
Morgan Gangwere GSOC 2016

*** FINAL DRAFT ***

A proposal for the NetBSD project
Inetd improvements

*** DRAFT HISTORY ***

2016-03-21: Initial proposal presented to Google Summer of Code 2016
2016-03-21: Fix some wording, make the whole document fit in 80cols
2016-03-21: Add better contact information
2016-03-22: Refine information about goals (prefork) and bonuses.
2016-03-24: Clarify certain points
2015-03-25: Final edits + argument for proposed path.

*** PROVIDING FEEDBACK / CONTACTING THE AUTHOR **

Feedback can be provided directly to me via email < morgan.gangwere@gmail.com >
as well as via the NetBSD tech-userlevel mailing list (where this will be
posted for public review)

I may be reached over XMPP via indrora@rows.io or via email (see above). I can
also be found on Freenode as indrora (responses may be delayed due to ZNC/idle)

A copy of my resume is available at http://tsunami.zaibatsutel.net/cv.pdf

This document is available in plain text at
http://tsunami.zaibatsutel.net/gsoc16_netbsd_proposal.txt

-~-=-~-

1. Introduction (About the author)

I'm Morgan Gangwere, a student from the University of New Mexico in Albuquerque,
New Mexico. I've worked on lots of things, but some of my favorites are open
source contributions. I've made contributions in my past to the libmtp project,
fixed bugs in Travis (in the end, the patch wasn't accepted -- my solution was
against the documented details, but it sparked the conversation), worked around
Android's limitations in OpenKeychain, then helped people communicate when I
forked yaaic as Atomic and started making it better.

I've built my own kernel, booted UEFI by hand, forged raw TCP sockets from raw
hot bits with a hex editor and a sledge, helped port Android to a new phone,
pushed new binaries to devices over TFTP. I've worked out what a "bad cast to
std::Allocator<_T, alloc<T>>(std::stream_t<T>)" means. I've created ext2 fs
superblocks by hand, beaten U-Boot into compliance, slimmed down FAT filesystems
and run rsync over amateur radio.

My editor is vim, my shell is zsh, my work? it makes the internet happen. My
goal is to make the internet a good place for data to live.

I got a degree in network administration (focusing on Cisco networks) back when
I was in high school. I automated my work in my sophomore year of classes, often
spending the second half of the class playing quake. I learned how to build
networks that are reliable and consistent in those classes. Since then, my work
has focused on making things reliable and easy to manage.

I'm familiar with NetBSD; I typically run it in virtual machines but also use it
on the odd 'I need a lightweight not-linux' system. I'm not afraid of digging
through manpages. I'm definitely not afraid of digging into /usr/src to find the
function I'm looking for.

I believe in Systems Engineering -- the idea that all things should be done with
a requirements document, a plan and a good debate as to whether this is the
right direction to go. I believe in change requests, in code that is built well

and comments that explain what is being done. I believe in code quality, secure design, portability and expected execution. I believe in clever design over clever implementation, a value I know the NetBSD project holds dearly.

2. The problems with inetd

Inetd is plagued with a few problems that the NetBSD projects wants to address through Google Summer of Code, as well as one that I personally wish to tackle early on to tackle a maintainability issue that plagues the NetBSD project.

There's a few bonuses here and there that I'd like to implement. Implementing a new per-service configuration format makes adding features to inetd simple and paves the way for other tools to do the same.

2.1 No early service availability

In conversation with Riastradh on IRC, this is sort of a hot-button topic. The GSoC project page specifies the following:

> Prefork: Support pre-forking multiple children and keeping them alive
> for multiple invocations.

How this should be implemented or otherwise handled is left very much as an "exercise to the reader".

As it stands, inetd waits until a service is requested (through kqueue) to fork() and exec(). It's simple, straightforward and is a direct approach to the problem.

FreeBSD's inetd has a "max children" but has nothing to create children early, only a mechanism to limit the number of sessions at any one given time. [FREEBSD-INETD] However, no

In 1999, we faced the c10k problem; cdrom.com saw 10,000 concurrent connections to its FTP server. As more devices hit the net, we saw the c100k problem, then in 2014 or so, we began seeing the c10m problem: 10 million active connections in software like Facebook, Twitter, etc. [C10K] [C10M]

kqueue is a competent system to deliver messages among processes and inetd can use this in a manner similar to how nginx's event-driven worker system. This path works well for Nginx and for fairly direct, computationally inexpensive services such as those which inetd is built for.

