



# 从零构建操作系统的“苦”与“乐”

陈鹏飞 中山大学  
2024-08-02



- “道”（操作系统原理）和“术”（操作系统实践）
- “道”：抽象层面原理和规律；“术”：具体层面技术和实施
  - 老子说，有道无术，术尚可求也。有术无道，止于术。
  - 庄子说，以道驭术，术必成。离道之术，术必衰。
  - 《孙子兵法》说，道为术之灵，术为道之体；以道统术，以术得道。
  - 古话说，上人用道，中人用术，下人用力。
  - 韩愈说，闻道有先后，术业有专攻。
- 道是思想，术是方法，道术合二为一，才是正道；
- 十堰是一个人称“道”的城市；



## ➤ 操作系统实现路径





# ➤ 操作系统实现路径

独立设计并实现面向特定CPU架构的原型操作系统！！！

```

1  [org 0x7c00] ; bootloader offset
2      mov bp, 0x9000 ; set the stack
3      mov sp, bp
4
5      mov bx, MSG_REAL_MODE
6      call print ; This will be written after the BIOS messages
7
8      call switch_to_pm
9      jmp $ ; this will actually never be executed
10
11     %include "../05-bootsector-functions-strings/boot_sect_print.asm"
12     %include "../09-32bit-gdt/32bit-gdt.asm"
13     %include "../08-32bit-print/32bit-print.asm"
14     %include "32bit-switch.asm"
15
16     db 32
17     BEGIN_PM: ; after the switch we will get here
18     mov ebx, MSG_PROT_MODE
19     call print_string_pm ; Note that this will be written at the
20     jmp $
21
22     MSG_REAL_MODE db "Started in 16-bit real mode", 0
23     MSG_PROT_MODE db "Loaded 32-bit protected mode", 0
24
25     ; bootsector
26     times 510-$-$ db 0
27     dw 0xaa55

```

几行

几十行

几千行

几千万行

The Linux Kernel Archives

About Contact us FAQ Release Signatures Archives

Protocol: Location  
HTTP: <http://www.kernel.org/pub/>  
GIT: <https://git.kernel.org/>  
RSYNC: <rsync://rsync.kernel.org/pub/>

Latest Release  
5.11.1

Version	Date	Commit	Author	Changes
5.11	2021-02-14	[v5.11] [commit] [log]	[v5.11] [commit] [log]	[v5.11] [commit] [log]
5.11.1	2021-02-17	[v5.11.1] [commit] [log]	[v5.11.1] [commit] [log]	[v5.11.1] [commit] [log]
5.11.17	2021-02-17	[v5.11.17] [commit] [log]	[v5.11.17] [commit] [log]	[v5.11.17] [commit] [log]
5.11.100	2021-02-23	[v5.11.100] [commit] [log]	[v5.11.100] [commit] [log]	[v5.11.100] [commit] [log]
5.19.177	2021-02-23	[v5.19.177] [commit] [log]	[v5.19.177] [commit] [log]	[v5.19.177] [commit] [log]
5.14.222	2021-02-23	[v5.14.222] [commit] [log]	[v5.14.222] [commit] [log]	[v5.14.222] [commit] [log]
5.14.223	2021-02-23	[v5.14.223] [commit] [log]	[v5.14.223] [commit] [log]	[v5.14.223] [commit] [log]
4.19.258	2021-02-23	[v4.19.258] [commit] [log]	[v4.19.258] [commit] [log]	[v4.19.258] [commit] [log]
4.4.256	2021-02-23	[v4.4.256] [commit] [log]	[v4.4.256] [commit] [log]	[v4.4.256] [commit] [log]
[v5.11] [commit] [log] [v5.11.1] [commit] [log]				
[v5.11.17] [commit] [log] [v5.11.100] [commit] [log]				
[v5.19.177] [commit] [log] [v5.14.222] [commit] [log]				
[v5.14.223] [commit] [log] [v4.19.258] [commit] [log]				
[v4.4.256] [commit] [log] [v4.4.256] [commit] [log]				

Tag next: [new-20210223](#) [2021-02-23](#)

聚沙成塔、集腋成裘



## ➤ 操作系统实现目标

- 实现面向*i386 (32位)*平台的原型操作系统；基于*Rust*的*X86\_64*操作系统作为试点。

用户空间  
——  
内核空间

- 涉及的内容包括：内核、启动、*I/O*、中断、*CPU*调度、多线程并发、物理/虚拟存储、文件系统、*Shell*等多方面的内容。





## ➤ 操作系统实验安排

### □ 面向i386处理器架构的32位操作系统实现

- ① 编译内核/利用已有内核构建OS；
- ② 汇编语言实验；
- ③ 实模式和保护模式下OS启动；
- ④ OS中断/异常；
- ⑤ 处理器调度；
- ⑥ 系统调用；
- ⑦ 同步与互斥；
- ⑧ 物理/虚拟内存管理；
- ⑨ 内核/用户模式线程管理模式；
- ⑩ 文件系统（可选）；
- ⑪ Shell（可选）；
- ⑫ 迁移到Risc-V架构/ARM架构（可选）；

appendix/debug_with_gdb_and_qemu	merge remote
docs	add documents and template
env	add how to install nasm 2.15
images	update
lab1	fix bug in lab1
lab2	fix bug in lab-2-test.cpp
lab3	add lab4
lab4	add assignment to lab4
lab5	review lab7
lab6	add assignment to lab6
lab7	add assignments to lab7
lab8	fix bug
lab9	Merge branch 'main' of gitee
.gitignore	add system call write
LICENSE	Initial commit
README.md	add other documents
本科生实验报告模板.docx	add documents and template

<https://gitee.com/nelsoncheung/sysu-2021-spring-operating-system>



# ➤ 操作系统实验安排

## □ 面向i386处理器架构的32位操作系统实现

- Lab2 实验入门
- 实验要求
- Assignment 1 MBR
  - 1.1
  - 1.2
  - **Assignment 2 实...**
  - 2.1
  - 2.2
  - 2.3
- Assignment 3 汇编
  - 3.1 分支逻辑的...
  - 3.2 循环逻辑的...
  - 3.3 函数的实现
- Assignment 4 汇...
- 实验概述
- 参考资料
- IA-32处理器
  - 从汇编开始
  - 体系结构
  - 实地址模式
- 汇编基础
  - x86汇编 注释
  - nasm汇编 标识符
  - nasm汇编 标号
  - x86汇编 数据传送...
  - nasm汇编 内存寻...
  - x86汇编 算术和逻...
  - x86汇编 控制转移...
  - x86汇编 栈操作指令
  - x86汇编 过程调用
- 计算机开机启动过程
- Example 1 Hello W...

### Assignment 2 实模式中断

参考资料如下。

- OSDev 关于 BIOS 的介绍
- BIOS 中断表
- VIDEO - WRITE CHARACTER ONLY AT CURSOR POSITION
- VIDEO - WRITE CHARACTER AND ATTRIBUTE AT CURSOR POSITION
- VIDEO - WRITE STRING (AT and later, EGA)
- VIDEO - GET CURSOR POSITION AND SIZE
- 10h中断

2.1和2.2使用的都是实模式中断 int 10h，由于功能号不同，执行的结果也就不同。在 int 10h 中断的资料

[https://zh.wikipedia.org/wiki/INT\\_10H](https://zh.wikipedia.org/wiki/INT_10H) 中，其只给出10h中断下各个功能号的用途，并未给出实际的用法。因此，同学们可能一开始会感觉不知所云，教程下面给出同学们完成本次实验需要用到的功能号。

功能	功能号	参数	返回值
设置光标位置	AH=02H	BH=页码, DH=行, DL=列	无
获取光标位置和形状	AH=03H	BX=页码	AX=0, CH=行扫描开始, CL=行扫描结束, DH=行, DL=列
在当前光标位置写字符和属性	AH=09H	AL=字符, BH=页码, BL=颜色, CX=输出字符的个数	无

注意，“页码”均设置为0。

一般地，中断的调用方式如下。

将参数和功能号写入寄存器  
int 中断号  
从寄存器中取出返回值

### 2.1

请探索实模式下的光标中断，利用中断实现光标的位置获取和光标的移动。说说你是怎么做的，并将结果截图。



# ➤ 操作系统实验安排

## 口 基于Rust的x86\_64操作系统实现 中山大学 YatSenOS v2 操作系统实验教程

### 1 实验说明

本文档提供了一套基于 Rust、面向 UEFI 和 x86\_64 的操作系统课程实验方案。

本实验设计期望基于低汇编、避免重复造轮子的宗旨，利用 Rust 语言优秀的包管理和底层支持，借助现有的优秀工程化底层封装，为学生提供一个低负担、现代、面向高级语言的操作系统实验方案。

### 2 实验大纲

每一次实验区间为两周。

- 实验零：环境搭建与实验准备
- 实验一：操作系统的启动
- 实验二：中断处理
- 实验三：内核线程与缺页异常
- 实验四：用户程序与系统调用
- 实验五：fork 的实现、并发与锁机制
- 实验六：硬盘驱动与文件系统
- 实验七：更好的内存管理
- 实验八：扩展实验

Name	Last commit
..	
0x00	chore(deps)
0x01	chore(deps)
0x02/pkg/kernel	chore: upgr...
0x03/pkg/kernel	Lab 7: 更好...
0x04	Lab 6: 硬盘...
0x05/pkg	fix(lab/5): h...
0x06/pkg	fix(lab/6): t...
0x07/pkg	fix(lab/7): r...

