x86_64 port and BSP

Google Summer of Code Program 2018 Project Proposal

Amaan Cheval
amaan.cheval@gmail.com
Thakur College of Engineering
Mumbai, India
+91 9833025795
IRC on freenode: amaan

Table of Contents

Project Abstract

The creation of a new x86_64 port and BSP to run on off-the-shelf consumer hardware, common and popular emulators, and development boards. This will replace the 32-bit i386 BSP and allow for further progress to be made on the new BSP, while abandoning legacy features in favor of features that can be built upon to take advantage of all that RTEMS has to offer.

Project Description

This project allows for RTEMS to support commonly used x86_64 consumer-grade processors, which should reduce the barrier to entry for newcomers, while at the same time ridding the project of legacy code for the i386 that RTEMS currently supports.

The project sets the basis for future development on the Intel x86-64 target, allowing it to eventually catch its RTEMS feature-set up to other architectures without unnecessary legacy baggage.

I chose this project because (1) I'm quite interested in studying computer architectures, and the x86 architecture in particular and (2) in embedded systems and operating system theory (scheduling, architecture, so on); this project meets that intersection.

Project Deliverables

- GSoC Timeline: https://developers.google.com/open-source/gsoc/timeline
- **May 14 (coding begins):**
  To ensure coding begins at this date, here are some of the preparatory tasks:
  - Have a working RTEMS development environment. I do already have a working development environment setup for RTEMS, including tools built for:
    - SPARC
    - ARM
    - i386
    - x86_64
    - RISC V
    - qemu-system-i386 and qemu-system-x86-64 (including compiled versions in case we need features from QEMU's master or ones behind compile-time flags)
  - Research and read about all prerequisites to confirm that plan is appropriate.
    - Read about UEFI
    - Read up on APIC and SMP system requirements
    - Read more of RTEMS' source and understand the contracts setup between a BSP, the SuperCore API, the executive itself, and user applications
    - Determine specifics of boot/init code requirements (protected mode, paging setup, video init, extended page tables, etc.)
  - Setup test environment
    - QEMU (or any other emulator used during development) will need OVMF running to test a UEFI-aware OS (done and documented)
    - Hardware; test a UEFI-aware OS or diagnostics image on real hardware to have a baseline for when we'll want to verify that the

x86_64 BSP will boot and initialize hardware - work with community members to get this environment setup and remotely accessible (such as iPXE and network booting)
        - Create Wiki documentation on setting this test environment up
    - Find existing sources for features we need
        - FreeBSD has a compatible license and likely already has some code for:
            - UEFI that we can use. Eg.
                - https://svnweb.freebsd.org/base?view=revision&revision=264095
                - https://wiki.freebsd.org/UEFI
            - Multiboot (optional, but it might be relatively simple to integrate)
            - APIC setup and SMP
                - See references

- **June 11-15 (Phase 1 Evaluation):**
    - Linker script for x86_64
    - A minimal port (stubs in cpukit's context-switching and interrupt setup code)
    - Beginning of the BSP boot initialization code
    - The x86_64 tools' issues should all be resolved (TLS, newlib support, etc. - no multilib support unless someone makes a great case for it)
    - All tests should be linkable with stub code for the port and BSP.
- **July 9-13 (Phase 2 Evaluation):**
    - UEFI boot should be complete (with multiboot depending on existing open-source code and ease of integration)
    - **(Partial)** Parts of context-switching (and basic ISR if required by tests below) (port)
    - **(Partial)** Video-based console driver (BSP)
    - **(Partial)** Idle thread based clock driver (BSP)
        i. BSPs for gdb simulators like `v850` and `sh` have `tcfg` lines to disable tests known to fail with this implementation; we can use the same for starters (based on Joel's comment)
    - Enough code to let `hello.exe` and `ticker.exe` tests pass after RTEMS boots
    - Basic documentation on how to setup and run these tests for x86_64 targets.
- **August 14-21 (Final Evaluation):**
    - Complete context-switching code (with SMP if time allows)
    - Complete ISR code (basic APIC setup, leaving I/O APIC / extended device interrupt routing out)
    - Complete COM1 support for console driver
    - Complete hardware-based clock driver
    - Complete benchmark timer driver (for timing test suite)
    - 90% of basic testsuite passing (excluding bonus features such SMP, ACPI, extended interrupt support through APIC, etc.) in at least a simulator, and potentially on hardware (based on community help)
    - The code should have gone through at least a few phases of cleanup through community / mentor reviews
    - Performance should be improved too - no more "development-only" hacks and WIP code in the required components
    - If time allows, SMP support should be underway too

