Binding Examples

Binding

The process of determining where in the physical memory the subroutine should go and making the reference in the main routine point to the subroutine.

Binding Model

- Coding the program
- Translating into object module
  - Compiler, Assembler, Interpreter
- Linking with other modules
- Loading into primary memory
- Running the process

Binding at Coding Time

- In a hypothetical embedded system
- Could locate things by hand
  - Put main module at location 100
  - Put the subroutine at location 500
- ORG assembler directive
  - In main module make call to ORG 500 to point to subroutine code
  - In subroutine make call to ORG 100 to point to main module.
- Dedicated hardware locations

Binding at Linking Time

- Subroutines we really don't care where they are located.
  - Use symbolic names
  - Assembler outputs object module that includes references that need to be fixed or linked.
  - Linker loads our module, processes it and finds the items it needs to find. Once found it will go back and link the references in the main module to the addresses found.

Binding at Linking Time

- Benefits of binding at link time?
  - Flexibility - We don’t change code
- Cons?
  - Objects are bigger. Carry references
  - Time

Compile Time Binding

```
int main()
{
  int x = 0xDEAD;
  int y = 0xBEEF;
  int z = x + y;
  return z;
}
```

Compile Time Binding

```
[tbakker@omega ~]$ objdump -d main

main:     file format elf64-x86-64

Disassembly of section .text:

00000000004000b0 <main>:
  4000b0: 55                      push   %rbp
  4000b1: 48 89 e5                mov    %rsp,%rbp
  4000b4: c7 45 f4 ad de 00 00    movl   $0xdead,0xfffffffffffffff4(%rbp)
  4000bb: c7 45 f8 ef be 00 00    movl   $0xbeef,0xfffffffffffffff8(%rbp)
  4000c2: 8b 45 f8                mov    0xfffffffffffffff8(%rbp),%eax
  4000c5: 03 45 f4                add    0xfffffffffffffff4(%rbp),%eax
  4000c8: 89 45 fc                mov    %eax,0xfffffffffffffffc(%rbp)
  4000cb: 8b 45 fc                mov    0xfffffffffffffffc(%rbp),%eax
  4000ce: c9                      leaveq
  4000cf: c3                      retq
```

Variables x, y, and z have an absolute address

Link Time Binding

```
#include "header.h"

int main()
{
  int x = 0xDEAD;
  int y = 0xBEEF;
  int z = 0;

  z = add_numbers( x, y );

  return z;
}
```

Link Time Binding

add_numbers has not been resolved to an address

```
main.o:     file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   48 83 ec 10             sub    $0x10,%rsp
   8:   c7 45 f4 ad de 00 00    movl   $0xdead,0xfffffffffffffff4(%rbp)
   f:   c7 45 f8 ef be 00 00    movl   $0xbeef,0xfffffffffffffff8(%rbp)
  16:   c7 45 fc 00 00 00 00    movl   $0x0,0xfffffffffffffffc(%rbp)
  1d:   8b 75 f8                mov    0xfffffffffffffff8(%rbp),%esi
  20:   8b 7d f4                mov    0xfffffffffffffff4(%rbp),%edi
  23:   e8 00 00 00 00          call   add_numbers
  28:   89 45 fc                mov    %eax,0xfffffffffffffffc(%rbp)
  2b:   8b 45 fc                mov    0xfffffffffffffffc(%rbp),%eax
  2e:   c9                      leaveq
  2f:   c3                      retq
```

Link Time Binding

We can use nm to look at the symbols in our object file

U means the symbol is undefined in our object file

```
[tbakker@omega ~]$ nm main.o
                 U add_numbers
0000000000000000 T main
```

Link Time Binding

```
[tbakker@omega ~]$ readelf --relocs main.o

Relocation section '.rela.text' at offset 0x540 contains 1 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000000024  000900000002 R_X86_64_PC32     0000000000000000 add_numbers + fffffffffffffffc
```

The compiler leaves behind a *relocation* (of type R_X86_64_PC32) which is saying "in the final binary, patch the value at offset 0x24 in this object file with the address of symbol add_numbers.

Runtime Binding

```
[tbakker@omega ~]$ gcc -shared header.c -fPIC -o libaddnumbers.so
[tbakker@omega ~]$ gcc main.c -laddnumbers -L. -o main -nostdlib
```

Let’s compile our example and tell the compiler we are going to use a shared library

Runtime Binding

The compiler has told the linker and loader that add_numbers can be found using the procedure linkage table which will then point to the global offset table

```
[tbakker@omega ~]$ objdump -d main

main:     file format elf64-x86-64

Disassembly of section .plt:

00000000004002e0 <add_numbers@plt-0x10>:
  4002e0: ff 35 b2 01 20 00      pushq  2097586(%rip)        # 600498 <_GLOBAL_OFFSET_TABLE_+0x8>
  4002e6: ff 25 b4 01 20 00      jmpq   *2097588(%rip)        # 6004a0 <_GLOBAL_OFFSET_TABLE_+0x10>
  4002ec: 0f 1f 40 00            nopl   0x0(%rax)

00000000004002f0 <add_numbers@plt>:
  4002f0: ff 25 b2 01 20 00      jmpq   *2097586(%rip)        # 6004a8 <_GLOBAL_OFFSET_TABLE_+0x18>
  4002f6: 68 00 00 00 00         pushq  $0x0
  4002fb: e9 e0 ff ff ff         jmpq   4002e0 <add_numbers@plt-0x10>
Disassembly of section .text:

0000000000400300 <main>:
  400300: 55                     push   %rbp
  400301: 48 89 e5               mov    %rsp,%rbp
  400304: 48 83 ec 10            sub    $0x10,%rsp
  400308: c7 45 f4 ad de 00 00   movl   $0xdead,0xfffffffffffffff4(%rbp)
  40030f: c7 45 f8 ef be 00 00   movl   $0xbeef,0xfffffffffffffff8(%rbp)
  400316: c7 45 fc 00 00 00 00   movl   $0x0,0xfffffffffffffffc(%rbp)
  40031d: 8b 75 f8               mov    0xfffffffffffffff8(%rbp),%esi
  400320: 8b 7d f4               mov    0xfffffffffffffff4(%rbp),%edi
  400323: e8 c8 ff ff ff         callq  4002f0 <add_numbers@plt>
  400328: 89 45 fc               mov    %eax,0xfffffffffffffffc(%rbp)
  40032b: 8b 45 fc               mov    0xfffffffffffffffc(%rbp),%eax
  40032e: c9                     leaveq
  40032f: c3                     retq
```