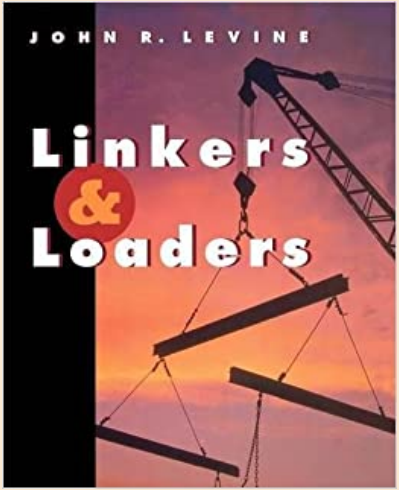
Compiler Construction

Lecture 15: Linking

Jeremy Yallop
jeremy.yallop@cl.cam.ac.uk
Lent 2026

Recommended book



Linkers & Loaders
John Levine
1st edition (October 25, 1999)
ISBN: 1558604960

Application Binary Interfaces

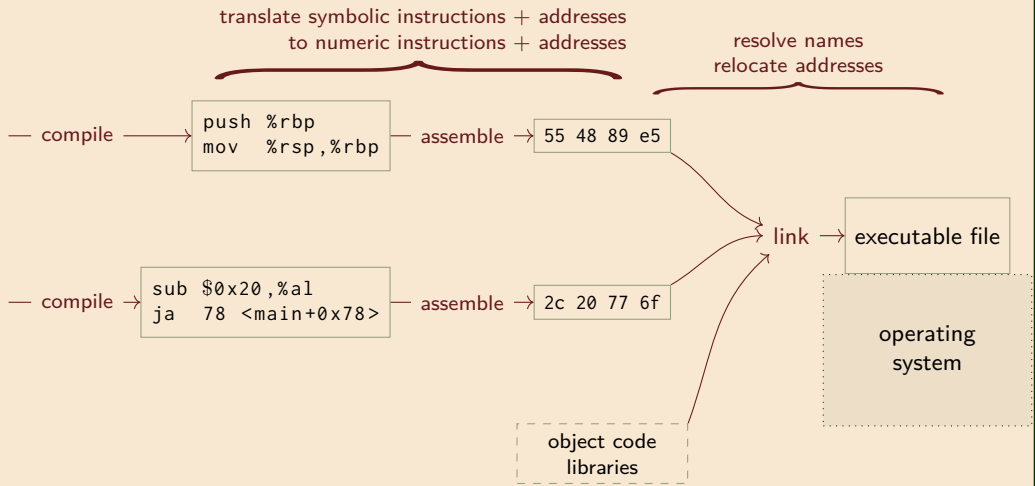
What happens after compilation?

ABI

Object files

Linking

Runtime



ABI: conventions that programs on a particular OS must follow:

- set of system calls (open, `read`, `write`, etc.)
- procedure for invoking the system calls
- what memory addresses a program can use
- how registers are used (e.g. passing parameters, returning results)
- stack frame layout
- data layout: endianness, alignment, etc.
- object file layout (e.g. ELF)
- linking, loading, name mangling

From *System V Application Binary Interface: AMD64 Architecture Processor Supplement*:

> The control bits of the `MXCSR` register are callee-saved (preserved across calls), while the status bits are caller-saved (not preserved). The x87 status word register is caller-saved, whereas the x87 control word is callee-saved.
>
> ### 3.2.2 The Stack Frame
>
> In addition to registers, each function has a frame on the run-time stack. This stack grows downwards from high addresses. Figure 3.3 shows the stack organization.
>
> The end of the input argument area shall be aligned on a 16 (32 or 64, if `__m256` or `__m512` is passed on stack) byte boundary. [11] In other words, the stack needs to be 16 (32 or 64) byte aligned immediately before the call instruction is executed. Once control has been transferred to the function entry point, i.e. immediately after the return address has been pushed, `%rsp` points to the return address, and the value of (`%rsp` + 8) is a multiple of

Object files

source file

```
let f x =
  M.g x + M.h x
```

compile

object file

Header information
(code size, source file name, etc.)

Object code
(binary instructions and data)

Relocation
(places for the linker to fix up)

Symbols
(exported & imported)

Debugging information
(line numbers, data structures, etc.)

ELF (executable and linkable format)

ELF is a common format for both linker input and output. Sections (not complete):

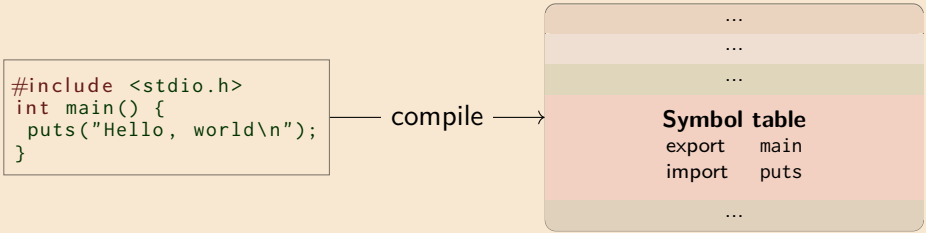
ELF header	
`.text`	code segment
`.data`	writable global data
`.rodata`	read-only global data
`.bss`	uninitialized data size
`.sym`	symbol table

`.rel.text`
`.rel.data`
`.rel.rodata`
	relocation tables: (offset, symbol) pairs

`.line`	maps source lines to object code locations
`.debug`	debugging information
`.strtab`	string names of symbols