代码框架

<https://gitee.com/nelsoncheung/sysu-2021-spring-operating-system>



## ➤ 操作系统实验安排

<https://ysos.cs.sustech.edu.cn/>

### □ 基于Rust的x86\_64操作系统实现

## 3 实验任务与要求

1. 请各位同学独立完成作业，任何
2. 请参考 [代码与提交规范](#) 进行实验
3. 依据 [实验任务](#) 完成实验。

- 代码编写任务：观察提供的预期。请在报告中介绍实现。
- 思考任务：完成“思考题”和题可能也是理解代码、实现。
- Bonus 加分项：学有余力的报告中进行展示。这部分内

4. 请在实验报告中涵盖相关任务的与解决方案等内容。

- [x64 数据结构概述](#)

## 8 思考题

1. 为什么需要在 `clock_handler` 中使用 `without_interrupts` 函数？如果不使用它，可能会发生什么情况？
2. 考虑时钟中断进行进程调度的场景，时钟中断的频率应该如何设置？太快或太慢的频率会带来什么问题？请分别回答。
3. 在进行 `receive` 操作的时候，为什么无法进行日志输出？如果强行输出日志，会发生什么情况？谈谈你对串口、互斥锁的认识。
4. 输入缓冲区在什么情况下会满？如果缓冲区满了，用户输入的数据会发生什么情况？
5. 进行下列尝试，并在报告中保留对应的触发方式及相关代码片段：
  - 尝试用你的方式触发 Triple Fault，开启 `intdbg` 对应的选项，在 QEMU 中查看调试信息，分析 Triple Fault 的发生过程。
  - 尝试触发 Double Fault，观察 Double Fault 的发生过程，尝试通过调试器定位 Double Fault 发生时使用的栈是否符合预期。
  - 通过访问非法地址触发 Page Fault，观察 Page Fault 的发生过程。分析 Cr2 寄存器的值，并尝试回答为什么 Page Fault 属于可恢复的异常。
6. 如果在 TSS 中为中断分配的栈空间不足，会发生什么情况？请分析 CPU 异常的发生过程，并尝试回答什么时候会发生 Triple Fault。
7. 在未使用 `set_stack_index` 函数时，中断处理程序的栈可能哪里？尝试结合 `gdb` 调试器，找到中断处理程序的栈，并验证你的猜想是否正确。



## ➤ 操作系统实验安排

充分利用大模型如ChatGPT、Llama、Gemma、ChatGLM等辅助学习和编程；

**Prompt:** Please write an assemble program used to boot kernel in the first section of disk for i386 CPU architecture

Let's break it down:

- \* We set up the BIOS data area (BDA) by moving some magic numbers into registers `ax` and `dx`, then calling the BIOS function with an interrupt (`int 0x15`).
- \* We get the disk parameters, including the boot sector's signature bytes (`boot\_msg`) using BIOS functions.
- \* We load the next sector's code using BIOS functions (`read\_sectors`). The next sector contains the rest of our bootloader code.
- \* We jump to the kernel's entry point (`kernel\_entry`), where your kernel code goes.
- \* The `print\_kernel\_msg` label is used to print a message before jumping to the kernel.

Assemble this code using NASM:

```
```bash
nasm -f bin boot_sector.asm -o mbr.bin
```
```

This will create a binary executable file `mbr.bin`. You can use a tool like `dd` or `copy` to copy this file onto your disk, for example:

```
```bash
dd if=mbr.bin of=/dev/fd0 bs=1 count=446 seek=0 conv=notrunc
```
```

This will put the MBR code at the beginning of the floppy disk.

Ollama + Llama3得出的结果



## ➤ 从零构建操作系统的“苦”之环境多样

### □ 学生电脑的CPU架构不统一，需要不同的工具栈；



- ✓ ARM64的GCC;
- ✓ ARM64的nasm;
- ✓ ARM64的qemu；
- ✓ 交叉编译：从ARM64 → i386；

### □ 开发操作系统环境复杂；

- ① Virtualbox，用于启动虚拟机；
- ② 安装Ubuntu 18.04桌面版，并配置清华安装源；
- ③ 安装VScode以及汇编、C/C++插件；
- ④ 安装nasm；
- ⑤ 安装qemu-system-i386；
- ⑥ 安装cmake、gdb工具；
- ⑦ 安装objdump, readelf；
- ⑧ 保存虚拟机镜像；

i386操作系统开发环境

- ① Virtualbox，用于启动虚拟机；
- ② 安装VScode以及汇编、C/C++插件；
- ③ 安装qemu-system；
- ④ 安装cmake、gdb工具；
- ⑤ 安装Rust编译和运行环境；
- ⑥ 安装CodeLLDB调试工具；
- ⑦ 安装objdump, readelf；

Rust操作系统开发环境



## ➤ 从零构建操作系统的“苦”之**编程语言**

```
**boot_sector.asm**
```nasm
section .data
    boot_msg db 'BOOTING...', 0x00, 0x55, 0
    sector_size equ 512
    kernel_addr equ 0x7c00
    kernel_size equ 0x4000

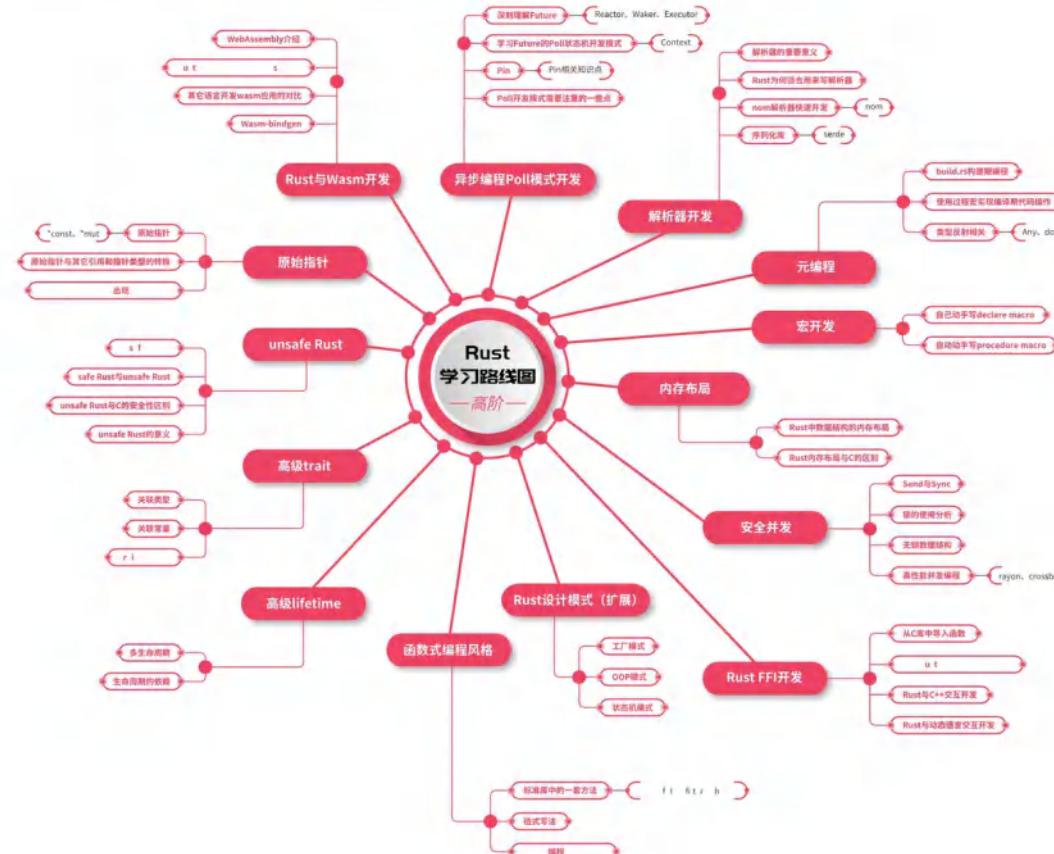
section .text
    global _start
_start:
    ; Set up the BIOS data area (BDA)
    mov ax, 0x2401
    mov dx, 0x0003
    int 0x15

