iRules Concepts: Connection States and Command Suspension

Colin Walker, 2012-04-01

Within iRules, in addition to mastering the language syntax and tricks therein, there is an added layer of understanding that can help in taking your iRules fu to a whole new level. Connection State. Connection states are relatively simple in concept, but aren't referenced very often other than by those intimately familiar with iRules underpinnings. Far more we hear people talking about "suspended" commands or "command suspension". Even this concept, though, is enshrouded in a fair amount of mystery to most with whom I've spoken of the topic.

What does command suspension mean? When do things suspend? What does that mean for my traffic? Hopefully the mysteries will be fewer and further between after reading through this explanation of what the available command states in iRules are, what they mean, and what this whole "Command Suspension" thing is. Throughout this article I'll be interchangeably talking bout connection and iRule state/suspension. I've heard both terms used, they mean the same thing - the current or suspended state of the connection. However where it pertains to an iRule you will often hear people use either term.

To start, there are multiple different states that your connection can technically live in. This article will be largely concerned with the Parked or "Suspended" state, but we'll cover them all to at least some degree. Those states and a brief description are:

- Pending
    - In which the byte code is available, but has not yet been called by an event.
- Executing
    - The byte code in which the iRule was stored is currently executing.
- Suspended or Parked
    - Execution of the current iRule is suspended.
- Resumed
    - Execution is resumed after a suspension.

Let's cover each of these states in turn in more detail so you can get some context and understanding of what's actually happening here. Keep in mind that the Parked or Suspended and Resumed states do not occur for every iRule, on

Pending

This basically means that nothing is happening. No really, that's what it means. In the pending state the iRule has been compiled into byte code and that byte code is available for execution via TMM. What is byte code? Well, when an iRule is saved to the system it is pre-compiled down into byte code. Byte code is simpler, more organized, and more efficient for TMM to execute, in loose terms. Perhaps if there's demand another article specifically discussing byte code will show up later. For now, what you need to know is that when you save your iRule, it gets transformed into this state, which ensures efficiency. This also answers the question that has been asked repeatedly: "Does the TCL parser have to spin up for each new connection/request that executes an iRule?". The answer is no, it does not. The parsing was done at load time.

Executing

For our second largely self explanatory state, the executing state of an iRule is when things are currently...shocking....executing. To get to this point, you wrote your iRule, saved it, it was parsed, compiled into byte code and sat in the pending state until it was executed by TMM. That is, when the appropriate event that you coded into your iRule was called. For instance if your iRule referenced HTTP_RESPONSE, the first HTTP traffic through from the client to the server would not execute your iRule, only the HTTP based response from the server to the client would. This is where iRules in action live 99% of the time (note: statistic fabricated), and is where all commands are executed, results are created and stored, etc. Anything that happens within an iRule happens here, unless suspension occurs. The TMM responsible for the iRule in question will be busy with the iRule being executed until it exits this state successfully or otherwise.

So if, for instance, you write while loop that never ends, you are permanently tying up a TMM each time your iRule executes. This was how some people, years ago, would induce delay in a connection to simulate a WAN environment, by forcing their iRule into a loop for a given amount of time, thereby creating that much latency for every connection. This is, however, on the "extremely un-recommended" list, and not a good idea at all. There are far better ways to achieve such things, and monopolizing TMMs is a bad practice in general.

Suspended or Parked

"Suspended" and "Parked" are different names for the same thing, the terms are used interchangeably. When an iRule is suspended it saves all of the state information necessary to resume processing, then sets the connection in question aside until the command that caused the suspension returns. In this way it allows the TMM responsible for that iRule's execution to resume processing other traffic. This is an important state for a couple of reasons. First, TMM is single threaded. Second, some iRules commands take a while to execute, relatively speaking, for various reasons. While they are in the executing state, remember that the TMM is tied up and not passing traffic.

Rather than leaving it there, we'd rather free it up to continue processing other traffic until the command that induced the suspension is ready to resume and complete the request or response. Hence the existence of the Suspended state. Years ago when dinosaurs walked the earth and I began iRule coding, this didn't' exist and there were murky issues surrounding some commands that really could have benefited from it. Now days we're able to easily set aside a connection for a little while to allow processing where necessary with no need for sorcery or occult practices in our code, and without slowing other traffic down.

Keep in mind that when I say "a little while" we're likely talking very, very miniscule amounts of time here (See: milliseconds at most unless specifically dictated by the iRule owner, in most cases). Even given that, some people are curious why some commands are suspend worthy. Different commands take different amounts of time to process, naturally, but the commands that induce suspension are almost solely waiting on some outside resource. Whether that is executing a DNS lookup or inducing intentional delay, there is enough time between execution and response that you don't want traffic sitting and waiting behind a busy TMM.

