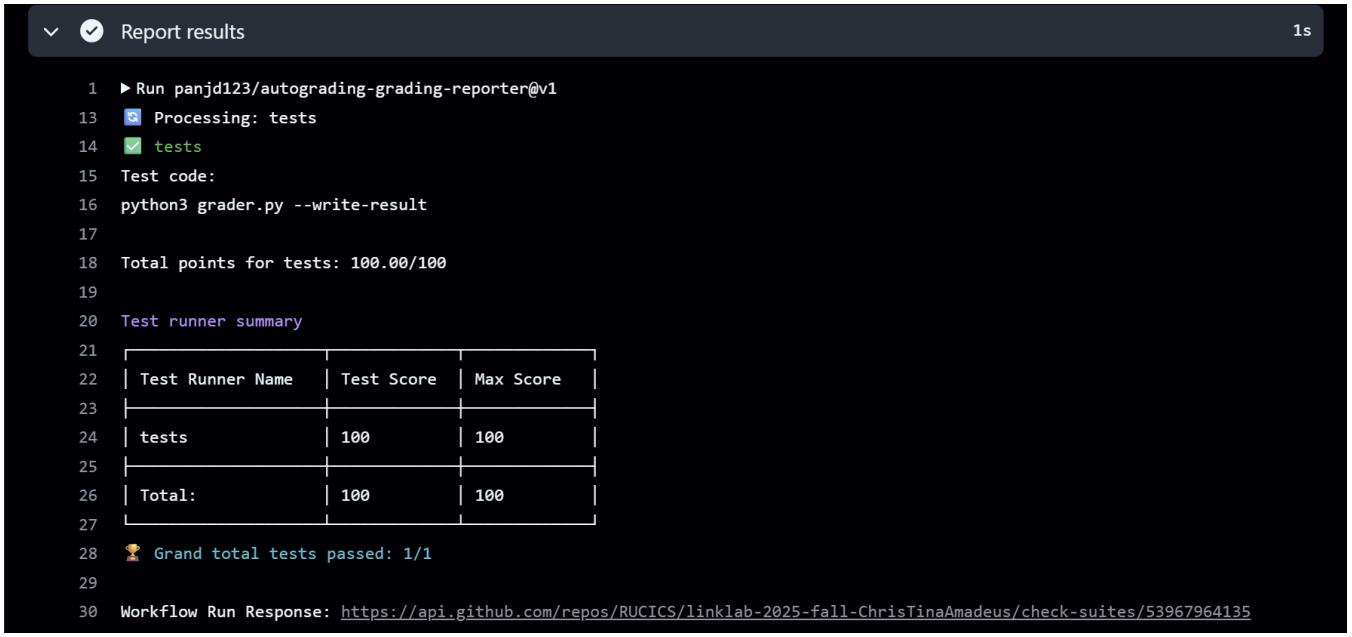
LinkLab 报告

姓名：王松宸
学号：2024201594



Part A: 思路简述

核心流程遵循典型的两遍扫描（Two-pass）策略：

1. **输入扫描与符号收集**：
   - 遍历输入文件，区分普通目标文件（`.obj`）、静态库（`.ar`）和共享库（`.so`）。
   - **静态库按需提取**：采用不动点迭代算法（Fixed-point iteration），反复扫描归档文件，仅提取能解析当前未定义符号的成员，直到无新成员加入。
   - **初始分类**：收集所有激活对象的段信息和符号定义，建立初始的局部与全局符号表。
2. **地址分配与段合并**：
   - 将相似的节分类合并为主要的输出段：`.text`（代码）、`.rodata`（只读数据）、`.data`（读写数据）、`.bss`（未初始化数据）。
   - **布局计算**：检测外部依赖，为动态链接分配 GOT（全局偏移表）和 PLT（过程链接表）所需的空间。
   - **内存对齐**：计算各段的虚拟地址（VMA），确保每个段的起始地址按 **4KB 页对齐**，以满足内存权限保护的要求。
3. **重定位与输出生成**：
   - **重定位应用**：遍历所有激活节的重定位条目。根据符号解析结果（本地定义或通过 GOT/PLT 访问），计算修正值并回填到对应段的数据缓冲区中。
   - **文件生成**：构造最终的 FLE 对象，设置程序头（Program Headers）的权限位（如 `.text` 为 `R|X`），处理 `.bss` 节（仅记录大小或占位），最终序列化为输出文件。

关键数据结构设计：

- `SectionMapping`：一个结构体记录，保存了输入节（Input Section）到输出段虚拟地址的映射关系，是重定

位计算的核心依据。

- **全局符号表**（`globals`）：使用 `std::map<string, GlobalSym>`，存储全局和弱符号。`GlobalSym` 包含符号类型和绝对地址，支持强弱符号规则的快速查重与覆盖。
- **局部符号表**（`locals`）：使用 `std::map<const FLEObject*, map<string, uint64_t>>`。采用双层映射（对象指针 -> 符号名 -> 地址），能够完美区分不同文件中的同名 `static` 变量，避免冲突。

