inetd enhancements

Arjun Bemarkar

About the project

Overview

`inetd` enhancements to allow it to babysit general daemons, not just networking ones: the ability to keep daemons alive (even when they crash), start them after various specified events/intervals and specify scheduling priority of them.

In addition, a utility will be added to start/stop these daemons.

Current State

In its current form, you can tell `inetd` to startup network daemons by editing `/etc/inetd.conf` and specifying an ingress port. You can also list program arguments that you want `inetd` to supply. When there is an incoming connection, `inetd` will startup the daemon with the arguments and supply the incoming data to it through `stdin`.

Deliverables

1. Update existing `inetd` code to support the new feature set
2. Feature documentation & User Guide
3. New utility to start/stop daemons

I am proposing to let `inetd` start other daemons based on events which aren't purely network events and keep them alive based on newly added parameters given to `inetd.conf`

Specifics

specific parameters to add to `inetd.conf`:

`keep_alive`

- `bool`
- repeatedly restart Whenever the program crashes (non zero exit), `syslogd(8)` will be notified.

`successful_exit`

- `bool`
- if true, the daemon will be restarted until it fails.
- if false, the daemon will be restarted until it succeeds

`network_state`

- `boolean`
- if true, the daemon will only (re)start when network is available

`path_state`

- `string`
- if true, the program will be alive as long as the given path exists
- if false, the program will be alive as long the path doesn't exist

`throttle_interval`

- `int` (some default value will be decided)
- time in seconds to wait between program restarts

`nice`

- `int`
- run he program at an altered scheduling policy

And a command line utility which will allow the user to make individual configuration files and start and stop them on demand.

Milestones and Schedule

Following the Google Summer of Code schedule, coding officially begins on June 13th. By this time, I should be familiar with my mentor and up to speed with the codebase from the "Community bonding period" (May 20 - June 12).

Until the first evaluation phase, I have 7 weeks. I predict this project to be finished in 9 - 10 weeks.

A 350 hour project means it will come to ~32hr work week. I don't have any other commitments for this summer, so this should be doable.

Here is a brief overview. Of course, it is likely to take longer than I anticipate and plan to, so I have left two weeks as a buffer.

Week 1(starting June 12)

Make changes to `inetd.conf` syntax so it can accept processes which aren't necessarily invoked by activity on a network port.

Week 2,3,4

Create and add support for a command line utility so it can add/remove daemons. This should deal with unix credentials and IPC with the daemon.

Make inetd start processes based on events other than network ones.

Week 5

add `keep_alive`

Week 6

add `successful_exit`

Week 7

add `network_state` and `path_state`

Week 8

add `throttle_interval` and `nice support`

Week 9

Week 10

Similar Software

MacOS has launchd Linux has systemd

Licensing

All features are inspired by launchd. Since that is under APSL, there shouldn't be any licensing issues.

About the project and NetBSD

Have you rebuilt the kernel and the userland, either in full or in parts?

Yes, I have rebuilt the inetd binary and got the tests running through atf.

How will your project integrate into NetBSD

`inetd` is ran at boot time by `/etc/rc`. I will be modifying this program, so it should start the daemons it is responsible for. `/etc/inetd.conf` will be used to use the new features

What interfaces in NetBSD will your project use?

in order to monitor and restart services when they crash, i will use various system calls including fork with `execve` and system libraries including `signal.h`, `wait.h` to setup signal handlers in order to handle the child daemons dying.

in order to check network availability, the `getifaddrs(3)` interface will be used. Here I can check whether the network is up with the `ifa_flags` (these flags are similar to the `ifconfig` utility)

Simple `access(2)` function will be used to wait until a file is created.

`setpriority(2)` will be used to set the "niceness" of the daemon.

there already exists a concept of time tracking in the `inetd` codebase through `ratelimit.c:rl_time()`. So, for `set_interval`, this interface can be used.

To what degree are you familiar with those interfaces?

I am relatively familiar with them. I have been playing around with them over the past month in preparation for this project.

I've also looked through Apple's launchd source code and feel comfortable with using them.

About you