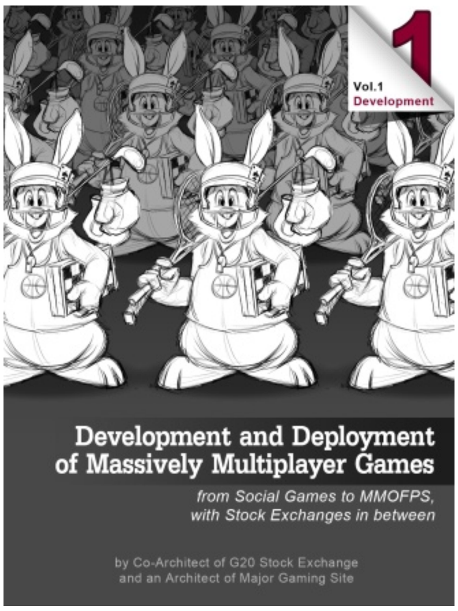
MMOG Server-Side. Eternal Linux-vs-Windows Debate

posted January 4, 2016 by "No Bugs" Hare, translated by Sergey Ignatchenko

[[*This is Chapter VI(c) from the upcoming book "Development&Deployment of Massively Multiplayer Online Games", which is currently being beta-tested. Beta-testing is intended to improve the quality of the book, and provides free e-copy of the "release" book to those who help with improving; for further details see "Book Beta Testing". All the content published during Beta Testing, is subject to change before the book is published.*

To navigate through the book, you may want to use Development&Deployment of MMOG: Table of Contents.]]


Vol.1
Development
1
Development and Deployment of Massively Multiplayer Games
from Social Games to MMOFPS, with Stock Exchanges in between
by Co-Architect of G20 Stock Exchange and an Architect of Major Gaming Site

Operating Systems

Please don't expect to find anything new in this section, especially in the context of "which OS is the best one out there". It is merely a summary of well-known things as they apply to MMOG server-side.

For the client-side, operating system is normally a big fat Business Requirement, which means that we as developers don't have much choice about it. If we need to support Android, iOS and Windows on the client-side – we just need to shut up and do it, plain and simple. With operating system for the server-side, situation is usually different – as nobody on the business side of things really cares (or at least SHOULD NOT care) about which OS is used to run our servers, it is more or less a developer's choice. What MAY (and actually SHOULD) interest business guys/gals though, is time-to-market and the cost of running servers, more on it below.

When it comes to server-side operating systems, there are actually only two realistic choices: Windows and Linux[1] (or "Linux and Windows", depending on your preferences, we'll discuss this in a moment). While in theory you can run an OS X server, or can dream about trying a 32-core SPARC M7 under Solaris, or (like myself) be eager to get your hands on the latest greatest POWER8 box, in practice all we'll ever get (except, maybe, for stock exchange guys) is x64 box with either Windows or Linux. And while there is nothing wrong about x64, it still often leaves us feel a bit sad about all those existing-but-never-available opportunities.

Leaving sentimental feelings aside, we need to take a look at two real contenders: Linux/BSD and Windows. Unfortunately, over the course of serveral last ~~centuries~~ decades, any attempt to to take such a look has invariably lead to almost-religious wars.

[1] For the purposes of our discussion, we'll consider BSD as a flavour of Linux (which is admittedly a sacrilege, but Linux programming and *BSD programming at our application level are that similar, that with a few narrow exceptions such as epoll/kqueue, we can pretty much ignore the difference until actual deployment).

