Linking

CS520

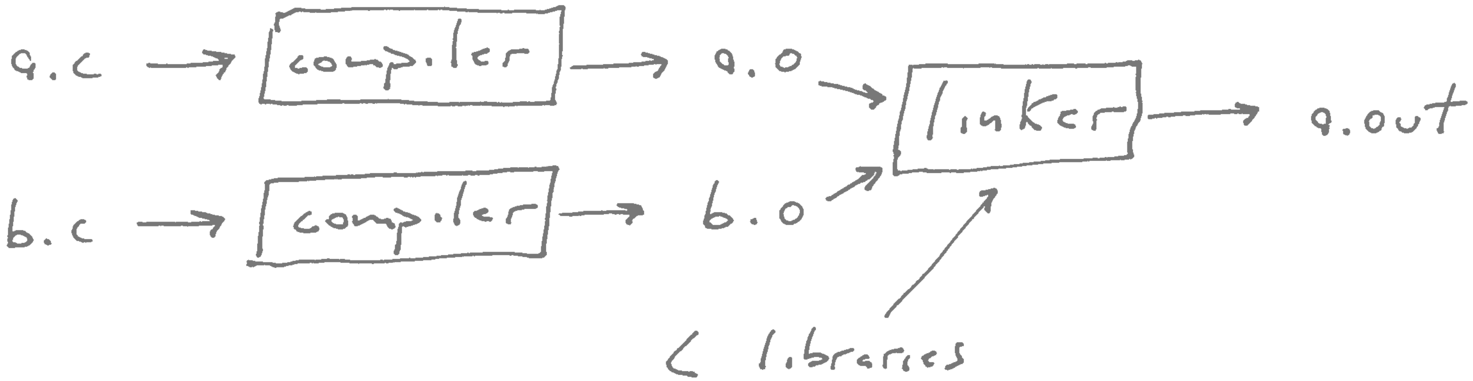
Dept. of Computer Science

Univ. of New Hampshire

Linking

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

ex gcc a.c b.c



gcc $\boxed{-v}$ both.c vm520.c -lpthread

```
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
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/u
sr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-bootstr
ap --enable-shared --enable-threads=posix --enable-checking=release --with-syste
m-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-
object --enable-linker-build-id --enable-languages=c,c++,objc,obj-c++,java,fortr
an,ada,go,lto --enable-plugin --enable-java-awt=gtk --disable-dssi --with-java-h
ome=/usr/lib/jvm/java-1.5.0-gcj-1.5.0.0/jre --enable-libgcj-multifile --enable-j
ava-maintainer-mode --with-ecj-jar=/usr/share/java/eclipse-ecj.jar --disable-lib
java-multilib --with-ppl --with-cloog --with-tune=generic --with-arch=i686 --bui
ld=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/ccG0ogt0.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
GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
ignoring nonexistent directory "/usr/lib/gcc/i686-redhat-linux/4.6.3/include-fix
ed"
ignoring nonexistent directory "/usr/lib/gcc/i686-redhat-linux/4.6.3/../../../..
/i686-redhat-linux/include"
#include "..." search starts here:
#include <...> search starts here:
 /usr/lib/gcc/i686-redhat-linux/4.6.3/include
 /usr/local/include
 /usr/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
GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
Compiler executable checksum: 3b913ff0ce1aa5afc1491f952f749bb7
COLLECT_GCC_OPTIONS='-v' '-mtune=generic' '-march=i686'
 as --32 -o /tmp/ccw6qTzo.o /tmp/ccG0ogt0.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/ccG0ogt0.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
GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
ignoring nonexistent directory "/usr/lib/gcc/i686-redhat-linux/4.6.3/include-fix
ed"
ignoring nonexistent directory "/usr/lib/gcc/i686-redhat-linux/4.6.3/../../../..
/i686-redhat-linux/include"
#include "..." search starts here:
#include <...> search starts here:
 /usr/lib/gcc/i686-redhat-linux/4.6.3/include
 /usr/local/include
 /usr/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
GGC heuristics: --param ggc-min-expand=100 --param ggc-min-heapsize=131072
Compiler executable checksum: 3b913ff0ce1aa5afc1491f952f749bb7
COLLECT_GCC_OPTIONS='-v' '-mtune=generic' '-march=i686'
 as --32 -o /tmp/ccHJheQM.o /tmp/ccG0ogt0.s
COMPILER_PATH=/usr/libexec/gcc/i686-redhat-linux/4.6.3/:/usr/libexec/gcc/i686-re
dhat-linux/4.6.3/:/usr/libexec/gcc/i686-redhat-linux/:/usr/lib/gcc/i686-redhat-l
```

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.

