

Introduction

Introduction

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

Mode of Operation

Mode of Operation

The mode of operation specifies how Céu-libuv captures events from the environment (e.g., timers and incoming network traffic) and redirects them to the Céu application. It is implemented in C and is part of Céu-libuv.

Céu-libuv maps each libuv request/callback to a corresponding request/input in Céu. As an example, instead of reading from a stream with `uv_read_start`, Céu-libuv uses `ceu_uv_read_start` which generates `UV_STREAM_READ` input events back to the application, as follows:

```
##define ceu_uv_read_start(stream) uv_read_start(stream,...,ceu_uv_read_start_cb);

void ceu_uv_read_start_cb(uv_stream_t* stream, ...) {
    <...>
    ceu_input(CEU_INPUT_UV_STREAM_READ, <stream>);
}
```

Under the hood, Céu-libuv uses one *event loop*, one *timer*, and one *async* libuv handles. The timer manages Céu timers. The async manages Céu asyncs and threads. The main event loop makes continuous calls to `uv_run` passing `UV_RUN_ONCE`:

```
int main (void) {
    ceu_start();
    while (<program-is-running>) {
        uv_run(&loop, UV_RUN_ONCE);           // handles all libuv callbacks
        ceu_input(CEU_INPUT__ASYNC, NULL);    // handles timers and asyncs
    }
    ceu_stop();
}
```

File System

File System

Provides file system operations.

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

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

A file handle.

```
data UV_FS_File with
  var&[] byte  buffer;
  var  usize offset = 0;
  var  int  handle = -1;
  event void ok;
end
```

- Fields:
 - `buffer`: alias to the read & write buffer
 - `offset`: current offset for read & write operations
 - `handle`: underlying operating system handle
 - `ok`: event signalled when the file is opened successfully

Code Abstractions

UV_FS_Open

Opens a file.

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

- Parameters
 - **path**: path to the file
 - **buffer_size**: size of the read & write ring buffer (default: 1024)
 - **flags**: access mode flags (default: `_O_RDONLY`)
 - **mode**: file permission mode (default: 0)
- Public fields
 - **file**: file handle
- Return
 - **int**: open error
 - * returns only in case of an error (always <0)

The file is only ready for use after `file.ok` is triggered.

Céu-libuv references: `UV_FS`.

libuv references: `ceu_uv_fs_open`, `uv_fs_close`, `uv_fs_req_cleanup`.

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_Open o = spawn UV_FS_Open("file.txt",_,_,_);
var int? err =
  watching o do
    await o.file.ok;
    _printf("open ok\n");    // file is ready for use
  end;
if err? then
  _printf("open error: %d\n", err!);
end

escape 0;
```

UV_FS_Read_N

Reads a specified number of bytes in the file handle to its buffer.

```
code/await UV_FS_Read_N (var& UV_FS_File file, var usize n) -> ssize
```

- Parameters
 - **file**: file handle to read
 - **n**: number of bytes to read

- Return
 - `ssize`: number of bytes read from `file`
 - * `>=0`: number of bytes (less than or equal to `n`)
 - * `<0`: read error

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

libuv references: `uv_buf_init`, `uv_fs_req_cleanup`.

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_Open o = spawn UV_FS_Open("file.txt", 11, _,_);
var int? err =
  watching o do
    await o.file.ok;

    loop do
      var ssize n = await UV_FS_Read_N(&o.file, $$o.file.buffer-1);
      if n == 0 then
        break;
      end
      o.file.buffer = o.file.buffer .. [{'\0'}];
      _printf("%s", &o.file.buffer[0]);
      $o.file.buffer = 0;
    end
  end;
_ceu_dbg_assert(not err?);

escape 0;
```

UV_FS_Read_Line

Reads a line from a file handle.

code/await `UV_FS_Read_Line` (`var& UV_FS_File file`, `var&[] byte line`, `var usize? by`) -> `ssize`

- Parameters
 - `file`: file handle to read
 - `line`: alias to destination buffer (excludes the leading `\n`)
 - `by`: size of read chunks in bytes (default: 128)
- Return
 - `ssize`: number of bytes read from `file`
 - * `>=0`: number of bytes (includes the leading `\n` and extra bytes)

* <0: read error

The file handle buffer advances to the byte after the `\n`.

Céu-libuv references: `UV_FS_Read_N`.

Example

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

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

var&? UV_FS_Open o = spawn UV_FS_Open("file.txt",_,_,_);
watching o do
  await o.file.ok;
  loop do
    var[] byte line;
    var ssize n = await UV_FS_Read_Line(&o.file,&line,_);
    if n <= 0 then
      break;
    end
    line = line .. [{'\0'}];
    _printf("%s\n", &&line[0], n);
  end
end

escape 0;
```

UV_FS_Write_N

Writes a specified number of bytes in the file handle from its buffer.

code/await `UV_FS_Write_N (var& UV_FS_File file, var usize? n) -> ssize`

- Parameters
 - `file`: file handle to write
 - `n`: number of bytes to write (default: current size of the file buffer)
- Return
 - `ssize`: 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`.

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&? UV_FS_Open o = spawn UV_FS_Open("hello.txt", _, _O_CREAT|_O_WRONLY, mode);
watching o do
    await o.file.ok;
    o.file.buffer = [] .. "Hello World!\n";
    var usize n1 = $o.file.buffer;
    var ssize n2 = await UV_FS_Write_N(&o.file,$o.file.buffer);
    _ceu_dbg_assert(n2>=0 and n2==n1);
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`.

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

Provides stream operations.

libuv reference: <http://docs.libuv.org/en/v1.x/stream.html>

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

UV_Stream

A stream handle.


