Libuv usages & internals

牟卫洋

Contents

- Libuv basics
- Libuv design overview
- Libuv usages
- Libuv internals
- Questions
- References

Libuv basics

- **Blocking VS non-blocking**
- **Synchronous VS asynchronous**
- **BIO, NIO and AIO**
- **Event & callback**
- **epoll**

Libuv basics

- **What is libuv?**
  multi-platform event-driven asynchronous I/O library

- **Multi-platform**
  Unix(Linux/BSDs/AIX/Solaris) and Windows
- **Event-driven**
  register callback for each event, callback will be called after event happens
- **Asynchronous I/O library**
  others include libevent and libev

Libuv basics

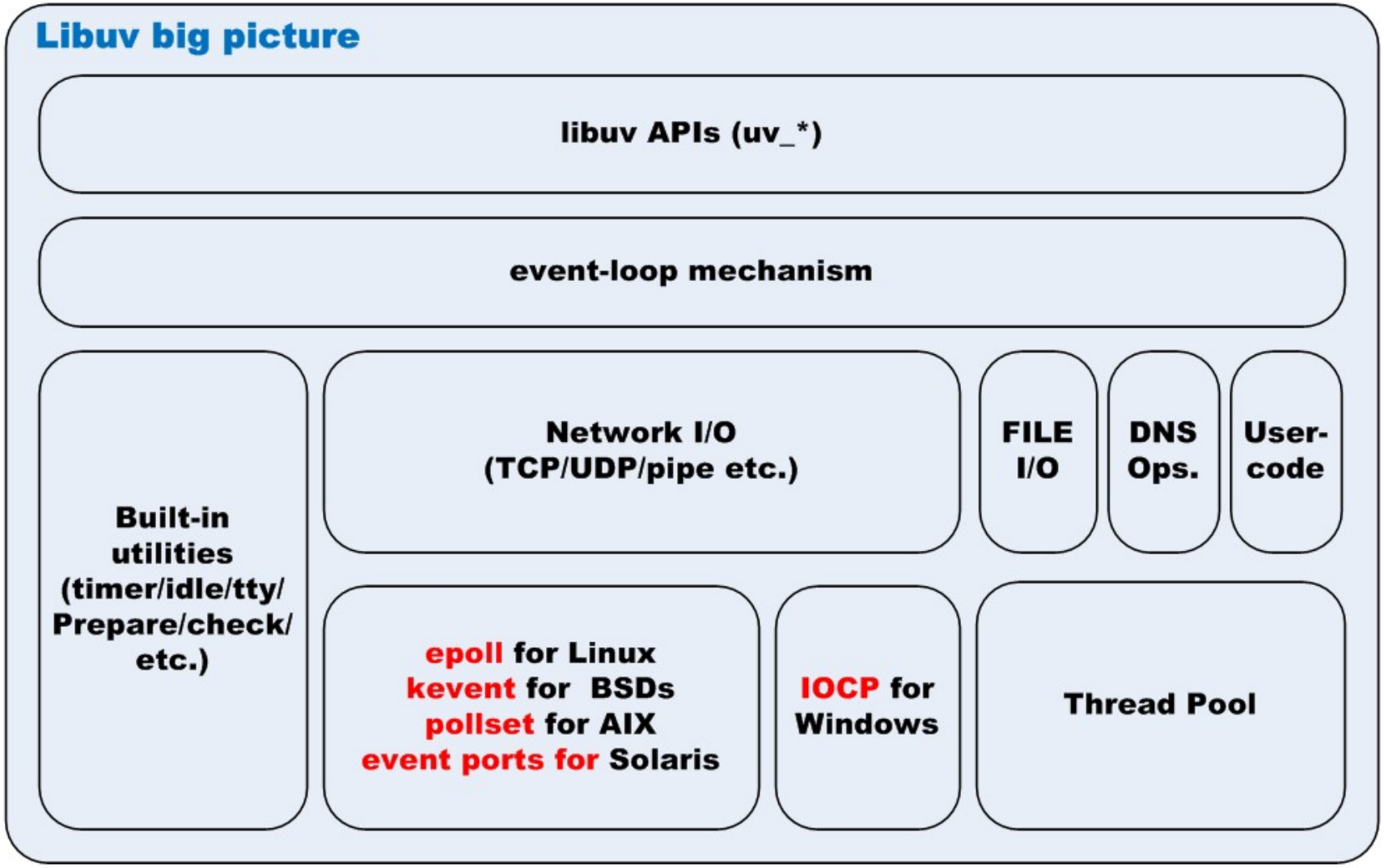
- **Handle**

  long-lived objects capable of performing operations while active (get its callback called), such as TCP/UDP, tty and timers