    ; Get the disk parameters
    mov si, boot_msg
    call print_string

    ; Load the next sector's code
    mov si, sector_size
    mov cx, si
    mov ah, 2      ; BIOS function: read sect
    int 0x10

    ; Jump to the kernel
    jmp short kernel_start
kernel_end:
    jmp kernel_end
```

## 汇编语言的不熟悉



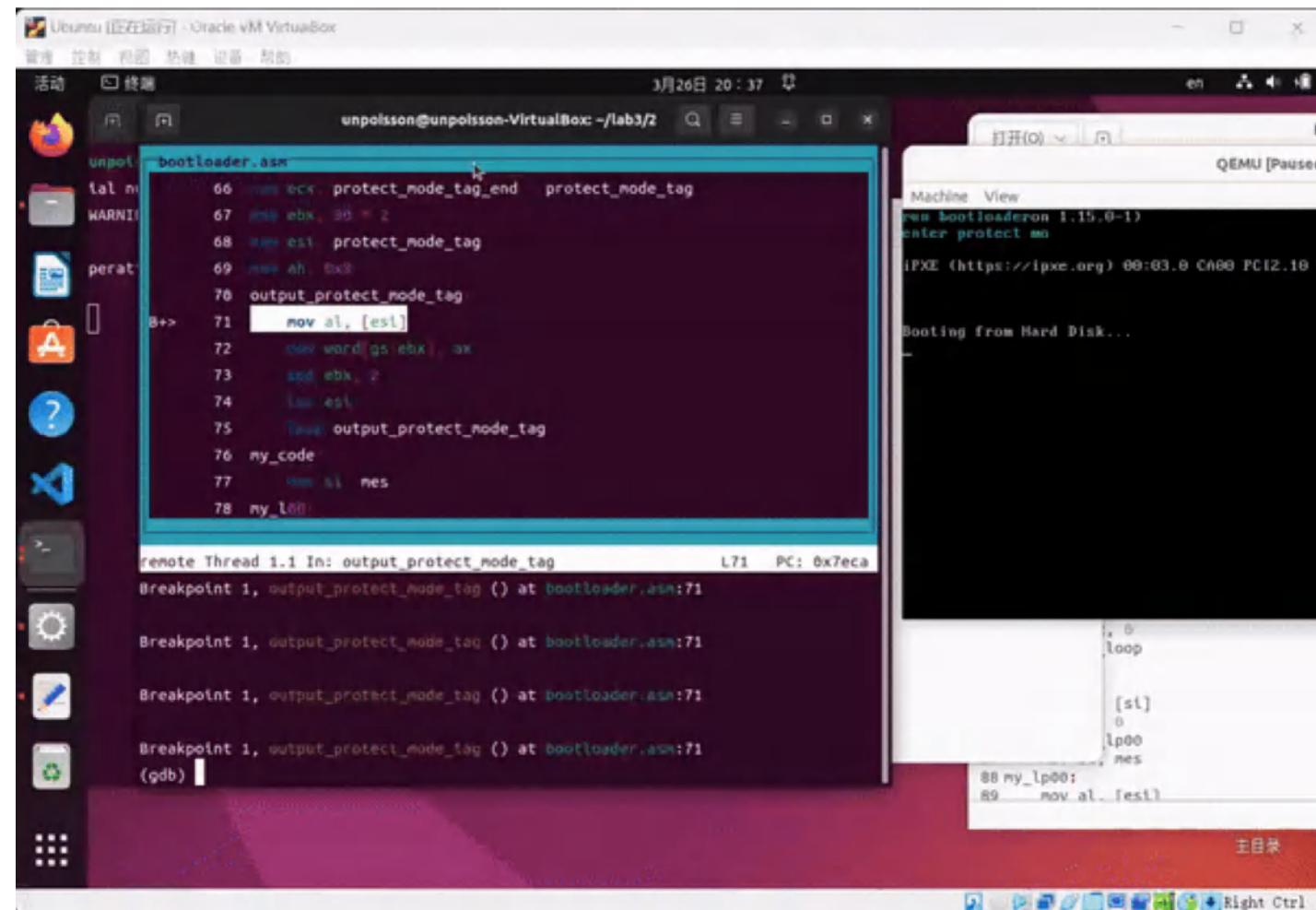
## Rust语言复杂的知识点

➤ 需要留出时间进行专门的编程语言的练习，完成编程任务；



## ➤ 从零构建操作系统的“苦”之程序调试

程序行为特别是寄存器操作复杂，难以调试；



问题：调试的时候有些指令直接跳过不输出寄存器内容（原因：GDB调试器问题） 13



# ➤ 从零构建操作系统的“苦”之硬件逻辑

- 直接操作底层硬件，需要熟悉硬件逻辑及相应的指令；
- 硬件命令执行容易出错，问题难以定位；

## 3 UART 与日志输出

### 3.1 串口驱动

在 [UART 串口通信](#) 部分中介绍了 UART 的基本原理，以及相关的基础知识。在这一部分实验中，你将会实现一个简单的串口驱动，并将其用于内核的日志输出。

由于这是第一次进行驱动的编写，你可以在 `pkg/kernel/src/drivers` 目录下看到一个基本的代码框架，你需要完成其中的 `uart16550` 驱动。

在 `pkg/kernel/src/drivers/serial.rs` 中存放了串口初始化的相关代码，你所实现的 `SerialPort` 结构体将会在这里被调用：

```

1  use super::uart16550::SerialPort;
2
3  const SERIAL_IO_PORT: u16 = 0x3F8; // COM1
4
5  once_mutex!(pub SERIAL: SerialPort);
6
7  pub fn init() {
8      init_SERIAL(SerialPort::new(SERIAL_IO_PORT));
9      get_serial_for_sure().init();
10
11     println!("{}",
12         crate::get_ascii_header());
13     println!("[+] Serial Initialized.");
14 }
15 guard_access_fn!(pub get_serial(SERIAL: SerialPort));

```

串口通信

```

asm_read_hard_disk:
; 从硬盘读取一个逻辑扇区

; 参数列表
; ax=逻辑扇区号0~15位
; cx=逻辑扇区号16~28位
; ds:bx=读取出的数据放入地址

; 返回值
; bx=bx+512

    mov dx, 0x1f3
    out dx, al    ; LBA地址7~0

    inc dx        ; 0x1f4
    mov al, ah
    out dx, al    ; LBA地址15~8

    mov ax, cx

    inc dx        ; 0x1f5
    out dx, al    ; LBA地址23~16

    inc dx        ; 0x1f6
    mov al, ah
    and al, 0x0f
    or al, 0xe0   ; LBA地址27~24
    out dx, al

    mov dx, 0x1f2
    mov al, 1
    out dx, al    ; 读取1个扇区

    mov dx, 0x1f7    ; 0x1f7
    mov al, 0x20    ; 读命令
    out dx, al

```

in al, 0x92 ; 南桥芯片内的端口  
or al, 0000\_0010B  
out 0x92, al ; 打开 A20

cli ; 保护模式下中断机制尚未建立  
mov eax, cr0  
or eax, 1  
mov cr0, eax ; 设置 PE 位

打开地址线

设置模态位

读写磁盘



# ➤ 从零构建操作系统的“苦”之隐性知识

## □ 操作系统构建需要的一些关键知识并没有在操作系统原理课上介绍：

```

30 mov dword [GDT_START_ADDRESS+0x18],0x80007fff ; 基地址为0x000B8000, 界限0x07FFF
31 mov dword [GDT_START_ADDRESS+0x1c],0x0040920b ; 粒度为字节
32
33 ; 创建保护模式下平坦模式代码段描述符
34 mov dword [GDT_START_ADDRESS+0x20],0x0000ffff ; 基地址为0, 段界限为0xFFFFFFF
35 mov dword [GDT_START_ADDRESS+0x24],0x00cf9800 ; 粒度为4kb, 代码段描述符
36
37 ; 初始化描述符表寄存器GDTR
38 mov word [pgdt], 39 ; 描述符表的界限
39 lgdt [pgdt]
40
41 in al,0x92 ; 南桥芯片内的端口
42 or al,0000_0010B
43 out 0x92,al ; 打开A20
44
45 cli ; 中断机制尚未工作
46 mov eax,cr0
47 or eax,1
48 mov cr0, eax ; 设置PE位
49
50 ; 以下进入保护模式
51 jmp dword CODE_SELECTOR:protect_mode_begin
52
53 ; 16位的描述符选择子: 32位偏移
54 ; 清流水线并串行化处理器
55 [bits 32]
56 protect_mode_begin:
57
58 mov eax, DATA_SELECTOR ; 加载数据段(0..4GB)选择子
59 mov ds, eax
60 mov es, eax
61 mov eax, STACK_SELECTOR

```

跨段长跳转

