ASPLOS
– Architectural Support for Programming Languages and Operating Systems

- The life cycle of a program
- Full-system virtualization
- OS API – system calls
- What do you still lack?

Bug fixed

- include/device/video.h

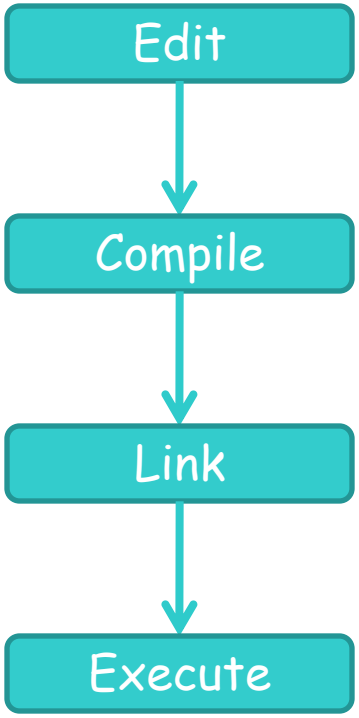
```
static inline void
draw_pixel(int x, int y, int color) {
-    assert(x >= 0 && y >= 0 && x < SCR_HEIGHT && SCR_WIDTH);
+    assert(x >= 0 && y >= 0 && x < SCR_HEIGHT && y < SCR_WIDTH);
```

- Fixing it will not effect the correstness of your program.
  - may expose some of your faults

- The life cycle of a program
  - from source to executable

Develop a program

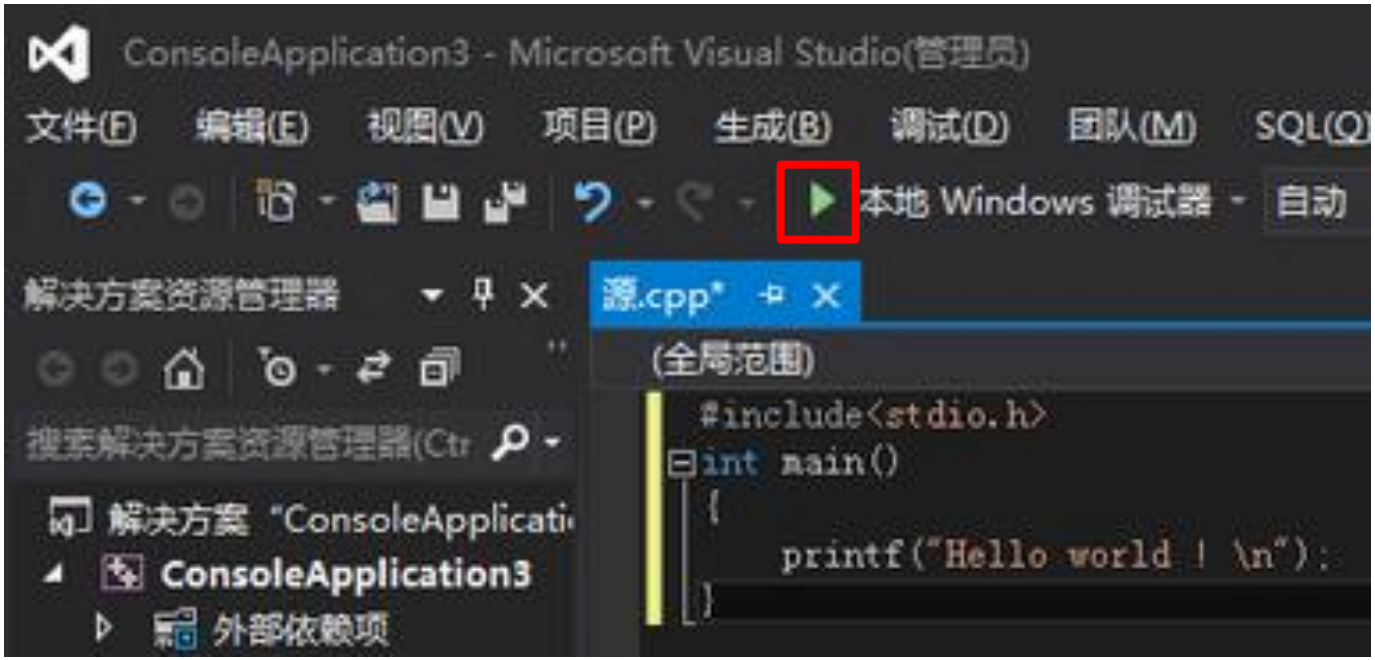
- Can you explain this process with more details?
  - to distinguish yourself from some coding farmers?
- Why not?
  - You may not have a closer look.
  - Now it is the time!



Compilation

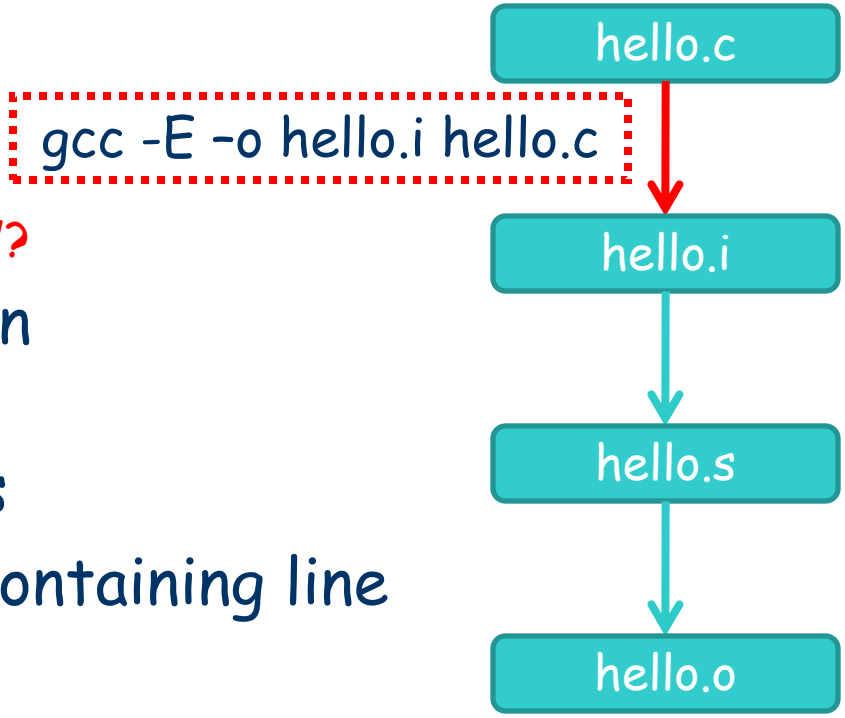
- What do they mean?