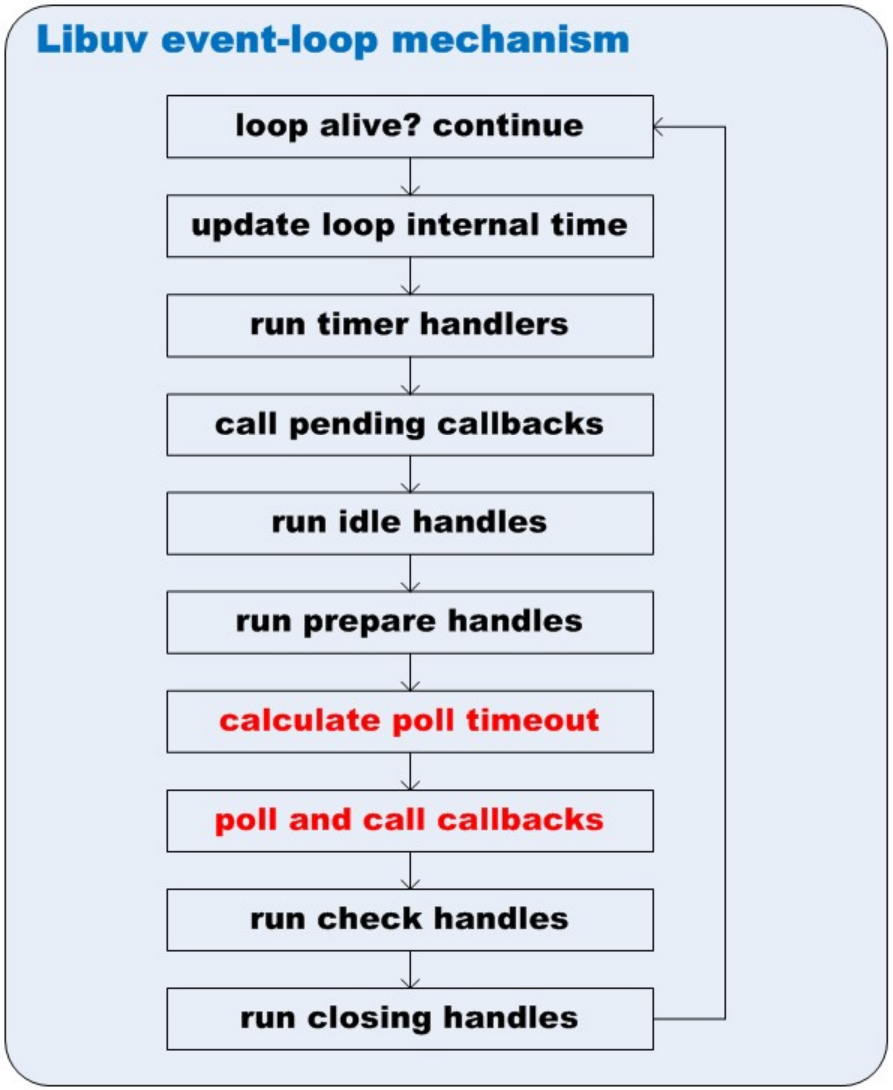
- **Request**

  short-lived objects, performed over handles (write/connect) or standalone (getaddrinfo)

Libuv design overview



Libuv design overview



Libuv usages

- **Download source code**

```
$ git clone https://github.com/libuv/libuv; cd libuv
```

- **Automation or benchmark test**

```
$ ./gyp_uv.py -f ninja
$ ninja -C out/Debug
$ cd out/Debug
$ ./run-tests
$ ./run-benchmarks
```

- **Install headers and shared library**

```
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
$ sudo ldconfig
```