```

int ProgramManager::executeThread(ThreadFunction function, void *parameter, const char *name, int priority)
{
    // 关中断, 防止创建线程的过程被打断
    bool status = interruptManager.getInterruptStatus();
    interruptManager.disableInterrupt();

    // 分配一页作为PCB
    PCB *thread = allocatePCB();

    if (!thread)
        return -1;

    // 初始化分配的页
    memset(thread, 0, PCB_SIZE);

    for (int i = 0; i < MAX_PROGRAM_NAME && name[i]; ++i)
    {
        thread->name[i] = name[i];
    }

    thread->status = ProgramStatus::READY;
    thread->priority = priority;
    thread->ticks = priority * 10;
    thread->ticksPassedBy = 0;
    thread->pid = ((int)thread - (int)PCB_SET) / PCB_SIZE;

    // 线程栈
    thread->stack = (int *)((int)thread + PCB_SIZE);
    thread->stack -= 7;
    thread->stack[0] = 0;
    thread->stack[1] = 0;
    thread->stack[2] = 0;
    thread->stack[3] = 0;
    thread->stack[4] = (int)function;
    thread->stack[5] = (int)program_exit;
    thread->stack[6] = (int)parameter;

    allPrograms.push_back(&(thread->tagInAllList));
    readyPrograms.push_back(&(thread->tagInGeneralList));

    // 恢复中断
    interruptManager.setInterruptStatus(status);

    return thread->pid;
}

```

CPU中断时保存指令地址，完成线程切换



## ➤ 从零构建操作系统的“苦”之千人千面

### □ 学生会遇到各种问题，需要针对不同的问题进行答疑解惑，工作量大；

1. 是否可以在内核线程中使用系统调用？并借此来实现同样的进程退出能力？分析并尝试回答。

xy3xy3 4月8日 问  
运行程序时发现了爆栈问题？  
个 1

2. 为什么需要克隆内核页表？在系统调用的内核态下使用的是哪一张页表？用户态程序尝试访问内核空间会被正确拦截吗？尝试验证你的实现是否正确。

3. 为什么在使用 `still_alive` 函数判断进程是否存活时，需要关闭中断？在不关闭中断的情况下，会有什么问题？

4. 对于如下程序，使用 `gcc` 直接编译：