gcc hello.c



From source to executable

- pre-process
  - macro expansion
    - #include, #define
      - What is "包含头文件"?
  - conditional compilation
    - #ifdef, #else, #endif
  - throw away comments
  - concatenate strings containing line breaking character
- Can you write a pre-processor?



Powerful macro

http://gcc.gnu.org/onlinedocs/cpp/Macros.html

- function-like macro
  - #define max(a, b) ((a) > (b) ? (a) : (b))
    - What if "max(++x, ++y)" ?
    - Can you modify the macro definition to fix this problem?
- stringifition
  - #define str(x) #x
- concatenation
  - #define glue(a, b) a##b
    - see include/adt/linklist.h for details

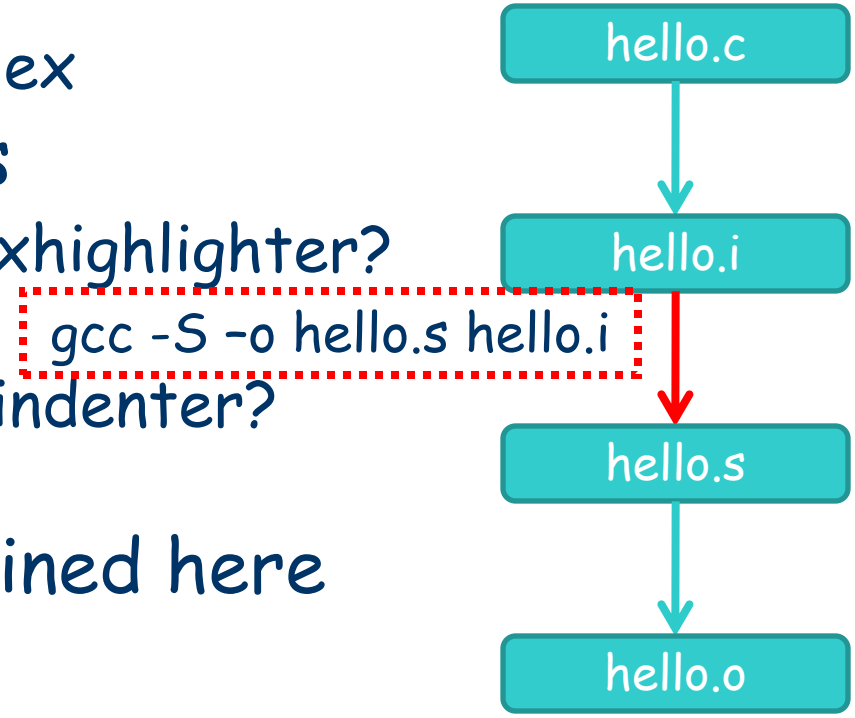
Powerful macro (cont.)

http://cslab.nju.edu.cn/acmicpc/wiki/Challenge13

```
#include <stdio.h>
#include <math.h>
#define clear 1;if(c>=11){c=0;sscanf(_,"%lf%c",&r,&c);while(*++_-c);}\
  else if(argc>=4&&!main(4-(*_++=='('),argv))_++;g:c+=
#define puts(d,e) return 0;}{double a;int b;char c=(argc<4?d)&15;\
  b=(*_%__LINE__+7)%9*(3*e>>c&1);c+=
#define I(d) (r);if(argc<4&&*#d==*_){a=r;r=usage?r*a:r+a;goto g;}c=c
#define return if(argc==2)printf("%f\n",r);return argc>=4+
#define usage main(4-__LINE__/26,argv)
#define calculator *_*(int)
#define l (r);r=--b?r:
#define _ argv[1]
#define x

double r;
        int main(int argc,char** argv){
  if(argc<2){
    puts(
      usage: calculator 11/26+222/31
      +~~~~~~~~~~~~~~~~~~~~~~~~~calculator-\
      !                            7.584,367 )
      +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+
      ! clear ! 0 ||l   -x  l   tan  I (/) |
      +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+
      ! 1 | 2 | 3 ||l  1/x  l   cos  I (*) |
      +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+
      ! 4 | 5 | 6 ||l  exp  l  sqrt  I (+) |
      +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~+
      ! 7 | 8 | 9 ||l  sin  l   log  I (-) |
      +~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~(0
    );
  }
  return 0;
}
```

From source to executable (cont.)

- compile
  - this is "rather" complex
- some guide questions
  - can you write a syntaxhighlighter?
    - lexical analysis
  - can you write a code indenter?
    - syntax analysis
- details are not explained here
  - semantic analysis
  - code generation



Operational semantic

https://class.coursera.org/compilers-selfservice

- use 人类的文明 to formally describe the interpretation of the language

$$\frac{so, E, S \vdash e_1 : v_1, S_1 \qquad so, E, S_1 \vdash e_2 : v_2, S_2}{so, E, S \vdash e_1 + e_2 : v_1 + v_2, S_2}$$

Code generation

- Translate the C code into assembly
  - For C, this is straightforward.
- variable ➔ address_of_variable
- function body ➔ instructions

Variables

- In assembly, every variable is a symbol.
  - x = 1 ➔ M[&x] = 1;
    - movl $1, address_of_x
  - a = b + c; ➔ M[&a] = M[&b] + M[&c];
    - movl address_of_b, %eax
    - addl address_of_c, %eax
    - movl %eax, address_of_a
- symbol table
  - use readelf to see the symbol table

