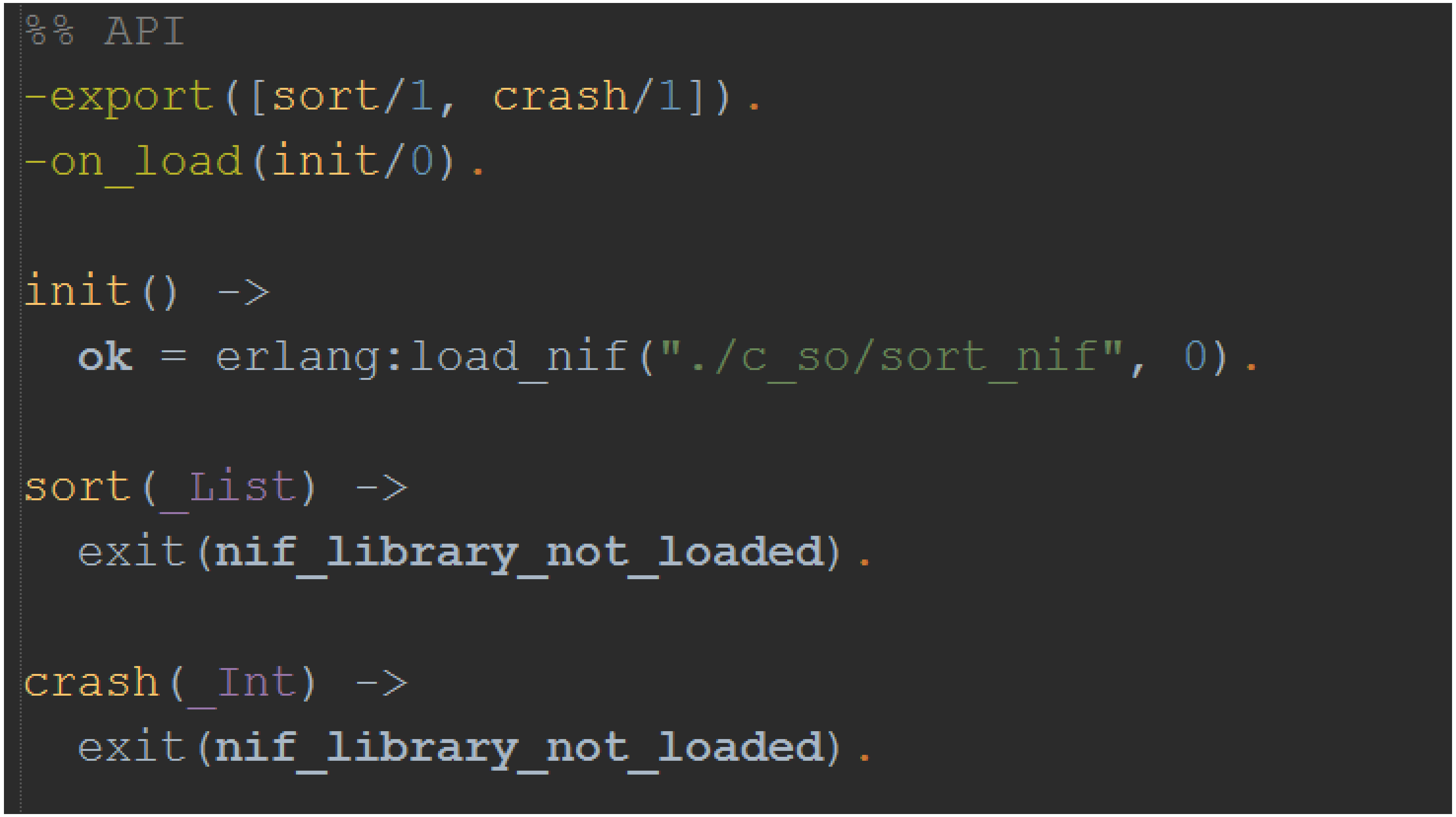
NIF

Overview

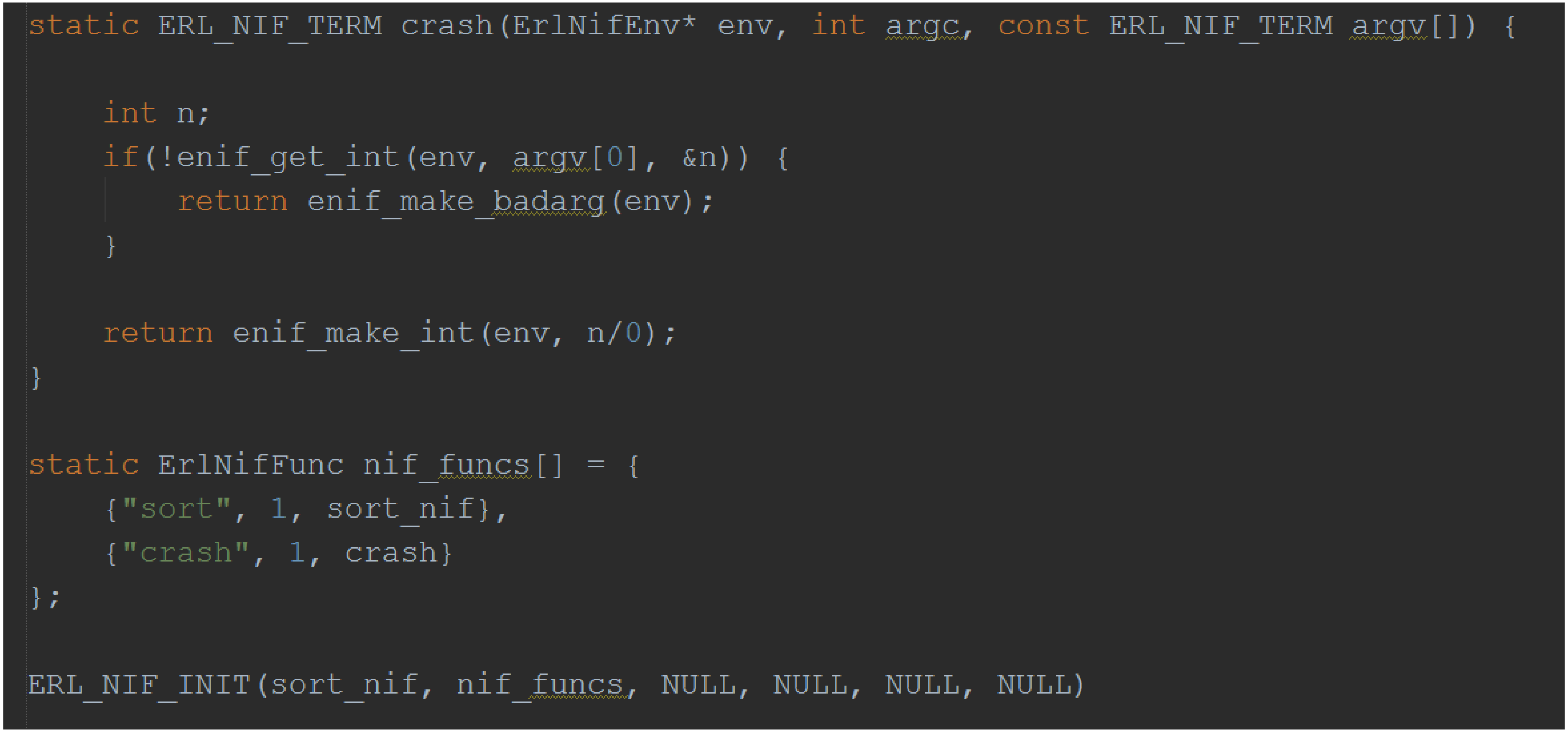
- A NIF (Native Implemented Function) is a function that is implemented in C instead of Erlang

