Troubleshooting: Putting it all together

Troubleshooting a process or system issue will quite often require exploring multiple aspects or factors. I've mentioned a lot in the last couple of lectures, but I want to go back over some again.

#1 on the list is Logs and Logging in general.

Generally, log text is created by the program author, and therefore can very precisely relay what is going on -- often eliminating a lot of the (potential) guesswork. But, this takes effort on the programmer's side: they must ensure they adequately log both expected and unexpected outcomes, and yet also be certain they don't over-log (or at least provide enough knobs to adjust the amount of logging).

That said, good logs can often instantly tell you what's wrong with a given piece of software or system, while bad logs can be worse that no logs (lead you on a wild goose chase!) For example, interleaved logging without identifying data tying log lines together is really bad. I talked about this before, but it's ridiculously important, so I'm reiterating it :)

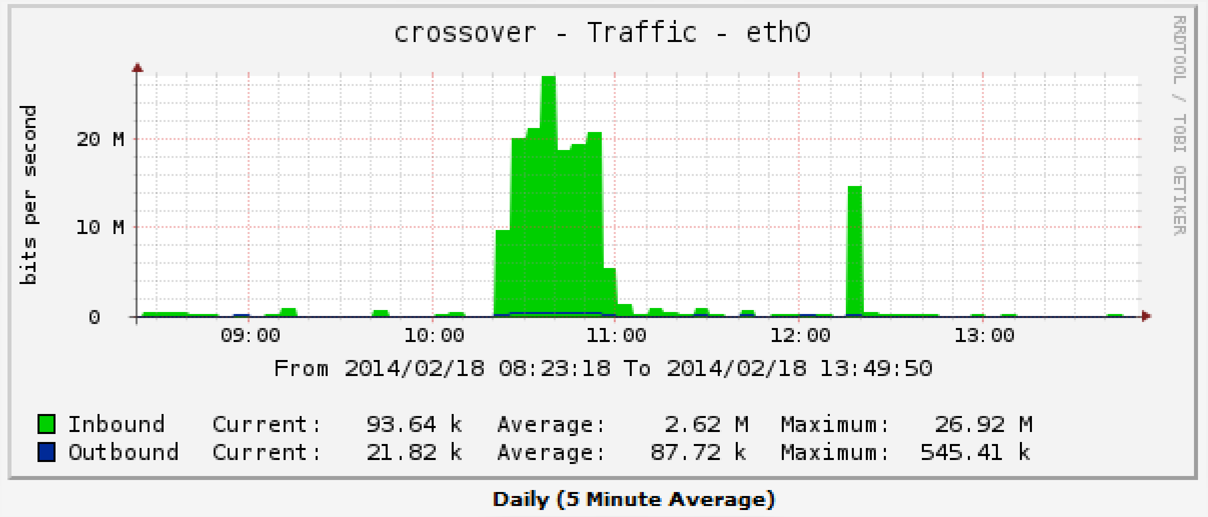
Unless you can be certain that either each process is logging to its own logfile, or that things are always (ALWAYS) single-threaded, you'll want to at least have a "thread id" (or other unique but consistent identifier) of sorts to tie log lines together. Other contextual data will also usually help -- for example a UserID or perhaps database connection ID, etc.

Logging is great when it's telling you exactly what's going on with a given process, but sometimes you need system-level data or logs. The utilities I've covered already (uptime, ps, vmstat, top, iostat, netstat) just give you an instantaneous snapshot of what's going on, but often you'll need/want historical information.

The collection of (and alerting on) system data is essential in larger architectures. One of the most basic way to do this is with SNMP (Simple Network Monitoring Protocol – UDP port 161), which can collect very basic data, but also has extensions that allow you to collect specific data as well. These can usually be extended to include particular process data, which can help in your troubleshooting!

Larger systems will often employ custom monitoring solutions -- either home-grown, or commercial off-the-shelf ones. While home-grown solutions are often the most "exact" when it comes to doing what you want them to do (since you control the code), they also demand resources (full-time programmers, often) -- and so some companies will opt to go for off-the-shelf, or at least some open source "packaged" solutions.

An example of what can be seen with a common open-source solution is on the next page -- where we're looking at the network traffic on my router machine one evening.



This data makes it evident I was downloading something for a little over half an hour or so.

Analyzing this sort of traffic can help even diagnose potential machine compromises (odd/unexpected traffic or system load, etc).

There are other system-level utilities that can help you track and monitor processes over time. You could use strict process accounting (`lastcomm`), which essentially logs every command run. Something like this would really help you see what's being run on the system -- but for busy systems, it can be a lot of data.

System utilities like '`sar`' will collect a ton of useful statistics (data similar to what is found from '`vmstat`', '`iostat`', '`netstat`', '`uptime`', etc) on regular intervals, for later examination. This can be a lot of data, too, but sometimes it's essential when tracking down issues. However, since it is a lot of data, sometimes it's only kept around for a few days or weeks -- meaning if you have an issue to troubleshoot, the sooner you can get to it, the better ;-)

A very powerful system utility that can really tell you a LOT about what’s going on, is “strace”.

This utility will often need to be run as ‘root’, but either way, it should give you a trace of all system calls. I’ll show you some examples.

Overall, when troubleshooting an issue, you will often times need to use all the available data you can find: System-level, Process-level, network-level, and (when applicable) user-experience-level ("what is the user experiencing?" Hopefully something a little more useful than "it doesn't work" ;-))