```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello, World!\n");
5     return 0;
6 }
```

调试在 fork 之后确实产生子进程了吗？

fuzzen 5月12日 问  
在尝试思考题6的时候发现，  
• 一个很小的数，大概小于  
• 一个比较小的数，大概 0  
这些在先前的实验中测试过是  
这样的现象在先前的 lab2 中 t  
定程度后变为开机就触发 Pag  
这两个现象感觉有一些相似之  
个 1

从本次实验及先前实验的所学内容出发，结合进程的创建、链接、执行、退出的生命周期，参考系统调用的调用过程（可以以 Linux 为例），解释程序的运行。

5. `x86_64::instructions::hlt` 做了什么？为什么这样使用？为什么不可以再用户态中的 `wait_pid` 实现中使用？

6. 有同学在某个回南天迷蒙的深夜遇到了奇怪的问题：

只有当进行用户输入（触发了串口输入中断）的时候，会触发奇怪的 Page Fault，然而进程切换、内存分配甚至 `fork` 等系统调用都很正常。

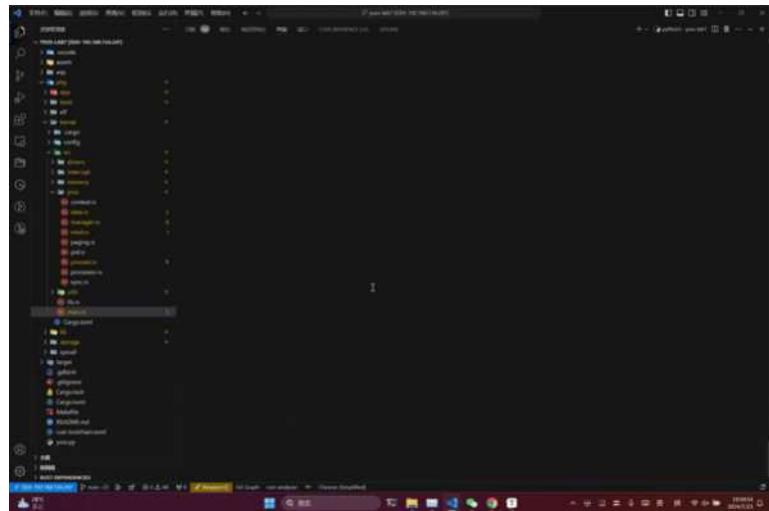
经过近三个小时的排查，发现他将 TSS 中的 `privilege_stack_table` 相关设置注释掉了。

请查阅资料，了解特权级栈的作用，实验说明这一系列中断的触发过程，尝试解释这个现象。

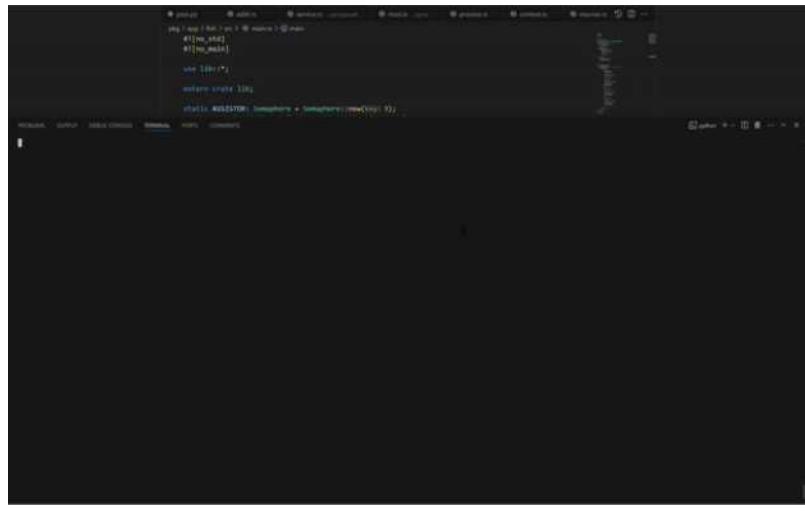
- 可以使用 `intdbg` 参数，或 `ysos.py -i` 进行数据捕获。
- 留意 `0x0e` 缺页异常和缺页之前的中断的信息。
- 注意到一个不应当存在的地址.....？



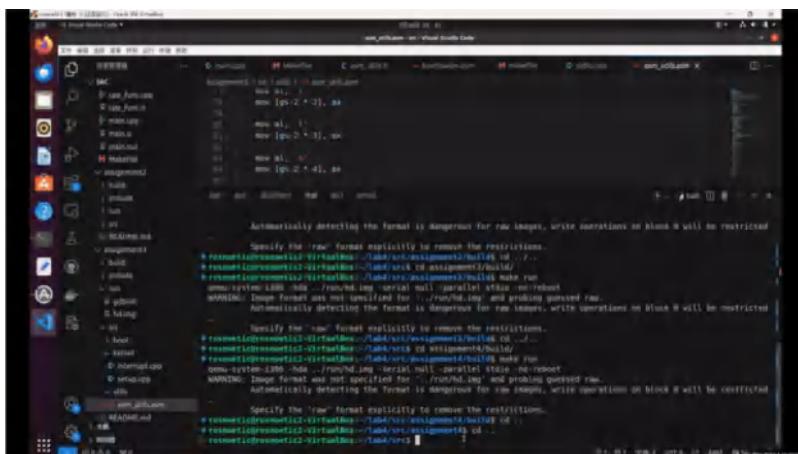
## ➤ 从零构建操作系统的“乐”之原型构建



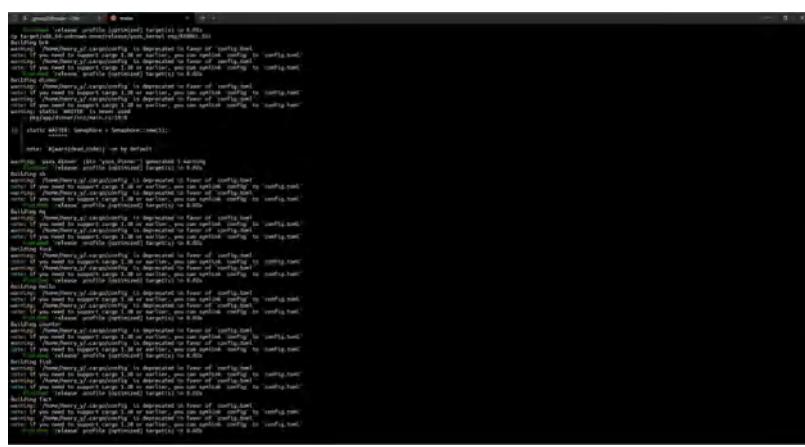
陈同学构建的Rust操作系统



王同学构建的Rust操作系统



丁同学构建的i386操作系统



黄同学构建的Rust操作系统



## ➤ 从零构建操作系统的“乐”之实验报告



中山大學  
SUN YAT-SEN UNIVERSITY

### 操作系统实验 4

用户程序与系统调用

中山大學



姓名：王宇  
学号：22330111  
教学班号：计科 1 班  
专业：计算机科学与技术  
院系：计算机学院

2023 学年第二学期

1

### 本科生实验报告

实验课程	:	操作系统
实验名称	:	YatsenOS v2
专业名称	:	计算机科学与技术
学生姓名	:	陈锐霖
学生学号	:	22330004
实验地点	:	南实验楼 D402
