LinkLab 报告

姓名：施素注
学号：2024201620

Part A: 思路简述

<!-- 200字以内简述你的链接器的关键实现思路,重点说明:

1. 核心方法的流程，依次做了什么
2. 关键数据结构的设计，如何组织和管理符号表
   -->

核心方法的流程：

1. 分离静态库`.ar`与目标文件`.obj`还有共享库`.so`
2. 进行按需链接，将所有会处理到的静态库目标文件加入待处理的目标文件中
3. 第一次遍历：合并节内容
4. 第二次遍历：构建全局符号表 + 处理符号冲突 + 分配内存地址
    1. 对于外部符号和弱变量符号生成GOT表，对外部函数生成PLT表。
    2. 填充PLT表
    3. 分配内存地址
5. 第三次遍历：处理重定位
    1. 共享库重定位
    2. 静态链接重定位
6. 构造输出节头 + 生成程序头 + 找到文件入口点
7. 填充符号表 + 返回最终文件

关键数据结构的设计：

1. 进行按需链接时，使用了队列进行搜索，还用了set进行去重
2. map哈希表也是程序中极为重要的一个数据结构，在查找符号，地址等操作时避免了遍历查找的O（n）复杂度，用O（1）的时间复杂度准确高效的进行查找。

组织和管理符号表：

1. 我使用`map<string, Symbol> global_symbols`作为全局符号表，用map的形式存储是为了方便后面重定位时对于符号的查找。
2. 我用`unordered_set<string> external_symbols`作为外部符号表，将 UNDEF 符号和弱变量符号加入其中，不过有些 UNDEF 符号 并不是 外部符号，更有 task4 中可能是未定义的符号。需要对`external_symbols`里面的符号遍历判断。

```
if(global_symbols.count(reloc.symbol) ||
(!so_symbol_section.count(reloc.symbol)))
{
    // 把未定义但不是外部符号的去掉了
    if(options.shared == false) external_symbols.erase(reloc.symbol);
}
```

之后的话符号表就差不多定型了。

Part B: 具体实现分析

符号解析实现

<!-- 300字以内,描述:

1. 如何处理不同类型的符号(全局/局部/弱符号)
2. 如何解决符号冲突
3. 实现中的关键优化
4. 关键的错误处理，一些边界情况与 sanity check
   -->

如何处理不同类型的符号：

GLOBAL 只能存在一个，而 WEAK 可以存在多个。有趣的是当没有GLOBAL而有多个WEAK时，就随机拿一个（lab中建议为第一个）WEAK来用。而局部符号要额外加上前缀`文件名::`。

还有外部符号`External`，对于外部符号最难的是判断他究竟是不是外部符号。具体筛选的代码在Part A中，最特别的还有`弱变量符号`,他虽然不是外部符号，但是同样需要GOT表，所以我也会把他加到`External`中。

```
else if(sym.type == SymbolType::WEAK && sym.section != ".text")
{
    external_symbols.insert(sym.name);
    // 弱变量符号
}
```

如何解决符号冲突：

冲突规则：GLOBAL > WEAK > LOCAL，但是 LOCAL 符号在这次lab里面会在符号名前面加上`文件名::`，所以不会和 GLOBAL，WEAK 会有命名上的冲突。

实现中的关键优化：

一个符号的地址由三个部分组成：

1. 最终文件中符号所在的合并节的起始地址
2. 符号原本所在的目标文件中的小节在合并节中的起始地址
3. 符号在原本所在的目标文件中的小节的偏移地址

为了快速得到这些地址，我用了：

```
map<string, SecInfo> global_sections; // 全局的大合并节的初始位置和大小
map<pair<size_t,string>, size_t> pre_sec_addr; // 原来的小节在原文件的起始地址
```

其中`global_sections`就是对应合并节的名称的起始地址，这个还比较简单。第二个`pre_sec_addr`则用了`pair`的技巧，一方面要记住符号是在哪个节的，另一方面还要记住符号在的这个节在合并节里面是第几小节，所以这相当于用合并两个键查一个值。而地址的第三部分就是原来符号本身的offset。

关键的错误处理，一些边界情况与 sanity check：

1:跳过未定义符号

```
if (sym.type == SymbolType::UNDEFINED) continue;
```

2: 同时存在两个相同的全局变量，抛出异常

```
else if (global_sym.type == existing_sym.type && global_sym.type ==
SymbolType::GLOBAL)
{
    throw runtime_error("Multiple definition of strong symbol: " + sym.name);
}
```

3: 如果重定位到了未定义的符号

```
if(!global_symbols.count(sym_name))
{
    throw runtime_error("Relocation points to an undefined symbol: " + sym_name);
}
```

4：我原本的代码中的错误与改正：

```
虽然符号的地址由三部分组成，但是就符号本身而言，其offset是相对其所在的节的偏移。
原本我的offset公式为：
global_sym.offset = merged_sec_addr + sec_off + sym.offset;
然后就会报错，访问到了未知地址。
后来检查发现是我原本对offset的理解有错，不能是全局的绝对地址：
global_sym.offset = sec_off + sym.offset;
```

