Remove set_vector() across all Architectures and BSPs

Google Summer of Code Program 2025 Project Proposal

Sunil Hegde
Discord: NotSoDumb

University of Visvesvaraya College of Engineering
Bengaluru, India

Project Abstract

The `set_vector()` function has been deprecated due to improper pointer usage. This project aims to remove its implementations, prototypes, and usages across all architectures and BSPs. It will be replaced with `rtems_interrupt_handler_install()`, `rtems_interrupt_catch()`, `_CPU_ISR_install_raw_handler()`, or alternative methods, depending on whether the interrupt should be unmasked or handled using non-RTEMS ISRs.

Project Scope

Medium(approx. 175 hours)

Project Description

When an interrupt occurs (either hardware or software) in the OS, they are handled by Interrupt service routines (ISR) or in lay terms a method to handle interrupts. In RTEMS, the interrupt handler takes care of this job. It is important to note here that different architectures or BSPs use different methods to handle these interrupts and historically, In RTEMS, `set_vector()` with different implementations is used for this purpose.

`set_vector()` function:

Function prototype:

```
rtems_isr_entry set_vector(
  rtems_isr_entry     handler,  /* ISR routine */
  rtems_vector_number vector,   /* Vector number */
  int                 type      /* RTEMS or RAW interrupt */
);
```

Parameters:
- `handler`: The ISR to be installed for the specified interrupt vector.
- `vector`: The interrupt vector number where the ISR will be installed.
- `type`: Specifies the nature of the interrupt handler. A value of `0` indicates a "raw" handler that bypasses RTEMS interrupt management like `_CPU_ISR_install_raw_handler()`, while `1` indicates an RTEMS-managed handler like `rtems_interrupt_catch()`.

Return Value:
The function returns the previous ISR associated with the specified vector, allowing for restoration if needed.

However, the set_vector() function has been identified as unsafe and is slated for removal across all architectures and Board Support Packages (BSPs) in RTEMS because:

- Incompatible Function Pointers: It has been reported to cause warnings due to incompatible pointer types.
- Ambiguous Implementation: The implementation of set_vector() varies across BSPs, leading to inconsistencies and potential misconfigurations. For instance, the RTEMS BSP and Driver Guide outlines how set_vector() is responsible for installing an interrupt vector and invokes support routines necessary to install an interrupt handler as either a "raw" or an RTEMS interrupt handler.

A simple grep shows that set_vector() is at least used 60 times.

BSPs Using set_vector() to Install ISRs

The following BSPs contain instances where set_vector() is used to install ISRs, along with the files where the function is referenced:

- SPARC
  - erc32
    - bsps/sparc/erc32/console/erc32_console.c
    - bsps/sparc/erc32/start/setvec.c
    - bsps/sparc/erc32/start/bspsmp.c
    - bsps/sparc/erc32/include/tm27.h
    - bsps/sparc/erc32/include/bsp.h
  - Leon2
    - bsps/sparc/leon2/console/console.c
    - bsps/sparc/leon2/start/setvec.c
    - bsps/sparc/leon2/include/tm27.h
    - bsps/sparc/leon2/include/bsp.h
  - Leon3
    - bsps/sparc/leon3/start/setvec.c
    - bsps/sparc/leon3/include/tm27.h
    - bsps/sparc/leon3/include/bsp.h
  - shared
    - bsps/sparc/shared/gnatcommon.c
- Microblaze
  - microblaze_fpga
    - bsps/microblaze/microblaze_fpga/include/tm27.h
- Nios2
  - nios2_iss (uses non-rtems ISRs for raw interrupt)
    - bsps/nios2/nios2_iss/start/setvec.c
    - bsps/nios2/nios2_iss/clock/clock.c
    - bsps/nios2/nios2_iss/include/bsp.h
    - bsps/nios2/nios2_iss/btimer/btimer.c
- M68k
  - mcf5235
    - bsps/m68k/mcf5235/clock/clock.c
    - bsps/m68k/mcf5235/include/tm27.h
    - bsps/m68k/mcf5235/include/bsp.h
  - av5282
    - bsps/m68k/av5282/clock/clock.c
    - bsps/m68k/av5282/include/tm27.h
    - bsps/m68k/av5282/include/bsp.h
  - mcf5329
    - bsps/m68k/mcf5329/clock/clock.c
    - bsps/m68k/mcf5329/include/tm27.h
    - bsps/m68k/mcf5329/include/bsp.h
  - uC5282
    - bsps/m68k/uC5282/clock/clock.c
    - bsps/m68k/uC5282/include/tm27.h

