GSoC 2023:
NetBSD Linux System Call Emulation:
"A Tale of Two Binaries"

Theodore Preduta

29 March 2023

1 Synopsis

NetBSD's Linux system call (syscall) emulation provides near seamless ability to run Linux binaries, but suffers from a predictability problem. That is, given an arbitrary Linux binary, it is extremely difficult, if not impossible, to predict whether or not that binary will run under syscall emulation, leading fairly quickly to frustration. By porting a commonly used testing facility, and using it as the benchmark rather than individual programs, we can take a more systematic approach to implementing new and fixing old system calls. Using this technique, this project aims to add support for a whole new class of Linux binaries while decreasing the frustration that comes with the predictability issue.

2 Contents

3 Project Description

The main issue with the current NetBSD Linux syscall compatibility is that its usefulness remains a constant mystery. “Will it work with X?” is not a question that can be easily answered [18]. And because of this the subsystem as a whole can be frustrating to work with. One of the driving causes behind this is that functionality has been written, in general, with specific applications in mind.

To help reduce this frustration, instead of taking the approach of find a broken application, fix it, repeat, this project will first introduce a common compatibility target to reach for, and then will take a decently sized step towards achieving it.

Enter the Linux Test Project (LTP) [7], which will act as the compatibility target. The theory here being that, since the LTP is currently used to test Linux itself [2], if its tests pass under NetBSD syscall emulation, any program that uses those syscalls should also work on NetBSD. And to cement this goal, and to allow other to use it more easily, the LTP will be packaged.

The issue with only looking at the LTP is that there are still hundreds of syscalls that have yet to be implemented, so the scope will still have to be limited somewhat arbitrarily. A target set of binaries is still used, but only to limit which syscalls are considered ‘in’ and ‘out’ of scope.

While in general it is hard to tell given a Linux binary whether it will work on NetBSD, there is an entire programming language’s worth of binaries that are guaranteed to immediately fail: Go Linux binaries. As a core part of its architecture, Go’s netpoller, which handles most IO, makes extensive use of the epoll set system calls on Linux [1], which the current compatibility code leaves unimplemented. Two Go programs I would personally like to see work[1] are Nebula [8] and Syncthing [20]. And so that will be where the line is drawn on syscalls to be implemented this summer.

4 Deliverables

First Evaluation Deliverables

- The LTP (and its possible dependencies) have been packaged.
- The getrandom, waitid, memfd_create, and the epoll family of system calls pass thir LTP system call tests.
- (Optional) Nebula functions, ie. nebula and nebula-cert Linux binaries can be used to setup an internal network, and then access some service behind that network.

[1]Yes, I know, those programs can actually be built natively under NetBSD.

Second (Final) Evaluation Deliverables

- The readahead, newfstatat, statx, close_range, ioprio_set, and inotify family of system calls pass the relevant LTP system call tests.
- (Optional) The Syncthing functions, ie. syncthing Linux binary can be used to sync a folder.

Bonus Deliverables (Time Permitting)

- Implement translation between the mount/umount2 Linux syscalls and the mount/unmount NetBSD syscalls (respectively).

5 Schedule

- Community Bonding Period: May 4 - 28
  - Set up a "better" build environment.
  - Sanity check the results from section 9 (the tables).
  - Fix any low hanging fruit (eg. wrong errno).
- Week 1: May 29 - June 4
  - Implement the getrandom and waitid syscalls.
  - Start packaging LTP.
- Week 2-3: June 5 - June 18
  - Implement for the epoll family of system calls (or adapt FreeBSD's implementation).
  - At this point the Nebula binary should run.
  - Continue packaging LTP.
- Week 4-5: June 19 - July 2
  - Implement the memfd_create syscall.
  - Finish packaging LTP[2].
- Week 6-8: July 3 - July 23
  - Implement the inotify family of system calls.
- Week 9-10: July 24 - August 6
  - Implement the readahead, newfstatat and statx syscalls.
- Week 11-12: August 7 - August 20
  - Implement the close_range and ioprio_set syscalls.

