Windows for Linux Nerds

Saturday, September 9, 2017

I recently started a job at Microsoft. In my first week I have already learned so much about Windows, I figured I would try to put it all into writing. This post is coming to you from a Windows Subsystem for Linux console!

New job and I got a Windows computer and a Linux computer! If you are new to my blog, let me tell you: I love setting up a perfect desktop experience. I've written a few posts on it (for Linux), you should check them out. Setting up a Windows computer is something I have not done in quite some time. I will describe a bit how to set up a windows machine in a reproducible way at the end of this post.

I would like to thank Rich Turner, John Starks, Taylor Brown, and

Sarah Cooley for taking the time to explain a lot of the following to me. :)

Windows Subsystem for Linux (WSL)

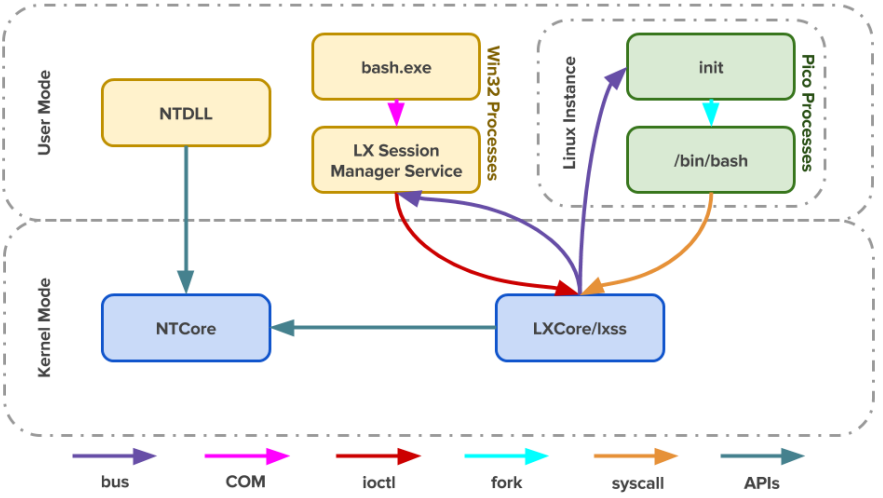
Let's start with Windows Subsystem for Linux, aka WSL. Even @monkchips wrote that since I joined Microsoft "Linux Subsystem for Windows will definitely be getting a workout." I am super excited about Windows Subsystem for Linux. It is one of the coolest pieces of tech I've seen since I started using Docker.

First, a little background on how WSL works…

You can learn a lot more about this from the Windows Subsystem for Linux Overview. I will go over some of the parts I found to be the most interesting.

The Windows NT kernel was designed from the beginning to support running POSIX, OS/2, and other subsystems. In the early days, these were just user-mode programs that would interact with `ntdll` to perform system calls. Since the Windows NT kernel supported POSIX there was already a `fork` system call implemented in the kernel. However, the Windows NT call for `fork`, `NtCreateProcess`, is not directly compatible with the Linux syscall so it has some special handling you can read about more under System Calls.

There are both user and kernel mode parts to WSL. Below is a diagram showing the basic Windows kernel and user modes alongside the WSL user and kernel modes.



The blue boxes represent kernel components and the green boxes are Pico Processes. The LX Session Manager Service handles the life cycle of Linux instances. LXCore and lxsys, `lxcore.sys` and `lxss.sys` respectively, translate the Linux syscalls into NT APIs.

Pico Processes

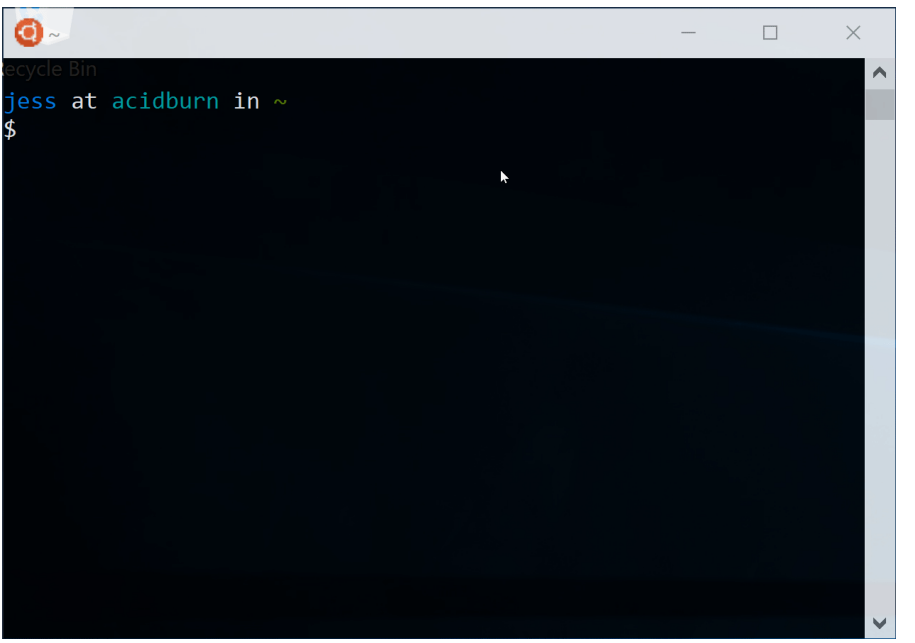
As you can see in the diagram above, `init` and `/bin/bash` are Pico processes. Pico processes work by having system calls and user mode exceptions dispatched to a paired driver. Pico processes and drivers allow Windows Subsystem for Linux to load executable ELF binaries into a Pico process' address space and execute them on top of a Linux-compatible layer of system calls.

