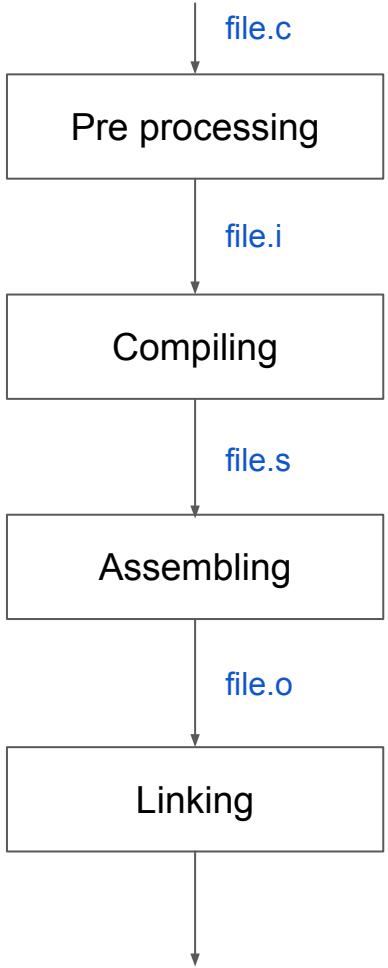
C Compilation process



gcc --save-temps file.c

-- it will output file.i file.s file.o and executable file

Pre processing

- Interpret preprocessor directives. preprocessing directive lines have been replaced with blank lines.
- Comments replace with spaces.
- Long runs of blank lines are discarded.
- Expand macros.

Preprocessor directives - Line starts with character #. **It gives instruction to the compiler to preprocess the information before actual compilation starts.**

In the C Programming Language, the #include directive **tells the preprocessor to insert the contents of another file into the source code at the point where** the #include directive is found.

cont..

linenum filename [flags]

flags:

‘1’ - This indicates the start of a new file.
‘2’ - This indicates returning to a file (after having included another file).
‘3’ - This indicates that the following text comes from a system header file, so certain warnings should be suppressed.
‘4’ - This indicates that the following text should be treated as being wrapped in an implicit extern "C" block.

```
# 1 "hello.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 1 "<command-line>" 2
# 1 "hello.c"
# 1 "/usr/include/stdio.h" 1 3 4
```

represents this process:

- Start reading hello.c, and then
- immediately read the built-in macros pseudo-file, and then
- immediately read the command line pseudo-file, and then
- immediately read the pre-defined macros file /usr/include/stdc-predef.h *as if* it was pre-included on the commandline by -include /usr/include/stdc-predef.h, and then
- resume and finish reading the command line pseudo-file, and then
- resume reading hello.c, and then
- start reading /usr/include/stdio.h

Compiling

- It takes the output of the preprocessor and generates assembly language, an intermediate human readable language, specific to the target processor.

Assembling

- Assembly is the third step of compilation. The assembler will convert the assembly code into pure binary code or machine code (zeros and ones). This code is also known as object code.
- Generates relocatable object file.

Linking

- Linking is the final step of compilation. The linker merges all the object code from multiple modules into a single one. If we are using a function from libraries, linker will link our code with that library function code.
- In static linking, the linker makes a copy of all used library functions to the executable file. In dynamic linking, the code is not copied, it is done by just placing the name of the library in the binary file. Modern operating system often support dynamic linking.

readelf

readelf - display information about ELF files.

To display all information - `readelf -a elf_file`

To display file headers of a elf file. - `readelf -h elf_file`

To display information about the different sections of the process' address space. - `readelf -S elf_file`

To display symbols table. - `readelf -s elf_file`

To display core notes. - `readelf -n elf_files`

To display relocation section. - `readelf -r elf_file`

elf - format of Executable and Linking Format (ELF) files. The ELF header is always at offset zero of the file. The program header table and the section header table's offset in the file are defined in the ELF header.

Strace

strace - trace system calls and signals.

It intercepts and records the system calls which are called by a process and the signals which are received by a process.

Each line in the trace contains the name of each system call, its arguments and its return value.

Uses - know your system call, what files your process opens, log files, Debugging Programs.

Some options -

-o filename

-e trace=open

-c --- count

-t --- absolute timestamp

-r --- relative timestamp

Eg.

strace -o strace.out ls

strace -e trace=write ls

strace -c ls

strace -r ls

strace -t ls

Linux system call

man syscalls