Erlang part



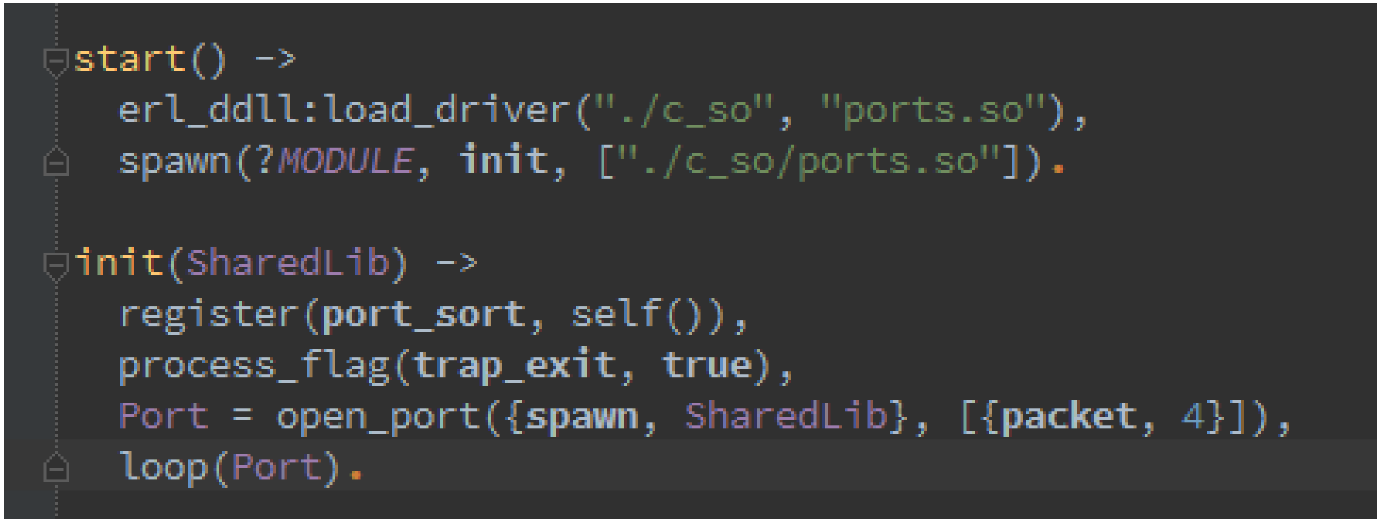
```erlang
%% API
-export([sort/1, crash/1]).
-on_load(init/0).

init() ->
  ok = erlang:load_nif("./c_so/sort_nif", 0).

sort(_List) ->
  exit(nif_library_not_loaded).

crash(_Int) ->
  exit(nif_library_not_loaded).
```

C part

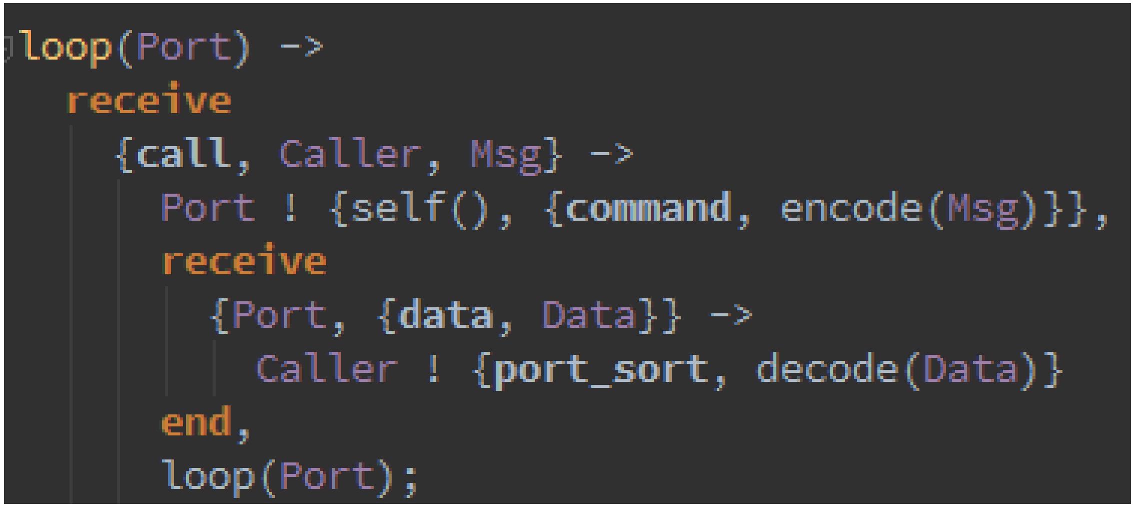


```c
static ERL_NIF_TERM crash(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[]) {

    int n;
    if(!enif_get_int(env, argv[0], &n)) {
        return enif_make_badarg(env);
    }

    return enif_make_int(env, n/0);
}

static ErlNifFunc nif_funcs[] = {
    {"sort", 1, sort_nif},
    {"crash", 1, crash}
};

ERL_NIF_INIT(sort_nif, nif_funcs, NULL, NULL, NULL, NULL)
```

