Shared Memory Based JITLink Memory Manager

- Google Summer of Code 2022 with LLVM Compiler Infrastructure (Project)
- Student: Anubhab Ghosh (GitHub)
- Mentors: Lang Hames, Vassil Vassilev, Stefan Gränitz

Introduction

LLVM provides Just In Time compilation APIs that are used by a number of open source projects (e.g. Julia, Cling, Postgres, LLDB). LLVM's JIT compilation APIs are based on the idea of "JIT Linking", parsing and linking a relocatable object file in one process and making them executable in another target process. These processes may be the same process, different processes on the same machine, or even different processes on different machines. To facilitate this LLVM's JIT APIs provide the `ExecutorProcessControl` interface to abstract calls to the target process (implemented via IPC/RPC in the cross-process case), and the `JITLinkMemoryManager` interface to handle paired memory allocation in the linker and the target processes. Depending on the JIT configuration that the client chooses, efficient memory management schemes may include direct allocation (when running in the same process), shared memory (when running in different processes on one machine), or RPC calls (when running in different processes across machine boundaries).

Goals

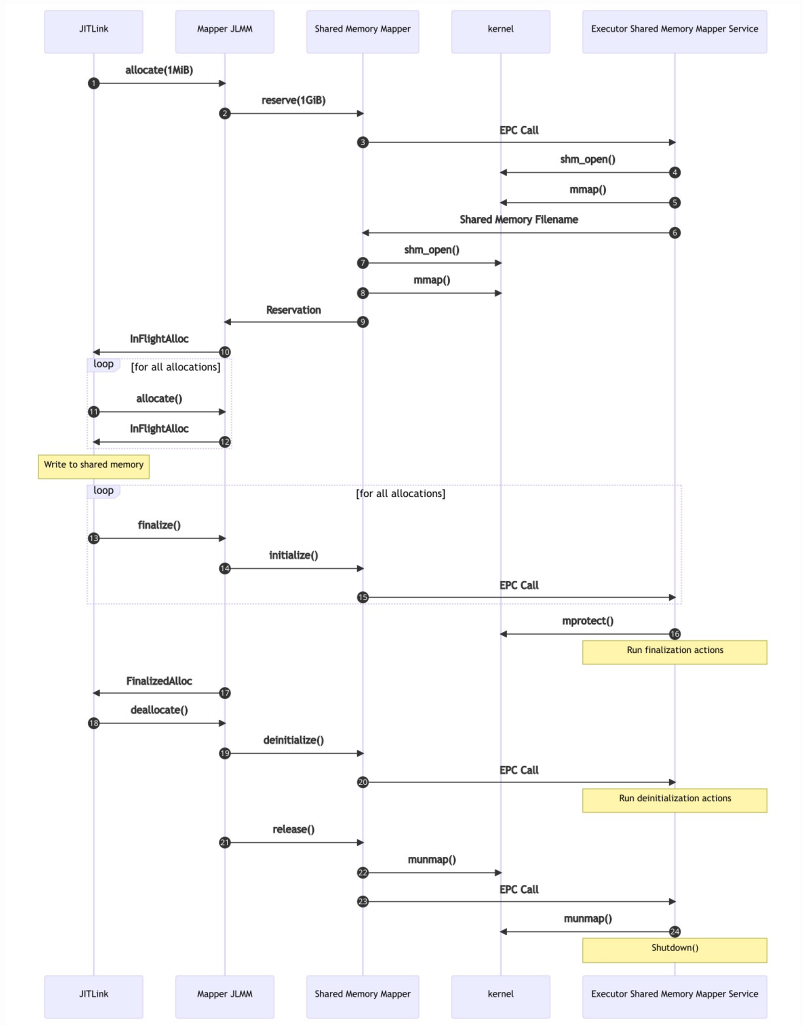
- When running out-of-process but on top of the same physical memory, implement support for shared memory-based transport of the generated code. The generated code will be directly written to shared memory regions. This should allow for faster transport.
- Implement a slab-based allocator that reserves large blocks of address space upfront in the executor process and allocates memory from that region. This guarantees that all allocations are close together in memory and meet the constraints of the default code model allowing the use of outputs from regular compilers. Without this, clients may be at risk of relocation-out-of-range errors if memory for different objects is allocated too far apart.

Design and Implementation

- `orc::MemoryMapper` interface: This is an interface to perform memory allocation, deallocation, setting memory protections etc. that handles most platform-specific operations. This abstraction allows us to decouple the transport for generated code from heap management making it simple for clients to use different transport mechanisms. (D127491)

    - `orc::InProcessMemoryMapper`: This implementation is used when running code in the same process where the JIT is running and uses `sys::Memory` API. (D127491)
    - `orc::SharedMemoryMapper`: This implementation is used when transferring code to a different executor process and uses POSIX or Win32 shared memory APIs. (D128544)
- `orc::MapperJITLinkMemoryManager`: This class implements the `jitlink::JITLinkMemoryManager` interface and handles all allocations within a slab. (D130392)
- Memory coalescing to join two consecutive free ranges and reuse them. (D131831)
- `llvm-jitlink` tool integration:
    - `MapperJITLinkMemoryManager` with an `InProcessMemoryMapper` is used by default when executing the code in the same process as the JIT. (D132315)
    - `MapperJITLinkMemoryManager` with a `SharedMemoryMapper` can be optionally used when `--use-shared-memory` is passed. (D132369)

The following diagram shows a possible interaction between them.



Sequence Diagram

Conclusion and Future Work

We observed a very low speedup (~4%) when using shared memory combined with the slab allocator but we expect to see more for updates to JIT'd code as the target memory can be directly accessed by the controller process now. The implementation was benchmarked on small projects like C-ray and the Python interpreter. Limitations of the current JIT implementation prevented us from testing on larger projects like Clang but we hope to resolve this soon and try the tests on larger programs.

▼ Some of the other smaller commits for reference

- [Orc] Disable use of shared memory on Android
- [Orc] Reorder operations in ExecutorSharedMemoryMapperService shutdown
- [Orc] Properly deallocate mapped memory in MapperJITLinkMemoryManager
- [Orc] Actually save the callback
- [Orc] Only unmap shared memory in controller process destructor
- [Orc] Provide correct Reservation address for slab allocations
- [Orc] Improve deinitialize and shutdown logic
- [Orc] Take offset inside slab into account in SharedMemoryMapper