重定位处理

<!-- 300字以内,描述:

1. 支持的重定位类型
2. 重定位计算方法
3. 关键的错误处理

-->

支持的重定位类型：所有都支持

重定位计算方法：

主要就两个，绝对地址和相对地址

重定位所指向的符号绝对地址：合并大节初始地址 + 原小节在合并大姐中的地址 + 符号原本的在小节中的偏移`offset`

相对地址:`S + A - P`,S为重定位所指向的符号绝对地址，A 为 `reloc.`addend,

P 为 重定位项所在地址

关键的错误计算：

忘记加 addend,然后相对地址时是谁减谁

bonus中的文档写的是

> 计算从当前位置到对应PLT stub的相对偏移，填入那个位置

这样感觉就很像是`reloc - 目标地址`,但是仔细思索，无论是`call`还是`jmp`，都是当前地址加 offset，如果是`reloc - 目标地址`,那就变成了`reloc + reloc - 目标地址`,这样就不对了。应该是填充`目标地址 + addend - reloc`

段合并策略

<!-- 300字以内,描述:

1. 如何组织和合并各类段
2. 内存布局的考虑
3. 对齐等特殊处理
   -->

合并各类段其实并没有那么难，我觉得最主要的是做这三件事情

```
// 合并内存
merged_sec[head].data.insert
(
    merged_sec[head].data.end(),
    sec.data.begin(),
    sec.data.end()
);
// 存下当前小节在合并大节后的初始位置
pre_sec_addr[{obj_idx, sec_name}] = global_sections[head].size;
// 相应的，更新到下一个小节的初始位置
global_sections[head].size += shdr.size;
```

内存布局的考虑：按照section_order的顺序来合并节

```
vector<string> section_order = {".text",".plt",".rodata",".got",".data",".bss"};
```

地址从给定的起始地址`0x400000`开始,然后按对齐一次填充

对齐：一行代码轻松搞定。

```
current_vaddr = (current_vaddr + page_size - 1) / page_size * page_size;
```

current_vaddr 表示当前节所在的初始地址

Part C: 关键难点解决

<!-- 选择2-3个最有技术含量的难点:

1. 具体难点描述
2. 你的解决方案

3. 方案的效果
示例难点:

- 重复符号处理
- 重定位越界检查
- 段对齐处理
-->

写的时候还是挺煎熬的，磕磕绊绊的，但是写完之后感觉倒也还好，其实代码部分对算法之类的可能比较难的就是按需链接队列那部分了，其他的最关键的是理解。

具体难点：

1.按需链接

解决方案：虽然总体上就是一个队列BFS的框架，但是具体实现起来还是要花一些时间的。以下是解决的简要流程：

```
区分静态库与目标文件与共享库
    将.ar 加入 curr.ars
    将.obj 加入 curr.objs 和 to_process
    略过 .so
进行按需链接BFS搜索
    取 to_process 的 front
    查看 front 里面还有没有未在 curr_objs 里面定义过的符号
    遍历 curr.ars 找到里面有符号依赖而且从来没被更新过的目标文件
    将目标文件添加入 curr_objs 和 to_process
```

方案效果：成功地实现了按需链接地功能，将静态库里面有符号依赖的目标文件加入待链接的文件vector中

2.分配节的地址

如何得到各个合并节的初始地址?

解决方案：对 global_sections 进行遍历，得到里面每个节的大小，然后使用 current_vaddr 进行累加，同时进行对齐。

```
// 分配节的地址
for (const auto& sec_name : section_order)
{
    // 读取节
    auto& sec = merged_sec[sec_name];
    // 记录当前节的初始位置
    global_sections[sec_name].addr = current_vaddr;

    // 更新节大小（.bss 节从输入节累计大小）
    if (sec_name.starts_with(".bss"))
    {
        // resize（a,b） 表示新增a个全部初始化为b的元素
        sec.data.resize(global_sections[".bss"].size, 0); // 填充0占位
    }

    // 推进当前地址（按节实际大小分配）
    // 也就是合并大节的初始位置
    // 而且地址要满足为 页大小 的整数倍
```

```
    current_vaddr += sec.data.size();
    current_vaddr = (current_vaddr + page_size - 1) / page_size * page_size;
}
```

方案效果：实现将各个节的初始地址赋值到 global_sections[].addr 中

Part D: 实验反馈

<!-- 芝士 5202 年研发的船新实验，你的反馈对我们至关重要
可以从实验设计，实验文档，框架代码三个方面进行反馈，具体衡量：

1. 实验设计：实验难度是否合适，实验工作量是否合理，是否让你更加理解链接器，链接器够不够有趣
2. 实验文档：文档是否清晰，哪些地方需要补充说明
3. 框架代码：框架代码是否易于理解，接口设计是否合理，实验中遇到的框架代码的问题（请引用在 repo 中你提出的 issue）
   -->