[2]While packaging LTP is not actually expected to take 4-6 weeks of work, it is also not the primary focus of this project.

6 Implementation Plans

Linux Test Project

As previously stated, the goal for the integration of the Linux Test Project (LTP) will be to package it. Any further requires a significant amount of labour that, on its own is enough work to be considered a separate project idea [21].

There are two possibilities for how this could be done, each with some benefits and drawbacks.

The first option involves creating four new pkgsrc packages, they are for the LTP [7], and its dependencies: gcc [14], make [12], and the kernel headers (provided on OpenSUSE by the kernel-source package) [13]. The creation of those last three should be very similar to that of suse15_base. That is to say, download the RPM locally and directly extract the LTP from it. With these packages it should be possible to directly build the LTP under the NetBSD emulation. The disadvantage here is the amount of packages needed, and the ending of the de facto current policy of only providing core emulation libraries in pkgsrc.

The second option is to package it directly as an RPM, on OpenSUSE, and create a new NetBSD package almost identical to that of suse15_base. The only real advantage to this approach is that it involves the creation of a single package (and an RPM). Otherwise, the downside of this approach is that it involves integrating both the Linux and NetBSD pkgsrc infrastructure, which at this point only supports RHEL 7 [5, 17]. And while building on RHEL 7 is possible, it will likely result in conflicts with the OpenSUSE version of glibc, which NetBSD packages. It is for these reasons that the first approach, while more complex, is preferred by myself.

System Calls

By limiting our scope from all Linux syscalls to those that are required for a usual use case of Nebula and Syncthing, we end up with a tractable list of syscalls. Those syscalls were then checked against the system call list in linux_syscalls.c [10]. Those that are unimplemented are outlined in table 1, and those that are were tested against the LTP, with results being in table 3.

While initially table 3 seems like it presents an impossibly large amount of syscalls to debug in the standard 12-week period, a more manual review of the tests show that the vast majority of the failing tests fall into one of two categories. The first being wrong errno, and the second being unable to run because of a dependency on some other (usually mount) syscall needed to setup the test.

Table 1, however, summarizes that majority of the work that must be done.

The simplest system calls of the bunch are close_range, getrandom, newfstatat, readahead, statx, and waitid, which have very direct NetBSD syscall counterparts. The implementation of these syscalls would be nearly identical to that of open [9], that is, translate the flags directly (if there are any), and call the equivalent NetBSD system call.

Next, the epoll and inotify series of syscalls can each be implemented with kevent and kqueue. Roughly, the epoll system calls need to be translated into kevent structs that are using the EVFILT_READ, EVFILT_WRITE, and EVFILT_EMPTY filters. It should be noted that FreeBSD uses this strategy in their Linuxulator [3], some of which can be adapted to NetBSD. Regarding the inotify syscalls, they can be roughly translated into kevent structs making use of the EVFILT_VNODE filter which has its fflags mappings outlined in table 2. The one item that stands out in table 2, is that of IN_CREATE can be approximated by watching the directory of the provided path, which is why it maps to NOTE_WRITE. Unfortunately, unlike epoll, FreeBSD's Linuxulator does not currently implement the inotify syscalls. But there is a userspace inotify library that makes use of kevent and kqueue [6], but a cursory read through suggests that this code would not be easily adaptable.

For memfd_create, the natural choice would be shm_open, but, unlike FreeBSD's implementation, NetBSD's implementation does not have a flag to allow for anonymous shared objects (SHM_ANON on FreeBSD) [4]. To avoid name conflicts, rather than just use it directly, it is better to mount another tmpfs, perhaps at /dev/memfd, and call open on a file located there (which is essentially what NetBSD's shm_open does anyways [11]). This is done so that this syscall would have its own 'namespace', controlled by the emulation code, thereby emulating anonymous files.

Finally ioprio_set, which modifies the scheduling of IO, NetBSD does not seem have any such functionality. However, if the goal is *just* to get the software to function, the nature of this system call is that it *could* be implemented as a no-op. So this one is left to the very end in the hopes by then I have enough knowledge to come up with a better idea.

