Out-Of-Process Execution for Clang-Repl

Gsoc 2024 by Sahil Patidar
Mentor : Vassil Vassilev, Matheus Izvekov

Project Final Summary

Introduction to Clang-Repl

- An interactive Just-In-Time (JIT) C++ interpreter that allows users to write, compile, and run code interactively.

Challenges in Current Design

- Resource Consumption: Heavy system usage, limiting performance on low-power devices like embedded systems.

- Instability: User code crashes can terminate the entire Clang-Repl session.

Project Objectives

Out-of-Process Execution: Transition Clang-Repl to an out-of-process execution model, separating the execution of code from the main Clang-Repl infrastructure. This will reduce the resource footprint on the main application and make it suitable for devices with limited resources.

Summary of Accomplished Tasks

Add Support for Out-of-Process Execution.

- Implement out-of-process execution for improved resource management and stability in `Clang-Repl`.
- **Solution:** Utilized `ORC JIT's` remote execution capabilities with `llvm-jitlink-executor`.
- **New Command Flags:**
  - **`--oop-executor`:** Launches the JIT executor in a separate process.
  - **`--oop-executor-connect`:** Connects Clang-Repl to an executor process over TCP.
- **Outcome:**
  - Enabled **isolated code execution**, reducing resource usage and enhancing stability by preventing crashes from terminating the entire Clang-Repl session.

Incremental Initializer Execution.

- Added `dlupdate` function to the ORC runtime for incremental execution of new initializers, improving efficiency in REPL environments.

- **Challenges with `dlopen`:**
  - Handles everything from initializers to code mapping, leading to inefficiencies.
  - Runs all initializers each time, even if only new ones are needed.

- **`dlupdate` Key Advantage:**
  - Executes **only new initializers**, streamlining interactive sessions.

- **Outcome:**
  - More efficient and focused initializer execution, especially beneficial for **interactive C++ sessions** in `clang-repl`.

Push-Request Model for ELF Initializers.

- Introduced a **push-request model** to handle ELF initializers for each **JITDylib**, improving management compared to the previous approach.

- **Challenges with ELF:**
  - Initializers were erased after being invoked, causing issues with re-running `dlopen`.
  - No ability to register or retain initializers for future executions.

- **Key Functions Added:**
  - **`__orc_rt_elfnix_register_init_sections`:** Registers the ELF initializer's for the **JITDylib**.
  - **`__orc_rt_elfnix_register_jitdylib`:** Registers the **JITDylib** with the ELF runtime state

- **Outcome:**
  - Enhanced efficiency and reliability in managing ELF initializers, enabling **out-of-process execution** in **clang-repl** for ELF targets.

Auto-loading Dynamic Libraries in ORCJIT (On-going)

- Enhance **ORC JIT** by implementing auto-loading dynamic libraries for improved symbol resolution.

- **Key Features:**
  - **Global Bloom Filter:** Skips unlikely symbols to speed up searches.
  - **Automatic Library Search:** Expands symbol resolution to unloaded libraries if not found in loaded ones.

- **How It Works:**
  - The JIT first checks loaded libraries.
  - If unresolved, it searches unloaded libraries, updating the bloom filter with their symbol tables.
  - Unresolved symbols are tracked and excluded from future searches.

- **Outcome:**
  - Efficient symbol resolution for both loaded and unloaded dynamic libraries by cutting down on unnecessary lookups.

Key Achievements

- Successfully integrated **Out-of-Process Execution** for Clang-Repl.
- **Improved stability:** Isolated user code crashes from the main Clang-Repl process.
- **Enhanced ELF and Mach-O runtime support:** Incremental initializer execution and dynamic library handling for **clang-repl**.

Conclusion & Future Work

Conclusion

- The project successfully integrated out-of-process execution into clang-repl, improving stability and performance.

Future Work

Crash Recovery and Session Continuation:

- Investigate and develop ways to enhance crash recovery so that if something goes wrong, the session can seamlessly resume without losing progress. This involves exploring options for an automatic process to restart the executor in the event of a crash.

Automatic Library Loading in ORC JIT:

- Wrap up the feature that automatically loads libraries in ORC JIT. This will streamline symbol resolution for both loaded and unloaded dynamic libraries by ensuring that any required dylibs containing symbol definitions are loaded as needed.

Thank You!