Linking

CS 520
Dept. of Computer Science
Univ. of New Hampshire
Linking

The linker combines multiple object code files together into one object code file.

ex  gcc a.c b.c

\[
\begin{align*}
\text{a.c} & \rightarrow \text{compiler} & \rightarrow \text{a.o} & \rightarrow \text{linker} & \rightarrow \text{a.out} \\
\text{b.c} & \rightarrow \text{compiler} & \rightarrow \text{b.o} & \rightarrow & \text{libraries}
\end{align*}
\]
Using built-in specs.

COLLECT GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/i686-redhat-linux/4.6.3/lto-wrapper
Target: i686-redhat-linux


Thread model: posix

gcc version 4.6.3 20120306 (Red Hat 4.6.3-2) (GCC)

COLLECT_GCC_OPTIONS='-v' '-mtune=generic' '-march=i686'

/usr/libexec/gcc/i686-redhat-linux/4.6.3/cc1 -quiet -v both.c -quiet -dumpbase both.c -mtune=generic -march=i686 -auxbase both -version -o /tmp/ccG0gt0.s

GNU C (GCC) version 4.6.3 20120306 (Red Hat 4.6.3-2) (i686-redhat-linux)
compiled by GNU C version 4.6.3 20120306 (Red Hat 4.6.3-2), GMP version 4.3.2, MPFR version 3.0.0, MPC version 0.9

GCC heuristics: --param gcc-min-expand=100 --param gcc-min-heapsize=131072

ignoring nonexistent directory "/usr/lib/gcc/i686-redhat-linux/4.6.3/include-fixed"

# include "..." search starts here:
# include <...> search starts here:
/usr/local/include

End of search list.

GNU C (GCC) version 4.6.3 20120306 (Red Hat 4.6.3-2) (i686-redhat-linux)
compiled by GNU C version 4.6.3 20120306 (Red Hat 4.6.3-2), GMP version 4.3.2, MPFR version 3.0.0, MPC version 0.9

GCC heuristics: --param gcc-min-expand=100 --param gcc-min-heapsize=131072

Compiler executable checksum: 3b913ff0ce1aa5aefc1491f952f749bb7

COLLECT_GCC_OPTIONS='--v' '-mtune=generic' '-march=i686'
as --32 -o /tmp/ccw6Tz2o.o /tmp/ccG0gt0.s

COLLECT_GCC_OPTIONS='--v' '-mtune=generic' '-march=i686'

/usr/libexec/gcc/i686-redhat-linux/4.6.3/cc1 -quiet -v vm520.c -quiet -dumpbase vm520.c -mtune=generic -march=i686 -auxbase vm520 -version -o /tmp/ccG0gt0.s

GNU C (GCC) version 4.6.3 20120306 (Red Hat 4.6.3-2) (i686-redhat-linux)
compiled by GNU C version 4.6.3 20120306 (Red Hat 4.6.3-2), GMP version 4.3.2, MPFR version 3.0.0, MPC version 0.9

GCC heuristics: --param gcc-min-expand=100 --param gcc-min-heapsize=131072

ignoring nonexistent directory "/usr/lib/gcc/i686-redhat-linux/4.6.3/include-fixed"

# include "..." search starts here:
# include <...> search starts here:
/usr/lib/gcc/i686-redhat-linux/4.6.3/include
/usr/local/include

End of search list.

GNU C (GCC) version 4.6.3 20120306 (Red Hat 4.6.3-2) (i686-redhat-linux)
compiled by GNU C version 4.6.3 20120306 (Red Hat 4.6.3-2), GMP version 4.3.2, MPFR version 3.0.0, MPC version 0.9

GCC heuristics: --param gcc-min-expand=100 --param gcc-min-heapsize=131072

Compiler executable checksum: 3b913ff0ce1aa5aefc1491f952f749bb7

COLLECT_GCC_OPTIONS='--v' '-mtune=generic' '-march=i686'
as --32 -o /tmp/compHJeO0C.o /tmp/ccG0gt0.s

types of addresses

absolute - does not change, no matter where program placed in memory

relative - assume program is placed in memory at address zero

PC-relative - added to PC to get actual address

linker/loader* may need to adjust relative address but would not need to adjust absolute or PC-relative addresses

* loader places program in memory for execution.
Key task for linker

handle situations where one file uses a symbol located in another file

instructions that reference the symbol contain "holes" that must be filled in
termination

in symbols - defined and exported

out symbols - referenced and imported
1. physically put object code together.