Several years ago (2014), a set of informal studies were done comparing nginx and Apache against each other with various configurations. One such found that nginx's pure event system was better for the static content that was being served in a specific case. [APACHE-PREFORK-NGINX]

2.2 A configuration format from ages ago

inetd's configuration format is right out of 1980. Whitepsace-delimited, it's a definite relic of how we used to do software. And it was good back when people didn't want to put their entire system's configuration into version control. But now we do want our system to be held inside a version control system and oh man, how are we going to do that?

The biggest question becomes "what happens when someone upgrades from NetBSD-N to NetBSD-N+1?"

FreeBSD just crammed more options on top of the wait/nowait column. This isn't a good design decision as it really encourages shoehorning features on top of a format. [FREEBSD-INETD]

... Choices need to be made ...

There's a *lot* of things inside the inetd configuration file that make it a hard to parse format that needs to be migrated away from in the long run. A format for setting bindhosts, specifying ipsec rules -- all these have been shoehorned onto the inetd configuration through special keywords that change the state of the parser as it's going on.

```
141
142 2.3 A solid log of a codebase
143
144 The codebase is pretty much one single file. Modern compilers and modern build
145 systems encourage smaller compilation units for massively parallel compiles.
146
147 I'd personally like to start a trend where NetBSD moves from where the tools
148 are maintained as only a few large files and broken into multiple compilation
149 units.
150
151 This has one upshot: Multiple authors can work on the same tool and not have
152 merge conflicts
153
154 3. Changes that should be made
155
156 3.1 preforking: A change to the fork()/execve() model:
157
158 We must take note of our fellow web daemons, study their actions. Their attack
159 is that of preforking. Implementing preforking in inetd requires a certain
160 restructuring of the codebase.
161
162 This restructuring means that we need to reconsider how inetd handles its child
163 pids -- notably, it currently just keeps a linked list of them and cleans up
164 some things after the children are done.
165
166 The proposed mechanism for preforking is restructuring the `servtab` such that
167 the number of fields is reduced. This would create (roughly) the following
168 C-style structures:
169
170 struct service {
171     char     *name;         /* service name */
172     uint8_t   type;         /* type of service */
173     int       proto;            /* protocol */
174
175     char     *progpath;     /* program path (not name) */
176     char     *argv;         /* arguments to pass to the program */
177     int       port;         /* tcp/udp port to run on (allows for override
178                                from /etc/services?) */
179
180     struct {
181         int children;       /* number of children to keep at hand at any time */
182         int overload;       /* Allow overflow (fallback to fork()/exec() on
183                                demand after pool is exhausted) */
184     } child_opts;
185     struct {
186         int hits;           /* Number of times a service can be used ... */
187         int time;           /* ... over a certain amount of time ... */
188         int cooldown;       /* ... before we wait for some time
189                                and let it cool */
190     } load_opts;
191
192     /* ... */
193
194 }
195
196 struct child_service {
197     char    *name;      /* tied to service name */
198     int     fd;         /* file descriptor we have to read/write (stdin
199                            and stdout bound to this) */
200     /* ... other control stuff */
201 }
202
203 (this isn't final by any means)
204
205 3.2 Per-Service configuration
206
207 This is by far the hardest part of what needs to happen to inetd. Inetd's
208 configuration hasn't been made with backwards (or really forwards) compatibility
209 in mind, and as a result we're having to collect on the technical debt.
210
```

3.2.1: The options on the table:

3.2.1.1: Fork inetd, creating inetd-legacy (default) and inetd-ng (not default).

In this situation, we can ignore the past and rebuild as neccesary, totally revamping the inner workings of inetd. However, this comes at a bit of a cost in that we've now got *two* versions of inetd to maintain! (I seriously doubt that he netbsd team wants to maintain two versions of inetd.)