For instance, the most basic and straight-forward suspending command is the after command. The after command is designed to cause suspension. It takes two (optionally 3) arguments: the amount of time to "sleep" for, and the code to be executed after that delay. For instance if you want to add a log entry 1 second after you get a request to "bob.com" you could run something like this in HTTP_REQUEST:

```
if {[HTTP::host] eq "bob.com"} {
  after 1000 {
    log local0. "Host: [HTTP::host]"
  }
}
```

This will effectively put your connection aside for 1 second, add a log entry, and then allow your connection to resume and continue processing as normal. This would be doable in other fashions such as loops and whatnot, but those older, not recommended practices would leave the iRule in an executing state. As such, traffic would begin to be delayed once all available TMMs were busy. This is an extreme and simple case, of course, but the concept applies elsewhere as well.

Take DNS lookups for instance, another common case for iRule suspension these days. You could, theoretically, build your own DNS request engine thanks to side-band connections, and could force the iRule to sit and wait for a response, again tying up TMM...but why? The RESOLV::lookup command is a command built specifically for DNS queries. With the knowledge in mind that these queries regularly take orders of magnitude longer to return than the average time it takes for an LTM to pass a request from client to server, the command was designed to suspend instead of allowing an iRule to tie up TMM and cause delays. The parent connection of the RESOLV::lookup command gets suspended, so you can reliably make decisions on the information returned from that command, since traffic processing halts until it returns, but you aren't binding up a TMM while you're at it.

These aren't the only commands that suspend, and I'm not going to try and give an exhaustive list here, but some examples are:

- after (always)
- RESOLV::lookup (if the result isn't cached)
- table (if the data in question isn't stored within the same TMM)
- session (if the data in question isn't stored within the same TMM)
- persist (if the data in question isn't stored within the same TMM)

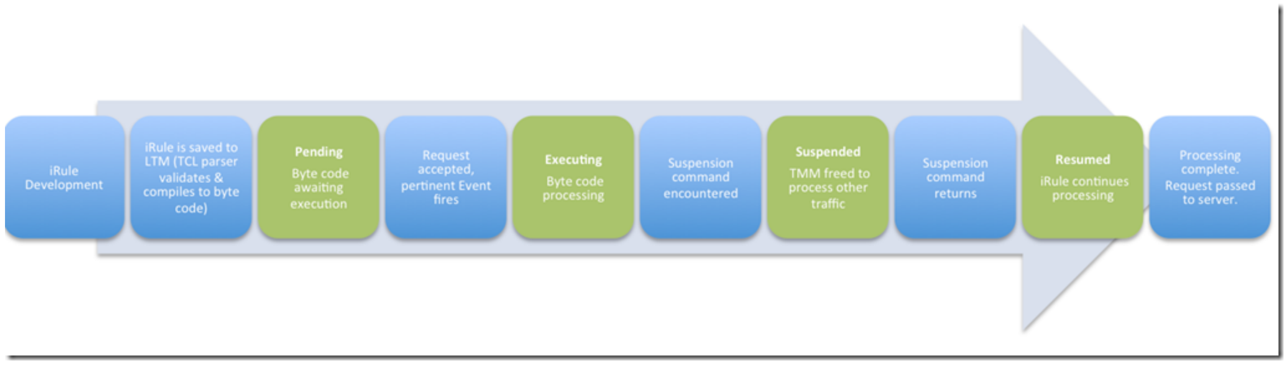
If there is enough demand, I'll work on getting a list together of all possible suspension cases, but for now hopefully this illustrates the concept and some of the most common cases.

Resumed

The final state in which a connection can exist is resumed. This is very simply an execution state that has been resumed after a suspension. Behavior here is the same as before, it is just differentiated by the fact that this connection has been suspended already. A connection will enter this state when a suspending command returns, whether successful or via timeout or error.

Conclusion

Once the connection is resumed, processing will complete as normal, and the traffic will be forwarded on to the destination server. This is unseen to either client or server, and often only involves miniscule amounts of latency, so it will largely go unnoticed. The connection state / suspension process isn't overly complex, but can be confusing to follow until you're comfortable with it. So, to recap what it looks like from beginning (iRule creation) to end (traffic passed through an iRule, suspended, resumed, and forwarded on to the destination server), I will close with a somewhat simplified graphical representation. I've highlighted the four connection states discussed above in green for clarity:


iRule Development
iRule is saved to LTM (TCL parser validates & compiles to byte code)
Pending
Byte code awaiting execution
Request accepted, pertinent Event fires
Executing
Byte code processing
Suspension command encountered
Suspended
TMM freed to process other traffic
Suspension command returns
Resumed
iRule continues processing
Processing complete. Request passed to server.

Hopefully this helps to de-mystify connection state and command suspension within iRules. As always, any questions are more than welcome in the comments.