insert the new blank replacement drive and proceed to install the new OS on this newly installed drive. It is critical that you first verify that the specific version of whatever operating system you plan to replace your factory installed one with has available the necessary hardware device drivers needed by your new computer/laptop's mainboard and peripheral chips. If not, you'll need to keep the factory installed host operating system, and instead install your alternate operating system as a virtualized guest OS.

A final hardware note. When a client's target system hardware utilizes specialized plug-in PCI adapter boards (i.e. high-end video frame-grabber or high bandwidth fiber-link communication adapter), it isn't possible to run the final software on one's development laptop. One option is to generate a thin software wrapper around the aforementioned I/O driver or vendor hardware library calls, and effectively simulate/stub-out invocations to these unavailable hardware interfaces. In any case, eventually you will need to perform integration and test on the target industrial PC computer chassis with the PCI adapters.

Jerry W. Rice

email: jerrywrice@fabnexus.com
12/2/2017 3:17 PM

Development IDEs, Tool-Chains, and other useful Utilities for Developers

When developing for Windows, Microsoft Visual Studio is the 'gold standard' development tool. Nevertheless, there are alternate tools available. Beyond simply the developer's preference, sometimes one must use these alternate development tools due to the client's legacy environment and other specific technical requirements. In recent years various Linux open-source development tool-chain environments (gcc, etc.) have been ported to Windows. There are also several non-Gnu open-source (free) development tools, and numerous commercial ones as well. The nature and details of these development tools vary based upon the target development language, and the host and target operating system. It goes without saying that Google is one of the developer's essential resources.

If developing for a mix of Windows, Linux, OS-10, Android, IOS, etc. (not necessarily at the same time or on the same project), there are benefits to 'tooling up' and becoming proficient with cross-platform development tools, some of which were mentioned above.

When developing software using languages other than c, c++, and c# the choice of development tools may be dictated by the language itself. Java, and many modern functional languages sometimes require downloading and using their built-in specialized development tools. A 'google' search will lead one to information on these, and of course Wikipedia is an excellent starting point to familiarize oneself with the available tools for a given programming language.

When developing embedded microcontroller software, typically the development tools will be supplied by the board or SOC chip vendor. Most times these projects involve coding with c/c++ and/or symbolic assembler.

For mobile development, typically IOS or Android, the development tools are typically supplied by the vendor or a third interested party such as Google. These tools sometimes leverage pre-existing open-source projects such as Eclipse. Another fairly popular open-source cross-development tool is Qt, originally developed by TrollTech, and now the 'Qt Company'. In recent years a variety of powerful mobile development (some cross-platform) tools have been introduced. One popular example is Xamarin's (now Microsoft) mono c# based mobile development platform.

For any development environment, the availability and capabilities of a symbolic debugger are critical. The major computer and operating system supplier's (Microsoft, Apple, Linux) provide powerful IDEs and debuggers, but one should scrutinize the availability and debugging features if evaluating a newer niche language or SOC environment. With certain SOC environments a hardware assisted JTAG or similar hardware assisted debug setup is required.

In most development projects there will be a need to compare and/or merge individual or directories of text or source files. For Windows I've found the free utility WinMerge to be extremely useful for this purpose. For Linux there are several similar tools, including DiffMerge, Kompare, and others (simply google 'Linux text file compare and merge tools'). Another powerful benefit of staging a development project in a virtualized guest operating system is that one can use file manipulation tools from an entirely different operating system by sharing file systems. I sometimes use Window's WinMerge to operate on my Linux project files simply by switching to an executing Windows virtual space, and utilize WinMerge since the Linux text files are accessible via file sharing.

Most development projects utilize some type of source code control tool. While Microsoft has its proprietary Team Explorer, I've found that many Windows development projects opt for one of the open source code control systems such as 'svn' or 'git'. These have run-time versions for all popular operating systems. Windows offers the 'Tortoise' graphical client versions for both (svn and git), which I find to be easier to use than the command line versions. This lends itself to the aforementioned technique of running the Windows TortoiseSVN or TortoiseGit client from a Windows VM even when undertaking a Linux development project.

I would be remiss to not mention code smart text editors, of which there are many. Some of these are Windows or Linux specific, but again using virtualization with shared files eliminates this matter as an impediment.

Tying it all together

Practicing software developers will certainly have their own preferred development tools, which can be integrated into the virtualized development environment as described herein. The range and choice of available and useful development tool options is quite large (and increasing over time), therefore any attempt to identify every available and deserving development tool would be an enormous, ever-changing and impractical undertaking. Simply stated, it is beyond the intended scope of this article.

Depending upon the specific scheduling and mix of your client projects, you'll need to consider:

1. The specific guest operating system images you should generate,
2. Where to install your individual development tools among the guest (and the host) operating systems, while factoring in any per seat license cost for your various commercial software tools.
3. Choose a mechanism for archiving and later retrieving guest operating system images once the associated project completes. The existing virtualization products make this relatively easy since the entire guest operating system is stored in a single folder hierarchy.
4. Tracking guest operating system login user-names and passwords.

The key concept conveyed herein is the usage of virtualized guest operating systems, coupled with shared file systems on one's development laptop, which opens a range of powerful benefits and value to the majority of software development consultants (and developers in general).

Since 1982 Jerry W. Rice has been self-employed (www.fabnexus.com) while consulting as a software engineer on a broad range of instrumentation, data network, machine control and robotic automation development projects