2. adjust address fields in IN₂ & OUT₂ by adding length of OBJ₁.

3. \( \text{IN} = \text{IN}_1 \cup \text{IN}_2 \) (if \( \text{IN}_1 \cap \text{IN}_2 \neq \emptyset \) then report errors.)

4. for \( i = 1 \) to \( 2 \)

   for each entry \( X \) in OUT₁

   insert \( \text{IN}[X] \) into instruction referencing \( X \)

   (if \( X \) not in \( \text{IN}_1 \), put \( X \) in OUT₁)

---

note: assuming addresses in instructions are PC-relative addresses or relative
this is the main program to be linked with another file
this file will call get42 twice and add the two return values together
the other file will supply the get42 function

import get42  # need to import the external function
addi fp,sp  # ie establish a fp (ie by copying the sp)
ldimm r0,4  # allocate a local for the return value
subi sp,r0
call get42  # first call: store return value in r0
ldind r0,-1(fp)
call get42  # second call: add return value to r0
ldind r1,-1(fp)
addi r0,r1
store r0,result  # store sum into result
halt  # halt

export result  # variable to store result

result:
word 0
# this is the code for the second file
# get42 just returns 42
# note that the routine uses r7!
# what are the register saving conventions?
#
    export  get42
get42:
    ldimm   r7,42
    stind   r7,-1(fp)
    ret
Insymbol Section (1 entries)
result 10

Outsymbol Section (2 entries)

get42 5
get42 3

Object Code (11 words)

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>0000ed0b</td>
<td>addi</td>
<td>fp, sp</td>
</tr>
<tr>
<td>00000001</td>
<td>00004000</td>
<td>ldimm</td>
<td>r0, 4</td>
</tr>
<tr>
<td>00000002</td>
<td>0000500c</td>
<td>subi</td>
<td>sp, r0</td>
</tr>
<tr>
<td>00000003</td>
<td>00000000</td>
<td>call</td>
<td>[undefined]</td>
</tr>
<tr>
<td>00000004</td>
<td>ffffff05</td>
<td>ldind</td>
<td>r0, -1(fp)</td>
</tr>
<tr>
<td>00000005</td>
<td>fffffff0f</td>
<td>call</td>
<td>[undefined]</td>
</tr>
<tr>
<td>00000006</td>
<td>fffff005</td>
<td>ldind</td>
<td>r1, -1(fp)</td>
</tr>
<tr>
<td>00000007</td>
<td>0000100b</td>
<td>addi</td>
<td>r0, r1</td>
</tr>
<tr>
<td>00000008</td>
<td>0001002</td>
<td>store</td>
<td>r0, 10</td>
</tr>
<tr>
<td>00000009</td>
<td>00000000</td>
<td>halt</td>
<td></td>
</tr>
<tr>
<td>00000010</td>
<td>00000000</td>
<td>halt</td>
<td></td>
</tr>
</tbody>
</table>
Insymbol Section (1 entries)

get42 → 11

Outsymbol Section (0 entries)

Object Code (3 words)

000000 41 0002a703 ldimm r7, 42
000001 42 fffff76 stind r7, -1(fp)
000002 43 00000010 ret

PC + 3 → set...

6 + stord → 11

stord @ 5 =

4 + stord = 11

stord = 7
collection of object files suitably indexed

i.e. when you use printf you need to be able to quickly determine which object file contains printf

note an object file pulled out of a library may require other object files to be extracted

object file from library may itself have outsymbols
Dynamic Linking

code is loaded into memory upon demand
i.e. while program is running
can be done via Procedure Linkage Table (PLT)
call to dynamically linked function is done via the PLT
initially each PLT entry contains the address of the dynamic linker
the first time a function is called the dynamic linker is actually called
it loads the code and re-sets the PLT entry to be the address of the new code
then the function call is re-executed