Function & pointer

- Function names are symbols, too.
  - fp = fun ➔ movl $address_of_fun, address_of_fp
  - fun() ➔ call address_of_fun
- Pointers are nothing special.
  - ***(p + 4) = 1 ➔
    - movl address_of_p, %eax
    - movl 4(%eax), %eax
    - movl (%eax), %eax
    - movl $1, (%eax)

Array & struture

- Array and structure are nothing.
  - a[n] = 1 ➔
    - movl address_of_n, %eax
    - movl $1, (address_of_a, %eax, 4)
  - s.age = 18 ➔
    - movl $18, offset_of_age(address_of_s)
  - t = s ➔
    - movl $address_of_s, %esi
    - movl $address_of_t, %edi
    - movl $(sizeof(struct stu)), %ecx
    - rep movsl %ds:(%esi),%es:(%edi)

```
struct stu {
    char name [80];
    double score;
    int age;
} s, t;
```

Branch & loop

- Conditional branch and loop are jumps.
  - if(a == 1) … ➔
    - R[eflags] = update_flags(1 - a);
    - if(R[eflags] & ELF_ZF) R[eip] = address_of_if_block;

    - cmp $1, address_of_a
    - jz address_of_if_block

Recursion

- Recursion is nothing.

```
########### if (n == 1) ###########
        cmpl    $1, 20(%ebp)
        jne     .L2
        movl    12(%ebp), %eax
        movl    %eax, 8(%esp)
        movl    8(%ebp), %eax
        movl    %eax, 4(%esp)
        movl    $.LC0, (%esp)
        call    printf
        jmp     .L1
############# else #############
.L2:
######## hanoi(a, c, b, n - 1) ########
        movl    20(%ebp), %eax
        subl    $1, %eax
        movl    %eax, 12(%esp)
        movl    12(%ebp), %eax
        movl    %eax, 8(%esp)
        movl    16(%ebp), %eax
        movl    %eax, 4(%esp)
        movl    8(%ebp), %eax
        movl    %eax, (%esp)
        call    hanoi
######## hanoi(a, c, b, n - 1) ########
        movl    $1, 12(%esp)
        movl    16(%ebp), %eax
        movl    %eax, 8(%esp)
        movl    12(%ebp), %eax
        movl    %eax, 4(%esp)
        movl    8(%ebp), %eax
        movl    %eax, (%esp)
        call    hanoi
######## hanoi(a, c, b, n - 1) ########
        movl    20(%ebp), %eax
        subl    $1, %eax
        movl    %eax, 12(%esp)
        movl    8(%ebp), %eax
        movl    %eax, 8(%esp)
        movl    12(%ebp), %eax
        movl    %eax, 4(%esp)
        movl    16(%ebp), %eax
        movl    %eax, (%esp)
        call    hanoi
############# end #############
.L1:
        leave
        ret
```

Portable assembly language

- Given C source code, can you translate it into assembly manually?
- C was designed to be a "portable assembly language".
  - hardware-friendly
  - directly translation to machine instructsions
    - inline assembly
  - minimal run-time support
- That is why C is widely used in system programming.

Other issues in code generation

- Register allocation
  - determine which variable uses which register
  - graph coloring
  - the "register" keyword
- Optimization
  - constant propagation: a = 1 + 2; b = a * 3;
  - redunant code elimination: x = 1; x = 2; x = 3;
  - invariant variable in loop: while(...) { x = 0; ... }
  - data flow analysis
    - reachability, active variable, common sub-expression...
  - ...
- Details are not explained.

The "volatile" keyword

- Prevent the compiler from optimizing certain variable.
- Why use it?

```
uint8_t *p = (uint8_t *)0x80325320;
*p = 0;
while(*p != 0xff);

*p = 0x33;
*p = 0x34;
*p = 0x86;
```

```
uint8_t *p = (uint8_t *)0x80325320;
*p = 0;
while(0);
```

The "volatile" keyword

- Prevent the compiler from optimizing certain variable.
- Why use it?

```
uint8_t *p = (uint8_t *)0x80325320;
*p = 0;
while(*p != 0xff);

*p = 0x33;
*p = 0x34;
*p = 0x86;
```

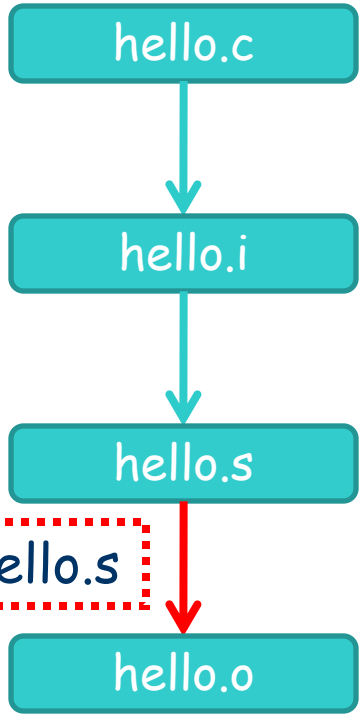
```
uint8_t *p = (uint8_t *)DEV_ADDR;
*p = DEV_INIT;
while(*p != DEV_READY);

*p = DEV_CMD1;
*p = DEV_CMD2;
*p = DEV_CMD3;
```

What about C++?

- How to implement the following?
  - encapsulation
  - operator overloading
  - inheretance & polymorphism
  - template
  - exception
- They are nothing to do with the machine.
- Try to implement a compiler for OOL in the future to have a closer look.

From source to executable (cont.)

- assemble
  - translate assembly into machine instructions
  - bijection
    - something like Huffman code
- gcc calls "as" (GNU assembler) to perform assembling. gcc -c -o hello.o hello.s
- Can you implement an asserbler?



What do we get now?

- hello.o
- It contains some binary machine instructions.
  - use “objdump” to disassemble the binary
  - use “readelf” to see more information

- Is hello.o executable? Why?