New Generation Chooses Cross-Platform! Well, at least it SHOULD…

One thing you should seriously consider before choosing one single OS as your development target, is "whether you can make your program cross-platform instead". In general, I strongly support cross-platfrom programs, even on the server-side, for several reasons:

- we don't need to go into Linux-vs-Windows debate right here, making it a deployment-time issue rather than development-time issue. Not only having cross-platform code postpones the debate, but also it makes the debate much less heated, as the cost of mistake at deployment-time is orders of magnitude lower
- cross-platform programs are, well, cross-platform, which gives you deployment-time freedom
    - for example, if you find that for the purposes of your game the latest greatest TCP stack from Linux (or Windows) works significantly better (see "Other Technical Differences (kernel scheduler, TCP stack, etc)" section below) – you can switch without that much hassle
    - moreover, you can have some servers on Windows and some on Linux at the same time (optimizing different audiences according to different parameters)
- cross-platform programming helps to keep dependencies in check
- cross-platform programs tend to have better structured codebases (I attribute it to better discipline, so it is not inherent to cross-platform programs, but a correlation)
- cross-platform programming help to test your code better. It has been observed that running a program which was considered perfectly error-free, on a different platform, helps to reveal quite a few subtle bugs which have never manifested themselves on original platform (but were sitting there, just waiting for the right moment to kick in).

How to achieve a holy grail of cross-platform code, is a separate story, which we'll discuss in [[TODO]] section below. For now, let's just make a note that going cross-platform does not necessarily mean going JVM (Python, Erlang, pick your poison), and that C++ can also be made perfectly cross-platform, so at least don't write it off on these grounds. On the other hand, let's keep in mind that outside of deterministic FSMs (and for pretty much any programming language), the best we can possibly hope for, is "run once – test everywhere", and "testing everywhere" takes time ☹ . Which, in turn, makes convincing managers going cross-platform route quite difficult (that is, unless you're using Java/Python/…), so you *may* need to choose your OS even if you would like to avoid it in the first place.

Eternal Windows-vs-Linux Debate

I realize that for the analysis below, I will be hit hard by zealots from both sides. On the other hand, as choosing server-side OS is an important part of the overall MMO exercise, I need to provide at least some observations in this regard, so I have no choice other than to brace myself and be prepared to all the punches from both Windows and Linux fans (with an occasional hit by BSD/Solaris proponents).

"I have no choice other than to brace myself and be prepared to all the punches from both Windows and Linux fans

Now, we can forget about the boring cross-platform stuff, and to concentrate on the classical Linux-vs-Windows flame war. BTW, most of the arguments routinely raised in such flame wars, do have some merit behind them, with the tricky part being to estimate applicability and impact of these arguments within the specific context. Let's take a closer look at some of them (only in the context of the server-side specifically for games):

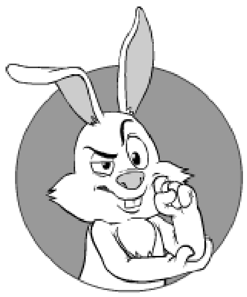
Open-Source

The practical argument here goes along the lines of "if you ever have a problem, you'll be able to fix it". However, being a game developer, I don't think it is realistic to expect that you'll be able to fix anything in Linux kernel (or, Linus forbid, driver). If you've done it before – of course, being able to fix things in kernel becomes an all-important argument, but otherwise – don't hold your breath over it.

Stability/Reliability

There are a lot of horror stories about Windows being unstable/unreliable, including (in)famous migration of London Stock Exchange from Windows to Linux in 2009. [http://www.itwire.com/opinion-and-analysis/the-linux-distillery/28359-london-stock-exchange-gets-the-facts-and-dumps-windows-for-linux] My personal experience, however, doesn't support this observation. In short – from what I've seen, if all you're using from Windows, is Windows kernel (without any fancy COM components or .NET) – Windows has been observed work perfectly fine (more on disabling unnecessary software in Chapter [[TODO]]). Add anything large on top of a bare Windows kernel – and if you're not careful enough, you're entering much riskier waters, to put it mildly. Pretty much the same goes for Linux, but as Linux doesn't try to cover everything-under-the-sun as a part of operating system, you can usually choose which software to use, more freely. Still, from my experience, if you're careful enough, it is more or less a tie between Linux and post-9x Windows in the stability realm.

Security



“Personally, I would agree that Linux is somewhat more secure (that is, if you're exercising at least basic caution and are not running your web server under root account).

