Figure 1: libuv

Overview

libuv is a multi-platform support library with a focus on asynchronous I/O. It was primarily developed for use by Node.js, but it's also used by Luvit, Julia, uvloop, and others.

Feature highlights

- Full-featured event loop backed by epoll, kqueue, IOCP, event ports.
- Asynchronous TCP and UDP sockets
- Asynchronous DNS resolution
- Asynchronous file and file system operations
- File system events
- ANSI escape code controlled TTY
- IPC with socket sharing, using Unix domain sockets or named pipes (Windows)
- Child processes
- Thread pool
- Signal handling
- High resolution clock
- Threading and synchronization primitives

Versioning

Starting with version 1.0.0 libuv follows the semantic versioning scheme. The API change and backwards compatibility rules are those indicated by SemVer.

libuv will keep a stable ABI across major releases.

The ABI/API changes can be tracked here.

Licensing

libuv is licensed under the MIT license. Check the LICENSE file. The documentation is licensed under the CC BY 4.0 license. Check the LICENSE-docs file.

Community

- Support
- Mailing list

Documentation

Official documentation

Located in the docs/ subdirectory. It uses the Sphinx framework, which makes it possible to build the documentation in multiple formats.

Show different supported building options:

```
$ make help
```

Build documentation as HTML:

```
$ make html
```

Build documentation as HTML and live reload it when it changes (this requires sphinx-autobuild to be installed and is only supported on Unix):

```
$ make livehtml
```

Build documentation as man pages:

```
$ make man
```

Build documentation as ePub:

```
$ make epub
```

NOTE: Windows users need to use make.bat instead of plain 'make'.

Documentation can be browsed online here.

The tests and benchmarks also serve as API specification and usage examples.

Other resources

- LXJS 2012 talk — High-level introductory talk about libuv.
- libuv-dox — Documenting types and methods of libuv, mostly by reading uv.h.

- learnuv — Learn uv for fun and profit, a self guided workshop to libuv.

These resources are not handled by libuv maintainers and might be out of date. Please verify it before opening new issues.

Downloading

libuv can be downloaded either from the GitHub repository or from the downloads site.

Before verifying the git tags or signature files, importing the relevant keys is necessary. Key IDs are listed in the MAINTAINERS file, but are also available as git blob objects for easier use.

Importing a key the usual way:

```
$ gpg --keyserver pool.sks-keyservers.net --recv-keys AE9BC059
```

Importing a key from a git blob object:

```
$ git show pubkey-saghul | gpg --import
```

Verifying releases

Git tags are signed with the developer's key, they can be verified as follows:

```
$ git verify-tag v1.6.1
```

Starting with libuv 1.7.0, the tarballs stored in the downloads site are signed and an accompanying signature file sit alongside each. Once both the release tarball and the signature file are downloaded, the file can be verified as follows:

```
$ gpg --verify libuv-1.7.0.tar.gz.sign
```

Build Instructions

For UNIX-like platforms, including macOS, there are two build methods: autotools or CMake.

For Windows, CMake is the only supported build method and has the following prerequisites:

- One of:
  - Visual C++ Build Tools
  - Visual Studio 2015 Update 3, all editions including the Community edition (remember to select "Common Tools for Visual C++ 2015" feature during installation).
  - Visual Studio 2017, any edition (including the Build Tools SKU). **Required Components:** "MSbuild", "VC++ 2017 v141 toolset" and one of the Windows SDKs (10 or 8.1).
- Basic Unix tools required for some tests, Git for Windows includes Git Bash and tools which can be included in the global `PATH`.

To build with autotools:

```
$ sh autogen.sh
$ ./configure
$ make
$ make check
$ make install
```

To build with CMake:

```
$ mkdir -p build

$ (cd build && cmake .. -DBUILD_TESTING=ON) # generate project with tests
$ cmake --build build                      # add `-j <n>` with cmake >= 3.12

# Run tests:
$ (cd build && ctest -C Debug --output-on-failure)

# Or manually run tests:
$ build/uv_run_tests                       # shared library build
$ build/uv_run_tests_a                     # static library build
```

To cross-compile with CMake (unsupported but generally works):

```
$ cmake ../..                  \
  -DCMAKE_SYSTEM_NAME=Windows \
  -DCMAKE_SYSTEM_VERSION=6.1  \
  -DCMAKE_C_COMPILER=i686-w64-mingw32-gcc
```

Install with Homebrew

```
$ brew install --HEAD libuv
```

Note to OS X users:

Make sure that you specify the architecture you wish to build for in the "ARCHS" flag. You can specify more than one by delimiting with a space (e.g. "x86_64 i386").

Running tests

Some tests are timing sensitive. Relaxing test timeouts may be necessary on slow or overloaded machines:

```
$ env UV_TEST_TIMEOUT_MULTIPLIER=2 build/uv_run_tests # 10s instead of 5s
```

Run one test The list of all tests is in `test/test-list.h`.

This invocation will cause the test driver to fork and execute `TEST_NAME` in a child process:

```
$ build/uv_run_tests_a TEST_NAME
```

This invocation will cause the test driver to execute the test in the same process:

```
$ build/uv_run_tests_a TEST_NAME TEST_NAME
```

Debugging tools When running the test from within the test driver process (`build/uv_run_tests_a TEST_NAME TEST_NAME`), tools like gdb and valgrind work normally.

When running the test from a child of the test driver process (`build/uv_run_tests_a TEST_NAME`), use these tools in a fork-aware manner.

Fork-aware gdb Use the follow-fork-mode setting:

```
$ gdb --args build/uv_run_tests_a TEST_NAME

(gdb) set follow-fork-mode child
...
```

Fork-aware valgrind Use the `--trace-children=yes` parameter:

```
$ valgrind --trace-children=yes -v --tool=memcheck --leak-check=full --track-origins=yes --l
```

Running benchmarks

See the section on running tests. The benchmark driver is `./uv_run_benchmarks_a` and the benchmarks are listed in `test/benchmark-list.h`.

Supported Platforms

Check the SUPPORTED_PLATFORMS file.

`-fno-strict-aliasing`

It is recommended to turn on the `-fno-strict-aliasing` compiler flag in projects that use libuv. The use of ad hoc "inheritance" in the libuv API may not be safe in the presence of compiler optimizations that depend on strict aliasing.

MSVC does not have an equivalent flag but it also does not appear to need it at the time of writing (December 2019.)

AIX Notes

AIX compilation using IBM XL C/C++ requires version 12.1 or greater.

AIX support for filesystem events requires the non-default IBM `bos.ahafs` package to be installed. This package provides the AIX Event Infrastructure

that is detected by `autoconf`. IBM documentation describes the package in more detail.

z/OS Notes

z/OS compilation requires ZOSLIB to be installed. When building with CMake, use the flag `-DZOSLIB_DIR` to specify the path to ZOSLIB:

```
$ (cd build && cmake .. -DBUILD_TESTING=ON -DZOSLIB_DIR=/path/to/zoslib)
$ cmake --build build
```

z/OS creates System V semaphores and message queues. These persist on the system after the process terminates unless the event loop is closed.

Use the `ipcrm` command to manually clear up System V resources.

Patches

See the guidelines for contributing.