Linking error

小百合系版"关于VS2010中的C++在编译过程中出现的LNK2005错误"

- What happen?

由于A.cpp、B.cpp、C.cpp等不同的cpp文件中需要同时调用一些全局变量，自定义宏，自定义函数等，因此我参考书上的作法，将这些变量、宏和函数统一写到了一个自定义的headline.h头文件中，然后再将所有的A.cpp、B.cpp和C.cpp文件中同时include "headline.h",结果经过编译时，就出现了大量的LNK2005 errors，然后经过查阅相关C++书籍和网络，得知在自定义的headline.h头文件的最开始，写入#pragma once命令，可以防止头文件被多次包含，可是楼主试了，仍然不灵；于是我又利用#ifndef、#define和#endif指令，可是结果仍然如出一辙。

①在楼主自定义的headline.h头文件中，关于我对#pragma once命令的使用：

```
#pragma once
int Lx=50,Ly=50,Lz=50;   /*不同cpp文件中需要调用的全局变量*/
#define PI 3.14159265;   /*不同cpp文件中需要调用的自定义宏*/
int nint(double x)       /*不同cpp文件中需要调用的自定义函数*/
{
    return (int)(x+0.5);
}
```

What does linking do?

- relocate *.o files
  - every *.o file starts at address 0
- determine the addresses of external symbols
  - "fill in the blanks" in *.o files
  - report unresolved symbols, re-defined symbols
- This is static linking.
  - gcc (indirectly) calls "ld" (GNU linker) to performing static linking.

Decleration & Definition

- Decleration
  - It is defined somewhere, but it will be used here.
- Definition
  - It is defined here.
- How to detect linking error?
  - symbol table
    - generated at compile time
    - can be seen by readelf

Linking

```
extern int x;
int main() {
    x = 1;
    srand(0);
    printf("%s:srand = %p, system = %p\n", __FUNCTION__, srand, system);
    return 0;
}
```

```
00000000 <main>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 e4 f0                and    $0xfffffff0,%esp
   6:   83 ec 10                sub    $0x10,%esp
   9:   c7 05 00 00 00 00 01    movl   $0x1,0x0
  10:   00 00 00
  13:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
  1a:   e8 fc ff ff ff          call   1b <main+0x1b>
  1f:   c7 44 24 0c 00 00 00    movl   $0x0,0xc(%esp)
  26:   00
  27:   c7 44 24 08 00 00 00    movl   $0x0,0x8(%esp)
  2e:   00
  2f:   c7 44 24 04 1c 00 00    movl   $0x1c,0x4(%esp)
  36:   00
  37:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
  3e:   e8 fc ff ff ff          call   3f <main+0x3f>
  43:   b8 00 00 00 00          mov    $0x0,%eax
  48:   c9                      leave
  49:   c3                      ret
~
/Templates/test.o.code[1]          unix utf-8 Ln 1, Col 1/20\
```

```
0804848c <main>:
 804848c:   55                      push   %ebp
 804848d:   89 e5                   mov    %esp,%ebp
 804848f:   83 e4 f0                and    $0xfffffff0,%es
 8048492:   83 ec 10                sub    $0x10,%esp
 8048495:   c7 05 3c 97 04 08 01    movl   $0x1,0x804973c
 804849c:   00 00 00
 804849f:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
 80484a6:   e8 d5 fe ff ff          call   8048380 <srand@
 80484ab:   c7 44 24 0c 60 83 04    movl   $0x8048360,0xc(
 80484b2:   08
 80484b3:   c7 44 24 08 80 83 04    movl   $0x8048380,0x8(
 80484ba:   08
 80484bb:   c7 44 24 04 8c 85 04    movl   $0x804858c,0x4(
 80484c2:   08
 80484c3:   c7 04 24 70 85 04 08    movl   $0x8048570,(%es
 80484ca:   e8 81 fe ff ff          call   8048350 <printf
 80484cf:   b8 00 00 00 00          mov    $0x0,%eax
 80484d4:   c9                      leave
 80484d5:   c3                      ret
~
~/Templates/test.code.drop[2]      unix utf-8 Ln 1, Col 1/20\
```

