Topic 5 – Loaders

Key Ideas

- This lecture is concerned with the issues raised when *loading a program into memory and running it.*
- What is a loader?
- relocation
  - what is it?
  - why do it?
  - when do we do it?
- What is object code?
- What is MERL file format?

Loaders

What is Loading?

- You now know how to convert an assembly language program to a machine language program (via an assembler).
- *But how do you actually run the program?*
- Some other program must be responsible for copying it from secondary storage (HDD or SSD) into primary storage (RAM) and then starting to execute the instructions in that program.
  - Processors can only execute code located in RAM.
- The *loader* is the program responsible for loading other programs into primary storage and preparing them for execution.

Loaders

Version 1.0 of Loading

```
repeat:
        P ← next program to run
load:   copy P to memory starting at 0x0
        jalr $0
        beq $0,$0,repeat
```

- *Key Problem:* programs are generally not loaded into memory at location 0x0
- *Solution:* addresses need to be adjusted depending on where the program is loaded.

Loaders

Version 2.0 of Loading

```
repeat:
        P ← next program to run
load:   $3 ← loader(P)
        jalr $3
        beq $0, $0, repeat
```

The loader takes program P as input and

- *finds a location* in RAM for P (i.e. a starting address α)
- *copies* P into RAM starting at address α
- returns α as the start address
- *starts executing* the code at that address

Loaders

Version 2.0 of Loading

```
loader(P):
 ;; input the machine code for P (called text)
 .word P(1) … .word P(k)

 ;; determine size of P and a location in RAM
 n = k + space for heap and stack
 a = first addr of n contiguous words of RAM

 ;; copy P into RAM and start executing P
 for i = 1..k
    MEM[a + (i-1)*4] ← P(i)
 $30 = a + 4*n           ;; set addr of stack
 $3 = a                  ;; $3 = start of P
 jalr $3                 ;; start executing P
```

Loaders

Version 2.0: the Details

- determine the length program,
- *allocate RAM* starting at, say address α, for the code and a stack (and possibly a heap)
- *copy the program* from secondary storage (HDD or SSD) into primary storage (RAM) starting at α,
- possibly set up the program, e.g. pass parameters to the program by placing them on P's stack
- load the address, α, into some register, say $3.
- *executes the program* (jalr $3)
- possibly do some work at the end, e.g. twoints will print out all the register values

Relocation

Changing a Program’s Location

- *Key Problem: If a program gets shifted around in memory (relocated) it affects the values of certain labels*

Assembly Language

```
 0   lis $3
 4   .word ft
 8   lw $3, 0($3)
 c   jr $31
10   ft: .word 41
```

Relocated Machine Code

```
α+0    0x0000 1814
α+4    0x0000 0010
α+8    0x8c63 0000
α+c    0x03e0 0008
α+10   0x0000 0029
```

- Initially the label `ft` referred to address 0x10 but when the code gets relocated it should refer to address α + 0x10

Relocation

Which Values Get Changed?

- *When .word refers to a location, you must add α to it.*

4	`.word ft`	$\alpha+4$	0x0000 00~~10~~
	`. . .`		
10	`ft: .word 41`	$\alpha+10$	0x0000 0029

- When .word refers to a constant: *do nothing.*

```
0   lis $1
4   .word 1
8   sub $2, $2, $1
```

- For beq, bne: *do nothing,* they jump forward or backward *i* instructions not to a certain address.
- All other instructions: *do nothing.*

Relocation

Finding those Values

- Problem: Machine code is just a sequence of bits
- Question: *How do we know which words are addresses that must be adjusted* (vs. constants which do not need to be adjusted).
- Answer: We don’t know.
- Approach: We *must augment the machine code* with information about which words need adjusting if the code is relocated.
- This modification of machine code called *object code*.

MERL

What is MERL?

- MERL is the *format for a program's machine code that includes information about what words need to be adjusted if the program in loaded into a location other than 0x0.*
- MERL = **M**IPS **E**xecutable **R**elocatable **L**inkable file
- It's CS241's own simplified format.
- Aside: Linux uses ELF and Linux provides tools (i.e. commands) like readelf that understand this format.
- MERL has three parts:
  1. a header
  2. the MIPS machine code
  3. the relocation information (with more coming later).

MERL

Part 1: The MERL Header

The header consists of three words (12 bytes)

1. *Cookie:*
   - the value is 0x1000 0002
   - it identifies the type of file
   - it can be interpreted as the MIPS instruction beq $0 ,$0, 2, which would skip over the header
2. *FileLength:* the length of the MERL file in bytes
3. *CodeLength:* the length of the header plus the MIPS machine code (also the offset to the Relocation Table)

MERL

Part 2: The MIPS Program

- *This is the program in MIPS machine code.*
- It works correctly if the program is loaded into RAM location 0x0c (i.e. the location immediately following the header).

Part 3: Relocation and External Symbol Table

- *It contains relocation information.*
- Format: the word 0x01 followed by the location of a word in the MERL file that needs to be adjusted if the file is relocated.
- It also contains the external symbol definition and external symbol reference (which we'll discuss later).

MERL Example

Assembly	Addr	MERL file	Comments
`beq $0, $0, 2`	0x00	0x1000 0002	; 1 - Header
`.word endfile`	0x04	0x0000 003c	; file length
`.word endcode`	0x08	0x0000 002c	; code + hdr
`lis $3`	0x0c	0x0000 1814	; 2 - MIPS
`.word 0x0abc`	0x10	0x0000 0abc	; no reloc
`lis $1`	0x14	0x0000 0814	
`r1: .word A`	0x18	0x0000 0024	; reloc1
`jr $1`	0x1c	0x0020 0008	
`B: jr $31`	0x20	0x03e0 0008	
`A: beq $0 ,$0, B`	0x24	0x1000 fffe	
`r2: .word B`	0x28	0x0000 0020	; reloc2

MERL Example

Assembly	Addr	MERL file	Comments
`endcode:`			; 3 - Relocation Table
`    .word 0x1`	0x2c	0x0000 0001	; format code
`    .word r1`	0x30	0x0000 0018	; relocation addr
`    .word 0x1`	0x34	0x0000 0001	; format code
`    .word r2`	0x38	0x0000 0028	; relocation addr
`endfile:`			

Comments about Relocation

- the instructions at r1: and r2: need to be relocated because **A** and **B** are addresses of instructions (not constants)
- the instruction at no reloc does not, 0x0abc is a constant

Loader Pseudocode

Loading in a CS 241 MIPS Program

```
read in MERL header
$\alpha$ = findFreeRAM(codeLength)          // find space
for each instruction                         // copy into RAM
    MEM[$\alpha$ + i] = instruction
for each relocation entry                    // fix addresses
    MEM[$\alpha$ + location] += $\alpha$
place $\alpha$ into $29                      // start executing
jalr $29
```

A MERL Assembler

Modifications to Create a MERL Assembler

For Pass 1

- record the size of the file
- start counting addresses at 0x0c (rather than 0x0)
- when you encounter a `.word <label>` instruction
  - record the location

For Pass 2

- output the header
- output the MIPS machine code (already do this step)
- output the relocation table

Loader Notes

Loading in CS 241 MIPS Program

- Notice how mips.twoints works:

```
% java mips.twoints
Usage: java mips.twoints <filename> [load_address]
```

- i.e. you can select the load address

Official Description of MERL

- The official description of the MERL format is in the CS241 web site in the Resource / Material for Assignment 4 (and beyond) section. https://www.student.cs.uwaterloo.ca/%7Ecs241/merl/merl.html