        - bsps/m68k/uC5282/include/bsp.h
    - genmcf548x
        - bsps/m68k/genmcf548x/clock/clock.c
        - bsps/m68k/genmcf548x/include/tm27.h
        - bsps/m68k/genmcf548x/include/bsp.h
- Powerpc
    - bsps/powerpc/shared/vme/README.md
- Template structure for new BSPs in no_cpu/no_bsp
    - bsps/no_cpu/no_bsp/start/setvec.c
    - bsps/no_cpu/no_bsp/clock/ckinit.c
    - bsps/no_cpu/no_bsp/include/bsp.h
- Shared code across all BSPs
    - bsps/shared/dev/serial/mc68681.c
    - bsps/shared/dev/serial/z85c30.c
    - bsps/shared/start/setvec.c

Additionally, `set_vector()` is used in some core functionalities within the cpukit. It is present in this file:

- cpukit/libgnat/ada_intrsupp.c

To address the depreciation of `set_vector()` usage, instructions are provided by Sebastian Huber in Issue #4171. The approach suggests 4 steps:

1. Unmasking corresponding interrupts in `rtems_interrupt_entry_install()` or `rtems_interrupt_handler_install()` or `rtems_interrupt_catch()`
2. Replacing function call having (*,*, 1) with `rtems_interrupt_catch()` or `rtems_interrupt_handler_install()`.
3. Replacing function call having (*,*, 0) with `_CPU_ISR_install_raw_handler()`
4. Removing implementations of `set_vector()`

`rtems_interrupt_handler_install()` function installs a specified interrupt entry at a given interrupt vector. It allows for the installation of either a unique or shared interrupt handler, as specified by the options parameter.
`rtems_interrupt_catch()` establishes an ISR for the system by installing the RTEMS interrupt wrapper in the processor's Interrupt Vector Table and the address of the user's ISR in RTEMS' Vector Table.
`_CPU_ISR_install_raw_handler()` directly installs a hardware ISR into the processor's trap table for a specified vector, bypassing the RTEMS interrupt management layer to minimize overhead.

These replacements provide several advantages over the existing function:

- Documentation: The new functions are well documented, facilitating easier implementation and maintenance.
- Type Safety: They provide type safety, reducing the likelihood of warnings and errors during compilation.
- Uniformity Across BSPs: Their adoption promotes consistency across different Board Support Packages (BSPs), simplifying development and debugging processes.

The approach is straightforward and the changes can be implemented incrementally by tackling one BSP at a time. Unmasking corresponding interrupts requires a new helper function to be written to reduce redundancy. I have explored different ways to implement the helper function with respect to its file structure and for now it felt best to follow how set_vector() was implemented previously.

I am particularly interested in this project because I have a keen interest in understanding how interrupts are managed within operating systems, a topic I enjoyed learning during my coursework. Given that interrupt handling is a core functionality in operating systems, I am eager to learn its implementation within RTEMS.

Project Deliverables

- **June 2 (coding begins)**
  - I already have an RTEMS development environment set-up for SPARC BSPs and I plan to start with it.
  - Finish researching on topics like interrupts and how it is handled across RTEMS.
  - Already get familiar with RTEMS coding practices to start coding right away.

- **July 14-18 (Midterm Evaluation)**
  - Remove the functions from at least 60% of the code base, get it reviewed and merged incrementally with one BSP at a time.
  - Also look into which method would be best to declare and implement a helper function.

