Operating Systems

Fall 2022, Stephan Ohl

Final Project - Event Loop Framework

Write a small framework to create communicating event loop systems in C. The system should start a main event loop, which then may create more event loops, which may communicate with one another through events. It should be possible to finish event loops. Each event loop should run in its own OS thread, which means that communication between event loops has to be made thread-safe.

Basic Interface

The basic interface of your system should support at least the following types and operations.

```
elf_status_t;  // a status or error code
elf_event_t;   // an event
elf_loop_t;    // an event loop handle
elf_handler_t; // an event handling function

elf_status_t handler(elf_loop_t loop, elf_event_t event);

elf_status_t elf_main(elf_handler_t handler);
elf_status_t elf_init(elf_loop_t* ref_loop, elf_handler_t handler);
elf_status_t elf_fini(elf_loop_t* ref_loop);

elf_status_t elf_send(elf_loop_t loop, elf_event_t event);
```

The `elf_status_t` should be a C-style status and error code. The `elf_loop_t` and `elf_event_t` should be types that are transparent to the user, which means that instances of these type can only be accessed by interface functions and not directly. The `elf_handler_t` should be a function pointer with above signature.

The `elf_init` function creates a new event loop and initializes an `elf_loop_t` handle. The `elf_fini` function stops an event loop. An event loop may stop itself using its handler's `loop` handle.

It should be possible to send an `elf_loop_t` to another event loop in an `elf_event_t`. It is possible as well to store `elf_loop_t` handles in the event loop's local state in-between of loop's event handler invocations. Note that is implies that multiple loops, and thus multiple threads, may have access to the same loop via handles. This requires careful implementation.

You can modify the interface slightly and you may have to add additional interface functionality to implement the aforementioned functionality.

Example Code

You should write an example program `example` that uses your framework. The interactive program should demonstrate a mix of IO and CPU bound operations distributed over multiple event loops. The IO may be via `stdin`, `stdout`, or files. The code should demonstrate how to communicate bidirectionally between event loops (requests and acknowledgments) and how to shut down event loops in a clean way. Be creative!

Code Requirements

- The framework must comply strictly to the **C11** standard. You can ensure compliance by using gcc flags such as `-std=c11` and `-pedantic`.
- The framework must be **POSIX** compliant. In particular, the threading has to be based on the `<pthread.h>` API.
- The user interface of your framework should be contained in the `"elf_interface.h"` header file. The framework should be compiled as a static library.
- The project must be organized in a folder `os_final_`*<name>* containing a **Makefile** with **build**, **run**, and **clean** targets. The folder has be submitted as compressed tar.gz or tar.bz2 archive. After downloading, the following commands should produce and run the executable.

```
$> tar -xf os_final_ohl.tar.bz2
$> cd os_final_ohl
```

```
$> make build
$> make run
```