# Lab 9: Linker - Pass I



## Lab Goal

In labs 7 and 8, you learned to manipulate object files. In this lab, you will do a simplified pass I of the linkage editor. Your input will be two relocatable files in ELF format. Your output will also be an ELF relocatable, but with all undefined symbols resolved, and all the relevant sections from both files merged.

## Simplifying assumptions

In order to make the task manageable within the time frame of a few hours, we will introduce several simplifying assumptions, as follows:

1. There will be only 2 input files: the first file is the primary (or "main") file, the second is the secondary file.
2. Only the primary file may have undefined global symbols in its symbol table.
3. If a section (e.g. ".text") from the secondary file is needed, i.e. it is the place where a symbol is defined, then there is always a section with the same name in the primary file.
4. The only sections that need to be merged (if they exist) are ".text", ".data", and ".rodata".
5. Only the primary file will have relocation tables (or if the secondary file has any such tables, they will be empty and thus can be ignored).

## Operation of the merging program

The basic operation of the merging program proceeds along the following steps:

1. Find undefined symbols in file1.
2. For each undefined symbol in file1, find a symbol corresponding to the same name in the symbol table of file2, e.g. if symbol *"foo"* is undefined in file1 and appears in the .text section of file2, we will append the .text section of file2 to the end of the .text section of file1. Consequently, we will have to fix the information in our new symbol table accordingly. In the case of symbol "foo", its entry in the symbol table should now (a) point to the .text section as numbered in file1; (b) point to the new offset, as it has now been appended to the end of the .text section of file1.
3. Write the output file: first the ELF header, then all the sections in order of appearance in the section header table (while merging sections ".text", ".data" and ".rodata" from the secondary file), and then the section header table at the end of the file.
   Finally, fix the ELF header, so that it points to the updated location of the section header table.

## Lab tasks

The lab tasks are not in order of execution in the merger program. The reason is to enable you to get an output file viewable by readelf as soon as possible, as an aid to debugging.

During this lab you must use **only** the files provided to you in task 0 and **not** your code from lab 8

## Task 0a - one file output (basic)

In this task we will write a preliminary version of the output function which writes out the new ELF file.
This function, *write_file(int fdOut, char *start1, char *start2)*, receives a file descriptor for the output file and two pointers to ELF file structures which were mapped to memory.
In task0 we will ignore the second ELF file.

1. Read the entire lab, make a simple drawing of the original files' sections and how the output should look like.
2. Download and compile the skeleton files:
   - The code skeleton - This is the file you will write your code in.
   - The utility functions
   - Utility functions header
3. Download and compile the test files (the object files will be used as input for our Pass I linker):
   - The test code
   - The syscall assembly code
   - Compile and link the files (using the ld linker) to see what the program does

4. Write a preliminary version of the write_file function. In this preliminary version, *write_file* will write to the output file the following data objects (from *start1*): the ELF header, then all the sections in order of appearance in the section header table, and then the section header table (which will be written at the end of the file). At this point all the above items will be written without modifying them, as follows:
   - Write out the ELF header to the beginning of the output file as is.
   - Loop over all section headers in the section header table, printing relevant fields (section name, offset, and length) to stdout (for debugging), and also write the respective section contents to the output file.
   - Write the section header table to the output file, and close the file
   - Finally, check your output file's correctness with hexedit

The new object file you've just assembled is **corrupt** (see Task 0b to understand why). Therefore, do not expect it to link properly.

### Task 0b - one file output (fixing internal pointers)

Note that in task 0a you wrote the ELF header, the sections, and the section headers, as they are, not necessarily in correct order in the file, and the file offsets may no longer be consistent with the actual file layout. Now you need to fix the ELF header and the section header table to account for this.