<u>Key task for linker</u>

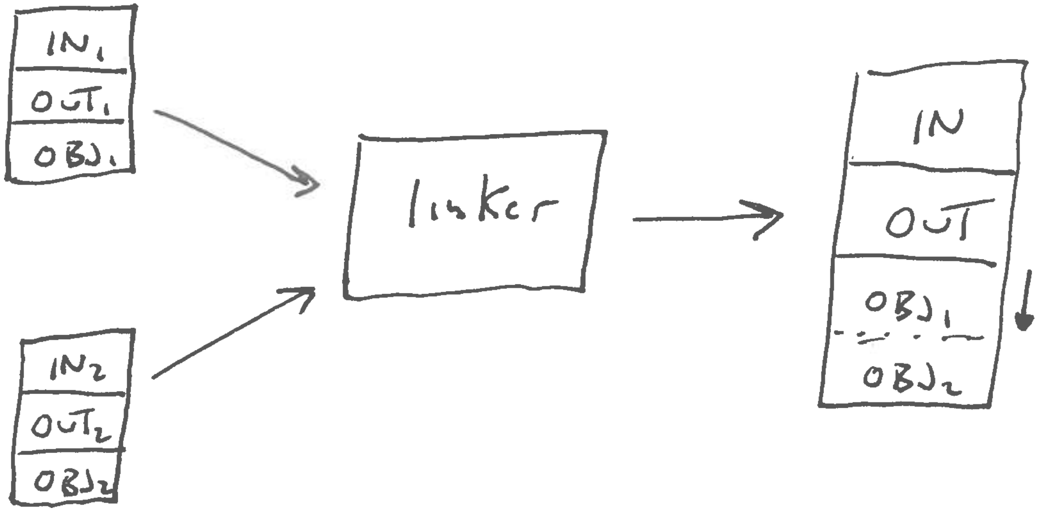
handle situations where one file uses
a symbol located in another file

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

terminology

insymbols – defined and exported

outsymbols – referenced and imported


IN₁
OUT₁
OBJ₁
IN₂
OUT₂
OBJ₂
linker
IN
OUT
OBJ₁
OBJ₂

1. physically put object code together.

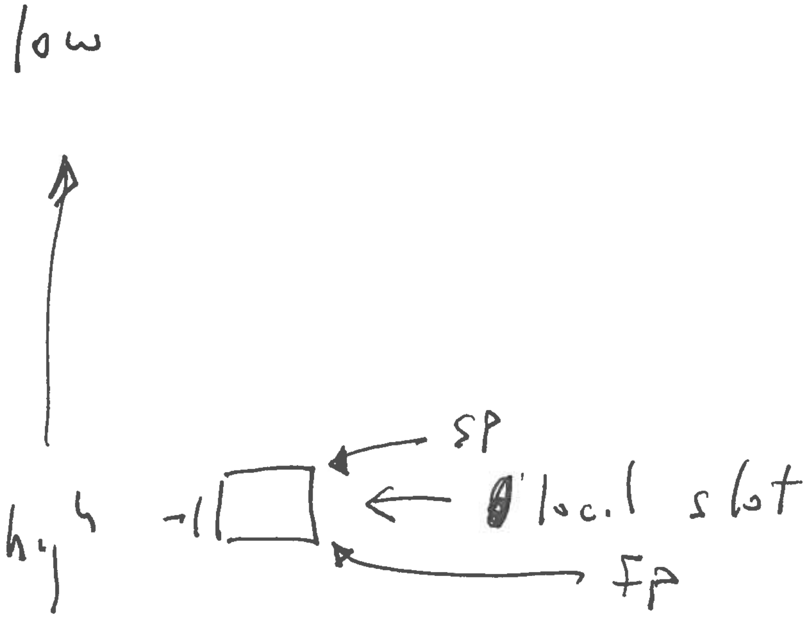
2. adjust address fields in IN_2 & OUT_2 by adding length of OBJ_1

3. $IN = IN_1 \cup IN_2$ (if $IN_1 \cap IN_2 \neq \phi$ then report errors.)

4.
```
for i = 1 to 2
    for each entry X in $OUT_i$
        insert $IN[X]$ into instruction referencing X
        (if X not in IN, put X in OUT.)
```

<u>note</u>: 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

00007
00005

Object Code (11 words)

```
0000000     0000ed0b     addi        fp, sp
0000001     00004003     ldimm       r0, 4
0000002     00000e0c     subi        sp, r0
0000003     0000000f     call        [undefined]
0000004     ffffd005     ldind       r0, -1(fp)
0000005     0000000f     call        [undefined]
0000006     ffffd105     ldind       r1, -1(fp)
0000007     0000100b     addi        r0, r1
0000008     00001002     store       r0, 10
0000009     00000000     halt
0000010     00000000     halt
```

holes

Insymbols

result 10

get42 11

Outsymbols

<none>

Insymbol Section (1 entries)

get42 ~~0~~ 11

Outsymbol Section (0 entries)

✓

Object Code (3 words)

```
~~0000000~~ 11  0002a703    ldimm       r7, 42
~~0000001~~ 12  ffffd706    stind       r7, -1(fp)
~~0000002~~ 13  00000010    ret
```

PC + stored addr → actual addr

6 + stored → 11

stored = 5

4 + stored = 11

stored = 7

libraries

collection of object files suitably indexed

ie 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