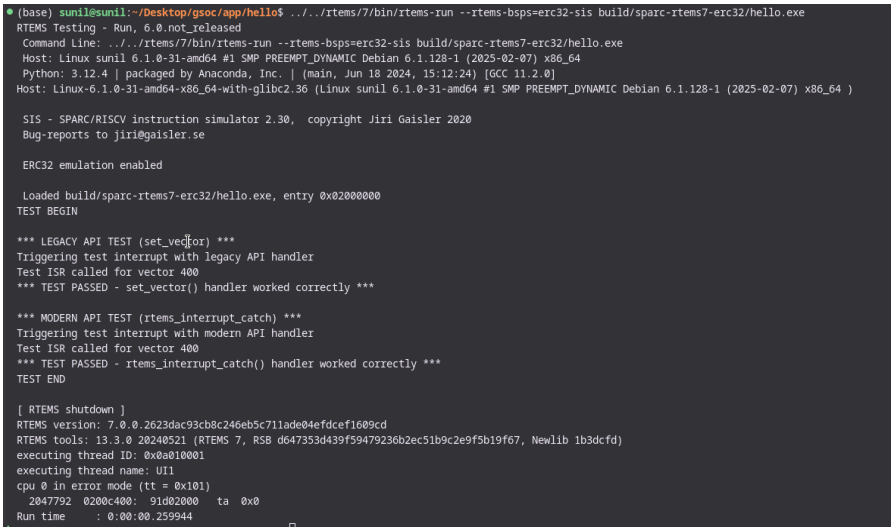
- **August 25- September 1 (Final Evaluation)**
  - Finish replacing the function in the entire code base with all the improvements and include all the suggested changes.
  - Finish cleaning up of the code and the documentation, fix formatting, recheck and remove if redundant code is present anywhere, remove confusion in the documentation by explicitly specifying that `set_vector()` is obsolete and lay out general replacement pattern for the function in user document and get all the changes approved for merging.

- **Post GSOC**
  - I will be maintaining this project and will definitely address issues if anything comes up in the future.
  - I plan to take up other tasks during GSoC outside of this project's scope(without compromising on the main project) like this issue or installing RTEMS on RaspberryPi3 and work on these well past GSoC as a contributor.

Proposed Schedule

March 24 - April 8 (Application Period)

- I already have set up my environment and have started working on the issue. I changed the code in sparc/erc32 BSP and submitted a draft MR for inputs, suggestions, and changes.
  Here is the link to the draft MR.
- I have created a simple test application which uses `set_vector()` and `rtems_interrupt_catch()` to install the same ISR to check if replacing the former with later gives out the same expected output. Here is a screenshot:



- I will focus on my proposal and improve it with suggestions from mentors.

April 8 - May 8 (Acceptance Waiting Period)

- Familiarize myself with installing and using multiple BSPs on my computer.
- Learning more about creating an RTEMS application and creating some tests specifically for this project.
- Research the use of `rtems_interrupt_handler_install()` and explore how `rtems_interrupt_handler_install()` provides a unified approach to handling both software and hardware interrupts, simplifying the interrupt management process and examine the potential of passing arguments to certain drivers using this function.

I will be having my Sem End exams during this time, so I will try my best to stay on track.

May 8 - June 1 (Community Bonding Period)

- I will learn how RTEMS interrupts work in detail referring to the documentation and other sources if necessary.
- I will discuss my proposed solution with my mentors and iteratively improve the implementation for 1 or 2 BSPs to understand the process and procedure.
- By this time I will also be sure of the benefits of `rtems_interrupt_handler_install()` over `rtems_interrupt_catch()` and `_CPU_ISR_install_raw_handler()` in different cases and how to use it.
- I plan to reach out to contributors who have worked on interrupts to learn how to test the changes, to detect and rectify if any new issues are created because of my changes.
- Making a roadmap at this stage would be best to stay on track and reach up to the expected goals.