Another quite popular argument is that Linux is more secure than Windows (what Microsoft vehemently objects, mostly on the basis of the number of reported bugs, which is a very convenient metrics for a closed-source company). Personally, I would agree that Linux is somewhat more secure (that is, if you’re exercising at least basic caution and are not running your web server under root account).

I tend to attribute it to the fact that Linux in general is more modular than Windows, so disabling unnecessary parts is easier (and it is these unnecessary parts that cause most trouble). While this is partially offset by an atrocious *nix permission system (with suid bit abuses being responsible of a substantial chunk of successful real-world attacks), being highly modular still helps even in this department. Also SE Linux, despite all the shortcomings, does provide an additional layer of protection.

On the other hand, it is clear that you do need a highly qualified and security-aware admin to run any operating system securely. Just one very recent real-world breach example involved default Amazon EC2 Linux image to run Apache under root (and while SE Linux was running, SE policies didn’t prevent attacker from taking the server over).[2] In short: it wasn’t a problem of Linux as such, but a problem of Linux being misconfigured. However, it leads us to an all-important bottom line:

Each server is only as secure as its admin

If you have highly qualified admins, then I’d probably prefer Linux from security perspective, but in practice security advantage over Windows will likely to be negligible (that is, if you’re using only “bare” Windows kernel, while disabling everything else, see above).

[2] if you don’t understand why running your services under root account is a problem – wait until Chapter [[TODO]], we’ll briefly discuss it there

Fast Network Packet Processing

If your game is a very latency-sensitive, all chances are that you’ll need to use UDP (see Chapter [[TODO]] for further discussion). And when you’re using UDP, you may easily run into your recvmsg() thread (or even recvmmsg() thread) becoming a bottleneck. One of the ways to deal with it in a cross-platform way, is to try multiple threads calling recvmsg() on the very same (non-blocking) socket, which has been reported to work pretty good (which has been briefly described in “UDP-related FSMs” section above). However, if this doesn’t help, you’re pretty much out of cross-platform options. It means using rather obscure and little-known platform-specific APIs, which may include the following.

[[TODO!: Linux netmap/DPDK, Windows RIO]]

[[TODO!: Interrupt balancing: Linux RSS/RPS/RFS]]

Other Technical Differences (kernel scheduler, TCP stack, etc)

There are quite a few debates out there related to comparisons between Linux and Windows kernel schedulers and network stacks. In short – at least for games, the differences between them are negligible. A tiny bit more detailed analysis follows.

Regarding kernel/thread schedulers – note that for the game you certainly want to keep your CPU utilization low (even for social games having CPU utilization at 100% is certainly not a good idea), and thread queue – as short as possible. It means that there should always be a free CPU in the system, which is ready to process incoming packet.[3] It means that the scheduler (almost) always doesn’t really have a choice which thread to schedule – *all* threads which are not waiting, will run, as there are (almost) always sufficient CPUs to run them. In practice, I don’t know of any significant differences between Windows and Linux schedulers when applied to games; moreover, the difference was non-observable in practice even in the days of Linux O(n) scheduler.[4]

NUMA

Non-uniform memory access (NUMA) is a computer memory design used in multiprocessing, where the memory access time depends on the memory location relative to the processor.

— Wikipedia —

One closely related topic is related to so-called NUMA scheduling. The thing here is the following. In production, you’re very likely to use 2-socket x64 servers, which are NUMA for the last 10 years or so. And for NUMA,[5] it is very important performance-wise to keep your threads’ physical memory on the same socket (NUMA node) where your thread is running (otherwise memory accesses will need to go across the QPI/Hypertransport, which is slow compared to local memory accesses). The topic of keeping NUMA locality when scheduling, is still very much in active development (see, for example, [Corbet2013]), and does have a potential to bring significant benefits for applications (due to removal of unnecessary round-trips via QPI/Hypertransport). However, the last time I’ve seen (at least somewhat appropriate) comparison, I wasn’t able to notice the difference between Windows and Linux in this regard (which might or might not be because of FSM-oriented architecture, which tends to exhibit very good memory locality and may be easier to handle by NUMA schedulers). In short – jury is still out on Windows-vs-Linux NUMA scheduling, it may or may not affect your game (though IMHO the differences are not going to be drastic, at least not for long). Good description of NUMA on Linux can be found in [Lameter2013]. A bit more on practical suggestions related to manipulating NUMA-related things from application level will be mentioned in Chapter [[TODO]].