This task modifies the *write_file* from task 0a so that the resulting output file is a valid and consistent object file. All this in preparation for the lab, where you will also be merging and otherwise modifying sections. Do this according to the following steps:

1. As you will be modifying the ELF header and the section header table, modify *write_file* to first create NEW copies of these two objects somewhere else in memory (preferably on the heap using malloc(), then memcpy()), and then change *write_file* to write the NEW copies to the output file instead of the original copies.
2. Insert the following preliminary step before the code that actually writes the output file: loop over the section headers, and for each of them compute the new and correct offsets (calculated as the size of the ELF header PLUS the accumulated size of the previous sections). Modify the offset field of the section headers (in the NEW copy) accordingly.
3. Insert the following additional step just before the code that writes the output file: modify the shoff field of the NEW copy of the ELF header according to the new offset of the section header table. Since *write_file* intends to write the section header table after the sections, the new location is simply the size of the ELF header PLUS the accumulated size of all sections.
4. Note that at this point, if you did follow the above steps correctly, the code that writes the file should write the modified versions of the ELF header and section header tables. **Check using readelf that the resulting file looks OK**, i.e. it should have the same sections, symbols, and relocations as the original file (but possibly in different file offsets)

**Note**
You are expected to complete all of task 0 before the lab. However, we are aware that it (especially part 0b) is non-trivial, and you may need help. Therefore, we require that you complete task 0a before the lab, and **make an honest effort** to do task 0b before the lab as well. If you do not succeed in task 0b before the lab, you can get help on this task during the lab, with no grade penalty, contingent on making an honest effort. However, please be aware that **not** doing this subtask before the lab may cost significant time during the lab, making you unable to complete the required tasks, **indirectly** getting a grade reduction as a result, so please **do** make an honest effort on this part as well.

### Task 1 - merge sections

Extend the *write_file* function from Task 0 to merge ".text", ".data" and ".rodata" (if they exist) of the two files.
Merging a section, e.g. ".text", means assembling the sections corresponding to the same name, s.t. the section of file2 is appended to end of the section of file1. This can be done in several ways, the simplest being to write the content of the each such section from the "start2" image to the output file immediately after writing the section from the "start1" image. Make sure to update section sizes and file offsets accordingly (header->e_shoff=accumulator), all updates must be done in the NEW section header table.

Note that at this point, your merged output object file is **corrupt**. Finalizing the Pass I of the linker requires updating relevant information in the symbol table, as in the following tasks.

### Task 2 - undefined symbols

Now, we need to update the symbol table - i.e. to write the *resolve_all_symbols(void *start1, void * start2)* function:
This function receives two pointers to memory regions ("buffers"), where in each, an ELF file structure had been mapped to memory.
The function changes the symbol table of the first file (*start1*) in place resolving all the undefined symbols in it.
If all undefined symbols are defined in the second file the function returns 1, otherwise -1.

### Task 2a - print undefined symbols

1. Loop over all symbols in the primary file's symbol table, and for every undefined global symbol, find it in the secondary file's symbol table.
   If the symbol is not in the secondary table, return -1.
2. Find the section in which the symbol appears in *start2*
3. Find the respective section number in *start1* (e.g. if ".text" is section 14 in *start2* and ".text" is section 9 in *start1* we want the symbol to point to section 9 in the output file).
4. Print: *symbol_name section_name section_index_in_file2 section_index_in_file1*

### Task 2b - update symbol table

1. Loop over all symbols in the primary file's symbol table, and for every undefined global symbol, find the symbol in the secondary file's symbol table. If the symbol is not there, return -1. (Same as step 1 in task 2a)
2. Fix the symbol table entry appropriately:
   - Find the section in which the symbol appears in *start2*, and find the respective section number in *start1*. (Same as steps 2 and 3 in task 2a)
   - Update the index of the symbol's section in the symbol table (from *start1*) to the correct section number (in the example above from task 2a it should be 9).
   - Update the offset of the symbol within the section (the section from *start2* will be appended at the end of the same section in *start1*).
3. Check using readelf that the resulting changes make sense.

### Task 2c - Final testing phase

At this point, if you have done everything correctly, the output files generated by your program result in a complete stand-alone relocatable file.

Test the results with by relocating and loading with ld (ld -melf_i386 [input file name] -o [output])

### Deliverables

Must finish task 1, 2a in the lab. Tasks 2b, 2c may be done in a completion lab if you run out of time.
Have fun!