11.29

Contents

- Cache problem last week
- RIP relative addressing
- Dynamic linking and PIC code
- Processes and ECF

Cache problem last week

- Method 1 (blocking, not strided) is faster, on my machine
- Tried to do some profiling...

- Result maybe different on different machine

- **Note:** The difference of memory accessing pattern is important!

RIP relative addressing

- Relative to "**NEXT INSTRUCTION**"
- However, linker only knows the address of relocation entry
  - Use "addend"
- Example: P480-P481, relocating function "sum"
  - Relocation entry's "addend" field is 4

RIP relative addressing

```
Sym. Name + Addend
global_func - 4
```

- Relative to "**NEXT INSTRUCTION**"
- Example: P480-P481, relocating function "sum"
  - Relocation entry's "addend" field is -4
- Another example
  - Function "global_func"
  - Addend is also –4

```
  int t = global_func(argc);
24:   8b 45 ec                mov    -0x14(%rbp),%eax
27:   89 c7                   mov    %eax,%edi
29:   e8 00 00 00 00          callq  2e <main+0x19>
2e:   89 45 fc                mov    %eax,-0x4(%rbp)
```

```
Relocation section '.rela.text' at offset 0x624c8 contains 5 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
00000000002a  001100000004 R_X86_64_PLT32    0000000000000000 global_func - 4
```

- "call" takes 4 byte offset value as input

Problem

- The offset from RIP in these instructions is limited to 32 bits.
- Which means it only works for binaries less than **2GB**
  - Use "–mcmodel=small" to tell *gcc* about this
- What will happen when the binary's size is larger than 2GB?

Code

```
 1 int global_arr[100] = {2, 3};
 2 static int static_arr[100] = {9, 7};
 3 int global_arr_big[50000] = {5, 6};
 4 static int static_arr_big[50000] = {10, 20};
 5
 6 int global_func(int param)
 7 {
 8     return param * 10;
 9 }
10
11 int main(int argc, const char* argv[])
12 {
13     int t = global_func(argc);
14     t += global_arr[7];
15     t += static_arr[7];
16     t += global_arr_big[7];
17     t += static_arr_big[7];
18     return t;
19 }
```

Without -fPIC, with –mcmodel=large

- Use "movabs"
- Generate a relocation entry

```
  int t = global_func(argc);
24: 8b 45 ec                    mov    -0x14(%rbp),%eax
27: 89 c7                       mov    %eax,%edi
29: b8 00 00 00 00              mov    $0x0,%eax
2e: 48 ba 00 00 00 00 00        movabs $0x0,%rdx
35: 00 00 00
38: ff d2                       callq  *%rdx
3a: 89 45 fc                    mov    %eax,-0x4(%rbp)
  t += global_arr[7];
```

PIC code

- Use PLT and GOT when relocating instead of real address
- Still the same code, compiled with "–mcmodel=small –fPIC"

```
Relocation section '.rela.text' at offset 0x624c8 contains 5 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
00000000002a  001100000004 R_X86_64_PLT32    0000000000000000 global_func - 4
000000000034  000f0000002a R_X86_64_REX_GOTP 0000000000000000 global_arr - 4
000000000040  000300000002 R_X86_64_PC32     0000000000000000 .data + 1b8
00000000004a  00100000002a R_X86_64_REX_GOTP 0000000000000340 global_arr_big - 4
000000000056  000300000002 R_X86_64_PC32     0000000000000000 .data + 31098
```

- **Note:** static data is relocated without GOT and PIC
- ** relative addressing still can be used here (small model)

PIC code with large model

- A little bit complicated here, because of no relative addressing
  - In the medium and large code models a register has to be allocated to hold the address of the GOT in position-independent objects, because the AMD64 ISA does not support an immediate displacement larger than 32 bits.
- Static data
  - Relocate with "symbol + addend – GOT", as cannot access GOT immediately
- Global data
  - Relocate with "symbol's offset in GOT"
- Global functions
  - Relocate with "symbol's offset in PLT – GOT", as cannot access PLT immediately

How does PLT and GOT works

- GOT is supposed to store "effective" address
- However, due to "lazy binding", the addresses are not "effective".
- Therefore, we need a "watchdog" or "reception" to deal with the situation
  - First time: register at "watchdog", "watchdog" turns to dynamic-linker to get the correct address.
  - Then, "watchdog" modify the GOT
  - After: "watchdog" tells you the correct address from GOT.
  - "watchdog" is PLT

Processes and signals

- ECF
  - Interrupt, fault, trap, abort. ==> differences?
- Process model
  - Process v.s. program.
- fork() & execve()
  - Characteristics of fork()
  - How to run a new program
- Signals
  - How many signals? Why signals?
- System calls
  - Remember the usage --> For lab and exam

Quiz – what's the output?

- List all possible outputs.

```c
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("hello");
    pid_t pid = fork();
    if(pid == 0)
    {
        printf(" from child!\n");
        _exit(0);
    }
    else
    {
        printf(" from parent!\n");
        _exit(0);
    }
}
```

How about this?

- List all possible outputs.

```c
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("hello\n");
    pid_t pid = fork();
    if(pid == 0)
    {
        printf(" from child!\n");
        _exit(0);
    }
    else
    {
        printf(" from parent!\n");
        _exit(0);
    }
}
```

Exercise

8． 下列关于静态库链接的描述中，错误的是（　　）

A． 链接时，链接器只拷贝静态库中被程序引用的目标模块

B． 使用库的一般准则是将它们放在命令行的结尾

C． 如果库不是相互独立的，那么它们必须排序

D． 每个库在命令行只须出现一次即可

Exercise

9. 在 foo.c 文件中的函数外，如果添加如下一条语句：

```
static int count = 0xdeadbeef;
```

那么它在编译为 foo.o 后，会影响到 ELF 可重定位目标文件中的除 .text 以外的那些字段？（　　）

A. .rodata

B. .data,.symtab,

C. .data,.symtab,.rel.data

D. .rodata,.symtab,.rel.data

Exercise

10. 在系统调用成功的情况下，下列代码会输出几个 hello？（　　）

```
void doit()
{
    if ( fork() == 0 ) {
        printf("hello\n");
        fork();
    }
    return ;
}

int main()
{
    doit();
    printf("hello\n");
    exit(0) ;
}
```

A. 3　　B. 4　　C. 5　　D. 6

Exercise

11．下列说法中哪一个是错误的？（　　）

A．中断一定是异步发生的

B．异常处理程序一定运行在内核模式下

C．故障处理一定返回到当前指令

D．陷阱一定是同步发生的