1.实验设计： 完成时间 35h左右

实验主要是 task2 和 bonus2 难度跨度有点大，首先task2直接要求手搓链接器，一下就两三百行的代码量，当时我还对link整个过程不太熟悉，还是在重新看了课件好几次，艰难的手搓出了 task2，接着是bonus2，手搓GOT表和PLT表，也是思索了好几次，终于才弄懂基本的过程，敲出了基本的代码，但是没想到每个点又只给一半的分，接着就debug近10h。。。。

总体来看`Id.cpp`的代码量其实还好，我代码比较稀疏，写了600行。主要是中间思考的过程和debug占了大多时间。不过就是从task2写到task7，期间要不断对代码许多地方进行调整，感觉还是可以给一个大体的框架，然后再往里面填代码。

链接器还是挺有趣的，也让我对链接器理解更深了，我bonus2卡住的那几天里面，睡觉都是GOT表和PLT表。

2.实验文档：

我觉得主要还是task2和bonus2的文档可以再写清晰一些，尤其是bonus2里面的21和22，情况都挺特殊的。比如说21里面有helper有多个，其中一个是UNDEF的，然后22里面`weak_value`是弱变量符号，也要用GOT表重定位，感觉不debug看文件的话，第一次写代码肯定考虑不到这些。

3.框架代码：。？？？好像没啥框架代码，fle.cpp还挺清晰的，不过里面有一些属性感觉多余，比如FLESection的has_symbols，感觉可以去掉。

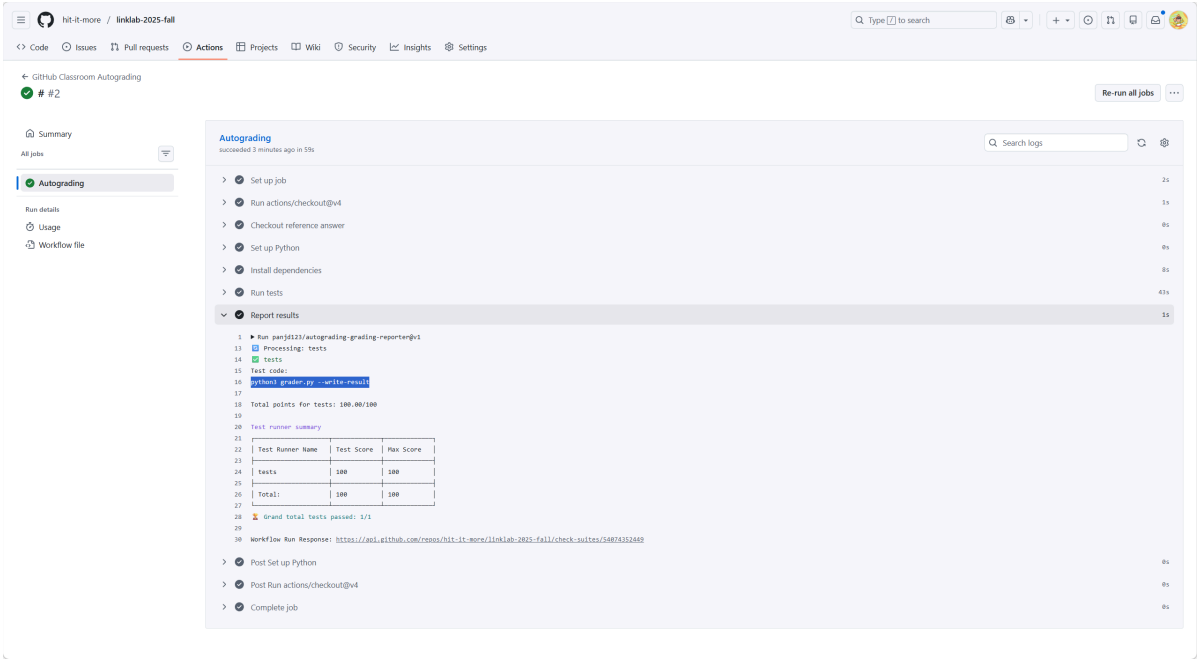
参考资料 （可不填）

1. [资料名] - [链接] - [具体帮助点]
2. ...还真没有，全靠docs和课本，老师的课件。

```
ssz@localhost:~/linklab-2025-fall$ make test
```

Complex Static Linking Test	PASS	0.37s	10.0/10.0	All steps completed
Shared Library Basic	PASS	0.09s	7.0/7.0	All steps completed
External Symbol Relocation	PASS	0.08s	6.0/6.0	All steps completed
Weak Symbol Export	PASS	0.08s	7.0/7.0	All steps completed
PLT/GOT Basic	PASS	0.12s	7.0/7.0	All steps completed
Multi-Library Dependency	PASS	0.18s	6.0/6.0	All steps completed
Complex PLT/GOT Offset	PASS	0.15s	7.0/7.0	All steps completed

```
Total Score: 200.0/200.0 (100.0%)
```



最终通过截图。