Pros:

- Erasure of the past's technical debt with regards to the configuration
- keeps upgrades simple: Not ready? Keep the old behavior as it is, get no new features

Cons:
- Two versions of inetd to maintain (with a consistently diverging codebase
- -ng has the very realy chance of just rotting and being left in the rain.
- Your daily dose of techncial debt at twice the cost: Fixing bugs in the old (-legacy) variant *plus* any bugs in the new (-ng) variant.

3.2.1.2: Want new features? Use the new format. Otherwise, fine whatever.

In this situation, we keep all the old parsing of /etc/inetd.conf as it stood. Any usage of new features is dependant on the new configuration format.

Pros:

- Existing installations can upgrade and not have to worry about it.
- Adding new features simple (new features are only in the old format)
- Zero upgrade cost

Cons:

- Support *nightmare* - you have to ask "are you using the old or the new format? What's your setup?" to see if there's a service collision. Ewww no.
- All the technical debt of the old version is still there, plus new, added complexity from the two formats.
- Encourages old users to keep doing what they're doing, "but it's always been done like this"
- Someone, somewhere is going to abuse some aspect of this combination to do something nasty.
- No incentive to move to the new format
- Documentation nightmare

christos@ suggests this as the best route.

3.2.1.3 Drop the old format, introduce the new format, keep a tool to spit out configuration files in the new format.

Here, we totally drop the old table-driven style configuration and move to a format that's tolerable for everyone.

As part of this, include a tool (using awk, python, whatever) that approxomately creates the new format to let people migrate a service.

Pros:

- Erasure of technical debt (inetd configuration can be gutted and rebuilt)
- Super-duper simple addition of configuration options
- Backwards compatible through the future (bring a newer service file in and all known configuration knobs will be brought in with it)
- a new, consistent format means easier documentation.

Cons:

- Documentation needs to be rewritten
- no backwards compatibility from the past *except* through a tool or manual configuration
- Need to maintain the tool to convert inetd lines into config lines.

```
  - Future needs to be told that the past changed the configuration format:
    Documentation needs to be rewritten but include a mechanism to find the
    old format's documentation.

    This is my preferred route.

3.2.1.4 shoehorn new options on top of the old ones as FreeBSD did

This is what several of our siblings have done. FreeBSD in particular has done
several things to try and cram more into the fields of the original inetd config
format.

FreeBSD has made the following the columns for /etc/inetd.conf:

service-name
socket-type
protocol
{wait|nowait}[/max-child[/max-connections-per-ip-per-minute[/max-child-per-ip]]]
user[:group][/login-class]
server-program
server-program-arguments

This makes sense in FreeBSD's case. Not so much in the case of NetBSD.

Pros:

 - Familiar format
 - Does not take a lot of time to implement similar extensions

Cons:

 - Shoehorning is only sustainable for so long
 - *not* future proof nor backwards compatible
 - Doesn't allow for a lot of future changes (e.g. chroot, etc)

3.2.2: Suggested new format

Up for suggestion is the following:

* Key-value pairs in the style of sysctl
* A set of directories within /etc, /usr possibly in /usr/pkg/etc and others
* A master configuration in /etc/rc.inetd

Grammar for /etc/rc.inetd would look like this:

```
comment
IncludeDir /usr/inetd/
IncludeDir /usr/pkg/inetd/

# to specify ssh over ipv4 but not ipv6
ssh.tcp4=yes
ssh.tcp6=no

# dns over defaults (ip4/ip6)
dns.udp=yes

alternately

dns=tcp4:dgram,bind=xx.xx.xx.xx
dns=tcp6:dgram,bind=dead:b33f:caf3:f00d::101

```

within /usr/inetd/ or /usr/pkg/inetd would be a file named "http" containing
something like

```
executable=/usr/pkg/bin/muhttpd
```
```

```
progname=muhttpd
arguments=--config muhttpd.conf
chroot=/srv/http/
```

If a configuration file exists in multiple directories, the *last scanned* wins.
That is, if the include order is

 * /usr/inetd/
 * /usr/pkg/inetd/
 * /usr/local/inetd/

and the files /usr/pkg/inetd/ssh and /usr/local/inetd/ssh exist, whatever is in
/usr/local/inetd/ssh overrides any settings from /usr/pkg/inetd/ssh

This means that a service on one system can override specific details (such as
environment variables or command line options) for a service whose binaries are
hosted over NFS.

