CMake 立大功：glibc 更新引出的陈年旧案

XieJiSS (Pan RZ)

PLCT Arch RISC-V Team

2022.4

Contents

The Problem

__atomic_* 报错

- Tips：语境为 riscv64gc；编译器为 gcc，提到的所有问题在 clang 中都不存在
- `/usr/bin/ld: foo.so: undefined reference to `__atomic_exchange_1'`
  `collect2: error: ld returned 1 exit status`
- 以往的修复方式：-pthread
- CMake: set(THREADS_PREFER_PTHREAD_FLAG ON) 会使得 -lpthread 变成 -pthread

Stack Overflow

-pthread tells the compiler to link in the pthread library as well as configure the compilation for threads.
Using the -lpthread option only causes the pthread library to be linked - the pre-defined macros don't get defined.

`__atomic_*` 报错 (Cont.)

- 今年（2022）年初我们发现一些使用 CMake 控制编译流程的软件包重新出现这个报错
- 使用黑魔法：`export CC="gcc -v"`，发现 `-pthread` 从参数中消失
- 怀疑是 CMake 的 bug
- `Kitware/CMake` commit `68285bc8a91`: FindThreads: Honor THREADS_PREFER_PTHREAD_FLAG when pthread is found in libc（被 revert 了，后面会说）

`__atomic_*` 报错 (Cont.)

- 手动编译出 target branch 中最新的 3.23.0-rc5，`-I` 传给某个出现该报错的包 `mongo-c-driver`
- 报错没有消失
- 怀疑 CMake 的修复有问题，没有起作用（其实并不是，后文会说到）
- 研究 CMake 源码

Debug & Research

研究 CMake 源码

- pthread 相关检测的核心是 FindThreads.cmake 模块
- 其中可以看到 PTHREAD_C_CXX_TEST_SOURCE，核心代码如下：

```
1 pthread_t thread;
2 pthread_create(&thread, NULL, test_func, NULL);
3 pthread_detach(thread);
4 pthread_cancel(thread);
5 pthread_join(thread, NULL);
6 pthread_atfork(NULL, NULL, NULL);
7 pthread_exit(NULL);
```

- 将其复制到 test-pthread.c，发现可以由 gcc 直接编译出二进制文件，无需 -pthread flag。
- 但以前就需要这个 flag，发生什么事了

研究 glibc

- 调查发现原本由 libpthread 提供的功能自从 glibc 2.34 起已经集成进入 libc

glibc 2.34 Release Note

...all functionality formerly implemented in the libraries `libpthread`, `libdl`, `libutil`, `libanl` has been integrated into `libc`. New applications do not need to link with `-lpthread`, `-ldl`, `-lutil`, `-lanl` anymore. For backwards compatibility, empty static archives `libpthread.a`, `libdl.a`, `libutil.a`, `libanl.a` are provided, so that the linker options keep working.

- 换言之原本需要 `-lpthread` 或 `-pthread` 的代码，现在不再需要额外的 flag 即可编译
- ……但是 `__atomic_*` 也是「原本需要 `-pthread`」的代码，去掉 `-pthread` 后仍然报错

研究 -pthread

- 首先，照着报错搞个测试样例出来：

```c
// test1.c
int main() {
    char u = 0, v = 1;
    __atomic_exchange_n(&u, &v, __ATOMIC_RELAXED);
}
```

- 尝试不带 pthread 编译，报错。添加 -pthread 后编译成功，使用 ldd 查看其依赖库：
- `libfakeroot.so => /usr/lib/libfakeroot/libfakeroot.so`
  `libc.so.6 => /usr/lib/libc.so.6`
  `/lib/ld-linux-riscv64-lp64d.so.1 =>`
  `/usr/lib/ld-linux-riscv64-lp64d.so.1`
  `libatomic.so.1 => /usr/lib/libatomic.so.1`
- 发现其实链接的是 libatomic

研究 -pthread (Cont.)