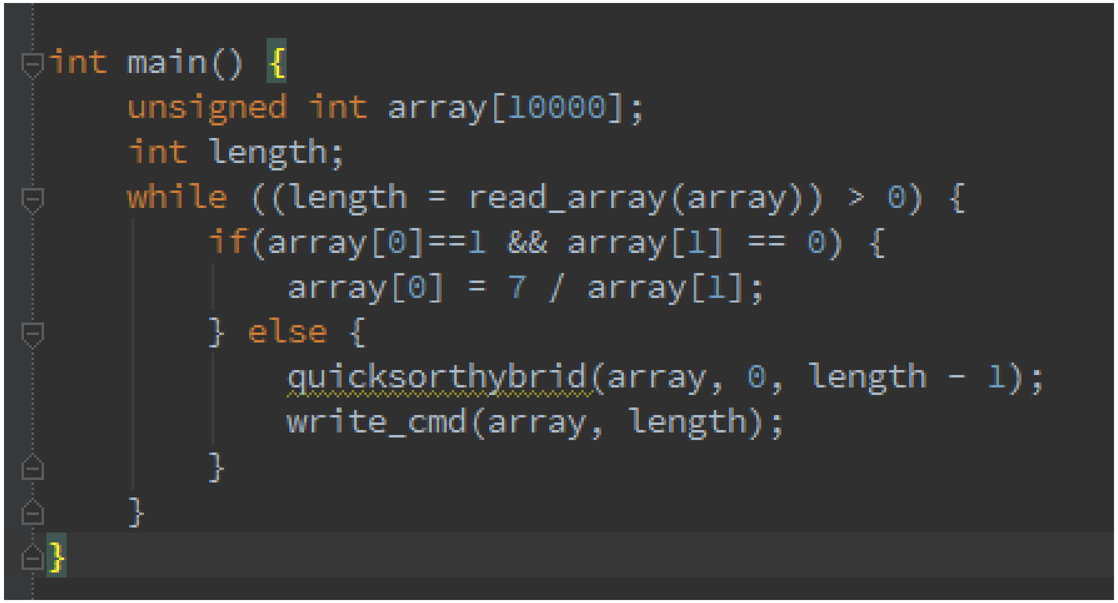
For more comparison: ports



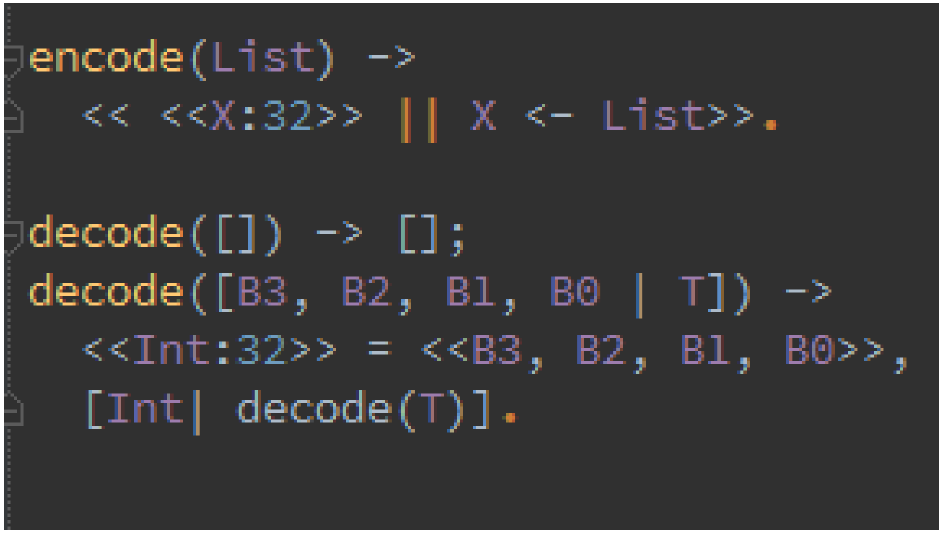
```
start() ->
  erl_ddll:load_driver("./c_so", "ports.so"),
  spawn(?MODULE, init, ["./c_so/ports.so"]).

init(SharedLib) ->
  register(port_sort, self()),
  process_flag(trap_exit, true),
  Port = open_port({spawn, SharedLib}, [{packet, 4}]),
  loop(Port).
```



```
loop(Port) ->
  receive
    {call, Caller, Msg} ->
      Port ! {self(), {command, encode(Msg)}},
      receive
        {Port, {data, Data}} ->
          Caller ! {port_sort, decode(Data)}
      end,
      loop(Port);
```