As for the TCP stack: all the TCP stacks out there start with the same RFC793 (yes, that's 1981 and still not obsolete); of course, there are several dozens RFCs on top of the basics described there, and sets of these RFCs and their defaults vary, but deep inside it is still pretty much the same thing (and even first several layers on top of it, such as Nagle's algorithm or SACK, are pretty much the same). Most of the differences between TCP stacks discussed out there, are actually about using different defaults/settings for TCP stack, which result in different throughput under different conditions (especially TCP performance over long fat pipes can be significantly different); however, these things, while interesting and important for video- and file-services, are not directly applicable to games, where average packet size is around 40-80 bytes (that's including 20 bytes of IP header). When it comes to latencies, network stack doesn't affect UDP latencies much, and TCP latencies will depend on lots of things, including TCP stack on the client side (not to mention that if you're into single-digit millisecond latencies, using TCP is probably not the best idea). One thing which *may* affect those games working over TCP, is a choice of TCP congestion algorithm (with Windows Server 2008+ using NewReno, and recent Linux reportedly using CUBIC); however, as of now, I don't have any information which demonstrates any advantage of any of them TCP-latency-wise (that is, with usual mixed-bag of clients, consisting of PCs and mobile phones); on the other hand, it is an area where development is still very much ongoing, so further changes are likely. Also note that as we cannot control client and there are tons of different clients with different TCP settings out there, any theoretical analysis becomes extremely complicated; the best we can do – is to try both in a real-world environment (the one with thousands of clients) and see whether there are any differences. Which makes yet another reason to have your code cross-platform.

When it comes to IPC (which you need to implement inter-FSMs interactions), both systems are very much the same. We'll discuss it in more detail in Chapter [[TODO]], but the rule of thumb is always the same: if you want it to be really fast – use shared memory, all the other mechanisms are inherently slower. Fortunately, shared memory is available on both Windows and Linux. If you don't care too much about achieving topmost available speed on the same machine – all common other methods (such as pipes and sockets) are readily available on both these platforms; and for our purposes, you won't need more than that. Fancy stuff such as completion ports and APC, *may in theory* provide some difference, but in practice for FSM-based architectures, it wasn't observed to provide any advantage (see also Chapter [[TODO]] for details). In short – IPC-wise, you will have quite a difficulty to find significant difference between Linux and Windows.[6]

As for file systems – for your Front-End Servers and Game Servers they don't really matter. Amount of file I/O on Front-End Servers and Game Servers should be kept negligible, mostly reading executables and configuration files; under these conditions all the differences between JFS, ZFS, ext4, and NTFS, won't play any significant role.

To summarize – technically (and from games perspective) both Windows and Linux kernel (and network stack) are doing really good job and (drivers aside) you're quite unlikely to observe significant differences because of these things. While you *may* see the some difference, if migrating from Windows to Linux or vice versa on the same hardware, experiences when migrating different server boxes will most likely be different, and I tend to attribute them (mostly) not to OS's as such, but rather to drivers, whose quality varies greatly. One potential exception is TCP congestion algorithm (that is, for TCP-based games), but its effects on games are yet to be seen.