- 要想知道为什么 -pthread 会链到 libatomic 上，就需要看 gcc spec
- 执行 `gcc -dumpspecs | grep pthread`，看到如下输出：
- `%{pthread:-lpthread} %{shared:-lc} %{!shared:%{profile:-lc_p}%{!profile:-lc}} %{pthread:--push-state --as-needed -latomic --pop-state}`
- 不难发现其中有 `--as-needed -latomic`
- 换言之，此前 `-pthread` 维修方案一直在利用副作用，本质上是 `-latomic`，随后由 libatomic 来提供 `__atomic_*`。

研究 atomic

- 真的是这样吗？
- 尝试另外的测试代码：

```
1 // test1.c
2 int main() {
3     int u = 0, v = 1;  // 这里从 char 修改为 int
4     __atomic_exchange_n(&u, &v, __ATOMIC_RELAXED);
5 }
```

- 用 gcc 直接编译，发现无需 `-latomic` 或 `-pthread` 也编译成功。ldd 查看依赖库，发现并不依赖 libatomic（当然，也不依赖 libpthread）。
- 继续测试其它位长，结果表明 `uint8_t` `uint16_t` 和 `__uint128_t` 均需要 libatomic，而 `uint32_t` 和 `uint64_t` 则不需要

研究 gcc

- gcc 似乎（目前）只支持给 32 位和 64 位的 atomic value 生成 inline code，其它的都会调用 libatomic 中的对应函数。
- 通过 `gcc -S` 可以看到对应的汇编：
- `call __atomic_exchange_1@plt`
- 溯源：`https://github.com/riscv-collab/riscv-gcc/issues/12`
- 通过该 issue，我们得知：gcc 不支持直接生成 subword atomic 的 inline asm（其实是只支持 word/double word，对应 `lr.w/sc.w` 和 `lr.d/sc.d`，但我并不确定 gcc 的 atomic 是不是通过 lrsc 实现）
- 但是该 issue 被 commit 2c4857d098 close 了，应当已被修复才对？

```
1 #define LIB_SPEC GNU_USER_TARGET_LIB_SPEC \
2   " " LD_AS_NEEDED_OPTION " -latomic "
      LD_NO_AS_NEEDED_OPTION \
```

研究 gcc (Cont.)

- git blame 发现，在这个 commit 之后，spec 还有修改：被修改为只对 `-pthread` 生效。这又是为什么呢？
- 一种可能是有的 linker 不支持 `--as-needed`，那样的话就会变成（fallback 到）不管有没有用到 atomic 都会被链 libatomic，存在 overhead
- 至于为什么放在 `-pthread` 的 spec 里，可能是因为当时做这个修改的人觉得 `-pthread` 和 atomic 绑定了吧，哈哈
- 随便举个反例：SIGNAL handler

Solution

Solution

- 解决方案和你想在什么程度上解决这个问题强相关
- 对于单个 package 层面而言，不追求优雅，可以直接 `export CFLAGS="$CFLAGS --as-needed -latomic"`，CXXFLAGS 类似。如果追求优雅，可以 patch 掉 Makefile 或者尝试通过 configure script 指定，等等。
- 对于多个 package（例如：都使用了 CMake），如果想采用一劳永逸的修复方案，可能需要编写 CMake 模块。目前我们采用的方案是：参考 LLVM 的 `CheckAtomic.cmake` 并将 subword atomic 加入了检测源码，最后在 `CMakeLists.txt` 中引入。

Solution (Cont.)

- 对于上游的单个项目，他们可能会需要考虑 users 使用非 gcc 的编译器，例如 clang，并且采用了不兼容 libatomic 的技术栈（如 compiler-rt）。故他们可能更倾向于依赖 `-pthread` 的副作用，`-pthread` 在 clang 里是不会链到 libatomic 的。
- 对于工具链上游，他们要考虑的情况更多。例如，某个叫 XL 的 linker 不支持 `-pthread`，所以 CMake 回滚了前面提到的修复 commit，才导致我们手动编译出 3.23.0-rc5 并替换 3.22.0 后仍然遇到这个错误。
- 最一劳永逸的办法当然是在 gcc 里添加 subword atomic inline 的支持。我们在 riscv-gcc 开了 issue #337 跟踪这一问题，并且已经参与到对应的 gcc patch（`gcc-patches@gcc.gnu.org/msg281336.html`）的 review 过程中。最近得知该 patch 只包含 `__atomic_fetch_add` 的支持，或许是一个好的开始。
- gcc 最终支持 subword atomic 后，`-pthread` 的 spec 大概率也会随之修改。我们已经在视频会议中将这个问题告知 gcc dev

Additional Info

- 需要注意的是，目前到 libatomic 的调用导致 subword atomic 在 gcc 中不是 always lock free 的。对应的 macro 如 `ATOMIC_BOOL_LOCK_FREE` 在 gcc 中目前被 define 为 `false`，类似地 `std::atomic<bool>::is_always_lock_free` 也是 `false`。这可能导致一些包的 `static_assert` guard 报错。
- 如果想尝试手动修复，目前可以暂时将 bool 类型 patch 成 int 类型，但需要阅读比较多的包代码，工作量较大。

Thank you!