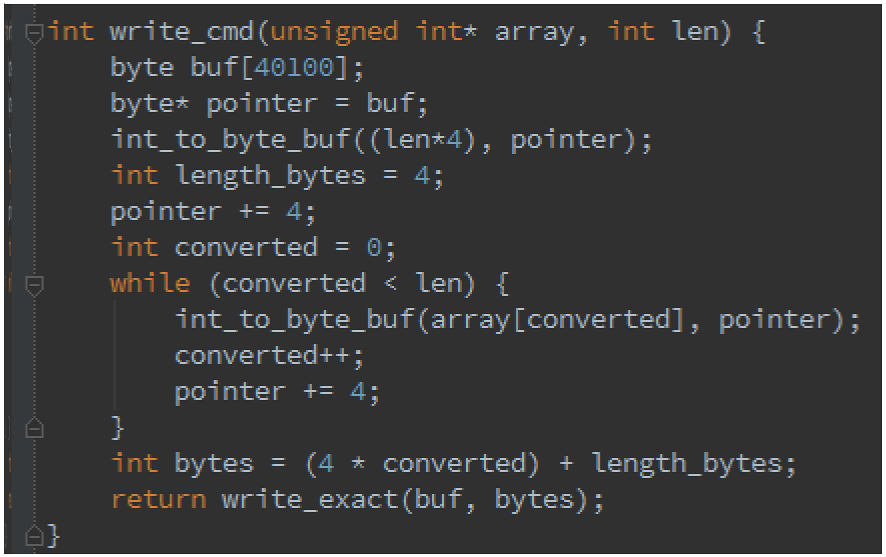


```
int main() {
    unsigned int array[10000];
    int length;
    while ((length = read_array(array)) > 0) {
        if(array[0]==1 && array[1] == 0) {
            array[0] = 7 / array[1];
        } else {
            quicksorthybrid(array, 0, length - 1);
            write_cmd(array, length);
        }
    }
}
```

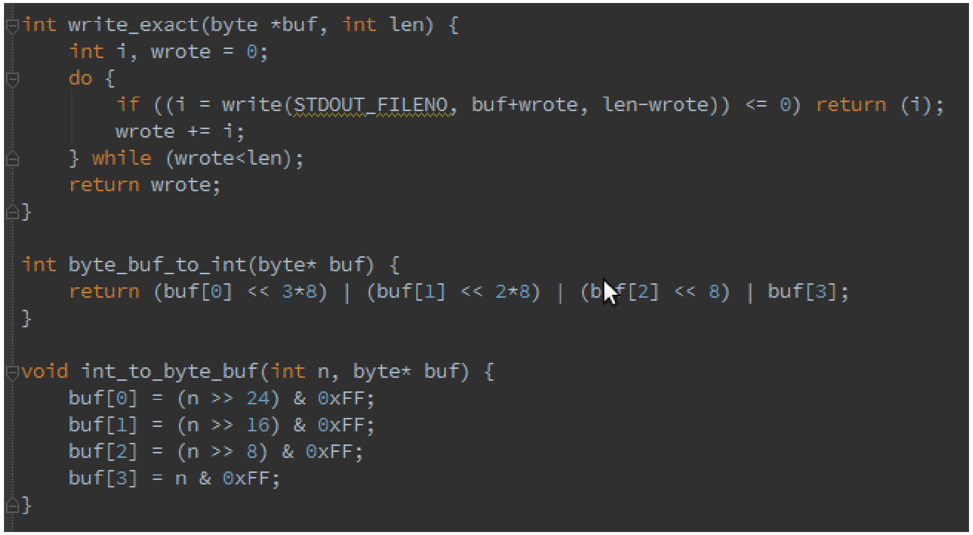


```
encode(List) ->
  << <<X:32>> || X <- List>>.

decode([]) -> [];
decode([B3, B2, B1, B0 | T]) ->
  <<Int:32>> = <<B3, B2, B1, B0>>,
  [Int| decode(T)].
```