Valid options within a service definition file would be, at the start

```
name        type        desc
type        enum        Socket type, one of (stream|dgram|raw|rdm|seqpacket)
wait        bool        Should inetd wait on this process to finish before
                        accepting another connection?
user        string      User to run as. Default gid is default gid of user,
                        unless specified with :group
server      string      full path to executable
progname    string      argv[0]
arguments   string      argv[1] and beyond
env         string      environment declaration
chroot      string      path to chroot() to before calling the executable
workers     int         Number of prefork threads to keep at any one given time
overflow    bool        allow the prefork pool to overflow.
```

*** NOT FINAL ***

if progname is empty or unspecified, the filename specified in server is used
in place.

Environment variables stack and overwrite each other. If /usr/pkg/inetd/ssh
has "env=foo=bar;baz=whatever" and later on .../ssh says "env=foo=wonka",
the final environment for the executable will be "foo=wonka;baz=whatever"

Other

3.3 a codebase in need of a breakup

This is actually the easiest part of the task.

At the moment, inetd.c is a huge 2ksloc file. Making each part of inetd's
component parts a separate compilation unit makes compilation easier on big
systems (modern systems can handle 2-3 compiler threads at a time) and makes
maintenance easier in future (less chance to trample something accidentially.)

Breaking up inetd.c into its constituent parts (builtins, configuration, inetd
itself, some kqueue stuff, structures, etc) improves code readability and
maintenance in the future.

3.4 integration, configuration, etc.

There are a few per-service configuration callouts in the project listing. These
are things like per-service rate limiting, blacklistd integration, etc.

I'd like to hit four major things:

* per-service ratelimiting
* blacklistd integration
* per-service logging configuration
* chroot() support

```
4. Timeline

Week 21 -> week 1 of GSOC work


WEEK (real) MONDAY           SUNDAY
Week 21     May 23      2016 May 29, 2016

Work pinning-down: A requirements document to define the grammar and specific
details of new configuration format.

Week 22     May 30      2016 June 5, 2016

Break up inetd into multiple compilation units, including
disabling of builtin services at runtime.

Implement each builtin as "inetd.<name>" and call depending on argv[0]

Week 23     June 6      2016 June 12, 2016
Week 24     June 13     2016 June 19, 2016
Week 25     June 20     2016 June 26, 2016

Implement per-service configuration:

* parser for service configuration
* parser for main service configuration
* new service structures to separate intances of a service from
  configuration.
* Starting parts for logging on a per-service basis
* Configuration parity to current design (incl. bindhosts & ipsec config)

Deliverable: inetd that reads from sparse configuration files
Deliverable: documentation for inetd configuration

At this point, inetd would be otherwise "at parity"

Week 26     June 27     2016 July 3, 2016  ** FOURTH OF JULY WEEKEND
Week 27     July 4      2016 July 10, 2016    FOURTH OF JULY WEEKEND **
Week 28     July 11     2016 July 17, 2016

Implement service availability + pool sizing

Deliverable: inetd that handles early service availability
Deliverable: documentation covering specifics and caveats of early service
              availability

** FULL IMLEMENTATION SHOULD BE COVERED IN REQUIREMENTS DOC **

Week 29     July 18     2016 July 24, 2016

Documentation week: Full documentation + conversion of a variety
of pkgsrc configurations

* openSSH
* bozohttpd
* BIND?

Also, a means to auto-generate from inetd classic lines into new-world
inetd service files. Suggesting awk (as it's common and kindof built for this)
or python (as stream parsers are easy to write in it)

Deliverable: A set of ready-to-run inetd service files for pkgsrc

Week 30     July 25     2016 July 31, 2016

* Logging configuration (log flags, etc)
* chroot() on a per-process basis

*** BONUS GOALS ***
```

```
491
492 * performance writeup
493 * blacklistd integration
494
495 Week 31     August 1    2016 August 7, 2016
496
497 ~ buffer week and writeups ~
498
499 ** DEFCON THIS WEEK / 4-7th IN LAS VEGAS, NV **
500
501 Week 32     August 8    2016 August 14, 2016
502
503 ~ buffer week and writeups ~
504
505 5. Prior work in the area
506
507 Some work has been done in the "overhaul inetd and friends" camp.
508
509 * Systemd: A total overhaul of Linux's init system. It has a complex, but
510   very comprehensive system for socket activation ( see system.socket(5) )
511   which can do wonderful things. Systemd's support includes adding pre-
512   and post-exec commands to setup/teardown a service, as well as dependencies
513   (X service must be running for Y server to run).
514 * xinetd: A *major* overhaul of inetd complete with a new configuration format.
515   this format is... Kindof JSON, Kindof not. Configuration is powerful, but
516   is poorly documented. THere hasn't been a huge amount of information on what
517   the future of xinetd looks like.
518 * launchd: OSX's precursor to systemd. There have been a few attempts at porting
519   it over to the BSD world, but that looks to be in vain [LAUNCHD-GODOT]
520
521 6. Specific thoughts on timeline, implementation
522
523 This timeline is intentionally a little... sparse. There is slop allowed for
524 each section of the work as I feel comfortable, with two weeks at the end for
525 the inevitable slippage that is software work.
526
527 I'm basing much of this timeline off conservative estimates off how comfortable
528 I am with each part. I fully expect that I'll be behind or ahead in the middle,
529 however I want to account for any other space that isn't well budgeted enough.
530
531 I'm not sure what the best direction to take is. There's evidence in both
532 directions on what's better. For example, in 2009, one test showed it really
533 matters on what's being done (e.g. static files vs. PHP) in Nginx vs. Apache
534 [APACHE-NGINX-2009] whereas another [APACHE-PREFORK-NGINX] really doesn't have
535 much to say positive for Apache.
536
537 I'd like to do, as part of the writeup, addressing performance before/after to
538 validate or invalidate this approach.
539
540 Basing the configuration on sysctl makes it easy for people to add new options
541 as well as graceful fallback when an unknown/unimplemented option is used.This
542 style is also consistent with other BSDs; notably, OpenBSD which has made many
543 system tunables a sysctl mechanism.
544
545 This also comes with a cheap bonus: adding features is easy.
546
547 7. Argument for preferred configuration change
548
549 My argument for a change in the configuration format, dropping the old format
550 fundamentally comes down to two factors:
551
552 * Documentation
553 * Maintaining
554
555 NetBSD is well documented -- it's a pride of the BSDs that the BSDs ship with a
556 book that defines how they act. Moreso, the documentation is clear and easy
557 to follow. We're not in the business of making things hard.
558
559 In NetBSD (and other BSDs) follow two basic forms of configuration: the table
560 form (see fstab) and the sysctl style. I personally feel that inetd was built
```