7 Biography

I am a second year student studying computer science at the University of Toronto who is particularly interested in systems-level programming. In terms of technological experience, I am quite comfortable with C, having used it regularly for coursework and competitive programming[3] over the last 5ish years, the one caveat with this is that all the C code I've written is in userland, and I expect the kernel to be somewhat different. For version control I use Git/Fossil/SVN on a daily basis for schoolwork, and personal projects [16].

Part of this proposal involves creating novel pkgsrc packages, and, while I do not have experience with the pkgsrc system in particular, I have had to package some piece of software for every Linux distribution I've had to use. This includes Debian, RHEL/Fedora, and Gentoo [15].

Finally, as part of the preparation of this document, I've spent some time getting to know NetBSD. Most of it is what would be expected for learning a

[3]So I don't have permission to release any of it, unfortunately.

new system: installing it in a virtual machine, installing packages, building a kernel and reading man pages. Most notably however, I spent a good chunk of time getting the LTP somewhat functional on NetBSD [19].

8 Personal Information

Name	Theodore Preduta
Preferred Name	Theo / Theodore
Email Address	theo@pta.gg
Website	www.pta.gg
Personal Projects	vcs.pta.gg / github.com/6167656e74323431
Timezone	EDT / UTC-4h

9 Tables

Linux Syscall	**NetBSD Syscall(s)**
close_range	close
epoll_create1 epoll_ctl epoll_pwait epoll_wait	kevent kqueue
getrandom	getrandom
inotify_add_watch inotify_init1 inotify_rm_watch	kevent kqueue
ioprio_set	
memfd_create	open
newfstatat	fstatat
readahead	posix_fadvise
statx	stat
waitid	waitid

Table 1: Unimplemented Linux syscalls and possible NetBSD translations.

inotify	**kqueue EVFILT_VNODE**
IN_ACCESS	*watch for everything*
IN_ATTRIB	NOTE_ATTRIB
IN_CLOSE_WRITE	NOTE_CLOSE_WRITE
IN_CLOSE_NOWRITE	NOTE_CLOSE
IN_CREATE	NOTE_WRITE
IN_DELETE	NOTE_DELETE
IN_DELETE_SELF	NOTE_DELETE
IN_MODIFY	NOTE_WRITE
IN_MOVE_SELF	NOTE_RENAME
IN_MOVED_FROM	NOTE_RENAME
IN_MOVED_TO	NOTE_RENAME
IN_OPEN	NOTE_OPEN

Table 2: Mapping between Linux's inotify's watch mask and NetBSD' kqueue's EVFILT_VNODE fflags.

Linux Syscall	Failing LTP Test(s)
access	access04
bind	bind01 bind04 bind05 bind06
brk	brk01
clock_nanosleep	clock_nanosleep01 clock_nanosleep02 clock_nanosleep03
clone	clone08 clone09
connect	connect01 connect02
execve	execve06
fallocate	fallocate04 fallocate05 fallocate06
fcntl	fcntl12 fcntl12_64 fcntl13 fcntl13_64 fcntl18 fcntl18_64 fcntl23 fcntl23_64 fcntl30 fcntl30_64 fcntl31 fcntl31_64 fcntl33 fcntl33_64 fcntl35 fcntl35_64 fcntl37 fcntl37_64 fcntl38 fcntl38_64 fcntl39 fcntl39_64
fsync	fsync01 fsync03 fsync04
futex	futex_cmp_requeue02 futex_wait03 futex_wait05 futex_wait_bitset01
getpeername	getpeername01
getpgid	getpgid02
getpid	getpid01
getppid	getppid01
getsockopt	getsockopt01 getsockopt02
getresgid	getresgid02 getresuid02
getrlimit	getrlimit01 getrlimit03
getuid	getuid03
ioctl	ioctl04 ioctl05 ioctl06 ioctl07
kill	kill03 kill11 kill13
link	link01
madvise	madvise01 madvise02 madvise03 madvise06
mincore	mincore01 mincore02 mincore04
mkdir	mkdir02 mkdir03 mkdir09
mkdirat	mkdirat02
mmap	mmap12 mmap13 mmap14 mmap15 mmap18
mprotect	mprotect01
munmap	munmap03
nanosleep	nanosleep01 nanosleep04
openat	openat02 openat04
pipe2	pipe2_01 pipe2_04
poll	poll02
readlink	readlink03
readlinkat	readlinkat02
renameat	renameat01
rmdir	rmdir02 rmdir03