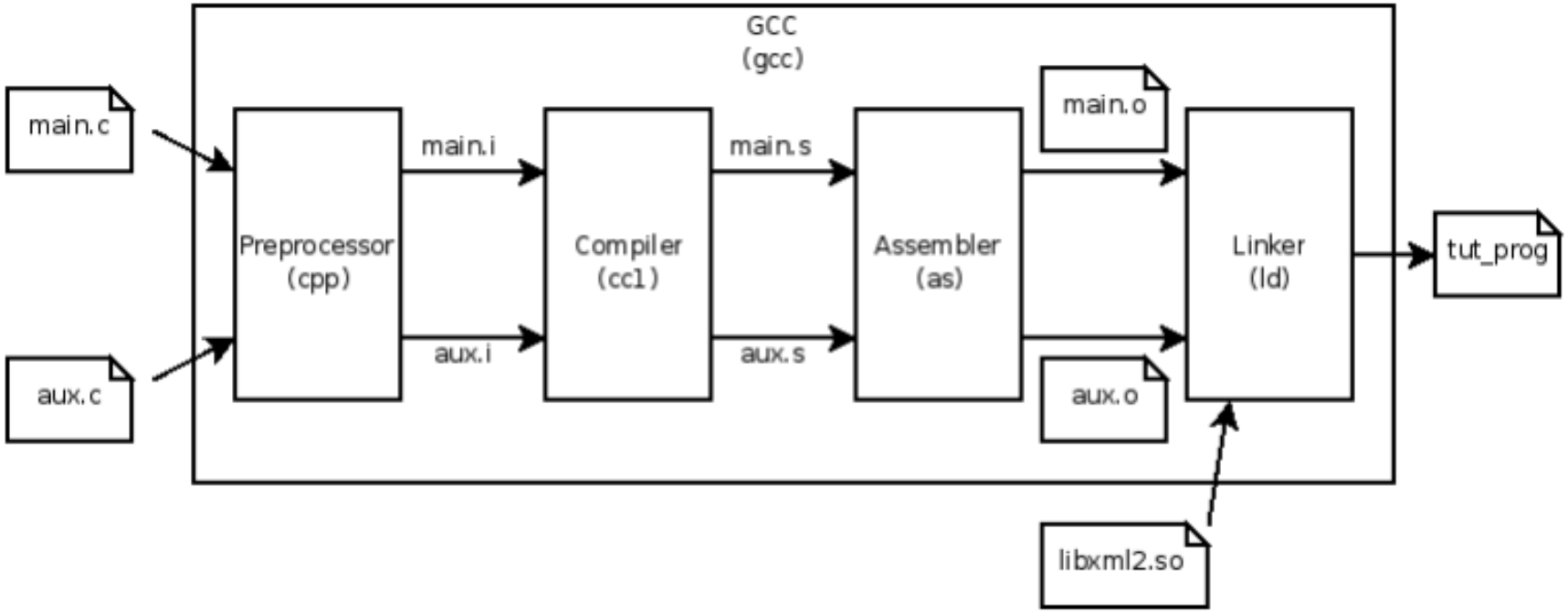
Linking in Lab0

- In Lab0, ld is calls explicitly to perform linking.
  - Entry point and start address are given explicitly.

```
$(LD) $(LDFLAGS) -e game_init -Ttext 0x00100000 -o game $(OBJS)
```

- The binary executable "game" is generated.
  - Have you run it directly?
- Can you write a program without main() ?

Birth of executables



Play with binaries

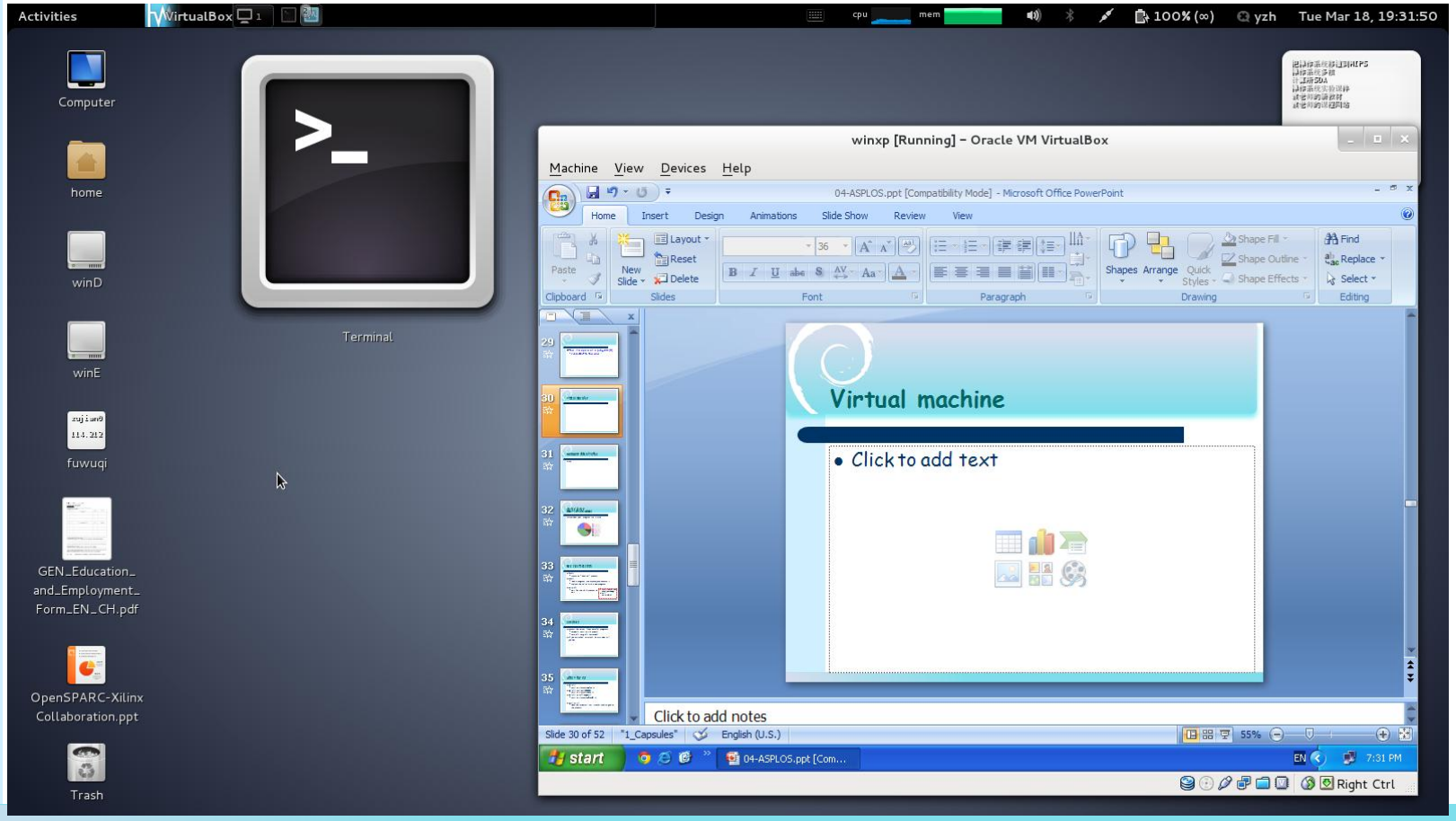
- vim + xxd

1, License 可以使用Windows下的License文件。
2, 找到/quartus/linux/下的libsys_cpt.so文件，使用gdb调入此文件，查找函数 l_pubkey_verify 的地址，记住它的地址，用ghex等编辑器打开此文件，抄写下从刚刚记下的地址开头的数据内容，在quartus 8.0中是 55 89 e5 53 81 ec 24 01 00 00 c7 45. 将此处字符串的前三个55 89 e5 修改为 31 c0 c3。（如何查找地址，参看附件Crack Quartus Linux.txt文件）
3, 用刚才记下的字符串为特征，查找quartus_map文件，你可以找到惟一的字符串，如法炮制，修改开头三个字节 55 89 e5为31 c0 c3。

- What is programming?

