Introduction

Introduction

Céu-libuv supports the development of libuv applications in the programming language Céu.

Mode of Operation

Mode of Operation

`TODO`

File System

File System

`TODO`

Input Events

UV_FS

```
input _uv_fs_t&& UV_FS;
```

- Occurrence:
  - Whenever a filesystem operation completes.
- Payload:
  - `_uv_fs_t&&`: pointer to the operation request

libuv reference: http://docs.libuv.org/en/v1.x/fs.html

Data Abstractions

UV_FS_File

```
data UV_FS_File with
    event void ok;
    var   int  fd;
end
```

Code Abstractions

UV_FS_Open

Opens a file.

```
code/await UV_FS_Open (var _char&& path, var int flags, var int mode)
                        -> (var& UV_FS_File file)
                            -> int
```

- Parameters
  - `path`: path to the file
  - `flags`: access mode flags
  - `mode`: file permission mode
- Initialization
  - `file`: created file handle
- Return
  - `int`: open error
    - returns only case of error (always `<0`)

The file is only ready for use after `UV_FS_Open` triggers `file.ok`.

Céu-libuv references: `ceu_uv_fs_open`, `UV_FS`.

libuv references: `uv_fs_close`, `uv_fs_req_cleanup`.

Note: all allocated libuv resources are automatically released on termination.

Example

Opens `file.txt` and prints *open ok* after the file is ready for use. In case of failure, prints *open error* along with the error code:

```
##include "uv/fs.ceu"

var& UV_FS_File file;

var int? err =
    watching UV_FS_Open("file.txt", _O_RDONLY, 0) -> (&file) do
        await file.ok;
        // file is ready for use
        _printf("open ok\n");
    end;
if err? then
    _printf("open error: %d\n", err!);
end

escape 0;
```

UV__FS__Read

Reads bytes from a file.

```
code/await UV_FS_Read (var& UV_FS_File file, vector&[] byte buf, var usize size, var usize o
                       -> ssize
```

- Parameters
  - `file`: file handle to read from
  - `buf`: destination buffer
  - `size`: number of bytes to read
  - `offset`: starting file offset
- Return
  - `ssize`: actual number of bytes read
    - `>=0`: number of bytes
    - `<0`: read error

Céu-libuv references: `ceu_uv_fs_read`, `UV_FS`.

libuv references: `uv_buf_init`, `uv_fs_req_cleanup`.

Note: all allocated libuv resources are automatically released on termination.

Example

Prints the contents of `file.txt` in a loop that reads the file in chunks of 10 bytes:

```
##include "uv/fs.ceu"

var& UV_FS_File file;

var int? err =
    watching UV_FS_Open("file.txt", _O_RDONLY, 0) -> (&file) do
        await file.ok;

        var usize offset = 0;
        loop do
            vector[11] byte buf;
            var ssize n = await UV_FS_Read(&file,&buf,$$buf-1,offset);
            if n == 0 then
                break;
            end
            buf = buf .. [{'\0'}];
            _printf("%s", &&buf[0]);
            offset = offset + ($$buf-1);
        end
    end;
_ceu_dbg_assert(not err?);
```

```
escape 0;
```

UV_FS_ReadLine

Reads a line from a file.

```
code/await UV_FS_ReadLine (var& UV_FS_File file, vector&[] byte buf, var usize offset)
                          -> ssize
```

- Parameters
  - `file`: file handle to read from
  - `buf`: destination buffer (excludes the leading `\n`)
  - `offset`: starting file offset
- Return
  - `ssize`: actual number of bytes read
    - `>=0`: number of bytes (includes the leading `\n`)
    - `<0`: read error

```
TODO: the file is currently read byte by byte.
```

Céu-libuv references: `UV_FS_Read`.

Example

Prints the contents of `file.txt` in a loop that reads the file line by line:

```
##include "uv/fs.ceu"

var& UV_FS_File file;

watching UV_FS_Open("file.txt", _O_RDONLY, 0) -> (&file) do
    await file.ok;

    var usize off = 0;
    loop do
        vector[] byte line;
        var ssize n = await UV_FS_ReadLine(&file,&line,off);
        if n <= 0 then
            break;
        end
        _printf("line = %s [%d]\n", &&line[0], n as int);
        off = off + (n as usize);
    end
end

escape 0;
```

UV__FS__Write

Write bytes from a file.

```
code/await UV_FS_Write (var& UV_FS_File file, vector&[] byte buf, var usize size, var usize
                        -> ssize
```

- Parameters
  - `file`: file handle to write to
  - `buf`: source buffer
  - `size`: number of bytes to write
  - `offset`: starting file offset
- Return
  - `ssize`: actual number of bytes written
    - `>=0`: number of bytes
    - `<0`: write error

Céu-libuv references: `ceu_uv_fs_write`, `UV_FS`.

libuv references: `uv_buf_init`, `uv_fs_req_cleanup`.