rt_sigprocmask	rt_sigprocmask01
sendmsg	sendmsg01 sendmsg03
sendto	sendto01 sendto03
setpgid	setpgid02
setrlimit	setrlimit03
setsockopt	setsockopt01 setsockopt02 setsockopt04 setsockopt05 setsockopt06 setsockopt07 setsockopt08 setsockopt09
sigaltstack	sigaltstack02
socket	socket01
socketpair	socketpair01
sockioctl	sockioctl01
statfs	statfs01 statfs01_64
statvfs	statvfs01
symlink	symlink01
symlinkat	symlinkat01
sysinfo	sysinfo03
tgkill	tgkill02 tgkill03
tkill	tkill02
unlinkat	unlinkat01

Table 3: Linux syscalls with failing LTP tests.

10 References

[1] *The Go netpoller*. 2013. URL: https://morsmachine.dk/netpoller.

[2] Chun Cui. *How we use Linux Test Project to test and improve Linux*. 2021. URL: https://www.redhat.com/sysadmin/linux-test-project.

[3] *FreeBSD - linux_event.c*. URL: https://svnweb.freebsd.org/base/head/sys/compat/linux/linux_event.c?revision=365080&view=markup.

[4] *FreeBSD - shm_open(2)*. URL: https://man.freebsd.org/cgi/man.cgi?query=shm_open&apropos=0&sektion=0&manpath=FreeBSD+13.1-RELEASE+and+Ports&arch=default&format=html.

[5] *Install on Linux*. URL: https://pkgsrc.smartos.org/install-on-linux/.

[6] *libinotify-kqueue*. URL: https://github.com/libinotify-kqueue/libinotify-kqueue.

[7] *LTP - Linux Test Project*. URL: https://linux-test-project.github.io/.

[8] *Nebula*. URL: https://github.com/slackhq/nebula.

[9] *NetBSD - linux_file.c*. URL: http://cvsweb.netbsd.org/bsdweb.cgi/src/sys/compat/linux/common/linux_file.c?rev=1.122&content-type=text/x-cvsweb-markup&only_with_tag=MAIN.

[10] *NetBSD - linux_syscalls.c*. URL: http://cvsweb.netbsd.org/bsdweb.cgi/src/sys/compat/linux/arch/amd64/linux_syscalls.c?rev=1.76&content-type=text/x-cvsweb-markup&only_with_tag=MAIN.

[11] *NetBSD - shm.c*. URL: http://cvsweb.netbsd.org/bsdweb.cgi/src/lib/librt/shm.c?rev=1.1.6.1&content-type=text/x-cvsweb-markup.

[12] *openSUSE Software - GNU make*. URL: https://software.opensuse.org/package/make.

[13] *openSUSE Software - The Linux Kernel Sources*. URL: https://software.opensuse.org/package/kernel-source.

[14] *openSUSE Software - The system GNU C Compiler*. URL: https://software.opensuse.org/package/make.

[15] *Personal Gentoo repository*. URL: https://github.com/6167656e74323431/turbo-octo-spoon.

[16] *Personal project repository*. URL: https://vcs.pta.gg/.

[17] *pkgsrc - install binary packages*. URL: https://www.pkgsrc.org/#index2h2.

[18] *Re: [GSoC] Emulating missing Linux syscalls project questions*. URL: https://mail-index.netbsd.org/tech-kern/2023/03/13/msg028779.html.

[19] *Re: [GSoC] Emulating missing Linux syscalls project questions*. URL: https://mail-index.netbsd.org/tech-kern/2023/03/19/msg028787.html.

[20] *Syncthing*. URL: https://syncthing.net/.

[21] *Test Linux emulation (350h)*. URL: `https://wiki.netbsd.org/projects/project/linuxtest/`.