---

[3] in practice, it is more complicated, as depending on the hardware, interrupt coming from NIC can be processed only on a dedicated CPU, which complicates things. However, this is normally not an OS restriction, but a hardware restriction, so there isn't much which can be done about it
[4] I also don't know of attempts to use different Linux schedulers for games, but based on reasoning above, I have my doubts whether they will make any difference
[5] I'm speaking about classical NUMA, with a node per socket
[6] ok, local sockets tend to be a tad slower on Windows than on Linux, but if you're really after speed, you still need to use shared memory, so it becomes pretty much a moot issue

C++ Compilers

If speaking about C++, a question of compiler becomes quite important. If you're going Windows route, your obvious choice would be MSVC, and for Linux it is probably GCC or LLVM/CLang. When comparing MSVC to GCC, GCC (especially GCC 4.8 and up) tends to produce better-quality code, which may amount (in practice) to as much as 5-10% overall performance difference.[7] This can be accounted for as 5-10% increase in number of servers you need to run your game; alternatively, you *may* try using MinGW (which is essentially GCC for Windows, I didn't try it myself, and can provide no warranties of any kind in this regard).

If comparing LLVM/CLang to GCC, in practice the difference (as of beginning of 2016) is pretty much negligible.

---

[7] individual functions can be *much* faster, but *on average* and taking into account such things as context switches and associated very severe cache misses (both

being inevitable on game servers), it is not that much as it may seem from "pure calculation" benchmarks

Is it Enough to Decide?

All the arguments above are repeated ad infinitum on the Internet, and as you see, I personally tend to favor Linux, but honestly, I don't really see that these arguments are sufficient to make a decision for our game servers (except, maybe, in some rare cases for interrupt-related stuff, see "Fast Network Packet Processing" section above). In practice, the real deal is usually about the following two reasons.

Free as in "Free Beer"

If your estimates show that you may need dozens and hundreds of servers – then the price of the license starts to hurt in a really bad way. And don't listen to those who say "Hey, RedHat license is about the same price as the Windows one, so it doesn't really matter"; in a price-conscious environment, you will likely use Debian, CentOS, or some other perfectly free distro, and will stay away from paying anything for Linux (except, maybe, for your DB server). And guess what – with zero price of free distros, there is absolutely no way for Windows to beat them price-wise, and even matching it looks very unlikely in foreseeable future.

"With zero price of free distros, there is absolutely no way for Windows to beat them price-wise, and even matching it looks very unlikely in foreseeable future.

TCO wars

TCO
Total cost of ownership (TCO) is a financial estimate intended to help buyers and owners determine the direct and indirect costs of a product or system.
— *Wikipedia* —

At some point around 10 years ago, Microsoft has pushed an argument that despite license costs, a long-term cost of ownership (known as TCO) is lower for Windows (mostly due to higher salaries of Linux guys). This argument was one of the cornerstones of Microsoft's highly controversial "Get the Facts" campaign. I certainly and clearly don't agree with Microsoft on TCO, and am of a very firm opinion that at least for not-too-small datacenter-hosted systems, pretty much regardless of how you calculate it, costs of Linux boxes will be lower. Fortunately, there are quite a few bits of research out there, which confirm my experience a.k.a. gut feeling in this regard. These start (surprisingly) from a Microsoft-sponsored(!) IDC report back from 2002 [IDC]; while Microsoft has made a lot of buzz about Windows TCO advantage found by this report, it usually conveniently omitted that for web servers Linux TCO was found to be lower (and our game servers are much more similar to web servers than to handling file or print jobs). Other studies supporting the same point of view include a report by Cybersource [Cybersource] and an IBM-sponsored report by RFG [RFG]. The latter one is especially interesting not only because it is exactly about application servers, and not only because it found Linux being 40% less expensive than Windows in the long run, but also because it has found that Linux admins, while more expensive, on average are able to handle more servers than their opposite numbers on the Windows side. To be honest, I need to mention that there are other reports which do claim that Microsoft TCO has an advantage, but also being honest, I need to say that I am not buying their arguments, agreeing with PCWorld's take on Linux-vs-Windows TCP for servers: "There's no beating Linux's total cost of ownership, since the software is generally free… The overall TCO simply can't be beat." [PCWorld]

To summarize the long text above:

Cost-wise, for game servers Linux is likely to provide a Significant Advantage

The importance of this observation, however, depends heavily on the number of servers you expect to run; if servers costs (not including traffic costs!) are going to be negligible, the whole line of argument about the server costs becomes much less important. More on it in "It is All about Money :-(" subsection below.