```

data UV_Stream with
  var&[] byte      buffer;
  var&  _uv_stream_t handle;
end

```

- Fields:
 - **buffer**: alias to the read & write buffer
 - **handle**: underlying operating system handle

Code Abstractions

UV_Stream_Listen

Starts listening for incoming connections in a stream handle.

```

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

```

- Parameters
 - **stream**: stream handle to listen
 - **backlog**: number of connections the kernel might queue (default: 128)
- Public fields
 - **ok**: event 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 TCP stream, binds it to port 7000, and then enters in listen mode. Each incoming connection triggers the event `ok`.

```

#include "uv/tcp.ceu"

var&? UV_TCP_Open tcp = spawn UV_TCP_Open(_);
watching tcp do
  var _sockaddr_in addr = _;
  _uv_ip4_addr("0.0.0.0", 7000, &&addr);
  _uv_tcp_bind(&&tcp.stream.handle as _uv_tcp_t&&, &&addr as _sockaddr&&, 0);

  var&? UV_Stream_Listen listen = spawn UV_Stream_Listen(&tcp.stream,_);
  watching listen do
    every listen.ok do

```

```

        <...>    // handle incoming connections
    end
end
end

escape 0;

```

UV_Stream_Read_N

Reads a specified number of bytes in the stream handle to its buffer.

code/await UV_Stream_Read_N (var& UV_Stream stream, var usize? n) -> ssize

- Parameters
 - **stream**: stream handle to read
 - **n**: number of bytes to read (default: whatever arrives in the stream)
- Return
 - **ssize**: number of bytes read from **stream**
 - * ≥ 0 : number of bytes (not related to **n**)
 - * < 0 : read error

After returning, if no errors occur, the stream handle buffer will contain at least **n** bytes. If the buffer already contains **n** bytes in the beginning, no read occurs and 0 is returned.

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

libuv references: `uv_read_stop`.

Example

Connects to 127.0.0.1:7000 and reads and writes in a loop:

```

#include "uv/tcp.ceu"

var&? UV_TCP_Connect c = spawn UV_TCP_Connect("127.0.0.1", 7000, _);
watching c do
    await c.ok;

    loop do
        await UV_Stream_Read_N(&c.stream,_);    // reads anything
        _printf("%s\n", &&c.stream.buffer[0]);    // shows it in the screen
        await UV_Stream_Write_N(&c.stream,_);    // writes it back
    end
end

escape 0;

```

UV_Stream_Read_Line

Reads a line from a stream handle.

code/await UV_Stream_Read_Line (var& UV_Stream stream, var&[] byte line) -> ssize

- Parameters
 - **stream**: stream handle to read
 - **line**: alias to destination buffer (excludes the leading \n)
- Return
 - **ssize**: number of bytes read from **stream**
 - * >=0: number of bytes (not related to **n**)
 - * <0: read error

Céu-libuv references: UV_Stream_Read_N.

Example

Connects to 127.0.0.1:7000 and reads and writes in a loop:

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

var&? UV_TCP_Connect c = spawn UV_TCP_Connect("127.0.0.1", 7000, _);
watching c do
  await c.ok;

  loop do
    var[] byte line;
    await UV_Stream_Read_Line(&c.stream,&line);      // reads a line
    _printf("%s\n", &&line[0]);                      // shows it in the screen
    line = line .. "\n" .. c.stream.buffer;
    c.stream.buffer = [] .. line;
    await UV_Stream_Write_N(&c.stream,_);           // writes it back
  end
end

escape 0;
```

UV_Stream_Write_N

Writes a specified number of bytes in the stream handle from its buffer.

code/await UV_Stream_Write_N (var& UV_Stream stream, var usize? n) -> ssize

- Parameters
 - **stream**: stream handle to write
 - **n**: number of bytes to write (default: current size of the **stream** buffer)
- Return

- **ssize**: number of bytes written
- * ≥ 0 : number of bytes
- * < 0 : write error

Céu-libuv references: `ceu_uv_write`, `UV_STREAM_WRITE`.