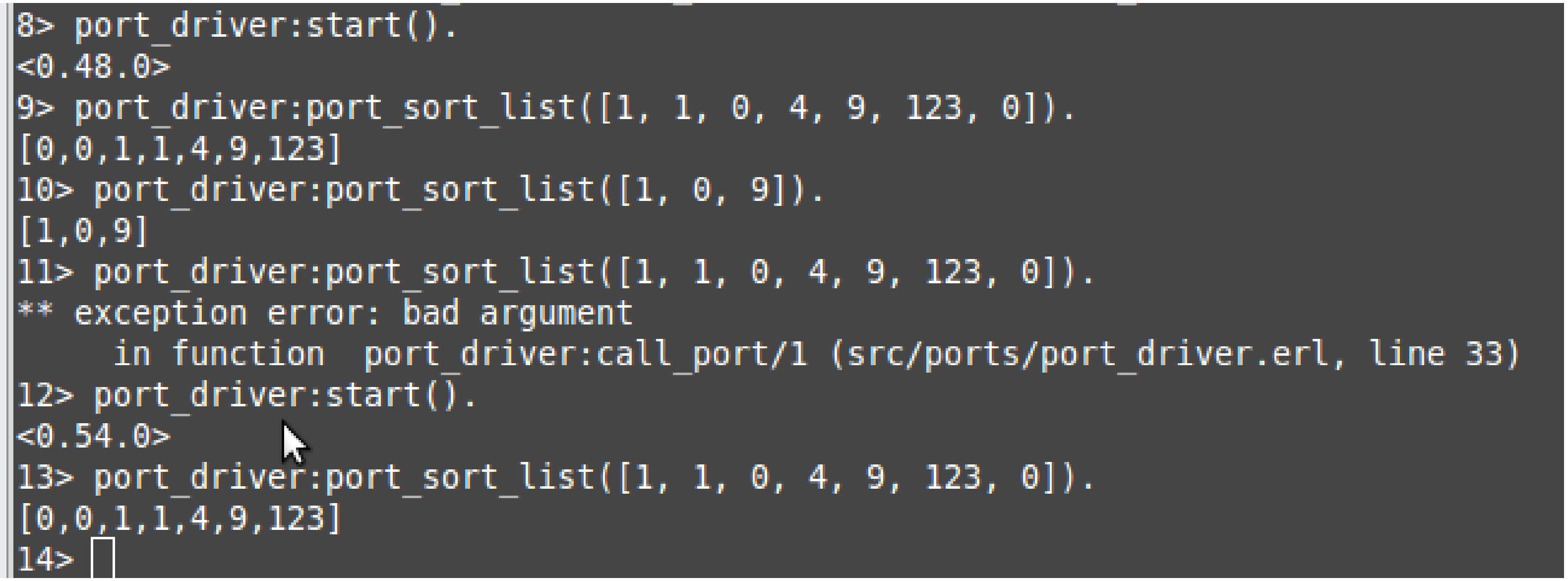
VS



```
int write_cmd(unsigned int* array, int len) {
    byte buf[40100];
    byte* pointer = buf;
    int_to_byte_buf((len*4), pointer);
    int length_bytes = 4;
    pointer += 4;
    int converted = 0;
    while (converted < len) {
        int_to_byte_buf(array[converted], pointer);
        converted++;
        pointer += 4;
    }
    int bytes = (4 * converted) + length_bytes;
    return write_exact(buf, bytes);
}
```