Note: all allocated libuv resources are automatically released on termination.

Example

Writes the string *Hello World* to `hello.txt`:

```
##include "uv/fs.ceu"

var& UV_FS_File file;

var _mode_t mode = _S_IRUSR|_S_IWUSR|_S_IRGRP|_S_IWGRP|_S_IROTH;

var int? err =
    watching UV_FS_Open("hello.txt", _O_CREAT|_O_WRONLY, mode) -> (&file) do
        await file.ok;
        vector[] byte buf = [] .. "Hello World!\n";
        var ssize n = await UV_FS_Write(&file,&buf,$buf,0);
        if (n<0) or (n as usize)!=$buf then
            _printf("write error\n");
        end
    end;
if err? then
    _printf("open error: %d\n", err!);
end

escape 0;
```

UV__FS__Fstat

Reads information about a file.

```
code/await UV_FS_Fstat (var& UV_FS_File file, var& _uv_stat_t stat)
                       -> int
```

- Parameters
  - `file`: file handle to write to
  - `stat`: destination buffer
- Return
  - `int`: operation status
    - `0`: success
    - `<0`: error

Céu-libuv references: `ceu_uv_fs_fstat`, `UV_FS`.

libuv references: `uv_fs_req_cleanup`.

Note: all allocated libuv resources are automatically released on termination.

Example

Prints the size of `file.txt` in bytes:

```
##include "uv/fs.ceu"

var& UV_FS_File file;

var int? err =
    watching UV_FS_Open("file.txt", _O_RDONLY, 0) -> (&file)
    do
        await file.ok;

        var _uv_stat_t stat = _;
        await UV_FS_Fstat(&file, &stat);
        _printf("size = %ld\n", stat.st_size);
    end;

if err? then
    _printf("open error: %d\n", err!);
end

escape 0;
```

Stream

Stream

`TODO`

Input Events

UV_STREAM_LISTEN

`input (_uv_stream_t&&, int) UV_STREAM_LISTEN;`

- Occurrence:
  - Whenever a stream server receives an incoming connection.
- Payload:
  - `_uv_stream_t&&`: pointer to the stream server

libuv reference: http://docs.libuv.org/en/v1.x/stream.html#c.uv_connection_cb

UV_STREAM_CONNECT

`input (_uv_connect_t&&, int) UV_STREAM_CONNECT;`

- Occurrence:
  - Whenever a connection opens.
- Payload:
  - `_uv_connect_t&&`: pointer to the connection
  - `int`: open status
    - `0`: success
    - `<0`: error

libuv reference: http://docs.libuv.org/en/v1.x/stream.html#c.uv_connect_cb

UV_STREAM_READ

`input (_uv_stream_t&&, ssize) UV_STREAM_READ;`

- Occurrence:
  - Whenever data is available on a stream.
- Payload:
  - `_uv_stream_t&&`: pointer to the stream
  - `ssize`: number of bytes available
    - `>0`: data available
    - `<0`: error

libuv reference: http://docs.libuv.org/en/v1.x/stream.html#c.uv_read_cb

UV_STREAM_WRITE

`input (_uv_write_t&&, int) UV_STREAM_WRITE;`

- Occurrence:
  - Whenever writing to a stream completes.
- Payload:
  - `_uv_write_T&&`: pointer to the write request
  - `int`: completion status
    - `0`: success
    - `<0`: error

libuv reference: http://docs.libuv.org/en/v1.x/stream.html#c.uv_write_cb

UV_STREAM_ERROR

`input (_uv_stream_t&&, int) UV_STREAM_ERROR;`

- Occurrence:
  - Whenever a read or write error occurs in a stream.
- Payload:
  - `_uv_stream_t&&`: pointer to the stream
  - `int`: error code

`UV_STREAM_ERROR` always occurs before the corresponding `UV_STREAM_READ` or `UV_STREAM_WRITE`.