- **August 22 (Final Results Announced):**
    - New documentation or updates to existing docs about
        i. Steps required to port or add a new BSP, with lessons learned, and documentation about current status of port and methods of reproduction
        ii. Future scope for growth for the x86_64 BSP
- **Post GSOC:**
    - Help fix bugs and potentially continue working on the BSP to add more BSP features, or work on improving developer experience (debugging capabilities, build process, tracing, logging, etc.) based on lessons learned from this project.

Proposed Schedule

March 12 - March 27 (Application Period)

Continue research into the prerequisites of having an x86_64 port - what tasks are essential, what kinds of discussions need community input, etc.

March 27 - April 23 (Acceptance Waiting Period / Research and Studying)

- Find gaps in x86_64 tools' feature-set by starting working on a stub port and BSP which will reveal gaps in toolset
- Continue to look at RTEMS' codebase and gain familiarity with the entire system
- Contribute patches for minor issues for more hands-on experience with RTEMS' review process and coding conventions
- Setup emulator with OVMF

April 23 - May 14 (Community Bonding Period)

- Fix issues that may exist in x86_64 tools (based on initial findings, there doesn't seem to be any work required, but if any pops up, we should handle it here)
- Reading FreeBSD's source for relevant features and the UEFI specification for things to be wary of
- Determining what features will be left for later (SMP support, ACPI, etc.) - more concretely hone in on what makes sense within the context of this summer project

May 14 - June 15 (First Phase)

- Bring x86_64 RTEMS tools up to speed if any issues still remain
- Look into ACPICA and how the required ACPI features can be kept in sync with the source
- Create the x86_64 stub source files based on no_cpu, and using the i386 source for reference (cleaning up anything that can be reused to be of an acceptable quality) - bring to a stage that all tests can be linked to the executive
- Reuse UEFI code from FreeBSD / write boot and initialization code (BSP) - determine the approach used to building the UEFI header.

Note: I will need **5 days off** (about one day a week) for exams during this period, which I

believe I'll make up for by working more per day / adding weekend work. I'll also make sure I let the mentors know about this when it's happening.

June 15 - July 13 (Second Phase)

- Complete boot / initialization code (BSP)
- Update cpukit context-switching code using setjmp/longjmp as guides, as much as is required for hello.exe (port)
- Add minimal ISR support if required by hello.exe (port)
- Implement simple console driver (using the easiest method to start with) and test with hello.exe (BSP) (shouldn't need ISR and may not even need complete context-switching support (floating-point context))
- Implement simple clock driver (idle thread) and test with ticker.exe (BSP)
- Stub documentation on testing x86_64 with QEMU (or other suitable emulator)
- Solicit reviews to confirm direction being taken

July 13 - August 14 (Third Phase)

- Complete context-switching and ISR code with as much APIC support as is required, confirm implementation (if we're doing well on time, the bonus task of SMP support can be completed here too since it's related). Regarding APIC support; the I/O APIC (which is required for external devices and PCI support for interrupts) will likely be left as a bonus feature.
- Have hardware-based clock driver, with counter support (`_CPU_Counter_read` function)
- Use counter support for benchmark driver (`./c/src/lib/libbsp/shared/timercpucounter.c`), to let us run timing tests in `./testsuites/tmtests`
- Test on hardware (community - Chris J has a Minnow Board, ITX atom board, and more; my primary desktop includes UEFI firmware and can be used if required)
- Add COM1 support and fix the `printk` dilemma for the console driver (switching video modes to take over serial device)

Bonus:

- SMP support:
  - Find what parts of this can be brought over from FreeBSD (see references for links to code)
  - Changes needed for boot; bootstrap processor initializes APIC, sends out SIPI (startup inter-processor interrupt) message to application processors
  - Changes to interrupt handling
  - Testing with `./testsuites/smptests`
- Leave stub code / TODO comments / create individual feature tickets (listing them in either the original ticket, or in a new "improve x86_64" ticket) for future features as apt (see future improvements below) - even implement some based on complexity and how we're doing on time
  - Documentation in rtems-docs if required
  - Update Tier level (rtems-source-builder and rtems-docs)

Future Improvements

- APIC support for non-trivial interrupt requirements (including I/O APIC and interrupt routing through ACPI or MP Tables)
- APIC prerequisite setup for SMP systems
- SMP support
- Support for booting in several forms (UEFI HDD, CD, multiboot, etc.)
- ACPI support

- libdl
- libdebugger

Continued Involvement

I'm particularly interested in two things, as I said earlier; studying various computer architectures, and also in embedded systems, and operating system theory.