实验成绩	:	
报告时间	:	20240525



## ➤ 比赛获奖

指导学生连续4年参加操作系统比赛，获二等奖2项、3等奖3项、连续3年评为优秀指导老师

CSCC 全国大学生计算机系统能力大赛

首页 成绩查询 我的报名 个人信息

上一页 / 2021全国大学生计算机系统能力大赛操作系统设计赛-功能挑战赛

大赛时刻 大赛章程与通知 功能赛要求 技术报告 作品开源 技术支持 常见问题

**2022全国大学生计算机系统能力大赛  
操作系统设计赛**

主办单位：全国高等学校计算机教育研究会 系统能力培养研究专家组 系统能力培养研究项目发起高校  
承办单位：天津大学  
协办单位：CCF系统软件专委 CCF教育专委 中科院计算所 中科院软件所 机械工业出版社  
赞助单位：(按拼音排序)  
钻石赞助：华为 腾讯软件 终信  
金牌赞助：龙旗科技 腾讯环宇 龙芯 蚂蚁 京东方 小米  
银牌赞助：墨腾 翼邦 指令集 指令集  
技术支持：基博

2023全国大学生计算机系统能力大赛操作系统设计赛

全国总决赛内核设计赛道、功能挑战赛道

获奖名单

**2024全国大学生计算机系统能力大赛  
操作系统设计赛**

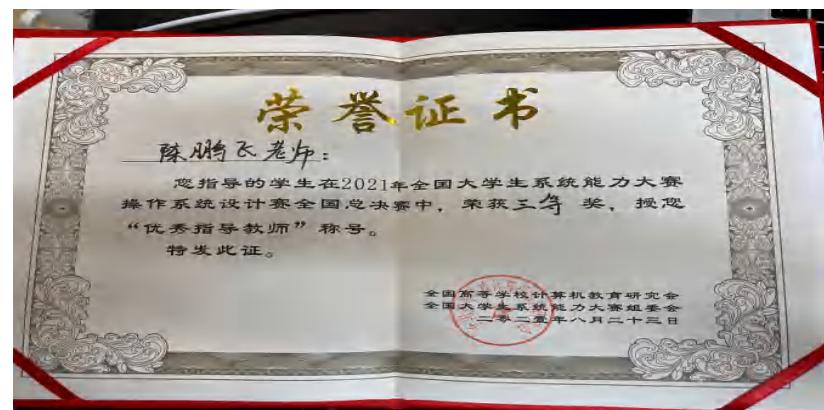
主办单位：全国高等学校计算机教育研究会 系统能力培养研究专家组 系统能力培养研究项目发起高校  
承办单位：中国电子技术标准化研究院  
协办单位：中国计算机学会系统软件专业委员会 北京计算所机房“101计划”操作系统及应用研究室 机械工业出版社  
钻石赞助：HUAWEI 腾讯 腾讯云 OPPO vivo  
金牌赞助：京东方  
银牌赞助：墨腾  
技术支持：基博

大赛组委会评选出 47 名优秀指导教师，名单如下。

优秀指导教师			
学校	姓名	学校	姓名
北京大学	刘先华	北京大学	杨春
北京工商大学	赵霞	北京航空航天大学	沃天宇
北京航空航天大学	牛虹婷	北京航空航天大学	姜博
北京航空航天大学	孙海龙	北京交通大学	瞿高寿
北京交通大学	杨武杰	浙江大学	申文博
中山大学	陈鹏飞	重庆大学	洪明坚
重庆大学	刘寄	重庆邮电大学	马彬

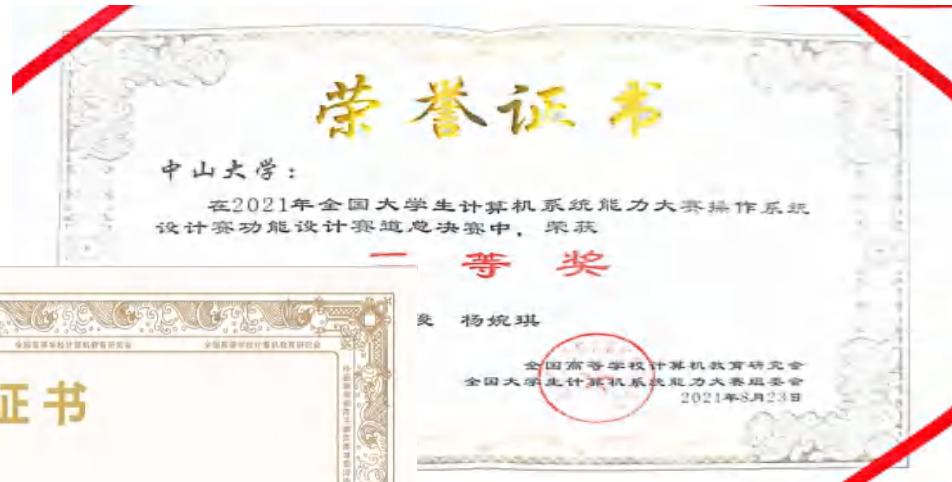
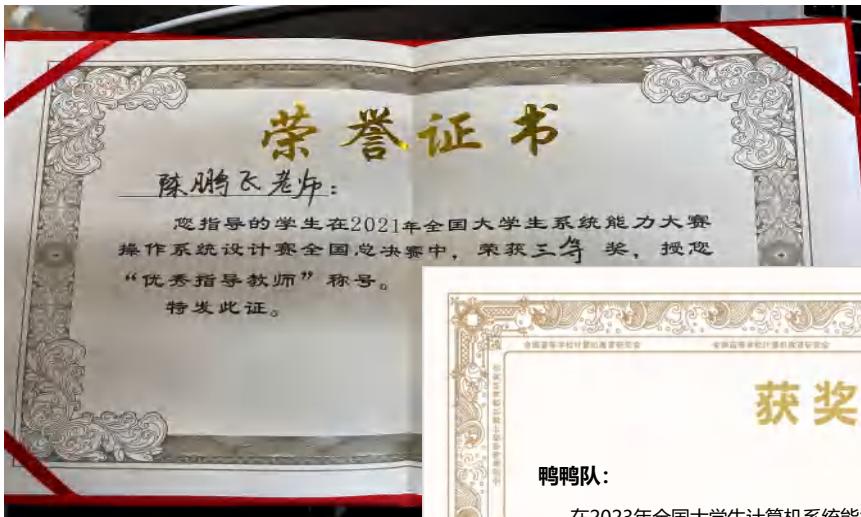
2022全国大学生系统能力大赛操作系统设计赛  
优秀指导教师名单  
(按姓氏笔画排序)

姓名	单位	姓名	单位	姓名	单位
安明伟	河南科技大学	姜博	北京航空航天大学	罗秋明	深圳大学
才振功	浙江大宝	高德钧	中国科学院大学	罗星	广东工业大学
董德泽	中国地质大学(武汉)	葛炎岩	南京大学	罗宇	国防科技大学
陈浩	湖南大学	陈鹏飞	重庆邮电大学	王波	厦门大学
陈莉君	西安邮电大学	李庆昌	浙江大学	邱锐	天津大学