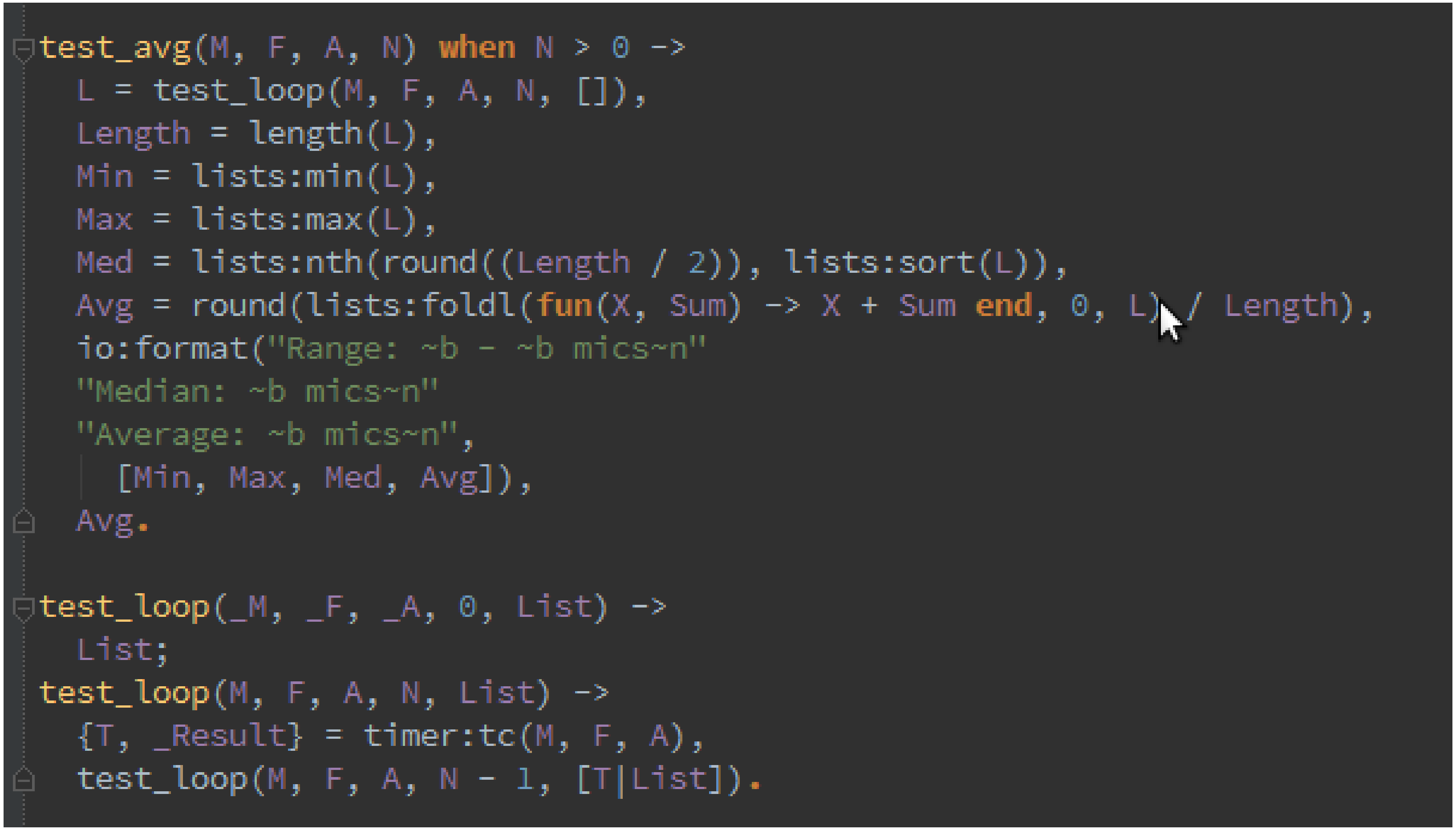


```
int write_exact(byte *buf, int len) {
    int i, wrote = 0;
    do {
        if ((i = write(STDOUT_FILENO, buf+wrote, len-wrote)) <= 0) return (i);
        wrote += i;
    } while (wrote<len);
    return wrote;
}

int byte_buf_to_int(byte* buf) {
    return (buf[0] << 3*8) | (buf[1] << 2*8) | (buf[2] << 8) | buf[3];
}

void int_to_byte_buf(int n, byte* buf) {
    buf[0] = (n >> 24) & 0xFF;
    buf[1] = (n >> 16) & 0xFF;
    buf[2] = (n >> 8) & 0xFF;
    buf[3] = n & 0xFF;
}
```



```
8> port_driver:start().
<0.48.0>
9> port_driver:port_sort_list([1, 1, 0, 4, 9, 123, 0]).
[0,0,1,1,4,9,123]
10> port_driver:port_sort_list([1, 0, 9]).
[1,0,9]
11> port_driver:port_sort_list([1, 1, 0, 4, 9, 123, 0]).
** exception error: bad argument
     in function  port_driver:call_port/1 (src/ports/port_driver.erl, line 33)
12> port_driver:start().
<0.54.0>
13> port_driver:port_sort_list([1, 1, 0, 4, 9, 123, 0]).
[0,0,1,1,4,9,123]
14> 
```