Example

Connects to 127.0.0.1:7000 and reads and writes in a loop:

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

var&? UV_TCP_Connect c = spawn UV_TCP_Connect("127.0.0.1", 7000, _);
watching c do
  await c.ok;

  loop do
    await UV_Stream_Read_N(&c.stream,_);    // reads anything
    _printf("%s\n", &&c.stream.buffer[0]); // shows it in the screen
    await UV_Stream_Write_N(&c.stream,_);    // writes it back
  end
end

escape 0;
```

TCP

TCP

Provides TCP operations.

libuv reference: <http://docs.libuv.org/en/v1.x/tcp.html>

Code Abstractions

UV_TCP_Open

Opens a TCP stream.

code/await UV_TCP_Open (var int? buffer_size) -> (var UV_Stream stream) -> int

- Parameters
 - **buffer_size**: size of the read & write ring buffer (default: 1024)
- Public fields
 - **stream**: opened and uninitialized TCP stream
- Return

- **int**: TCP error
- * returns only in case of error (always <0)

Céu-libuv references: `ceu_uv_tcp_init`, `ceu_uv_close`, `UV_STREAM_ERROR`.

Example

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

var&? UV_TCP_Open tcp = spawn UV_TCP_Open(_);
var int? err =
  watching tcp do
    <...> // use the raw `tcp` stream
  end;
if err? then
  _fprintf(_stderr, "%s\n", _uv_strerror(err!));
end
```

escape 0;

Opens a TCP stream and connects it.

```
code/await UV_TCP_Connect (var _char&& ip, var int port, var int? buffer_size)
    -> (var& UV_Stream stream, event void ok)
    -> int
```

- Parameters
 - **ip**: remote host
 - **port**: remote port
 - **buffer_size**: size of the read & write stream ring buffer (default: 1024)
- Public fields
 - **stream**: TCP stream
 - **ok**: event signalled when **stream** connects and is ready for use
- Return
 - **int**: TCP error
 - * returns only in case of error (always <0)

Céu-libuv references: `ceu_uv_tcp_connect`, `UV_STREAM_CONNECT`.

Example

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

var&? UV_TCP_Connect c = spawn UV_TCP_Connect("127.0.0.1", 7000, _);
watching c do
  await c.ok;
  <...> // use the connected TCP `c.stream`
```

end

escape 0;

UV_TCP_Open_Bind_Listen

Opens a TCP stream, binds it to an IP and port, and listens for incoming connections.

```
code/await UV_TCP_Open_Bind_Listen (var _char&&? ip, var int port, var int? backlog, var int?
                                     -> (var& UV_Stream stream, event& void ok)
                                     -> int
```

- Parameters
 - **ip**: local host (default: "0.0.0.0")
 - **port**: local port
 - **backlog**: number of connections the kernel might queue (default: 128)
 - **buffer_size**: size of the read & write stream ring buffer (default: 1024)
- Public fields
 - **stream**: TCP stream
 - **ok**: event signalled on every new incoming connection
- Return
 - **int**: TCP error
 - * returns only in case of error (always <0)

Céu-libuv references: `UV_TCP_Open`, `UV_Stream_Listen`.

Example

Listen on port 7000:

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

```
var&? UV_TCP_Open_Bind_Listen tcp = spawn UV_TCP_Open_Bind_Listen("0.0.0.0", 7000, _,_);
watching tcp do
  every tcp.ok do
    <...>    // handle incoming connections
  end
end
```

escape 0;

Opens a TCP stream, binds it to an IP and port, listens for incoming connections, and spawns a handler on every new connection.

```
code/await UV_TCP_Server (var _char&&? ip, var int port,
```

```

var int? backlog, var int? buffer_size,
var&? UV_TCP_Server_Data shared)
-> int

```

- Parameters
 - **ip**: local host (default: "0.0.0.0")
 - **port**: local port
 - **backlog**: number of connections the kernel might queue (default: 128)
 - **buffer_size**: size of the read & write stream ring buffer (default: 1024)
 - **shared**: an optional payload to be shared with all handlers
- Return
 - **int**: TCP error
 - * returns only in case of error (always <0)

The handler is a user-defined `code/await` with the fixed identifier `UV_TCP_Server_Handler`, which must be declared in between the includes for `uv/tcp.ceu` and `uv/tcp-server.ceu`, as follows:

```

#include "uv/tcp.ceu"
code/await UV_TCP_Server_Handler (var& UV_Stream stream, var&? UV_TCP_Server_Data shared) ->
  <...>          // handles a new client connection
end
#include "uv/tcp-server.ceu"
<...>

```

The handler receives a TCP stream of the connected client.

If the macro `UV_TCP_SERVER_HANDLER_MAX` is defined, the server uses a bounded pool of `UV_TCP_Server_Handler` of that size.

Céu-libuv references: `UV_TCP_Open_Bind_Listen`, `UV_TCP_Open`.

libuv references: `[_uv_accept]`.

Example:

Executes a server on 0.0.0.0:7000 and handles each connection inside `UV_TCP_Server_Handler`:

```

#include "uv/tcp.ceu"

data UV_TCP_Server_Data;    // empty data

code/await UV_TCP_Server_Handler (var& UV_Stream stream, var&? UV_TCP_Server_Data shared) ->
  <...>          // handles a new client connection
end

```

```
##include "uv/tcp-server.ceu"  
  
await UV_TCP_Server("0.0.0.0", 7000, _,_,_);
```

License

License

Céu-libuv 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.