On ISPs and Windows-vs-Linux Cost

If you by any chance think "hey, we will rent servers from ISP anyway, so license costs won't matter", you're deadly wrong. Yes, you will most likely rent servers from ISPs (see Chapter [[TODO]] for details), but ISPs (no real surprise here) need to factor in the license price into their server rental price. As of the beginning of 2016, kind of typical price difference between CentOS two-socket "workhorse" server and the-same-hardware server with Windows Standard,[8] was roughly between $35/month and $50/month. For cheaper servers, the difference between Windows and Linux can eat as much as 50% of the server rental price (though for those servers which are more or less optimal price-performance-wise observed difference was closer to 20-30%). And with cloud providers, it won't get any better: an instance which costs $52/month with Linux, went up to $77/month with Windows (that's almost 50% on top of Linux (!)).

"For cheaper servers, the difference between Windows and Linux can eat as much as 50% of the server rental price (though for those servers

Time To Market: Familiarity to your Developers

which are more or less optimal price-performance-wise observed difference was closer to 20-30%).

If your game is computationally intensive, and you can support only a thousand players per server (and therefore, if your game is a success, you will need hundreds of servers to run your game), costs become a very important factor, difficult to fight with. In such cases, there is IMHO only one consideration that can trump lower costs for Linux boxes. This one is time to market for your game.

In other words, if you don't have anybody on the team who has ever developed anything for Linux, it is usually a good enough reason to use Windows on the server-side (and yes, it will work, provided that you're careful enough). It is not that to exploit lower cost of Linux boxes you need *all* of your developers to be Linux gurus (after all, you're much better when you can keep your FSMs "pure" anyway, and being "pure" pretty much implies being cross-platform), but if the whole your team has *zero* Linux experience – it will probably qualify as a valid reason to use Windows (that is, if you've already calculated the associated price tag and are ok with it).

An additional (and quite similar) time-to-market-related pro-Windows argument arises if your game is PC-only (or PC-and-Xbox-only). In this case, if you keep your server under Windows, you can have the same code running on server and client quite easily. While such logic has a grain of truth in it, personally I don't really like this line of reasoning. First of all, there isn't that much code to share to start with (it is mostly about the framework which runs FSMs, Communications- and Routing-related FSMs, and client-side prediction if applicable). Second, your FSMs need to be "pure" and cross-platform anyway (see above). Third, even the code outside of FSMs can be made cross-platform without going into vendor-lock-in stuff rather easily. And last but not least, having the same code run on different platforms, while taking additional time, allows to test your code better, improving overall code quality.

It is All about Money 🙁

At the end of the day, if your team consists primarily (but not exclusively) of Windows developers, *and* your game is computationally intensive enough to support only thousands (or even worse – hundreds) of players per server (and you can count on income per player being very limited), you're facing quite a difficult decision.

Usually, under such circumstances time-to-market considerations will override lower server costs, so it is all about the balance of Windows-vs-Linux guys and gals on your team. On the other hand, it is clearly a Business Decision which needs to be made by Business People and is outside of scope of this book. Our job as developers is just to warn business-minded people that Windows servers are going to cost more than their Linux counterparts (and that server/cloud rental difference can be as large as 50%, though likely to be more in around 20-30%; note that these numbers *do not* include traffic, which will be the same regardless of the platform); the rest is not our decision anyway.

On the other hand, if your estimates show that you can handle a hundred thousands players per server – it looks unlikely that license costs will eat too much of your budget either way, so in this case you may be able to use Linux or Windows, whichever-platform-looks-better-for-you. The whole thing is all about numbers, pure and simple.

