Out-of-process execution for clang-repl

Lev E. Chechulin
l.e.chechulin@gmail.com

Abstract

This project aims to add out-of-process execution to clang-repl.

Motivation

The Clang compiler is part of the LLVM infrastructure, supporting languages such C/C++ and ObjC/ObjC++. The clang-repl tool utilizes clang and LLVM JIT API to enable to work with C++ in a more ”pythonic” way - with a REPL loop. However, the current design, where code is executed in the same process, brings benefits such as efficiency and ease of implementation but also has drawbacks, for example a crash in user code will crash the entire process, making clang-repl more difficult to use and reducing its reliability. Another problem is that the existing design cannot be used on devices with insufficient computing power to host entire infrastructure.[2]
The main goal of this project is to transition clang-repl to an out-of-process execution model and address both of these issues.

More about JITting

Clang, like other *ahead-of-time* (AOT) or *static* compilers, works in phases; that is source code is compiled to some other form and execution happens later. *Just-in-time*(JIT) compilers on the contrary are mixing together compilation and execution.
Clang-repl is build on top of the ORCv2 API to generate code on-the-fly. ORCv2 is a rich *orthogonal* API where different parts are not depending on each other. On the whole, the process can be narrowed down to this: generate machine code for given source code, allocate memory for it and run.

Possible approach

In general the implementation can be narrowed down to two main goals: executing JITted code in a separate process and implementing crash recovery/restart. Abstracting away the details and some utilities, the main work of clang-repl is done in:

```
    ClangRepl.cpp:127 auto Err = Interp->ParseAndExecute(*Line)
```

The first milestone will be that the new process is created and code executed there.

As of now, llvm repository has already some tools to implement external executors. Orcv2 provides useful utilities, namely `SimpleRemoteEPCUtils` [1]which has an API for memory allocations in EPC context, a memory manager, dynamic libraries manager and debugging tools. Stefan Gränitz demonstrated use of these in `RemoteJITUtils`. However, the existing `SimpleRemoteEPC` might need further development to be used in clang-repl, especially for communicating back errors.
The second milestone is behaviour in case of a crash. The challenge here is to restore the state after the crush with minimum cost. For instance, checkpointing will slow down the entire process - like ”stop the world” in garbage collection. Another approach might be to introduce scratch and stable areas: new code will be in a temporary execution space where the effects of the code can be isolated. Once the code in the scratch area executes sucesfully without errors and crashes, its effects (e.g., variable mutations, loaded modules) could be committed to the ”stable” area, the main session state. In event of a crash, the system would discard the scratch area and revert to the last known stable state, effectively isolating the crash impact and preventing it from corrupting the session. In this approach state commit mechanism should be carefully implemented and performance impact minimized
There is a draft pull-request implementing out-of-process execution that can be used as a reference in this project. It should be noted, that the author modified ELFNix platform to allow JITDylibs to be reinitialized through the `"__orc_rt_jit_dlupdate"` function, since clang-repl incrementally adds static initializers to a single JITDylib.

Timeline

- May 1 - 26
  *Community Bonding Phase.*This phase should be used for extensive preliminary work, i. e. for extensive design and API discussion, proof-of-concepts etc.
- May 27 - June 22
  Focus will be on enabling code execution in another process in clang-repl. In the best case, to achieve working remote executor on linux and passing tests in this phase.
- June 23 - July 14
  This period is dedicated to work on the problem of session restart/continuations in case of a crash.
- July 15 - July 29
  This time is reserved for implementing support on other platforms.
- July 30 - Aug 11
  The goal of this phase is to ensure that some of the ez-clang use-cases can be supported. Troubleshooting if needed.
- Aug 12 15 - Aug 19
  Final week.

This timeline can be changed, especially after the first 3 weeks, depending on the progressing speed, the number of issues arised and other factors.

References

[1] URL: `https://github.com/llvm/llvm-project/blob/release/18.x/llvm/examples/OrcV2Examples/LLJITWithRemoteDebugging/RemoteJITUtils.cpp` (visited on 03/29/2024).

[2] Stefan Gränitz. *ez-clang: experimental C++ REPL for bare metal.* URL: `https://compiler-research.org/meetings/#caas_10Mar2022` (visited on 03/29/2024).