陈鹏飞	中山大学	李罡	交通大学	邱元杰	电子科技大学
董志伟	重庆大学	李斯	西南交通大学	江伟	重庆大学





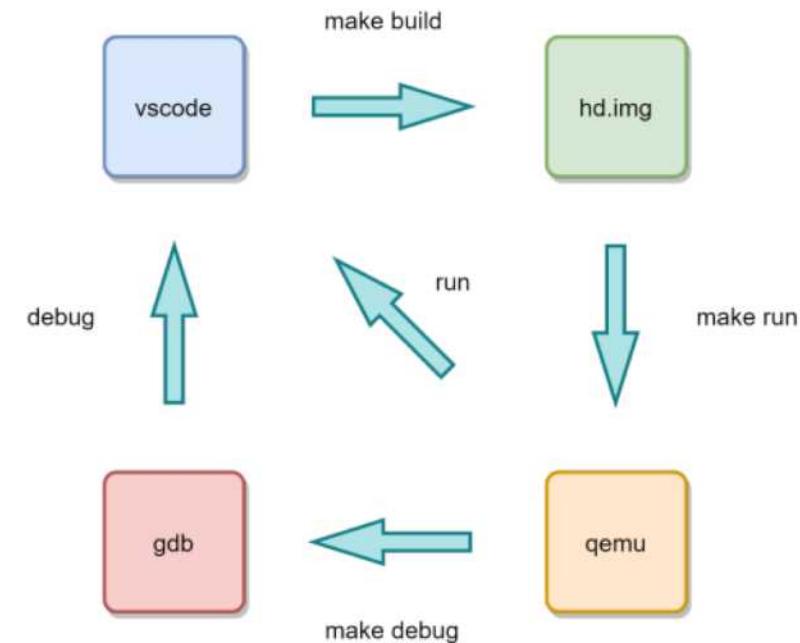
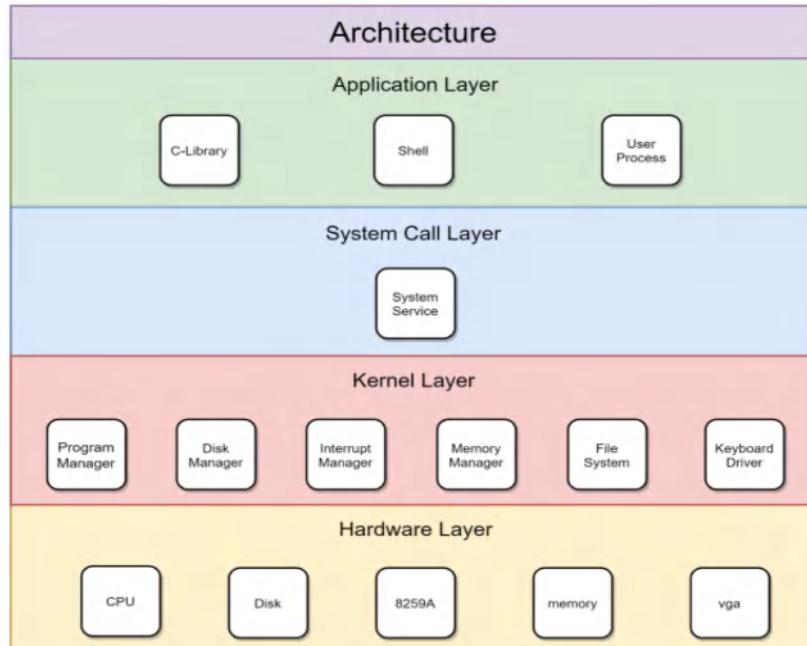
## ➤ 从零构建操作系统的“乐”之比赛获奖





## ➤ 操作系统比赛获奖作品：面向i386架构的教学原型操作系统

针对i386架构的处理器，实现了包含进程、内存、中断、I/O等模块的原型操作系统；



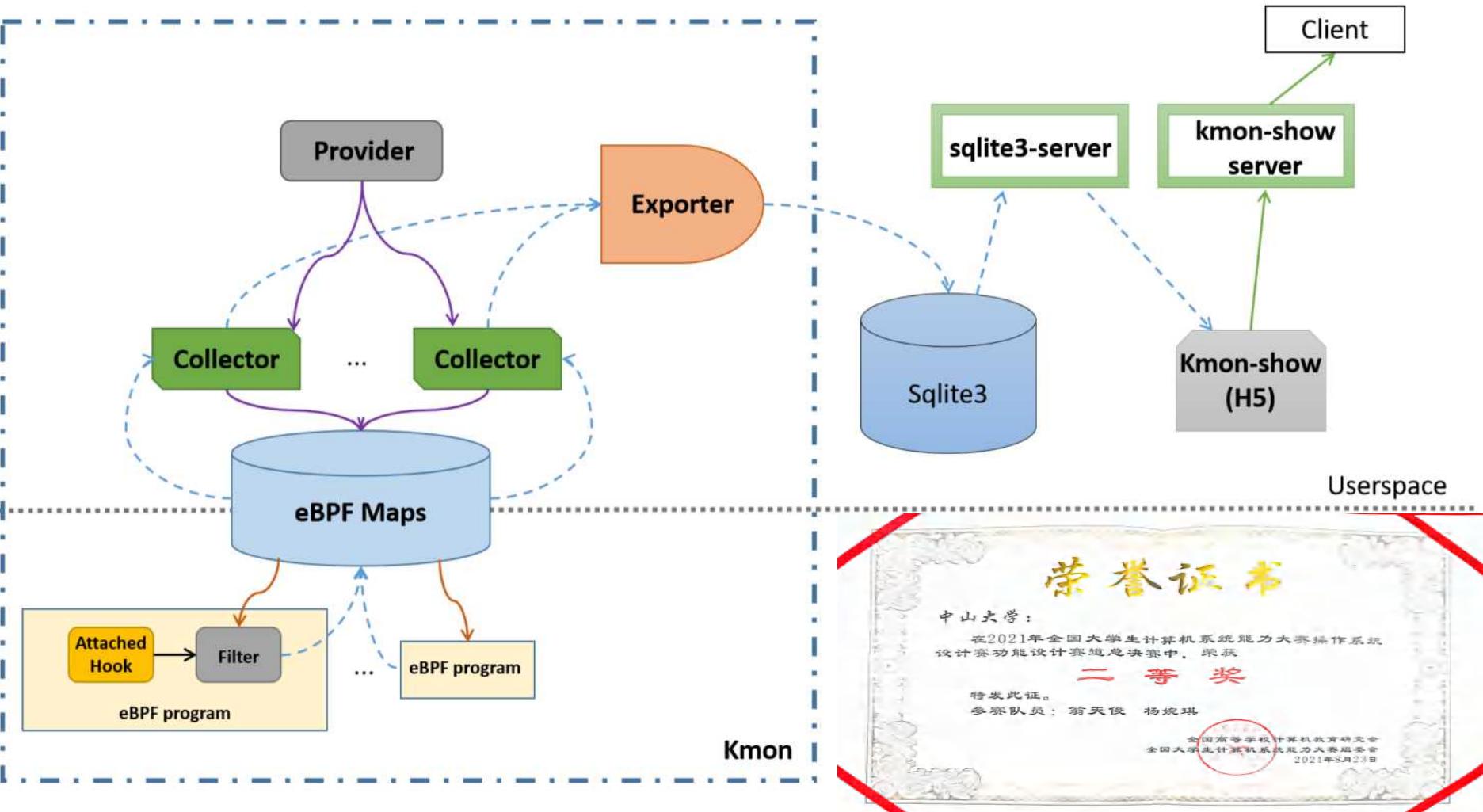
Language	files	blank	comment	code
C++	13	339	126	1529
Assembly	4	124	38	459
C/C++ Header	20	117	109	421
make	1	17	1	50
Markdown	1	4	0	17
<b>SUM:</b>	<b>39</b>	<b>601</b>	<b>274</b>	<b>2476</b>





## ➤ 操作系统比赛获奖作品：基于eBPF的性能检测和展示工具

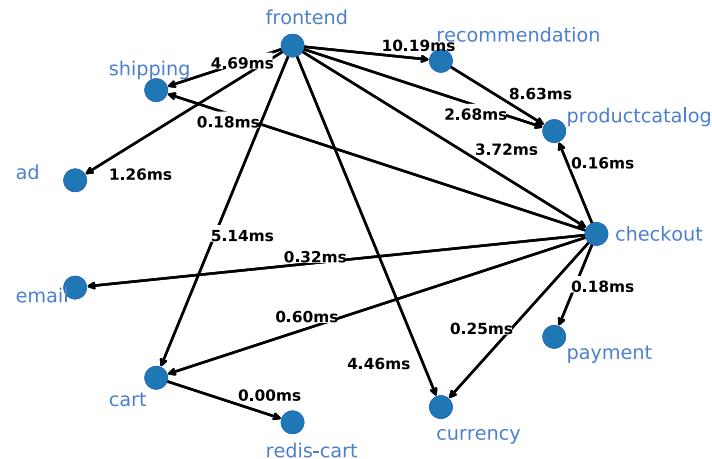
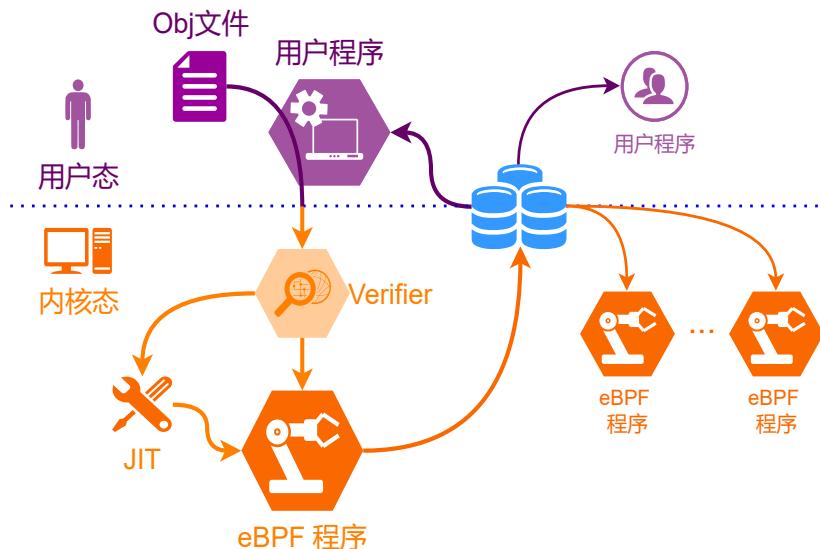
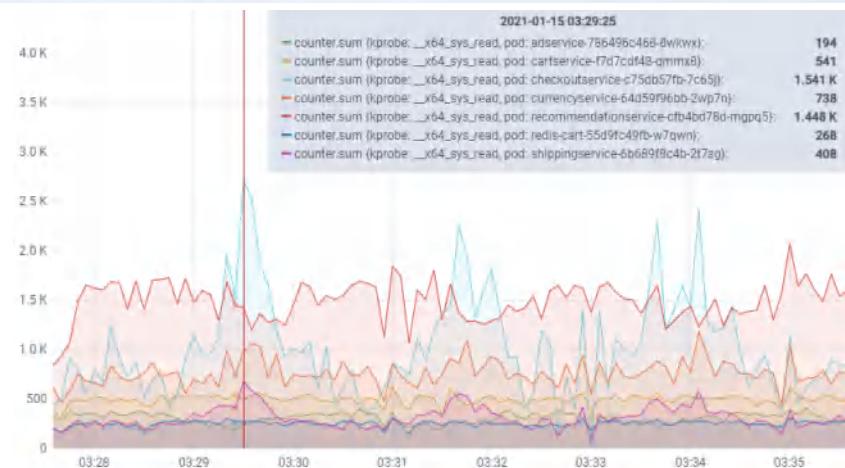
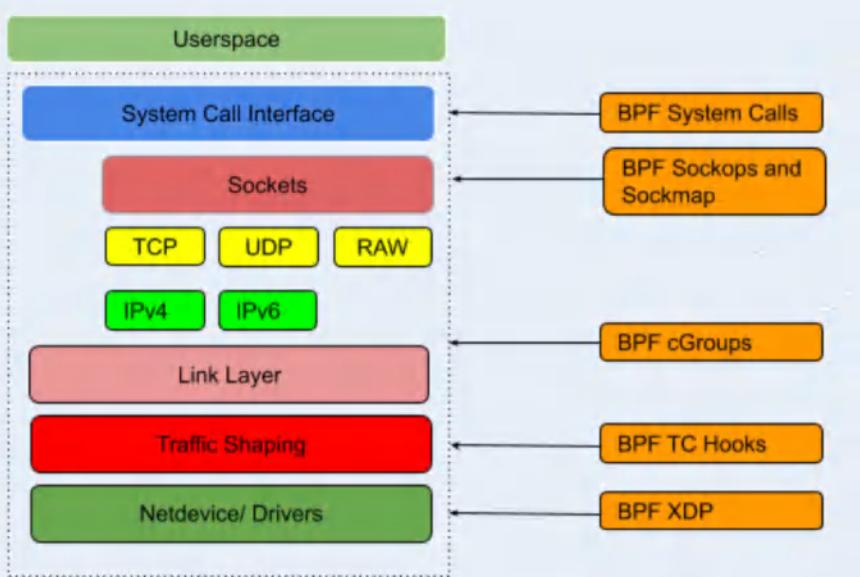
基于eBPF的内核级别的监控技术，采集拓扑、业务指标、日志、系统调用等全栈指标；





## ➤ 操作系统比赛获奖作品：基于eBPF的性能检测和展示工具

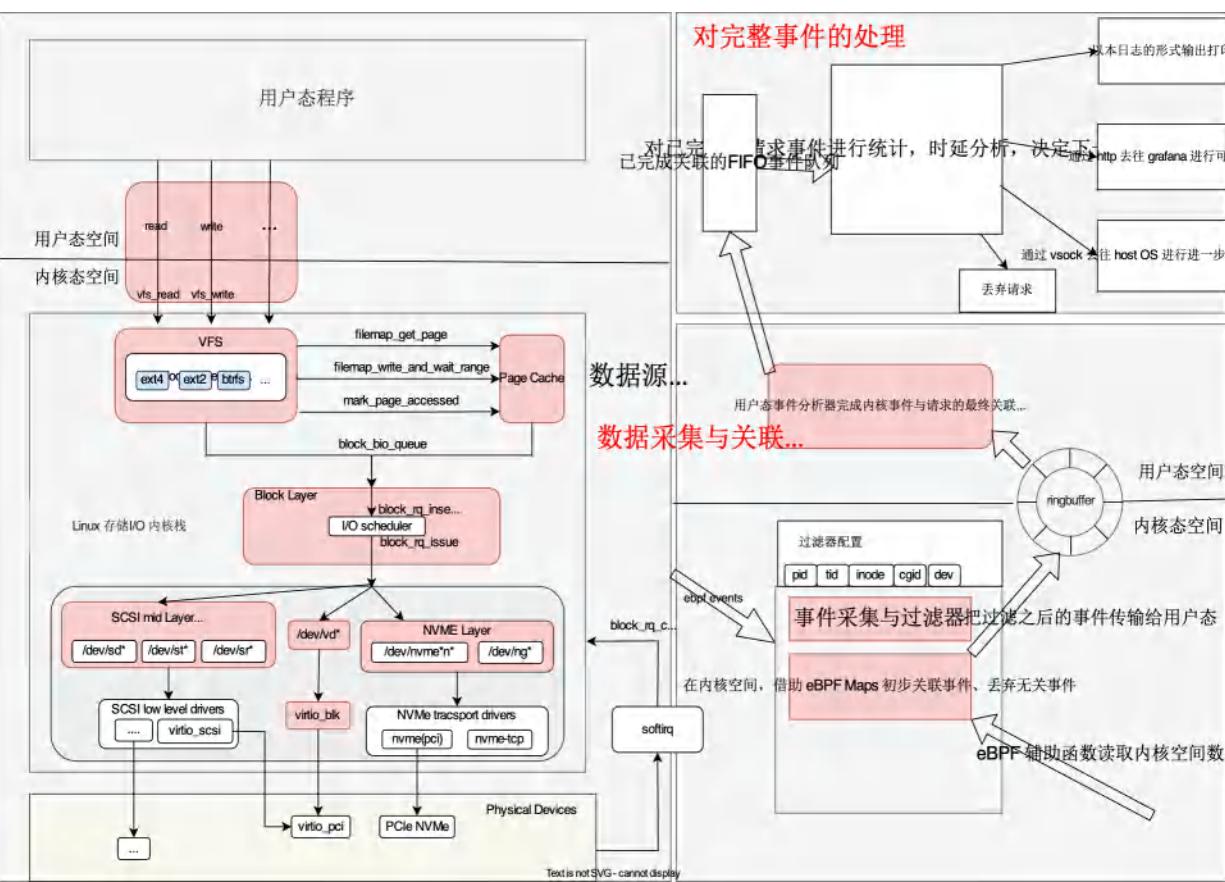
基于eBPF的内核级别的监控技术，采集拓扑、业务指标、日志、系统调用等全栈指标；





## ➤ 操作系统比赛获奖作品：基于eBPF内核路径性能分析

能够追踪内核存储 IO 涉及 fs、block、driver 层的操作；bio, request, page 等数据结构，read/write 等 IO 系统调用在内核中各个子模块的处理流程，系统开销不超过5%。拓展到了虚拟场景下virtio，从 guest 到 qemu 再到 host 的全链路追踪。





## ➤ 操作系统比赛获奖作品：基于eBPF内核路径性能分析

能够追踪内核存储 IO 涉及 fs、block、driver 层的操作；bio, request, page 等数据结构，read/write 等 IO 系统调用在内核中各个子模块的处理流程，系统开销不超过5%。拓展到了虚拟场景下virtio，从 guest 到 qemu 再到 host 的全链路追踪。