- **Simple API test**

```
$ gcc version.c -luv
$ ./a.out
$ 1.8.1-dev
```

```c
#include <stdio.h>
#include <uv.h>

int main()
{
    printf("%s\n", uv_version_string());
    return 0;
}
```

Libuv usages

- **A simple example – uvcat**

```c
int main(int argc, char **argv) {
    uv_fs_open(uv_default_loop(), &open_req, argv[1], O_RDONLY, 0, on_open);
    uv_run(uv_default_loop(), UV_RUN_DEFAULT);

    uv_fs_req_cleanup(&open_req);
    uv_fs_req_cleanup(&read_req);
    uv_fs_req_cleanup(&write_req);
    return 0;
}

void on_open(uv_fs_t *req) {
    // The request passed to the callback is the same as the one the call setup
    // function was passed.
    assert(req == &open_req);
    if (req->result >= 0) {
        iov = uv_buf_init(buffer, sizeof(buffer));
        uv_fs_read(uv_default_loop(), &read_req, req->result,
                   &iov, 1, -1, on_read);
    }
    else {
        fprintf(stderr, "error opening file: %s\n", uv_strerror((int)req->result));
    }
}
```

Libuv usages

- **A simple example – uvcat (cont.)**

```
void on_read(uv_fs_t *req) {
    if (req->result < 0) {
        fprintf(stderr, "Read error: %s\n", uv_strerror(req->result));
    }
    else if (req->result == 0) {
        uv_fs_t close_req;
        // synchronous
        uv_fs_close(uv_default_loop(), &close_req, open_req.result, NULL);
    }
    else if (req->result > 0) {
        iov.len = req->result;
        uv_fs_write(uv_default_loop(), &write_req, 1, &iov, 1, -1, on_write);
    }
}

void on_write(uv_fs_t *req) {
    if (req->result < 0) {
        fprintf(stderr, "Write error: %s\n", uv_strerror((int)req->result));
    }
    else {
        uv_fs_read(uv_default_loop(), &read_req, open_req.result, &iov, 1, -1, on_read);
    }
}
```

Libuv usages

Normal BIO VS **libuv AIO**

```c
// BIO mode
void main()
{
    fd = open(path, ...);
    while (read(fd, ...) != 0) {
        write(STDOUT_FILENO, ...);
    }
    close(fd);
}
```

```c
// libuv AIO mode
void main()
{
    fd = uv_fs_open(&open_req, path, ..., open_cb);
    uv_run(...);
}

void open_cb(uv_fs_t *req) {
    uv_fs_read(req->result, ..., read_cb);
}

void read_cb(uv_fs_t *req) {
    if (req->result > 0) {
        uv_fs_write(STDOUT_FILENO, ..., write_cb);
    } else if (req->result == 0) {
        uv_fs_close(open_req.result, NULL);
    }
}

void write_cb(uv_fs_t *req) {
    uv_fs_read(open_req.result, ..., read_cb);
}
```

Libuv internals

- **How cross-platform?**

- **For compilation, use cross-platform GYP build tool**
- **For asynchronous I/Os, use Unix libev and Windows IOCP**
  - Epoll for Linux (`uv__io_poll -> epoll_wait sycall`)
  - Kqueue for Mac OS X and other BSDs (`uv__io_poll -> kevent`)
  - Pollset for IBM-AIX (`uv__io_poll -> pollset_poll`)
  - Event ports for Sun-Solaris (`uv__io_poll -> port_getn`)
  - IOCP for Windows (`uv_io_poll -> GetQueueCompletionStatus`)
- **For threadpoll, use high-level abstracted APIs + MT libraries**
  - Pthread for POSIX Unix (`uv_thread_create -> pthread_create`)
  - Built-in MT APIs for Windows (`uv_thread_create ->_beginthreadex`)

