



Boost Context Module Implementation for s390x

Mentor: Dan Horak

Mentee: Naveen Naidu

naveennaidu479@gmail.com

Junior Year - Computer Science Undergraduate

Project Overview

TITLE: Boost Context Module Implementation on s390x architecture

Aim

To implement the Boost's Context library on s390x architectures so that it can reach parity with other platforms.

What is Boost.Context ?

boost.context is a foundational library that provides a sort of cooperative multitasking on a single thread. boost.context uses a concept called as fiber. A fiber provides the means to suspend the current execution path and to transfer execution control, thereby permitting another fiber to run on the current thread. This state full transfer mechanism enables a fiber to suspend execution from within nested functions and, later, to resume from where it was suspended.

The advantage of this library is that it reduces the number of CPU cycles drastically for context switching when compared to the context switching via system calls which usually involves OS kernel and thousands of CPU cycles. boost.context with it's faster execution time and better implementation proves to be a great tool for developers.

fcontext_t

boost.context() **depends heavily on fcontext_t** to deal with context switching. A *fcontext_t* provides the means to suspend the current execution path and to transfer execution control, thereby permitting another *fcontext_t* to run on the current thread.

The following are the functions which work alongside fcontext_t to make context switching possible:

I. `jump_fcontext()`

Calling `jump_fcontext()` **invokes the context_function in a newly created context** complete with registers, flags and stack and instruction pointers.

II. `make_fcontext()`

A new context supposed to execute a context-function (returning void and accepting void * as argument) **will be created on top of the stack** (at 16 byte boundary) by function `make_fcontext()`.

Different architectures have different stack structures and different ABI (*Application binary interface*) interfaces. It is not possible to implement the context switching on different architectures by a high level programming language, because - it won't be efficient nor would it allow us to use the special functions, an architecture might have. Hence it becomes necessary to write the executing code for the above functions in assembly.

The assembly codes for the above functions[`jump_fcontext()` and `make_fcontext()`] are stored in the directory [context/src/asm/](#). There are three assembly files for each of the different architectures which are implemented and optimized for the particular architecture.

These assembly files are not present for the s390x architecture as a result of which Boost.Context does not build and run on s390x. This project aims at implementing these files.

The **naming convention** for each assembly file is:

< function > _ < arch > _ < ABI > _ < Binary format > _ < assembler > . S

Eg: An example assembly file for s390x architecture would be: **jump_s39x_sysv_elf_gas.S**

Let's look at what the each assembly file do:

1. **jump_s390x_sysv_elf_gas.S** : context switch between two fibers/fiber_contexts
2. **make_s390x_sysv_elf_gas.S** : prepares stack for the first invocation
3. **ontop_s390x_sysv_elf_gas.S** : execute function on top of a fiber/fiber_context

Implementation

Before I start implementing the project, I would have to gather the knowledge regarding:

- The binary format
- Which ABI is used (ABI describes the calling convention)
- Information about the Stack frame layout

From my research I was able to find the answers to the above two questions.

- **The binary format** - ELF (since we are focusing on Linux)
- **The ABI** - System V (sysv)
- **Stack frame layout** - Yet to read about the s390x architecture.

Goals

1. Include s390x architecture in the boost's build property.
2. Implement **jump_s390x_sysv_elf_gas.S** assembly file.
3. Implement **make_s390x_sysv_elf_gas.S** assembly file.
4. Implement **ontop_s390x_sysv_elf_gas.S** assembly file
5. Test the working on a s390x hardware
6. Alternatively test if the libraries which were depended on context(coroutine,hhvm etc) are able to build
7. Document all the steps in the blog

Stretch Goals

Package the context module and the other modules which would build after successful implementation and send it upstream.

Planning

The implementation of each Assembly file would be done in one sprint having three stages.

Stage 1: Planning

- I would like to take some time to understand how the assembly code works by reading the documentation and by looking at the implementation of these in existing architecture

Stage 2: Implementation

- Implement the assembly codes by following the standard coding practices
- Document the steps

Stage 3: Testing

- Test whether the particular assembly file works.
- Get feedback from mentor
- Implement the feedback


Timeline

Coding Phase 1 (July 1 - August 2nd)

#WEEK	Tasks	Deliverables
Week 1 and 2 (July 1 - Jun 15)	Implement the architecture in Jamfile.v2 Implement make_fcontext()	Boost.context can detect the s390x arch and try to build using the assembly files. Make_fcontext assembly file implemented
Week 3 (June 15 - 27)	Implement jump_fcontext()	jump_fcontext() assembly file is implemented
Week 4 (June 27 - Aug 2)	Buffer Week	Partially completed project.

Coding Phase 2 (August 2 - September 2)

#WEEK	Tasks	Deliverables
-------	-------	--------------



Week 1 and 2 (Aug 2 - 16)	Implementing on_top context()	On_top context is implemented
Week 3 (Aug 16 - 25)	Build the entire Boost.Context library using the newly created assembly files	Boost.Context should now be able to be built on s390x architectures.
Week 4 (Aug 25 - Sep 2)	Buffer Week.	The Project is entirely completed