NAME

blink — headless blinkenlights x86-64-linux virtual machine

SYNOPSIS

blink [**-hvjemZs**] [**-L** *logfile*] [**-C** *chroot*] *program* [*argv1...*]
blink [**-hvjemZs**] [**-L** *logfile*] [**-C** *chroot*] **-0** *program* [*argv0...*]

DESCRIPTION

blink is a JITing virtual machine that uses x86_64 as its byte code language and implements the Linux Kernel ABI. **blink** may be used to run prebuilt binaries intended for x86_64 Linux on other POSIX platforms.

ARGUMENTS

The *program* argument is a `PATH` searched command, which may be:

- An x86_64-linux ELF executable (either static or dynamic)
- An Actually Portable Executable (either MZqFpD or jartsr magic)
- A flat executable (if *program* ends with .bin, .img, or .raw)
- A shell script whose #!interpreter meets the above criteria

The **-0** flag allows `argv[0]` to be specified on the command line. Under normal circumstances,

```
blink cmd arg1
```

is equivalent to

```
execve("cmd", {"cmd", "arg1"})
```

since that's how most programs are launched. However if you need the full power of execve() process spawning, you can say

```
blink -0 cmd arg0 arg1
```

which is equivalent to

```
execve("cmd", {"arg0", "arg1"})
```

OPTIONS

The following options are available:

-h Prints condensed help.

-v Shows **blink** version and build configuration details.

-e Log to standard error (fd 2) in addition to the log file. If logging to *only* standard error is desired, then `-eL/dev/null` may be passed as flags.

-s Enables system call logging (repeatable).

This will emit to the log file the names of system calls each time a SYSCALL instruction in executed, along with its arguments and result. System calls are logged once they've completed, so that the result can be shown.

System calls are also sometimes logged upon entry too, in which case `syscall(arg) -> ...` will be logged to show that the system call has not yet completed. Whether or not this happens depends on how many times the **-s** flag is supplied.

- Passing **-s** (once) will only log the entries of system calls that are defined as having read + write parameters (e.g. poll, select)

- Passing **-ss** (twice) will also log the entries of cancellation points that are likely to block (e.g. read, accept, wait, pause).
- Passing **-sss** (thrice) will log the entries of system calls that POSIX defines as cancellation points but are unlikely to block (e.g. write, close, open) which we try to avoid doing in order to reduce log noise.
- Passing **-ssss** (4x) will log the entry of every single system call, even harmless ones that have no business being emitted to the log twice (e.g. sigaction). This should only be useful for the Blink dev team when the rare need arises to troubleshoot a system call that's crashing.

System call logging isn't available in `MODE=rel` and `MODE=tiny` builds, in which case this flag is ignored.

-m Enables full memory virtualization. Under normal circumstances, **blink** aims to share the same address space as the guest program. Depending on the program's memory requirements (i.e. if it uses MAP_FIXED) and the constraints imposed by the host platform, this so-called linear memory optimization may lead to an mmap() crisis that causes Blink to not work properly. Using this option will cause **blink** to virtualize the x86_64 address space precicely and reliably, however the use of this flag carries the tradeoff of causing **blink** to go at least 2x slower.

-j Disables Just-In-Time (JIT) compilation. Using this option will cause **blink** to go ~10x or possibly even ~100x slower.

-L *path*
Specifies the log path. The default log path is *blink.log* in the current directory at startup. This log file won't be created until something is actually logged. If logging to a file isn't desired, then -L /dev/null may be used. See also the **-e** flag for logging to standard error.

-C *path*
Launch *program* in a chroot'd environment. This flag is both equivalent to and overrides the `BLINK_OVERLAYS` environment variable.

-Z Prints internal statistics to standard error on exit. Each line will display a monitoring metric. Most metrics will either be integer counters or floating point running averages. Most but not all integer counters are monotonic. In the interest of not negatively impacting Blink's performance, statistics are computed on a best effort basis which currently isn't guaranteed to be atomic in a multi-threaded environment. Statistics aren't available in MODE=rel and MODE=tiny builds, in which case this flag is ignored.

ENVIRONMENT

The following environment variables are recognized:

`BLINK_LOG_FILENAME`
may be specified to supply a log path to be used in cases where the **-L** *path* flag isn't specified. This value should be an absolute path. If logging to standard error is desired, use the **-e** flag.