June 2 - July 14 (First Half)

- I will start with removing all the `set_vector()` calls in the SPARC BSPs targeting parts of the shared code and one BSP at a time. I will then remove its implementation and declaration.
- The current tests are not detecting the warnings, so I need to create new tests to validate these changes. If an issue occurs, I will:
    - Analyze the failure logs to pinpoint the cause.
    - Compare with previous behavior to determine if the issue arises from the replacement approach or an underlying dependency.
    - Engage with mentors and maintainers to discuss possible fixes.

  By following these steps, I will ensure the stability of the code before submitting each MR.
- I will then move on to m68k and microblaze one after the other following the same steps.
- I will finish documentation concerning these changes and add new information if necessary.

July 20 - August 23 (Second Half)

- With all the previous changes in place, I will move on to nios2, powerpc(here a README file suggests to use `set_vector()`), no_cpu, and finally shared code if anything is left in there.
- After finishing work on all the BSPs, i will do the same in cpukit and in testsuites if uses are found in it.
- All this while I will be taking rough notes on the changes I made and this will be helpful in writing the documentation at this point.

Future Improvements

As this project aims to remove the set_vector() function, areas of improvement in the future can be better handling of unmasking and clearing the interrupt vectors. Other than this, improvement can be done in tests or the documentation.

Continued Involvement

Working on Operating Systems is interesting and I plan to explore memory protection improvements in RTEMS as a potential future project, given its complexity and importance. I plan to learn more about RTEMS with the help of the community and work my way to hopefully become a maintainer too.

Other Commitments

I'm a third-year engineering student with no major commitments besides my semester-end exams. My upcoming exams run from April 15 to the first week of May, which falls within the waiting period, so they won't be an issue. The next semester-end exams are in October, which also won't interfere. While I have two internal exams (each lasting two days), they won't affect my contributions, as my next semester has easier subjects, requiring less time off.

Eligibility

I confirm that I am eligible.

Major Challenges foreseen

- One of the major challenges will be installing the BSPs on my computer. It will take a lot of time to build a BSP and then test it later.
- Since the changes are being made to existing code, I need to be cautious to avoid unintended side effects. Modifying existing code is more challenging than writing new code, as it requires understanding the current implementation and preventing any unintended side effects.
- Initially, learning the code and navigating the project may take time, but I am confident I will improve with familiarity.
- This will be my first contribution to a large project, so I will need to invest extra effort in understanding the codebase structure.

References

- RTEMS Classic API Guide
- Operating System Concepts by Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin
- Issue #5215 and Issue #4171

Relevant Background Experience

- I have taken relevant coursework such as Computer Organisation and Architecture, Microprocessors and Microcontrollers, and Operating systems.
- During my last semester, I started building an OS during my Operating Systems coursework. I developed a bootloader and a simple kernel using x86 assembly and C, gaining experience with QEMU and other essential OS development tools. The project is available on my GitHub.
- Currently, I'm working on a side project to emulate the 6502 processor in C to deepen my understanding of hardware. This was inspired by my Microprocessors coursework, where I studied the 8086 processor, x86, and ARM assembly.
- I have learnt a lot about GitLab and can work my way around it easily.

Personal

I am a third-year Information Science and Engineering student at UVCE, Bengaluru. Over time, I have explored various domains, including web development, Android development, machine learning, competitive programming, and IoT. However, I have come to realize that my true interest lies in understanding computers at a fundamental level.

I started with assembly language, gradually shifting towards hardware-software interaction. While my coursework has provided valuable insights, I prefer a hands-on approach, which has deepened my interest in low-level programming.

As a space enthusiast, I was fascinated to learn that Perseverance runs on an open-source OS. While exploring different types of operating systems, I came across RTOS and eventually discovered RTEMS.

Experience

Free Software Experience/Contributions (optional):

- I have tried to fiddle with the linux kernel when I was having issues with my WiFi card on my computer. This helped me to learn a lot about the Linux kernel's structure.
- As mentioned I built a basic bootloader for my os and here is the link.
- This will be my first open source project.

Language Skill Set

- C and C++: Intermediate
- Java: Beginner
- Javascript and Typescript: Intermediate
- Python: Advanced
- x86 Assembly: Beginner

Reference Links and Web URLs (optional):

- My GitHub page
- My Blog