I'd like to continue to learn from the RTEMS community - how design decisions are made, what efficiency requirements can be guaranteed, etc. I believe I can do this in the form of continuing to track the ongoings on the mailing list and submitting patches myself.

I think as I learn more, I'd like to contribute more to the core executive portions too, where portability, maintainability, and performance requirements are a lot more important.
I'd also be interested in diving into the specifics of another architecture's BSP (ARM and RISC-V are particularly interesting to me).

Eventually, I think I'd also be interested in being on the other side of the GSoC process, potentially mentoring future students, once I've learned and grown enough.

Conflict of Interests or Commitment

I've already spoken to my employer (part-time) about taking time off during GSoC, and they've been very supportive.
Besides that, I'll need 5 days off during the entire 3-month period for exams (including time to prepare for them!) - I've worked with my employer through exams in the past and have a pretty good gauge on how to not lose too much work time to them.

I *do* have a potential wrench-throwing issue: repetitive strain injury. The injury has been under control the past few months, allowing me to work 25-30 hour work-weeks relatively comfortably, but if it flares up, that may require a few extra days off. I'm free of commitments after GSoC and more than willing to see my project to completion afterwards if need be, spreading the work out over a longer period (even if it means failing the GSoC evaluations).

The code I write for RTEMS cannot be laid claims on either by my employer or my university.

Major Challenges foreseen

- A blind-spot in what the architecture needs for a specific feature; you can't know what you don't know, and the Intel x86 architecture has *a lot* of features one may not know of. I aim to skim significantly larger chunks of the Intel manual (especially volume 3; system programmer's guide) to combat this issue.
- Difficulties in testing / verifying implementation - initialization / boot code may work in the emulator, but not on hardware; this could be significantly time-consuming. I hope to read up on hardware debugging setups as I can as well to combat this as much as possible. (For eg. the Multiprocessor Initialization protocol differs on different models of the Intel family of processors.) The community will likely prove useful in this regard.

References

- https://devel.rtems.org/ticket/2898
- Intel architecture:
    - https://www.intel.in/content/www/in/en/architecture-and-technology/64-ia-32-architectures-software-developer-manual-325462.html
    - http://x86asm.net/articles/x86-64-tour-of-intel-manuals/
    - https://www.cs.cmu.edu/~410/doc/minimal_boot.pdf
- UEFI:
    - http://www.uefi.org/specifications
    - https://wiki.osdev.org/UEFI
    - https://svnweb.freebsd.org/base?view=revision&revision=264095
- APIC / Interrupts / SMP:
    - https://wiki.osdev.org/APIC
    - https://github.com/freebsd/freebsd/blob/master/sys/x86/x86/local_apic.c
    - https://github.com/freebsd/freebsd/blob/master/sys/amd64/amd64/apic_vector.S
    - https://people.freebsd.org/~jhb/papers/bsdcan/2007/article/article.html
    - https://forum.osdev.org/viewtopic.php?f=1&t=21745&start=0#p174219
    - https://www.cheesecake.org/sac/smp.html
- ACPI
    - https://github.com/acpica/acpica (Intel's reference implementation)

Relevant Background Experience

- I've been working on an open-source Intel x86 emulator for nearly a year now. This is the project I've been working on part-time as well, with paid work contributions being private for the time being. See Experience section later for more.
- As part of an intercollege competition, I've worked with embedded devices to make (1) a pizza delivery bot which can schedule multiple deliveries at once and partial work on (2) an incomplete self-balancing bot, involving heavy UART usage.
- RTEMS contributions:
    - https://devel.rtems.org/ticket/3328
    - https://devel.rtems.org/ticket/3331

Personal

I'm in the final year of my Bachelor's in IT (the closest degree to Computer Science they offered) at Thakur College of Engineering & Technology in Mumbai.