`BLINK_OVERLAYS`
specifies one or more directories to use as the root filesystem. Similar to `PATH` this is a colon delimited list of pathnames. If relative paths are specified, they'll be resolved to an absolute path at startup time. Overlays only apply to IO system calls that specify an absolute path. The empty string overlay means use the normal / root filesystem. The default value is `:o` which means if the absolute path `/$f` is opened, then first check if `/$f` exists, and if it doesn't, then check if `o/$f` exists, in which case open that instead. Blink uses this convention to open shared object tests. It favors the system version if it exists, but also downloads `ld-musl-x86_64.so.1` to `o/lib/ld-musl-x86_64.so.1` so the dynamic linker can transparently find it on platforms like Apple, that don't let users put files in the root folder. On the other hand, it's possible to say `BLINK_OVERLAYS=o:` so that `o/...` takes precedence over `/...` (noting again that empty

string means root). If a single overlay is specified that isn't empty string, then it'll effectively act as a restricted chroot environment.

QUIRKS

Here's the current list of Blink's known quirks and tradeoffs.

Flags

Flag dependencies may not carry across function call boundaries under long mode. This is because when Blink's JIT is speculating whether or not it's necessary for an arithmetic instruction to compute flags, it considers `RET` and `CALL` terminal ops that break the chain. As such 64-bit code shouldn't do things we did in the DOS days, such as using carry flag as a return value to indicate error. This should work fine when `STC` is used to set the carry flag, but if the code computes it cleverly using instructions like `SUB` then EFLAGS might not change.

Faults

Blink may not report the precise program counter where a fault occurred in `ucontext_t::uc_mcontext::rip` when signalling a segmentation fault. This is currently only possible when `PUSH` or `POP` access bad memory. That's because Blink's JIT tries to avoid updating `Machine::ip` on ops it considers "pure" such as those that only access registers, which for reasons of performance is defined to include pushing and popping.

Threads

Blink doesn't have a working implementation of `set_robust_list()` yet, which means robust mutexes might not get unlocked if a process crashes.

Coherency

POSIX.1 provides almost no guarantees of coherency, synchronization, and durability when it comes to `MAP_SHARED` mappings and recommends that msync() be explicitly used to synchronize memory with file contents. The Linux Kernel implements shared memory so well, that this is rarely necessary. However some platforms like OpenBSD lack write coherency. This means if you change a shared writable memory map and then call pread() on the associated file region, you might get stale data. Blink isn't able to polyfill incoherent platforms to be as coherent as Linux, therefore apps that run in Blink should assume the POSIX rules apply.

Signal Handling

Blink uses `SIGSYS` to deliver signals internally. This signal is precious to Blink. It's currently not possible for guest applications to capture it from external processes.

Memory Protection

Blink offers guest programs a 48-bit virtual address space with a 4096-byte page size. When programs are run on (1) host systems that have a larger page (e.g. Apple M1, Cygwin), and (2) the linear memory optimization is enabled (i.e. you're *not* using `blink -m`) then Blink may need to relax memory protections in cases where the memory intervals defined by the guest aren't aligned to the host system page size. Is is recommended, when calling functions like mmap() and mprotect(), that both `addr` and `addr + size` be aliged to the true page size, which Blink reports to the guest in `getauxval(AT_PAGESZ)`. This value should be obtainable via the portable API `sysconf(_SC_PAGESIZE)` assuming the C library implements it correctly. Please note that when Blink is running in its fully virtualized mode (i.e. `blink -m`) this concern does not apply. That's because Blink will allocate a full system page for every 4096 byte page that gets mapped from a file.

EXIT STATUS

The **blink** command passes along the exit code of the *program* which by convention is 0 on success or >0 on failure. In the event that **blink** fails to launch *program* the status 127 shall be returned.

SEE ALSO

`blinkenlights`(1)

STANDARDS

The **blink** command implements a superset of the IEEE Std 1003.1-2008 ("POSIX.1") specification, intended to emulate the behaviors of the Linux Kernel.

AUTHORS

Justine Alexandra Roberts Tunney <jtunney@gmail.com>