- Full-system virtualization

Virtual machine



Hardware virtualization

- Virtual machine softwares perform full-system virtualization to simulate a real machine.
- full-system
  - Everything about hardware is simulated.
  - CPU, interrupt, memory, I/O device…
- We take qemu as an example.

Simulate CPU

```
typedef struct CPUX86State {
    /* standard registers */
    target_ulong regs[CPU_NB_REGS];
    target_ulong eip;
    target_ulong eflags;
    ……
    /* segments */
    SegmentCache segs[6];
    SegmentCache ldt;
    SegmentCache tr;
    SegmentCache gdt;
    SegmentCache idt;
    target_ulong cr[5];
    ……
}
```

```
case 0xe8: /* call im */
    {
        if (dflag)
            tval = (int32_t)insn_get(env, s, OT_LONG);
        else
            tval = (int16_t)insn_get(env, s, OT_WORD);
        next_eip = s->pc - s->cs_base;
        tval += next_eip;
        if (s->dflag == 0)
            tval &= 0xffff;
        else if(!CODE64(s))
            tval &= 0xffffffff;
        gen_movtl_T0_im(next_eip);
        gen_push_T0(s);
        gen_jmp(s, tval);
    }
    break;
```

```
case 0xfa: /* cli */
    if (!s->vm86) {
        if (s->cpl <= s->iopl) {
            gen_helper_cli(cpu_env);
        } else {
            gen_exception(s, EXCP0D_GPF, pc_start - s->cs_base);
        }
    } else {
        if (s->iopl == 3) {
            gen_helper_cli(cpu_env);
        } else {
            gen_exception(s, EXCP0D_GPF, pc_start - s->cs_base);
        }
    }
    break;
```

Simulate interrupt

```
    if (is_int) {
        old_eip = next_eip;
    } else {
        old_eip = env->eip;
    }

    dt = &env->idt;
    if (intno * 8 + 7 > dt->limit) {
        raise_exception_err(env, EXCP0D_GPF, intno * 8 + 2);
    }
    ptr = dt->base + intno * 8;
    e1 = cpu_ldl_kernel(env, ptr);
    e2 = cpu_ldl_kernel(env, ptr + 4);
    /* check gate type */
    type = (e2 >> DESC_TYPE_SHIFT) & 0x1f;
+--- 38 lines: switch (type) {-------------------------------
    dpl = (e2 >> DESC_DPL_SHIFT) & 3;
    cpl = env->hflags & HF_CPL_MASK;
    /* check privilege if software int */
    if (is_int && dpl < cpl) {
        raise_exception_err(env, EXCP0D_GPF, intno * 8 + 2);
    }
```

- Give you the i386 Manual, can you write a program to simulate the behavor of x86 CPU?

Simulate VGA

- Simulation is slower than the real hardware.

```
cursor_offset = ((s->cr[VGA_CRTC_CURSOR_HI] << 8) |
                 s->cr[VGA_CRTC_CURSOR_LO]) - s->start_addr;
if (cursor_offset != s->cursor_offset ||
    s->cr[VGA_CRTC_CURSOR_START] != s->cursor_start ||
+ 11 lines: s->cr[VGA_CRTC_CURSOR_END] != s->cursor_end) {-------
cursor_ptr = s->vram_ptr + (s->start_addr + cursor_offset) * 4;
+  4 lines: if (now >= s->cursor_blink_time) {-------------------

depth_index = get_depth_index(surface);
if (cw == 16)
    vga_draw_glyph8 = vga_draw_glyph16_table[depth_index];
else
    vga_draw_glyph8 = vga_draw_glyph8_table[depth_index];
vga_draw_glyph9 = vga_draw_glyph9_table[depth_index];

dest = surface_data(surface);
linesize = surface_stride(surface);
ch_attr_ptr = s->last_ch_attr;
line = 0;
offset = s->start_addr * 4;
+ 82 lines: for(cy = 0; cy < height; cy++) {---------------------
```

How does a program execute?

- How does a program execute
  - on a physical machine?
  - on a virtual machine?
- Why can we simulate the execution on a physical machine?

Other issues

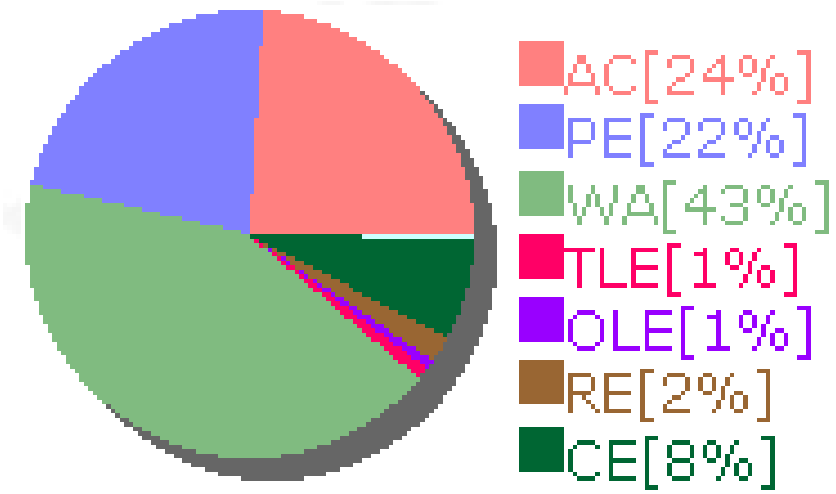
- snapshot
  - take a "photo" for the virtual machine
  - onekey recovery
- replay
  - record the execution flow of the virtual machine
    - how?
    - what makes the execution undeterministic?
- dynamic migration
  - change the physical machine without pausing the virtual one
  - significant for data centers
- cloud

- OS API – system calls

Online judge

https://code.google.com/p/hustoj

- How does your program execute on OJ?