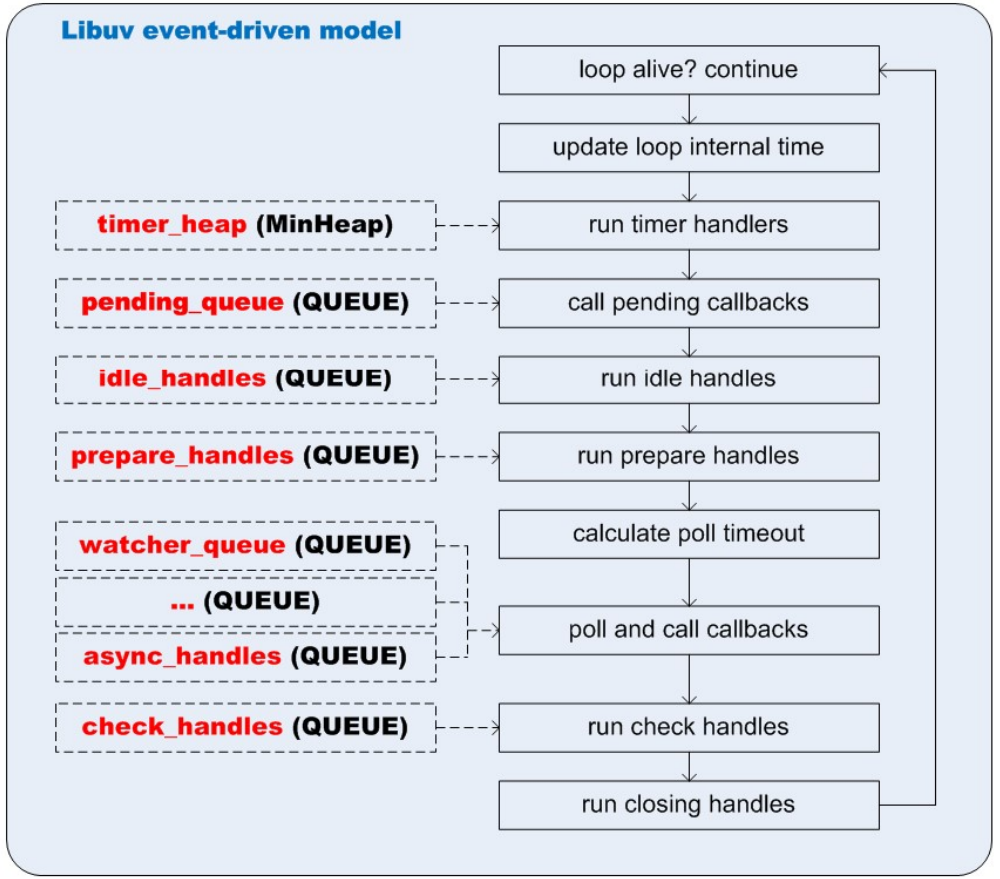
Libuv internals

- **Event-driven model**

- **Components:**
  - **Registered callbacks**
  - **Event queues**
  - **Poll mechanism**

- **Steps**

1. registered callback
2. add to queue when event happens
3. poll queues in loop
4. call registered callbacks