“Usually, under such circumstances time-to-market considerations will override lower server costs

Mixed Bags

In the context of the discussion above, a logical question arises: “Can we develop our servers for Windows to get it faster, and migrate to Linux later to save costs?“ The answer is “yes, you can, but you need to be extremely vigilant to avoid unnecessary dependencies”. In general, QnFSM model with deterministic FSMs stimulates cross-platform development, so it might be not that difficult, but you still should remember about your intention to migrate later (this, for example, pretty much excludes using fancy-but-Windows-specific things such as completion ports; not that you really need them anyway for FSM-based architecture, see Chapter [[TODO]] for details).

It is also possible to run both Windows and Linux servers on the server-side not just as a part of migration from one to another one, but because of different reasons. Just to give an idea how it may happen: you may need to integrate with a payment provider, that requires you to use DLL, available only on Windows.[9] Ok, you can have that-provider’s-FSM on a different server running under Windows, while having everything else running under Linux. Been there, done that.

---

[8] No “Essentials” edition was observed as a rental option, probably because of license restrictions
[9] and while I hate such providers, throwing them away is not my decision to make ☹

Linux-vs-Windows: Time to Decide

To summarize my arguments above:

- if you want to use Linux because you're familiar with it – you're fine regardless of number of servers you'll need
- if you want to use Windows because you're familiar with it – take a look at the number of servers you expect to be using
    - the price of Windows license is far from negligible (making up to 50% of the rental cost of the server, though usually the price difference is more in 20-30% range), so it can make a significant difference for your ongoing costs after you launch the game
    - in this case, you may want to develop for Windows first (to speed time-to-market), and to migrate to Linux later
        - extreme vigilance to avoid being inadvertedly locked-in is required (see Chapter IV for details). On the other hand, FSMs tend to make dependency fighting simpler.
- if you're in doubt – use Linux, it is safer that way[10]

"if you want to use Linux because you're familiar with it – you're fine regardless of number of servers you'll need

Things to Keep in Mind: Windows

When developing for a specific platform, there are always platform-specific things which you need to keep in mind. For Windows my own favorite list of DO's and DON'Ts goes as follows (note that this is a language-agnostic list, for C++-specific stuff see Chapter [[TODO]]):

- DO fight 3rd-party dependencies. Unnecessary dependencies tend to make Windows less stable, less secure, the code less manageable, etc. Refer to Chapter IV, "DIY vs Re-use: In Search of Balance" for details on "what to DIY and what to re-use".
- DO fight 3rd-party dependencies. Re-use MUST NOT be taken lightly, and extreme vigilance is required.
- DO fight 3rd-party dependencies. In spades. While all the developers are prone to taking some "nice" 3rd-party component and to using it without telling anybody, from my experience Windows developers are more likely to do it than Linux ones.
- DON'T use .NET-based stuff unless absolutely necessary. .NET in production will cause you quite a lot of trouble. If you want to use .NET as your own platform – well, at least you (I hope) know why you're using it, and will be able to configure it to minimize the impact. If you're programming in not-a-.NET-language, running .NET

"DO fight 3rd-party dependencies. In spades.

unless absolutely necessary, is a recipe for several different disasters (ranging from security problems to run-away 3rd-party not-really-necessary .NET component eating all-the-available-resources).

  - Stay away from web services (that is, unless you're into Web-Based Architecture), at the very least for time-critical pieces. In general, any technology that has a blocking RPC interface, should be avoided, as blocking inter-process (and even worse, inter-server) calls don't fit well into our perfectly-non-blocking no-unnecessary-context-switching highly-optimized FSMs, and will cause significant performance degradation compared to them.

- Stay away from COM.[11] COM components have two pretty bad properties. First, it is yet another technology based on blocking RPC calls (see above about them). Second, if you're using COM for your own components – it is quite silly (ok, unless you're using Visual Basic), and if you're using it for 3rd-party components – it is a 3rd-party dependency, you should fight as stated above. Consider an offense of using DCOM as just an aggravated form of the offense of using COM.