using the table form without a lot of thought of what the future was going to look like (especially with regard to SMP, manycore, highly parallel systems) whereas fstab was definitely built for that future oriented style.

This brings us to maintenance. Trying to keep the old format on life support is inviting two major factors: Poor support for both (see also "do thing well", a typical UNIX philosphy) and potential nasty bugs cropping up in one.

The table driven form has acrued a certain amount of technical debt. The OpenBSD project has worked to slowly find ways to make the inetd form less bug-ridden, but the problem still exists: Technical debt needs to be paid; I personally will argue that the table-driven format carries too much technical debt and possibly doubling that debt is not the direction that NetBSD should take. It is not my belief that the NetBSD project wants to be known for the type of hacks that are normally associated with the Linux kernel.

Dropping support for the table driven style of configuration and pushing on a new, well understood form fundamentally makes it easier to maintain in the near future. Including an awk script that ingests the old form and spits out he new form is not a huge problem, and could even be made to understand different kinds of inetd configurations.

A. End references

[FREEBSD-INETD] https://www.freebsd.org/doc/handbook/network-inetd.html
[C10K] http://www.kegel.com/c10k.html
[C10M] http://c10m.robertgraham.com/p/manifesto.html
[APACHE-PREFORK-NGINX] http://www.eschrade.com/page/performance-of-apache-2-4-with-the-event-mpm-compared-to-nginx/
[APACHE-NGINX-2009] https://blog.a2o.si/2009/06/24/apache-mod_php-compared-to-nginx-php-fpm/
[LAUNCHD-GODOT] http://homepage.ntlworld.com/jonathan.deboynepollard/FGA/launchd-on-bsd.html