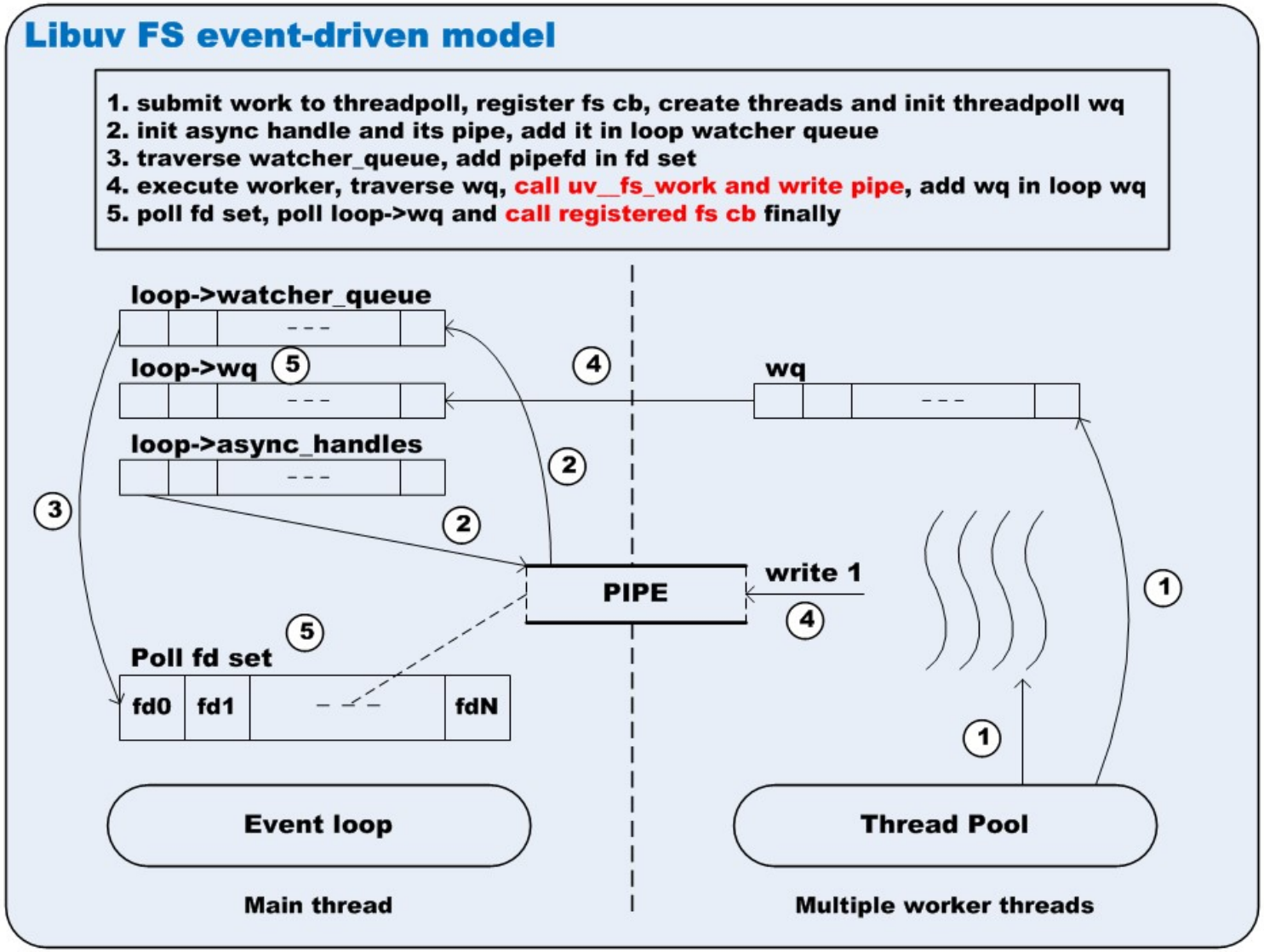


Libuv internals

- **FS event-driven model**

- **Components**
  - **Blocking FS Operations: worker threads in threadpoll**
  - **Event loop: main thread**
  - **Inner-Thread Communication : pipe-based async handle**

Libuv internals



Libuv internals

- **FS event-driven model (cont.)**

- **Steps and source code trace**

1. **submit work to threadpoll, register fs cb, create threads and init threadpoll wq**

   `uv_fs_XXX(CB) -> POST -> uv__work_submit(uv__fs_work, uv__fs_done)`

2. **init async handle and its pipe, add it in loop watcher queue**

   `uv_loop_init -> uv_async_init -> uv__async_start`

3. **traverse watcher_queue, add pipefd in fd**

   `setuv_run -> uv__io_poll -> uv__epoll_ctl`

4. **execute worker, traverse wq, call uv__fs_work, write pipe, add wq in loop wq**

   `worker -> uv__fs_work -> uv__async_send`

5. **poll fd set, call registered fs cb finally**

   `Uv__io_poll -> uv__async_io -> uv__async_event -> uv__work_done -> uv__fs_done -> fs cb`

Questions

- **epoll is NIO or AIO ?**
- **What is the relationship between epoll and libuv ?**
- **What is the difference between libev and libuv ?**

References

- **Libuv source code**

  https://github.com/libuv/libuv

- **uvbook (intro ebook, examples and ex-examples)**

  http://nikhilm.github.io/uvbook/An%20Introduction%20to%20libuv.pdf
  https://github.com/nikhilm/uvbook
  https://github.com/thlorenz/libuv-dox/

- **Libuv official API documentation**

  http://docs.libuv.org/en/v1.x/

Thank you !