Execute your program

- fork()
  - create an "identical" process
- exec()
  - load a program into memory and execute it
  - replace the caller with a new program
- waitpid()
  - wait for the child process to exit

```
if( (pid = fork()) == 0) {
    exec(your_prog);
} else {
    waitpid(pid);
}
```

Sandbox

- protect the server from harmful programs
- put your program into a sandbox
  - isolated evironment
  - minimal privileges

- If you are asked to attack an OJ, what will you do?

Attack the OJ

- while(1);
  - setrlimit(RLIMIT_CPU, …)
- while(1) malloc(1000000);
  - setrlimit(RLIMIT_AS, …)
- while(1) printf(long_str);
  - setrlimit(RLIMIT_FSIZE, …)

- setrlimit()
  - When the resource limit is reach, send a signal to the process.

Attack the OJ (cont.)

- asm ("cli");
  - your program is running in ring3
- fopen("/home/judge/junk", "w");
  - chroot(...)

- chroot()
  - set the root directory to a temporal one
  - your program can never jump out of it
  - when your program exits, remove this directory

Using system()

- For an OJ server, system() is very dangerous.
  - OJ deamon is run as root.
- system("pkill -9 judged");
- system("rm –rf /etc/init.d/judged");
- system("head –c 512 /dev/random > /dev/sda");
- system("poweroff");

Drop the root privileges

- setuid(), setgid(), setresuid()
  - change the user ID and group ID to a non-root one
- When your program runs, it cannot execute commands requiring root privileges.

- But system() is still powerful.
- How do you reject a program using system()?

Prevent against system()

- scan source code
- #define hello sys##tem
  - hello("poweroff");
- scan the source code after pre-processing
- scan the symbol table in the executable

Magical call

provided by jyy

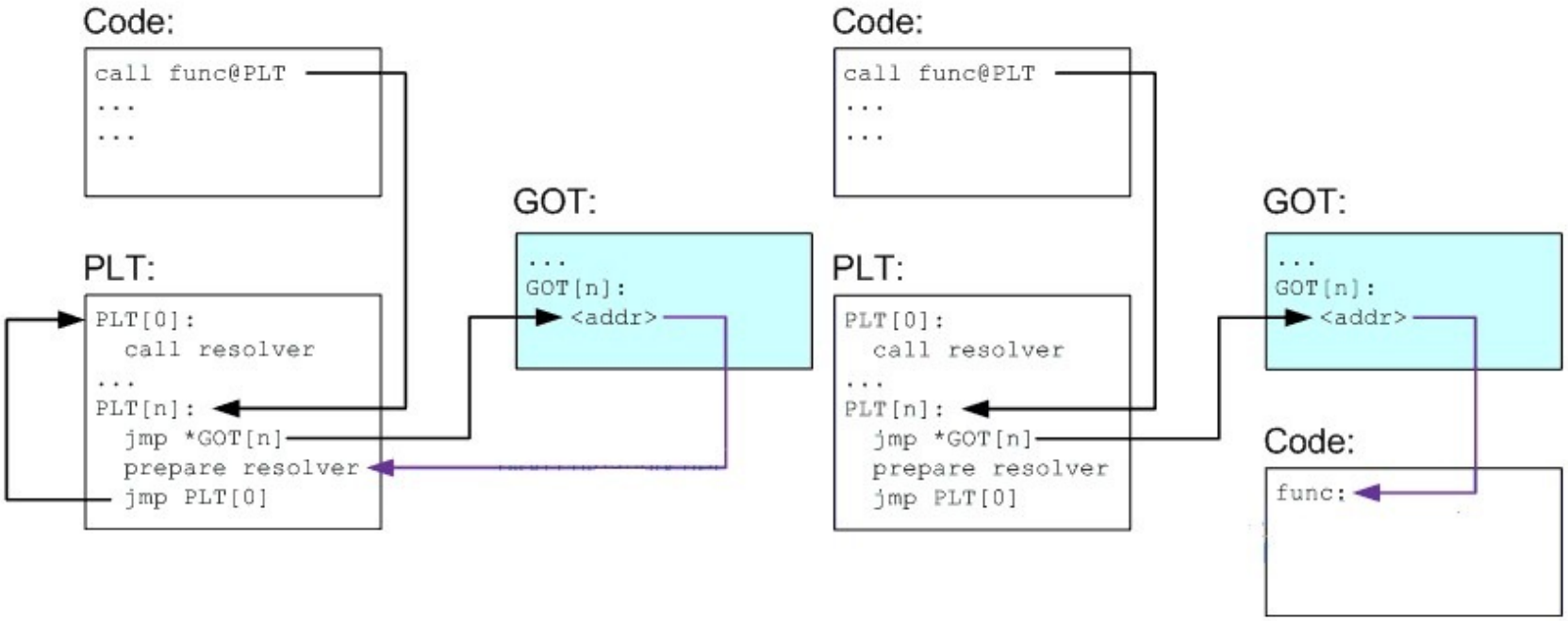
```
#define OFFSET (-0x555c85b0 + 0x555d4430)

int main() {
    srand(0);
    char *addr = **(char***)((char*)srand + 2) +
OFFSET;
    ((void(*)(const char *))addr)("poweroff");
}
```

Dynamic linking

- Dynamic linking keeps only one copy of the target function in memory.
  - shared by all executables
- The “hello” program does not contain the actual code of printf().
- What happen when “hello” calls printf()?