You can read even more in depth on this from the MSDN Pico Processes post.

System Calls

One of the first things I did in WSL was run a syscall fuzzer. I knew it would break but it was interesting for the purposes of figuring out which

syscalls had been implemented without looking at the source. This was how I realized PID and mount namespaces were already implemented into `clone` and `unshare`!



The WSL kernel drivers, `lxss.sys` and `lxcore.sys`, handle the Linux system call requests and translate them to the Windows NT kernel. None of this code came from the Linux kernel, it was all re-implemented by Windows engineers. This is truly mind blowing.

When a syscall is made from a Linux executable it gets passed to `lxcore.sys` which will translate it into the equivalent Windows NT call. For example, `open` to `NtOpenFile` and `kill` to `NTTerminateProcess`. If there is no mapping then the Windows kernel mode driver will handle the request directly. This was the case for `fork`, which has `lxcore.sys` prepare the process to be copied and then call the appropriate Windows NT kernel APIs to create and copy the process.

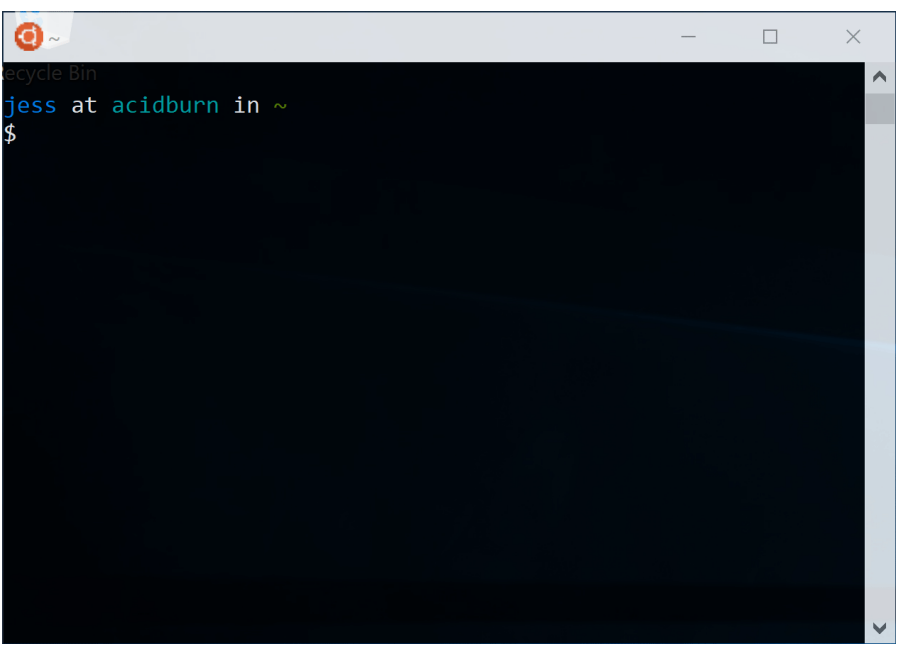
You can learn more from the MSDN System Calls post.

Launching Windows Executables

Since WSL allows for running Linux

binaries natively (without a VM), this allows for some really fun interactions.

You can actually spawn Windows binaries from WSL. Linux ELF binaries get handled by `lxcore.sys` and `lxss.sys` as described above and Windows binaries go through the typical Windows userspace.

