Boost Context module implementation for s390x

11 messages

Naveen Naidu <naveennaidu479@gmail.com> 17 February 2019 at 23:10
To: oliver.kowalke@gmail.com

Hello Oliver,

I am Naveen, a Junior Year Computer Science Undergraduate from India. I really apologize for disturbing you. I have very recently developed interest in the s390x architectures. I am planning to apply for *Open Mainframe Project Internship* program, whose one of the proposed project is to **Implement the Boost Context module for s390x**. I really liked the idea and would like to work on it, because of it's impact for the mainframe community. IIRC, the lack of this module blocks *coroutine* and *ceph* for the s390x architecture.

I have been going through the Boost.Context Manual to understand what the library is all about. The Manual is very resourceful, thank you very much for that. Me being a novice in s390x architecture - I am unable to grasp the content so quickly, It would be really kind of you if you help me get an idea about what has to be done, in order to implement the Context Library for s390x. Just a little hindsight is all I request, rest I'll put my best efforts into finding out what needs to be done.

In the little inspection I had, One part of the process to implement Boost Context in s390x architecture would be to provide assembly codes for s390x just like those present for x86_64, i386, arm64.

But I seem to be a little lost. It would be really grateful of you, if you could help me figure out what is it, that I need to do.

Thank you very much for your time and I again ask for your pardon if I have cause any inconvenience.

P . Naveen Naidu

Oliver Kowalke <oliver.kowalke@gmail.com> 18 February 2019 at 12:38
To: Naveen Naidu <naveennaidu479@gmail.com>

Hi,

Am So., 17. Feb. 2019 um 18:41 Uhr schrieb Naveen Naidu <naveennaidu479@gmail.com>:

> I am Naveen, a Junior Year Computer Science Undergraduate from India. I really apologize for disturbing you. I have very recently developed interest in the s390x architectures. I am planning to apply for *Open Mainframe Project Internship* program, whose one of the proposed project is to **Implement the Boost Context module for s390x**. I really liked the idea and would like to work on it, because of it's impact for the mainframe community.

That would be create - unfortunately I've never had access to a s390x system.
After looking into the boost build system it seams that some parts are not implemented. For instance to select the proper assembler file boost build's property `architecture` is used but it doesn't contain s390x at the moment -> needs to be fixed.

> But I seem to be a little lost. It would be really grateful of you, if you could help me figure out what is it, that I need to do.

First you need to figure out following infos:
- the binary format; if we focus on Linux on s390x it should be ELF
- which ABI is used; might be sysv
- the ABI describes the calling convention

The calling convention defines how a function invokes another, e.g. it describes which CPU registers are scratch registers and which one needs to be preserved during a subroutine call.
Additionally you need to know the stack frame layout.
With this data you could start implementing 3 assembler files:
- jump_s39x_sysv_elf_gas.S: context switch between to fibers/fiber_contexts
- make_s39x_sysv_elf_gas.S: prepares stack for the first invocation
- ontop_s39x_sysv_elf_gas.S: execute function on top of a fiber/fiber_context

> Thank you very much for your time and I again ask for your pardon if I have cause any inconvenience.

you are welcome

Oliver

Naveen Naidu <naveennaidu479@gmail.com> 28 February 2019 at 10:15
To: pavank830@gmail.com

[Quoted text hidden]

Naveen Naidu <naveennaidu479@gmail.com> 6 April 2019 at 22:03
To: Oliver Kowalke <oliver.kowalke@gmail.com>

Hello Oliver,

I am Naveen. I would like to thank you for really helping me get started with the information regarding how *Context can be implemented on s390x architecture*. I would like to inform you that I have been **accepted as the Intern** for the *OpenMainFrame Project* (which comes under The Linux Foundation) for the project **Implement the Boost Context module for s390x.** I would be starting my work pretty soon on the project.

I again would like to take this opportunity to thank you for all your support. This couldn't have been possible without you explaining me in brief about the project :)

Thanking you,
P . Naveen Naidu

[Quoted text hidden]

Oliver Kowalke <oliver.kowalke@gmail.com> 7 April 2019 at 10:07

To: Naveen Naidu <naveennaidu479@gmail.com>

Hi,
congratulations - I wish you all the best.
If you have questions feel free to ask.
Oliver
[Quoted text hidden]

Naveen Naidu <naveennaidu479@gmail.com> 13 April 2019 at 18:11
To: Oliver Kowalke <oliver.kowalke@gmail.com>

Hello Oliver,

I wanted to ask, if it's appropriate for me to create a issue regarding the s390x implementation for context module, to let other contributors know that I am working on it?
[Quoted text hidden]

Oliver Kowalke <oliver.kowalke@gmail.com> 13 April 2019 at 20:32
To: Naveen Naidu <naveennaidu479@gmail.com>

Yes, go ahead.
[Quoted text hidden]

Naveen Naidu <naveennaidu479@gmail.com> 25 April 2019 at 23:30
To: Shivansh Handa <avihs.29@gmail.com>

P . Naveen Naidu

---------- Forwarded message ---------
From: **Naveen Naidu** <naveennaidu479@gmail.com>
Date: Sun, 17 Feb 2019 at 23:10
Subject: Boost Context module implementation for s390x
To: <oliver.kowalke@gmail.com>

[Quoted text hidden]

Naveen Naidu <naveennaidu479@gmail.com> 26 April 2019 at 23:46
To: Oliver Kowalke <oliver.kowalke@gmail.com>

Hello Oliver,

Sorry to disturb you again. I have been going through the documentations of Boost.Context, Papers on Fibers(N4024), Fibers without scheduler (P0876R0) and The ELF ABI documentation in order to get a clearer understanding of what the *context* library actually does. So, far I have understood the following points. It would be really kind of you if you could please have a look to see if I understood it correctly.