Dynamic linking & PLT & GOT

- PLT – procedure linkage table
- GOT – global offset table



Hacking

Anyway to defend?

use to obtain the address of real srand() in GOT

```
08048380 <srand@plt>:
 8048380:   ff 25 18 97 04 08    jmp    *0x8049718
 8048386:   68 18 00 00 00       push   $0x18
 804838b:   e9 b0 ff ff ff       jmp    8048340
```

the PLT entry

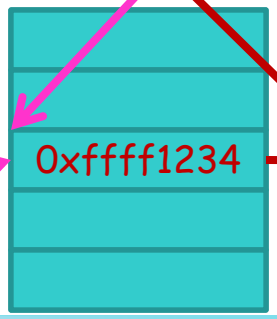
skip the opcode

```
srand(0);
char *addr = **(char***)((char*)srand + 2) + OFFSET;
```

the real srand() GOT

the GOT entry

the real system()



0x8049718

```
system() {
    ...
}
srand() {
    ...
}
```

OFFSET can be pre-computed in your machine using symbol table of libc.so

Trace your program

- Why is system() so powerful?
  - system calls!
    - use clone() to create a child process
    - use exec() to execute new programs
    - new programs use other system calls to do other things…
- Tracing system calls is the best way to protect the OJ server against attacks.

Trace system calls

- ptrace()

```
// check the system calls
ptrace(PTRACE_GETREGS, pidApp, NULL, &reg);

if (!record_call && call_counter[reg.REG_SYSCALL] == 0) {
    ACflg = OJ_RE;
    char error[BUFFER_SIZE];
    sprintf(error,"[ERROR] A Not allowed system call: runid:%d
callid:%ld\n", solution_id, reg.REG_SYSCALL);
    write_log(error);
    print_runtimeerror(error);
    ptrace(PTRACE_KILL, pidApp, NULL, NULL);
}
```

Summary

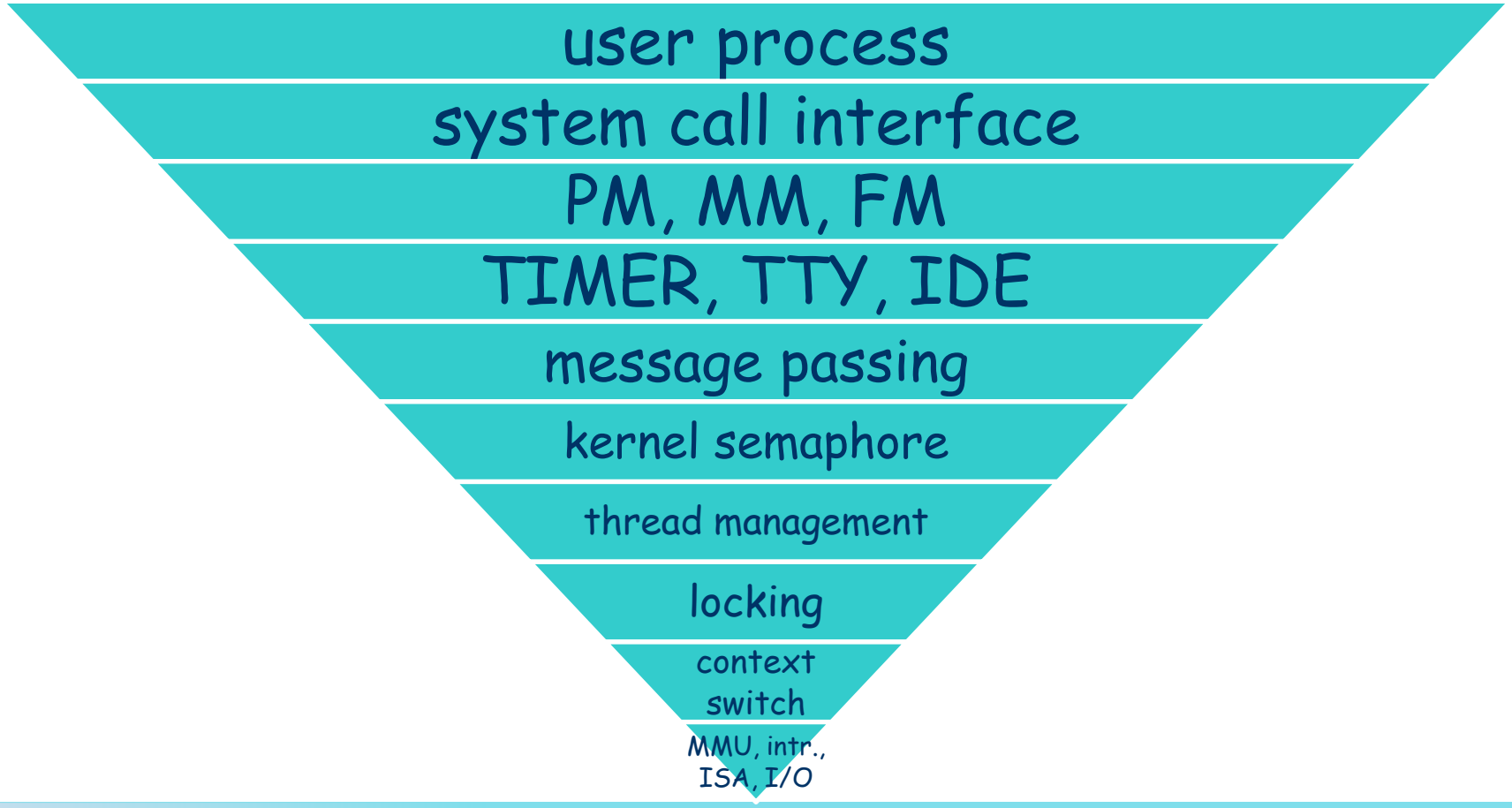
- System call is the unique way to request for services beyond the program's ability.
  - Without system calls,
    - your program can only "compute".
    - OJ is nothing.
- OS tries to provide a minimal set of safe system calls to serve requests from user processes.
  - It is a challenge to design efficient and safe system calls.

- What do you still lack?

You have known

- You have known
  - what is a program
  - how a program is born
  - how a program executes
- You also have known
  - how hardware is built
  - what hardware has provided
- But you still do not know
  - how to build this wonderful world?
  - precisely, after you issue command to run a “hello world” program, what happen to the OS exactly?

The architecture of Nanos



The real journey

- start now!
- full of thrill, shock, suspense

- but remember
  - 机器总是对的
  - 未测试代码总是错的