LinkLab 报告

Part A: 思路简述

1. 核心方法的流程：核心方法基本上是根据注释的顺序。
    1. 先按照任务六的要求分开不同的section，遍历不同FLE文件中的同一个section。这一次遍历可以同时完成注释的一二部分，即收集段和处理符号。按照符号处理的需求，我把符号另存在了两个map即Global和Local symbol中。
    2. 在收集了所有重定位信息后就可以进行重定位操作，不同section的重定位都在自己section内，所以可以分开section对每个section内的重定位项进行操作。重定位完成后按照第四部分的要求处理返回的fle文件即可。
2. 关键数据结构的设计：
3. 使用了unordered_set来处理所有的section name：因为section name不能有重复项；
4. 使用map<std::string, Symbol> 来处理Symbol：因为使用键值对能够保证Global Symbol不会出现重复的情况，而且键值对可以根据键快速找到Symbol，根据键的变化替换Undefined项，从而可以方便的查看是否有没有被定义的Symbol。之后仿照ELF的处理方式，把所有的Local Symbol后面都加上了他的来源，防止重复。
5. reloc_info是一个用于标识重定位项信息的数据结构。这个数据结构考虑到使用比较灵活，可能需要根据任意几个数据项查找，所以使用struct。

Part B: 具体实现分析

符号解析实现

由于是先按照section遍历，后按照obj遍历，因此先找到这个obj中本section的符号。符号主要分三种：Global，Local和其他。使用Switch分类这三类，其中：Global出现两次会符号冲突，没出现就新建一项，出现是weak或者undefined就可以替换成global类型；出现Local就按照section内容和obj信息进行新建Local（Local不可能重复）；出现其他如果没出现在Global中就新建项，否则看出现过的项是不是Undefined，是就替换，否则跳过。

关键优化主要有使用字符串连接的方式使Local类型能够只保存一个字符串，同时能和Global类型的Symbol进行相似的处理（因为数据结构相同）；Switch可能能稍微加快跳转速度。关键的错误是重复的Global符号，能够在出现两次的时候输出runtime error。

重定位处理

支持所有的四种重定位。重定位因为只有重定位的Symbol名字，而且Local的Symbol名字可能重复，因此需要先从Relocation Information中找到对应的符号是Local还是Global，以及这个Reloc属于的Object。按照是Global还是Local在Global和Local的符号表内查找需要的Symbol。找到后就进行计算。

如果是绝对地址的截断处理，就对地址进行取掩码处理，这种处理方式下需要进行错误检查，如果有无符号的截断处理无法还原原先的数据就扔出一个runtime error；如果是相对的截断处理，就根据任务三给出的公式进行计算相对地址；如果是原64位绝对地址，就直接将数据填入data段。

段合并策略

段合并是按照section的顺序来的，每次将不同obj中的同一个section进行合并，合并的同时处理relocate信息，计算relocate的offset并且把relocate的基本信息记录到Relocation Information里，并且根据不同部分的头部表维护offset变量。每个section遍历后，按照任务六给出的权限和段的基本信息生成头部表，并且将合并完成的段以及头部表加入到FLE文件即可。

内存对齐是由于每一节的大小都不大于4KB，因此只需要把每个section都对齐到第n个4KB即可。

整个文件完成之后需要在Global符号中找到start符号并生成入口，按照issue中的要求清除relocate信息等，并最终返回。文件所有查找符号的地方都进行了错误检查，找不到符号会导致程序抛出runtime error。

Part C: 关键难点解决

- 数据结构设计和处理：个人感觉这个任务需要不断的检查offset和Symbol Type，因此不仅需要使用到Global和Local符号表，还需要记录重定位项的Object，同时有的项目需要考虑到重复（如section name）的问题，因此设计和维护数据结构很困难。我的解决方法是使用多种适用的数据结构进行处理，如Global Local Symbol使用map，reloc_info使用自己设计的struct等。
- 地址计算：题目中offset的计算非常多，对数据结构不熟悉很容易计算错误，并且很难找出错误。在计算offset的时候需要仔细的阅读FLE文件的诸多数据结构，了解每一部分的数据记录。

Part D: 实验反馈

1. 实验设计：实验整体感觉难度不高，但是实际编写有时非常困难，主要是由于以下几个问题：
    1. 不同任务基本上没有线性关联，甚至有的任务需要把之前的代码一部分推翻重来（如任务六），导致需要重新设计数据结构，就像是搭积木的时候有的任务要求在积木的最底层加上一层，在积木（代码）已经基本完成之后就十分困难。建议可以调整部分任务的顺序，将相对底层的任务放到最初，或者每个task只需要完成链接器的一部分，逐步结合成完整的链接器，从而不会出现“推翻重来”的情况。
    2. 调试困难：基本上没有很好的调试方式，无法像正常的代码进行断点单步调试，或是cout或printf输出数据，导致想要得到程序运行的某个数据结构问题需要写大量的文件输出流输出数据结构信息，而且无法完整的输出，导致很多时候出现bug都需要猜测bug的原因。
    3. 测试点和任务并不完全匹配：部分测试点如task2的测试点没有Local符号的重定位测试，但是Local的重定位是在task2中要求的。在task3测试点出现local的重定位问题时，学生很容易简单的认为是task3新加入的代码出现问题（比如我），因此非常难以找出bug。
2. 实验文档：文档在更新过一次后相对清晰，但是还是不是很清楚。部分task要求的步骤非常多（如task2和6），建议文档能够清楚的列举出需要完成的内容，避免出现不知道自己是否已经完成task的情况。同时部分task的要求非常少（如task3和5），建议可以将任务较多的task拆解成很多子任务，或者按照子任务的方式分点写出需求。同时，建议文档能强调比较重要的部分，如**调试工具**等。
3. 框架代码：readfle等调试工具非常好用，但是调试方式依旧较少，修理bug的成本比较高。

参考资料