Boost.Context is a library that provides the users an ability to perform cooperative multitasking on a single thread. It does this by providing an abstraction of the low level stuff such as the stack information, the execution state of the current thread, registers, stack pointers etc. This library provides the users with three ways using which they can control the execution flow of the program.
Three ways being:

1. Fibers
2. call\cc
3. continuation

Fibers, call\cc and continuations are implementations similar to that of a thread on a user space level that provides the means to suspend the current execution path and transfer execution control which in turn permits another context to run on the current thread.

In order for the above three means to actually save the context, this library uses something called as *fcontext* (which is an implementation whose interface is similar to that of *ucontext_t*). In a more specific way, Boost.Fiber uses call/cc from the Boost.Context as building block. The implementation uses *fcontext_t* per default. *fcontext_t* is based on assembler and not available for all platforms. The thing here being, *fcontext_t* is actually responsible for dealing with the context of the functions.

While reading the fcontext.hpp , I got a vague idea that the functions (*jump_context, make_context etc..)* provided by the header file are actually in someway connected to the Assembly files. That means when *Boost.context* is built, It first detects the architecture and on the basis of this it associates the function provided by fcontext.hpp with the appropriate assembler. For eg: whenever *jump_context* is called internally by fibers, it transfers the control to the assembly file to actually handle the low level stuff of storing the parameters into registers, initialization of stack frames etc. Having the assembly implementation though might be a little problem in adding new architectures but the speed provided using this approach is phenomenal as it reduces a lot of CPU cycles.

I also, have the following question:

How does Boost.context actually associate jump_context with the assembly files. I mean, how does Boost.context know which assembly file to launch when jump_context is run. It would be really great if you could point me to the file where this is actually being done. I know that we use *Jamfile.v2* and *architecture.jam* to select the assembly files, but I have not been able to understand how they get connected. The only thing that I see in *fcontext.hpp* is the functional definitions of the three functions.

Thank you very much for your time and I apologize for any inconvenience caused.
[Quoted text hidden]

Oliver Kowalke <oliver.kowalke@gmail.com> 29 April 2019 at 12:36
To: Naveen Naidu <naveennaidu479@gmail.com>

Am Fr., 26. Apr. 2019 um 20:16 Uhr schrieb Naveen Naidu <naveennaidu479@gmail.com>:

> Hello Oliver,
>
> Sorry to disturb you again. I have been going through the documentations of Boost.Context, Papers on Fibers(N4024), Fibers without scheduler (P0876R0) and The ELF ABI documentation in order to get a clearer understanding of what the *context* library actually does. So, far I have understood the following points. It would be really kind of you if you could please have a look to see if I understood it correctly.
>
> Boost.Context is a library that provides the users an ability to perform cooperative multitasking on a single thread. It does this by providing an abstraction of the low level stuff such as the stack information, the execution state of the current thread, registers, stack pointers etc. This library provides the users with three ways using which they can control the execution flow of the program.
> Three ways being:
>
> 1. Fibers
> 2. call\cc
> 3. continuation

it is fiber (actual API), call/cc (deprecated by the C++ committee), execution_context (v1/v2 - was deprecated by the C++ committee)

> In order for the above three means to actually save the context, this library uses something called as *fcontext* (which is an implementation whose interface is similar to that of *ucontext_t*).

yes, it is a faster implementation than ucontext_t

> In a more specific way, Boost.Fiber uses call/cc from the Boost.Context as building block.

actually both libs use the fiber API of boost.context

> The implementation uses *fcontext_t* per default. *fcontext_t* is based on assembler and not available for all platforms.

right, for platforms not supported by fcontext_t you could use ucontext_t instead (bjam property context-impl=ucontext)

> The thing here being, *fcontext_t* is actually responsible for dealing with the context of the functions.

right

> While reading the fcontext.hpp , I got a vague idea that the functions (*jump_context, make_context etc..)* provided by the header file are actually in someway connected to the Assembly files.

fcontext.hpp tells the function signature of jump_fcontext, make_fcontext ...

> That means when *Boost.context* is built, It first detects the architecture and on the basis of this it associates the function provided by fcontext.hpp with the appropriate assembler.

the assembler files are compiled and included in the binary/library - so the functions jump_fcontext, make_fcontext, ontop_fcontext are contained in the compilation unit(s)/library
the header fcontext.hpp tells the compiler the function signature if you call for instance jump_fcontext in a C++ file

> I also, have the following question:
>
> How does Boost.context actually associate jump_context with the assembly files. I mean, how does Boost.context know which assembly file to launch when jump_context is run.

the boost.context library contains only one compiled jump_fcontext function (compiled from assembler) - please note that C/C++ code is transformed into assembler by the compiler (you could see the resulting assembler code if you call `gcc/g++ -S ...`)

> It would be really great if you could point me to the file where this is actually being done. I know that we use *Jamfile.v2* and *architecture.jam* to select the assembly files, but I have not been able to understand how they get connected. The only thing that I see in *fcontext.hpp* is the functional definitions of the three functions

the compiled library already contains the symbols for jump_fcontext , make_fcontext, ontop_fcontext - the linker knows the function entries of those functions

---

Naveen Naidu <naveennaidu479@gmail.com>	29 April 2019 at 16:27
To: Oliver Kowalke <oliver.kowalke@gmail.com>

Thank you very much for the detailed explanation. It really helped me understand how Boost.Context actually works :)
P . Naveen Naidu

[Quoted text hidden]