I've been programming since I was about 13, and I've toured a few domains in the time; web development, game development, backend web dev, machine learning, embedded systems, software security.

I find embedded systems enticing because they're so much closer to the hardware - abstractions are great, except most end up being leaky. I like knowing how the lowest layers of the hardware function because the context makes me a better programmer.

Besides, I've always dreamed of contributing to space research, and that's why finding RTEMS on the GSoC org list was so exciting to me! It aligns extremely well with my technical interests and personal motivations too!

Irrelevant, but I've also been learning German online for about 2 years.

Experience

Free Software Experience/Contributions:

- Some open source work on v86:
  - https://github.com/copy/v86/pull/162
  - https://github.com/copy/v86/pulls?q=is%3Apr+author%3AAmaanC+is%3Aclosed
- RTEMS contributions:
  - https://devel.rtems.org/ticket/3328
  - https://devel.rtems.org/ticket/3331
- Work on a Slack chatbot (for my former employer, Auth0, but in my personal time):
  - https://github.com/auth0/concierge-droid/pulls?q=is%3Apr+is%3Aclosed+author%3AAmaanC
- A simple SAML (identity management protocol) debugging tool (for Auth0, but in my personal time):
  - https://github.com/AmaanC/saml-idp
- A workshop on x86 assembly, and reverse engineering for a local OWASP meetup:
  - https://github.com/AmaanC/x86-reveng-pres
  - http://g.whatthedude.com/x86-reveng-pres/presentation/#/
- Other miscellaneous projects:
  - https://github.com/AmaanC?utf8=%E2%9C%93&tab=repositories&q=&type=source&language=

Language Skill Set

- C (semi-advanced)
  - v86 (we use Emscripten to compile C to WebAssembly) (~6 months of heavy usage, including patching existing projects like Wine and QEMU)
  - Reading source-code of open-source projects (Linux kernel, Radare, RTEMS)
- x86 assembly (advanced)
  - Several reverse engineering projects (crackmes, CTFs, etc.) and puzzles
  - v86 code
- JavaScript (Node.js - advanced, client-side - rusty)
  - Several canvas browser-based games
  - Backend for a real-time Android app (published)
- Python (rusty)
  - Several basic Machine Learning scripts
  - Binary exploitation scripts during CTFs
- Common Lisp & Emacs Lisp (beginner)

Related Research and Work Experience:

- Systems Programmer on v86 (an Intel x86 emulator) (~1 year)
  - Work goes into a private repo, not the public repo (eventually open-sourced, perhaps)
  - For starters, we ported most of the JavaScript modules into C so we could

compile to WebAssembly and get speedups
    - Multi-stage bootloaders for the emulator in x86 assembly as unit-tests (private).
    - Implemented significant portions of the emulator's MMX features.
    - Read significant portions of the Intel x86 developer manuals as part of my work.
    - Debugged several issues with OS's malfunctioning in our emulator (v86), including temporary workaround patches to Wine, and diving into portions of libc that malfunctioned.
    - Implementation of a Just-In-Time compiler (in C) to detect hot-code blocks in x86 assembly, hot-compile them to WebAssembly, and cache and reuse them, while accounting for cache consistency and consistent memory writes for self-modifying code
    - Numerous JIT optimizations (compiling basic blocks for common branch conditions, avoiding updates to EIP in cached blocks when not required (i.e. for non jump instructions which don't need EIP), etc.)
    - Instrumentation profiler custom-made for the project to benchmark and improve performance
    - Code coverage analysis engine that evaluates how many basic blocks in every "relevant" function are being covered by tests
    - Auto-generated instruction-specific unit tests using QEMU / real hardware as an oracle, extracting all relevant state (lots of scripting, assembly, and setup code)
- Developer Success Engineer (technical support) at Auth0 (~1 year)
    - Debug customer issues in a variety of environments - got good at asking the right questions and replicating unknown environments based on "problem symptoms"

Both jobs I've held have been work-from-home, with coworkers in several time zones. I've gotten good at communicating excessively, as is required when working remotely, and at being evaluated upon actual work done, not number of hours spent.

Reference Links and Web URLs (optional):

- http://whatthedude.com/
- https://github.com/AmaanC
- Old irrelevant personal blog: http://wiki.whatthedude.com/
- IRC: amaan on Freenode