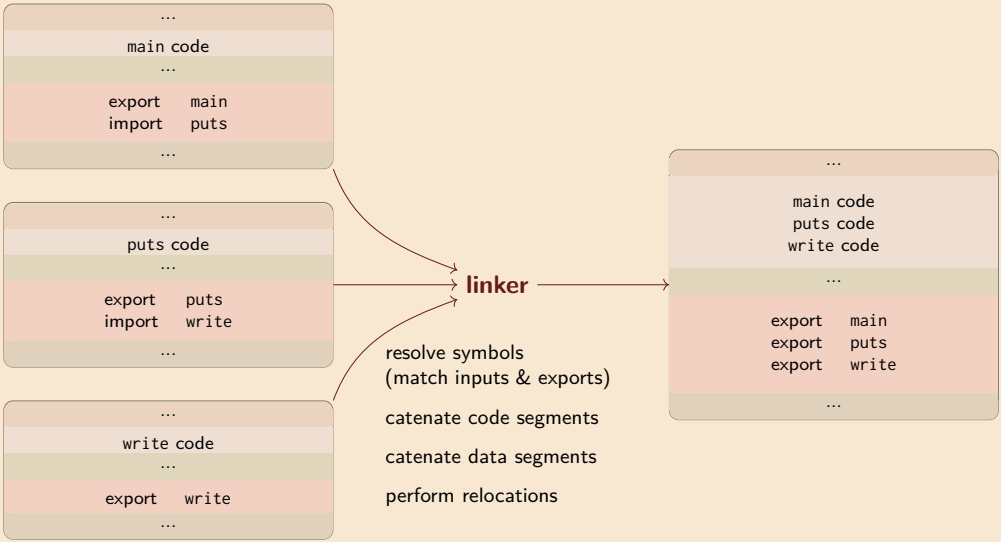
The symbol table in an object file may include various types of symbols:

- Global symbols defined (& perhaps referenced) in the module
- Global symbols referenced, but not defined
- Segment names
- Optional: non-global symbols, line number information (for debugging purposes)



Linking

(Static) linking



Static *vs* dynamic linking

Linking may be **static** (compile-time) or **dynamic** (run-time).

Dynamic linking: object files contain stubs; the OS links the code on demand.

Static linking

- \- Executables are larger
- \- Libraries can't easily be changed (e.g. for bug fixes)
- \+ Libraries can't change unexpectedly (silently updating program behaviour)
- \+ Loading (starting) programs is faster

Dynamic linking

- \+ Executables are smaller
- \+ Libraries can easily be changed (e.g. for bug fixes)
- \- Libraries can change unexpectedly (silently updating program behaviour)
- \- Loading (starting) programs is slower

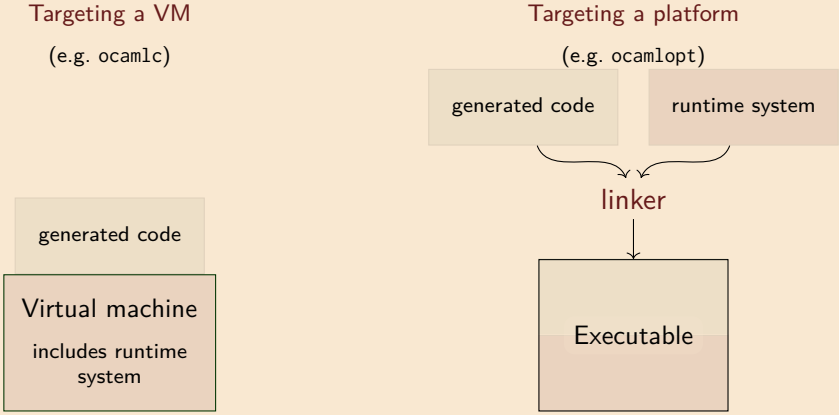
Runtime systems

Runtime system: a library needed to run compiled code
Provides support for a particular language ("the OCaml runtime")
Implemented for a particular operating system

The runtime system may offer:

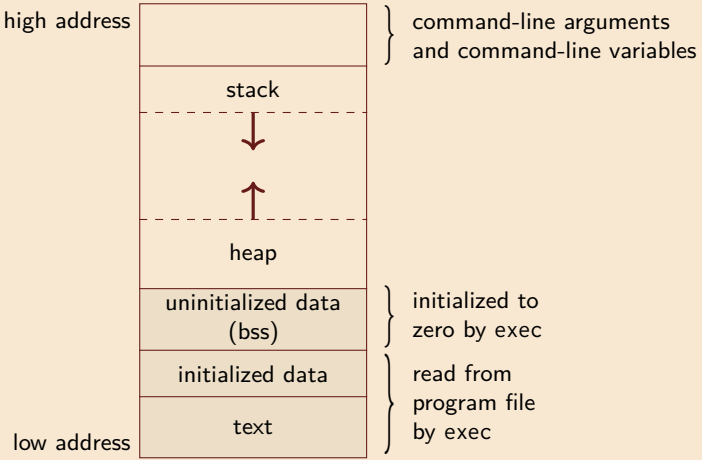
- an interface between the language and the operating system (system calls)
- an interface to other languages (foreign function interface)
- efficient implementations of primitive operations
- runtime type checking, method lookup, security checks, &c.

Targeting a VM *vs* targeting a platform



In both cases: compiler & runtime implementers must agree on low-level details (memory layout, data representation)

Typical memory layout (UNIX)



(Adapted from *Advanced Programming in the Unix Environment*, W. Richard Stevens)

Next time: bootstrapping