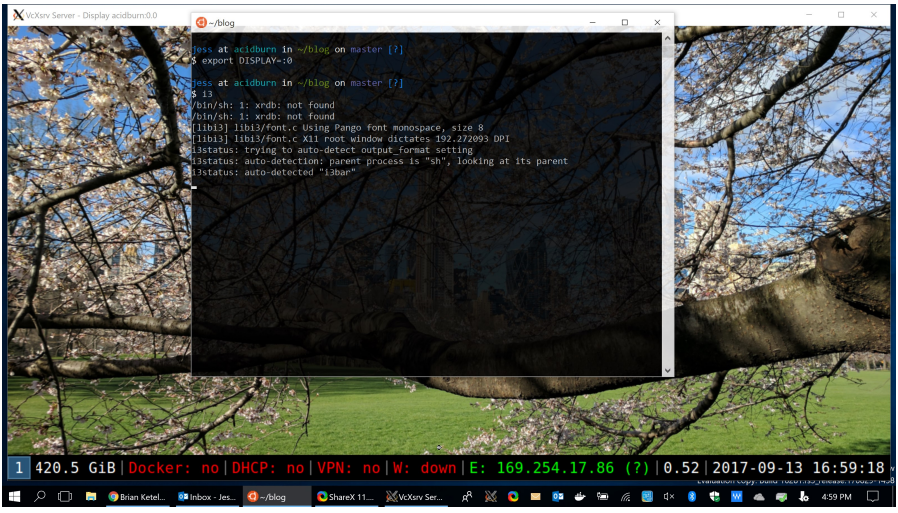


You can even launch Windows GUI apps as well this way! Imagine a Linux setup where you can launch

PowerPoint without a VM…. well this is it!!

Launching X Applications

You can also run X Applications in WSL. You just need an X server. I used `vcxsrv` to try it out. I run `i3` on all my Linux machines and tried it out in WSL like my awesome coworker Brian Ketelsen did in his blog post.



The hidpi is a little gross but if you play with the settings for the X server you can get it to a tolerable place.

While I think this is neat for running whatever X applications you love, personally I am going to stick to using `tmux` as my entrypoint for WSL and using the Windows GUI apps I need vs. Linux X applications. This just feels less heavy (remember, I love minimal) and I haven't come across an X application I can not live without for the time being. It's nice to know X applications can work when I do need something though. :)

Pain Points

There are still quite a few pain points with using Windows Subsystem for Linux, but it's important to remember it is still in it's beginnings. So that you all have an idea of what to expect I will list them here and we can watch how they improve in future builds.

Each item links to it's respective GitHub issue.

Keep in mind, I am using the default Windows console for everything. It has improved significantly since I played with it 2 years ago while we were working on porting the Docker client and daemon to Windows. :)

- **Copy/Paste**: I am used to using `ctrl-shift-v` and `ctrl-shift-c` for copy paste in a terminal and of course those don't work. From what I can tell `enter` is copy… supa weird… and `ctrl-v` says it's paste. Of course it doesn't work for me. I can get paste to work by two-finger clicking in the term, but that does not work in `vim` and it's a pretty weird interaction.
- **Scroll**: This might just be a *huge*

pet peeve of mine but the scroll should not be able to scroll down to nothing. This happens all the time by accident for me with the mouse and I have no idea why the terminal is rendering more space down there. Also typing after I have scrolled should return me back to the console place where I am typing. It unfortunately does not.

- **Files Slow**: Saving a lot of files to disk is super slow. This applies for example to git clones, unpacking tarballs and more. Windows is not used to applications that save a lot of files so this is being worked on to be more performant. Obviously the unix way of "everything is a file" does not scale well when saving a lot of small files is super slow.
- **Sharing Files between Windows**

and WSL: Right now, like I pointed out, your Windows filesystem is mounted as `/mnt/c` in WSL. But you can't quite yet have a git repo cloned in WSL and then also edit from Windows. The VolFS file system, all file paths that don't begin with /mnt, such as /home, is much closer to Linux standards. If you need to access files in VolFS, you can use `bash.exe` to copy them somewhere under `/mnt/c`, use Windows to do whatever on it, then use `bash.exe` to copy them back when you are done. You can also all Visual Studio code on the file from WSL and that will work. :)

Setting Up a Windows

Machine in a Reproducible Way

This was super important to me since I am used to Linux where everything is scriptable and I have scripts for starting from a blank machine to my exact perfect setup. A few people mentioned I should check out boxstarter.org for making this possible on Windows.

Turns out it works super well! My gist for my machine lives on github. There is another powershell script there for uninstalling a few programs. I love all things minimal so I like to uninstall applications I will never use. I also

learned some cool powershell commands for listing all your installed applications.

```
#--- List all installed programs --#
Get-ItemProperty
HKLM:\Software\Wow6432Node\Microsoft\
Windows\CurrentVersion\Uninstall\*
| Select-Object DisplayName,
DisplayVersion, Publisher,
InstallDate
|Format-Table -AutoSize

#--- List all store-installed
programs --#
Get-AppxPackage | Select-Object Name,
PackageFullName, Version |Format-
Table
-AutoSize
```

I am going to be scripting more of this out in the future with regard to pinning applications to the taskbar in powershell and a bunch of other settings. Stay tuned.

Overall, I hope you now understand

some basics around Windows Subsystem for Linux and are as excited as I am to see it grow and evolve in the future!