libuv reference: http://docs.libuv.org/en/v1.x/errors.html

Data Abstractions

`TODO`

Code Abstractions

UV_Stream_Listen

Starts listening for incoming connections in a stream.

```
code/await UV_Stream_Listen (var& _uv_stream_t stream, var int backlog)
                                -> (event& void ok)
                                    -> int
```

- Parameters
  - `stream`: stream to listen
  - `backlog`: number of connections the kernel might queue
- Initialization
  - `ok`: signalled on every new incoming connection

- Return
  - `int`: operation status
    - `0`: success
    - `<0`: error

Céu-libuv references: `ceu_uv_listen`, `UV_STREAM_LISTEN`.

Example

Opens a `server` tcp handle, binds it to port `7000`, and then enters in listen mode. Each incoming connection triggers `ok_listen` whose reaction accepts the client, prints its address, and closes the connection.

```
##include "uv/tcp.ceu"

var& _uv_tcp_t server;
watching UV_TCP_Open() -> (&server) do
    var _sockaddr_in addr = _;
    _uv_ip4_addr("0.0.0.0", 7000, &&addr);
    _uv_tcp_bind(&&server, &&addr as _sockaddr&&, 0);

    event& void ok_listen;
    watching UV_TCP_Listen(&server,128) -> (&ok_listen) do
        every ok_listen do
            var _uv_tcp_t client = _;
            var int err = _ceu_uv_tcp_init(&&client);
            _ceu_dbg_assert(err == 0);
            var int ret = _uv_accept(&&server as _uv_stream_t&&, &&client as _uv_stream_t&&)
            _ceu_dbg_assert(ret == 0);

            vector[20] _char ip = _;
            var _sockaddr_in name = _;
            var int len = _;
            _uv_tcp_getsockname(&&client, &&name as _sockaddr&&, &&len);
            _uv_ip4_name(&&name,&&ip[0],20);
            _printf("new incoming connection from %s\n", &&ip[0]);
            _uv_close(&&client as _uv_handle_t&&, null);
        end
    end
end

escape 0;
```

UV__Stream__Read

Reads bytes from a stream continuously.

```
code/await UV_Stream_Read (var& _uv_stream_t stream, vector&[] byte buf)
                          -> (event& usize ok)
                              -> int
```

- Parameters
  - `stream`: stream to read from
  - `buf`: destination buffer
- Initialization
  - `ok`: signalled whenever new data is read to the destination buffer
- Return
  - `int`: read error
    - returns only case of error (always `<0`)

Céu-libuv references: `ceu_uv_read_start`, `UV_STREAM_READ`.

libuv references: `uv_read_stop`.

Note: all allocated libuv resources are automatically released on termination.

Example

Connects to `127.0.0.1:7000` and waits reading 10 bytes in a loop:

```
##include "uv/tcp.ceu"

var& _uv_tcp_t tcp;
var int? err =
    watching UV_TCP_Open() -> (&tcp) do
        var _uv_connect_t connect = _;
        var _sockaddr_in dest = _;
        _uv_ip4_addr("127.0.0.1", 7000, &&dest);
        _ceu_uv_tcp_connect(&&connect, &&tcp, (&&dest as _sockaddr&&));

        var _uv_connect_t&& c;
        var int status;
        (c,status) = await UV_STREAM_CONNECT until c==&&connect;
        _ceu_dbg_assert(status == 0);

        vector[11] byte buf;

        event& usize ok_read;
        var int? err2 =
            watching UV_TCP_Read(&tcp,&buf) -> (&ok_read) do
                loop do
                    await ok_read;
                    if $buf == 10 then  // assumes server sends exactly 10 bytes
                        break;
                    end
```

```
                end
            end;
        _ceu_dbg_assert(not err2?);

        buf = buf .. [{'\0'}];
        _printf("buf: %s\n", &&buf[0]);
    end;
_ceu_dbg_assert(not err?);

escape 0;
```

TCP

TCP

TODO

Input Events

TODO

Data Abstractions

TODO

Code Abstractions

TODO

License

License

Céu-Arduino is distributed under the MIT license reproduced below:

```
Copyright (C) 2012-2017 Francisco Sant'Anna

Permission is hereby granted, free of charge, to any person obtaining a copy of
this software and associated documentation files (the "Software"), to deal in
the Software without restriction, including without limitation the rights to
```

use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.