Time for some testing



```
test_avg(M, F, A, N) when N > 0 ->
  L = test_loop(M, F, A, N, []),
  Length = length(L),
  Min = lists:min(L),
  Max = lists:max(L),
  Med = lists:nth(round((Length / 2)), lists:sort(L)),
  Avg = round(lists:foldl(fun(X, Sum) -> X + Sum end, 0, L) / Length),
  io:format("Range: ~b - ~b mics~n"
  "Median: ~b mics~n"
  "Average: ~b mics~n",
    [Min, Max, Med, Avg]),
  Avg.

test_loop(_M, _F, _A, 0, List) ->
  List;
test_loop(M, F, A, N, List) ->
  {T, _Result} = timer:tc(M, F, A),
  test_loop(M, F, A, N - 1, [T|List]).
```

	NIF	PORTS
SPEED	Very fast, in specific situations could be faster than bif	Much slower bacause of byte communication
CRASH	Brings the virtual machine down	Behaves almost like a process – makes managing crash easy.
CODE	Erlang code very simple, focus on code in C, conversion between C types and Erlang terms	Much more work to manage communication between Erlang and C parts

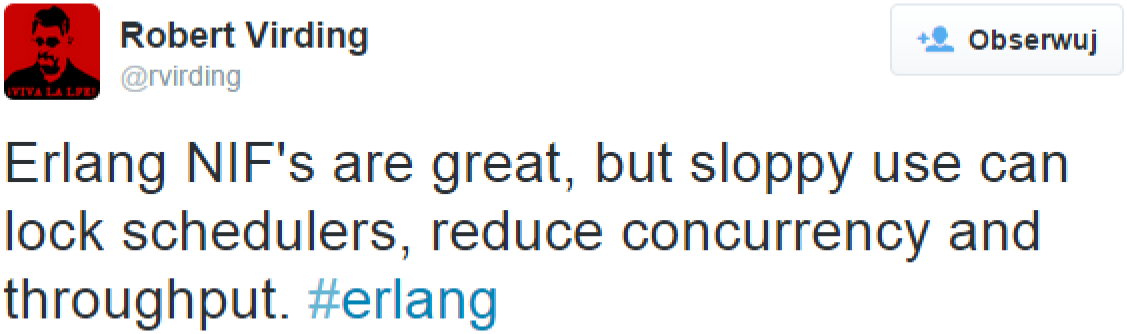
Pros

- NIFs let you achieve the highest speed
- Minimises amount of Erlang code to manage C
- Library providing Erlang ↔ C communication and types convertion.

Cons

- If crash when running NIF whole virtual machine is down
- Long-running NIFs will take over a scheduler and prevent Erlang from efficiently handling many processes
- Problems with scaling

Summary



Robert Virding
@rvirding

Erlang NIF's are great, but sloppy use can lock schedulers, reduce concurrency and throughput. #erlang

References

All code and this presentation is available on: https://github.com/fib1123/nif_erlangproject

For more information check:

- http://www.erlang.org/doc/tutorial/nif.html
- http://www.erlang.org/doc/man/erl_nif.html