Part B: 具体实现分析

符号解析实现

1. **分层查找策略**：解析符号时，优先在当前对象的 `locals` 表中查找（处理 `static` 局部符号），若未找到则查全局表 `globals`。
2. **符号冲突决议**：
    - **强+强**：报错 `Multiple definition`。
    - **强+弱**：强符号覆盖弱符号（更新地址表）。
    - **弱+弱**：保留原有定义（遵循 First-fit 原则）。
3. **未定义检查**：在链接结束前，对于非共享库输出，检查所有未解析的符号。若符号未在内部定义且不由共享库提供，则抛出 `Undefined symbol` 异常。
4. **优化**：在静态库扫描阶段，维护 `unresolved` 集合，避免重复的全量符号表扫描。

重定位处理

1. **支持的重定位类型**：
    - `R_X86_64_64` / `32` / `32S`：绝对地址修正，计算公式为 `S + A`。
    - `R_X86_64_PC32`：相对地址修正，计算公式为 `S + A - P`。
    - `R_X86_64_GOTPCREL`：GOT 相对寻址，计算公式为 `GOT_SLOT + A - P`。
2. **动态链接支持**：
    - 对于外部函数调用（`PC32`），重定位目标指向本地生成的 PLT 条目。
    - 对于外部数据访问（`GOTPCREL`），重定位目标指向本地生成的 GOT 槽位。
    - 生成 `.so` 时，未能内部解析的符号引用会转化为动态重定位项（`dyn_relocs`），留待加载器处理。

段合并策略

1. **分类与映射**：通过辅助函数识别节名称前缀（如 `.text.startup` 归入 `.text`），将所有输入节分配到四个标准类中。
2. **内存布局**：
    - 顺序：`.text` (含 PLT) -> `.rodata` -> `.data` -> `.got` -> `.bss`。
    - 使用 `text_data`, `rodata_data` 等 `vector<uint8_t>` 缓冲区进行物理拼接。
3. **对齐与保护**：
    - 使用 `align_up(vaddr, 4096)` 计算每个段的基址。这虽然会在虚拟地址空间中产生“空洞”，但确保了操作系统能对不同段（如代码段 RX，数据段 RW）独立设置页表权限。
    - `.bss` 段在文件中仅记录头部信息（或全零占位），加载时由 OS 清零。

Part C: 关键难点解决

1. 静态库的循环依赖与按需加载

难点：归档文件包含多个 `.obj`，链接器不应加载未被引用的成员；但库成员之间可能相互依赖（A 引用 B，B 又引用 A），简单的顺序扫描无法解决。
解决方案：实现不动点迭代算法（Fixed-point iteration）。在一个 `while(changed)` 循环中，每一轮收集当前所有的未解析符号，遍历所有归档文件成员。只要发现某个成员定义了未解析符号，就将其加入"激活列表"并标记 `changed=true`。
效果：能够正确处理复杂的库依赖关系，仅链接必要的代码，符合标准链接器 `ld` 的行为。

2. 局部符号的同名冲突

难点：多个 C 源文件可能定义同名的 `static` 变量（如 `static int counter`）。它们在逻辑上是隔离的，但在简单的符号表中会发生命名冲突。
解决方案：放弃单一的符号表结构。构建了二级映射 `locals[Object*][SymbolName]`。在处理某个对象的重定位时，强制先在该对象的专属局部表中查找符号。
效果：彻底隔离了不同文件作用域，无需对符号名称进行重命名（如 `file.c::var`）即可通过所有本地符号测试，逻辑更清晰。

3. GOT/PLT 的动态链接构建 (Bonus)

难点：需要在链接阶段预判哪些符号是外部的，并为其分配转接代码桩（Stub）和数据槽（Slot），同时处理 `PC32` 到 PLT 的跳转转换。
解决方案：

- **预扫描**：在地址计算前，扫描所有外部重定位。若符号为外部函数，分配 6 字节 PLT Stub；若为外部数据，分配 8 字节 GOT Slot。
- **代码生成**：手动构造 PLT 的机器码 `jmp *got_offset(%rip)`，并在重定位阶段将原本指向符号的 `PC32` 修正为指向 PLT Stub。
  效果：成功支持了可执行文件调用共享库函数，实现了地址无关代码（PIC）的正确链接与运行。

Part D: 实验反馈

1. **实验设计**：实验设计非常出色，由浅入深。从基础的符号表查看（Task 1）到模拟完整的静态链接过程，最后扩展到动态链接（Bonus），学习曲线平滑。FLE 格式去除了 ELF 格式中繁琐的二进制偏移计算，让我能将精力集中在链接器的核心逻辑（符号决议、重定位、内存布局）上，极大地提升了学习效率。
2. **实验文档**：文档质量很高，尤其是关于 Bonus 部分的原理讲解，配合图示和代码示例，清晰地解释了 GOT 和 PLT 的协作流程，对完成 Bonus 任务帮助巨大。
3. **个人反思**：完成本次实验的过程中我对链接的理解更上一层楼。报告写到这里的时候突然有点感慨，如果能在期末以前就完成这个实验就好了，说不定考场上能多会点链接的题...

参考资料

1. 教材
2. 北大 ICS 课程资料