Out of process execution for Clang-Repl

Name: Sahil Patidar
Email: sahilpatidar60@gmail.com
Github: https://github.com/SahilPatidar
Mentors:

- Vassil Vassilev
- Matheus Izvekov

Size of Project: medium **or** large**.**

1. Project Description:

Clang, a key component of LLVM, supports various languages and is designed as a library, fostering a compiler-assisted ecosystem. The relatively accessible Clang codebase, coupled with LLVM's JIT advancements, enables innovative C++ processing methods. However, challenges like incremental compilation and integrating compile/link time optimizations persist.

Incremental compilation processes code in chunks, building a growing translation unit and enabling efficient interpreters. Clang-Repl, using the Orcv2 JIT infrastructure within the same process, exemplifies this concept. Yet, this design has limitations for resource-constrained devices and can lead to process-wide crashes on user code failure.

This project aims to transition Clang-Repl to an out-of-process execution model to address these issues.

This project aims to enhance Clang-Repl, an interactive C++ interpreter, by transitioning it from an in-process to an out-of-process execution model. This shift will address two key limitations of the current design:

1. **Resource Constraints:** The current in-process model requires devices with sufficient resources to host Clang and the Just-In-Time (JIT) infrastructure. This limits Clang-Repl's usability on resource-constrained platforms like Arduino Due.
2. **Crash Resilience:** In-process crashes within user code can bring down the entire Clang-Repl process, hindering stability and user experience.

Proposed Solution:

This project proposes migrating Clang-Repl to an out-of-process execution model. This will establish a separation between the executor and the Clang-Repl infrastructure, leading to several benefits:

- **Lightweight Execution:** By isolating Clang-Repl in a separate process, the resource footprint on the main application will be reduced, enabling its use on devices with limited resources.
- **Improved Stability:** User code crashes will be contained within the Clang-Repl process, preventing them from affecting the main process. This will enhance the overall reliability and robustness of the system.

Expected Outcomes:

1. Implement out-of-process execution of statements in Clang-Repl.
2. Demonstrate Clang-Repl's capability to support some of the ez-clang use-cases.
3. Research approaches to restart or continue the session upon a crash.
4. As a stretch goal, design a versatile reliability approach for crash recovery.

Technical Skills: C/C++, LLDB, Familiar with LLVM IR, x86, Git

2. Implementation Plan:

A. Leveraging Existing RPC Utilities

To establish out-of-process execution, we'll utilize the RemoteJitUtils RPC utilities. These utilities provide a foundation for creating a separate process (executor) and enabling communication between Clang-Repl (main process) and the executor. We'll explore two communication options: pipes and TCP sockets.

1. Pipe-Based Communication

- Inter-Process Communication (IPC) Setup: We'll create a pipe for data exchange between Clang-Repl and the executor.
- Forking a Child Process: will fork a child process using the fork system call. This creates a copy of the current process, allowing independent execution in the child.
- Executor Process Configuration: Within the child process:

- We'll construct arguments for passing the pipe file descriptor to the llvm-jitlink-executor tool using functions like execvp.
- The child process will then call execvp to launch the llvm-jitlink-executor, replacing itself with the executor process. The llvm-jitlink-executor will utilize the provided pipe file descriptor for communication with Clang-Repl.

2. TCP Socket-Based Communication

we can potentially explore using TCP sockets for communication. This might be suitable for scenarios where processes reside on different machines or when more flexibility is required.

B. Checkpoint-Based Restoration

To address potential crashes in the executor during user code execution, we'll investigate integrating the CRIU (Checkpoint/Restore in Userspace) feature into Clang-Repl.
The approach will involve creating checkpoints of the executor process every time a new user statement is about to be executed. This checkpointing will allow for restoration in the event of a crash.
CRIU commands will be executed in a separate process from Clang-Repl, utilizing the executor process ID for targeted checkpoint management.

Timeline: The expected timeline of the project is as follows:

- **Week 1:**

  Task: Design and implement out-of-process execution using existing RPC utilities (RemoteJitUtils).

- **Week 2:**

  Task: Complete the out-of-process execution interface for Clang-Repl.

  Deliverable: Clang-Repl out-of-process execution interface implementation.

- **Week 3:**

Fix bugs and address any issues encountered during testing. Address any delayed tasks from previous phases.

- **Week 4:**

  Task: Modify ORC runtime to support `dlupdate` for efficient `JitDylib` initialization in the out-of-process context (focusing on MachO platform for initial proof-of-concept).

  Deliverable: Modified ORC runtime code with MachO-specific `dlupdate` support.

- **Week 5:**

  Task: Develop a plan for implementing `dlupdate` support for additional platforms (ELF, COFF).

  Deliverable: dlupdate support for various platforms.

- **Week 6:**

  This week, we will focus on performance studies and benchmarking tests for the out-of-process execution in Clang-REPL. The goal is to evaluate the efficiency and effectiveness of the implemented changes, identifying any performance bottlenecks and areas for improvement.

  Deliverable: Benchmark results comparing performance with traditional in-process execution on various code samples.

- **Week 7:**

  Research and evaluate crash recovery options. Start planning the implementation of crash recovery mechanisms.

- **Week 8:**

  Begin implementing CRIU features for clang-repl to enable executor restoration after a crash.

  Develop unit tests for the CRIU integration.

- **Week 9:**

Complete the implementation of CRIU integration with clang-repl.

Conduct comprehensive testing of the crash recovery functionality.

- **Week 10-11:**

  Task: Test and benchmark the out-of-process execution functionality with different types of code to identify and fix performance bottlenecks.

  Deliverable: Benchmark results comparing performance with traditional in-process execution on various code samples.

- **Week 12:**

  Task: Document the entire project, highlighting benefits, challenges, and design decisions.

  Deliverable: Comprehensive technical documentation explaining the implemented functionalities (out-of-process execution, ORC runtime support, optional crash recovery) and user instructions for leveraging the new features.