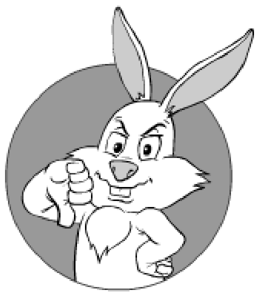
[10] "safer" here can be interpreted in several different ways: from "a little bit safer security-wise" to "safer in case if your profits are much lower than expected, so price of the servers becomes more critical".
[11] yes, I know lots of people consider COM long-dead; unfortunately, it is not

Things to Keep in Mind: Linux

Linux also has it's fair share of DO's and especially DON'Ts. My favourite ones are as follows:

- DO fight 3rd-party dependencies. While from my experience, the danger of 3rd-party dependencies is lower for Linux than for Windows, it still exists.

- DON'T program for one single distribution. Your code should be generic enough to allow jumping around different distros; there is no reason to depend on package manager or exact directory structure. If you need these badly, move this kind of stuff into config files (or into rarely-executed shell scripts), so your admins can adjust directories if necessary.



"DON'T program for one single distribution.

  - As long as we're speaking about Linux (not including BSD), all you really need to use on your Game Server is Linux kernel and glibc. Both will be very much the same for all the distros (with the only difference being kernel/glibc version).

  - If considering *BSD family, they are somewhat different, but as long as you're using POSIX APIs (and that covers 99% of what you'll really want in practice[12]), the differences are negligible

- DON'T use shell scripts for frequently-performed tasks. While an occasional shell script to install your daemon is fine, invoking shell 1000 times a second is rarely a good idea.
    - Pretty much the same goes for cron – DON'T try to get around cron's 1-minute restriction by playing tricks such as running 60 cron jobs every minute, with the first job waiting for one second, the second one waiting for another second, and so on.
- DON'T think that threads are much faster than processes on Linux (at least not that much as they are on Windows). And BTW, it is not that threads on Linux are slow, it is that process creation (fork()) is fast. On the other hand, you may still want to use threads if you're after cross-platform development.

[12] the remaining 1% includes things such as epoll/kqueue

Things to Keep in Mind: All Platforms

In addition, there are a few things to remember about, which apply regardless of the platform you're developing for:

- DON'T use platform-specific APIs within your FSMs (see below about using them outside of FSMs). Leaving aside a few narrow exceptions, your FSMs need to stay "pure" (see Chapter V for discussion of the associated benefits), and platform-specific APIs is #1 enemy of the code being "pure".
- DO consider cross-platform code even outside FSMs. The whole QnFSM can be written in a fully cross-platform manner.[13] Even if you find platform-specific optimizations, it is better to have a purely cross-platform version (at the very least, to have a baseline to compare your optimizations against).[14]

"DO consider cross-platform code even outside FSMs.

[13] been there, done that
[14] I've seen quite a few "platform-optimized" versions which were actually slower than cross-platform ones, and even more platform-optimized stuff which was exactly on par with the cross-platform one

[[To Be Continued…

This concludes beta Chapter VI(c) from the upcoming book "Development and Deployment of Massively Multiplayer Games (from social games to MMOFPS, with social games in between)". Stay tuned for beta Chapter VI(d), "Modular Architecture: Server-Side. Programming Languages.]]

[–] References

[Corbet2013] Jonathan Corbet, "NUMA scheduling progress"
[Lameter2013] Christoph Lameter, "NUMA (Non-Uniform Memory Access): An Overview"
[IDC] "Windows 2000 Versus Linux in Enterprise Computing"
[Cybersource] "Linux vs Windows. Total Cost of Ownership Comparison"
[RFG] "TCO for Application Servers: Comparing Linux with Windows and Solaris"
[PCWorld] Katherine Noyes, "Five Reasons Linux Beats Windows for Servers"

Acknowledgement

Cartoons by Sergey Gordeev from Gordeev Animation Graphics, Prague.

« *Chapter VI(b). Server-Side Architecture. Front-End Servers a..*

Asynchronous Processing for Finite State Machines/Actors:... »

Filed Under: Distributed Systems, Programming, System Architecture
Tagged With: game, Linux, multi-player, server, Windows