```
hrpccs@ubuntu:~/workspace/project1466467-176089/build$ sudo [REDACTED]
hrpccs@ubuntu:~/workspace/project1466467-176089/runbenchmark$ python3 launch_benchmark
.py [REDACTED]
```



## ➤ 操作系统比赛获奖作品：基于eBPF的Linux进程崩溃信息收集

实现基于eBPF的Linux进程崩溃信息收集，包括进程元信息、进程状态、进程退出信号、调用栈、依赖库等

```
Copyright (c) 2018, Intel Corporation. All rights reserved.
Brought to you by Prof. Pengfei Chen
Modified by Pengfei Chen
Developed by Pengfei Chen
and Students of SISI in Grade 2016
Haojun Wang
Haojun Wang
Dai Tao
Dai Tao
Hongyu Yu

Basic System Information
- os: Linux
- system.release: 4.15.0-40-generic
- system.version: 4.15.0-40-generic
- system.machine: Intel(R) Xeon(R) CPU E-12850 CPU @ 1.80GHz
- system.processor: Intel(R) Xeon(R) CPU E-12850 CPU @ 1.80GHz
- system.cpu: 1
- system.total_memory: 16384
- system.free_memory: 12248
- system.total_swap: 0
- system.free_swap: 0
- system.total_swap_usage: 0
- system.swap_usage: 0
Memory Information
- memory.total: 16384 MB
- memory.free: 12248 MB
- memory.balanced: 16384 MB
- memory.page_size: 4096 KB
- memory.stack_size: 16384 KB
Disk Information
- disk.total: 16384 MB
- disk.free: 16384 MB
- disk.balanced: 16384 MB
- disk.page_size: 4096 KB
- disk.stack_size: 16384 KB
Graphics Card Model: Intel(R) Xeon(R) CPU E-12850
Processor Manufacturer: Intel Corporation
Processor Model: Intel(R) Xeon(R) CPU E-12850 Processor
Processor Version: 4.0.0
Processor Revision: 0.0.0
Processor Time: 07:27:00
Processor Status: Intel(R) Corporation K80000000000000000000000000000000
Processor Status: Initializing
```

```
Copyright (c) 2018, Intel Corporation. All rights reserved.
Brought to you by Prof. Pengfei Chen
Modified by Pengfei Chen
Developed by Pengfei Chen
and Students of SISI in Grade 2016
Haojun Wang
Haojun Wang
Dai Tao
Dai Tao
Hongyu Yu

Basic System Information
- os: Linux
- system.release: 4.15.0-40-generic
- system.version: 4.15.0-40-generic
- system.machine: Intel(R) Xeon(R) CPU E-12850 CPU @ 1.80GHz
- system.processor: Intel(R) Xeon(R) CPU E-12850 CPU @ 1.80GHz
- system.cpu: 1
- system.total_memory: 16384
- system.free_memory: 12248
- system.total_swap: 0
- system.free_swap: 0
- system.total_swap_usage: 0
- system.swap_usage: 0
Memory Information
- memory.total: 16384 MB
- memory.free: 12248 MB
- memory.balanced: 16384 MB
- memory.page_size: 4096 KB
- memory.stack_size: 16384 KB
Disk Information
- disk.total: 16384 MB
- disk.free: 16384 MB
- disk.balanced: 16384 MB
- disk.page_size: 4096 KB
- disk.stack_size: 16384 KB
Graphics Card Model: Intel(R) Xeon(R) CPU E-12850
Processor Manufacturer: Intel Corporation
Processor Model: Intel(R) Xeon(R) CPU E-12850 Processor
Processor Version: 4.0.0
Processor Revision: 0.0.0
Processor Time: 07:27:00
Processor Status: Intel(R) Corporation K80000000000000000000000000000000
Processor Status: Initializing
```

kprobe\_do\_exit

挂载点触发bpf程序

eBPF功能强，安全，但难移植

- Libbpf + CO-RE
- BTFGen + BTFHub

可移植版本分发

```
/* The DATA, ID, TIME, PRIORITY, and CPUINFO need to be avoided
 * SEC("bpf/sched/sched_process_exit")
 * int trace_event_raw_sched_process_exit(struct trace_event_raw_sched_process *ctx)
 * {
 *     struct task_struct *task;
 *     struct event *e;
 *
 *     u64 pid = bpf_get_current_pid_tgid();
 *     u32 tid = (u32)pid;
 *     pid = pid >> 32;
 *     /* get information in task_struct
 *     task = (struct task_struct *)bpf_get_current_task();
 *     int exitcode = __bpf_core_read(task,exit_code);
 *
 *     if(exitcode == e) //exit code
 *         return 0;
 *
 *     /* bpf event to ringbuffer can be read by user space
 *     e = bpf_ringbuf_reserve(skb, sizeof(*e), 0);
 *     if(e)
 *     {
 *         e->stack_id = bpf_get_stackid(ctx, &map_stack_traces, 0);
 *         e->exitcode = exitcode >> 8;
 *         e->sig = exitcode & 0xf;
 *         e->pid = __bpf_core_read(task, parent, pid);
 *         bpf_get_current_com(task, &e->com), TASK_COMM_LEN);
 *         bpf_ringbuf_submit(e, 0);
 *     }
 *
 *     return 0;
 * }
```

BPF程序过滤，追踪内核数据

组件部署阶段

加载eBPF程

```
exitcatch_bpf__open();
exitcatch_bpf__load();
exitcatch_bpf__attach();
```

进程崩溃信息收集

从perf event或ringbuffer中获取信息

```
static int handle_event(void *ctx, void *data, size_t data_sz)
{
    const struct event *e = data;
    /* ... */
    /* get_stack_from_task
     * int ret = bpf_map_lookup_elem(
     *     &skel->maps.map_stack_traces,
     *     &e->stack_id,
     *     sizeof(unsigned int),
     *     &stacks,
     *     VALUE_SIZE,
     *     0);
     */
    /* event_handling
     * ... */
    return 0;
}
```

信息整合与处理

用户态

内核态

Helper Function

```
8 //Use a ringbuffer map to send data down to user space
9 struct {
10     __uint(type,BPF_MAP_TYPE_RINGBUF);
11     __uint(max_entries,256 + 1024);
12 }rb SEC(".maps");
13
14 //Use to store stacktrace and will be used by bpf_set_stackid()
15 struct {
16     __uint(type,BPF_MAP_TYPE_STACK_TRACE);
17     __uint(key_size,sizeof(u32));
18     __uint(value_size,VALUESIZE);
19     __uint(max_entries,1024);
20 } map_stack_traces SEC(".maps");
21
```

BPF maps  
存储数据  
用户程序与内  
核交换数据



## ➤ 操作系统比赛获奖作品：基于eBPF的Linux进程崩溃信息收集

实现基于eBPF的Linux进程崩溃信息收集，包括进程元信息、进程状态、进程退出信号、调用栈、依赖库等

```
tz@tangzhe:~/es/competition/OScomp/build$ sudo ./exitcatch.bin
```

```
=====Basic System Information=====  
-----OS Information-----  
System Name: Ubuntu 20.04.4  
System Bits: 64  
Kernel Version: 5.15.0-46-generic  
-----CPU Information-----  
Basic CPU Information: Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz  
CPU_Architecture: x86_64  
Physical CPUs Number: 4  
Cores Per CPU: 1  
Logical CPUs Number: 4  
L1d cache: 128 KiB  
L1i cache: 128 KiB  
L2 cache: 1 MiB  
L3 cache: 24 MiB  
-----Memory Information-----  
Memory Information: 1*4896 MB  
Total Memory: 3.80GB  
Memory Balance: 767.99MB  
Memory Usage(%): 80.26%  
Virtual Memory Usage: 59.52MB  
-----Disk Information-----  
Disk Model: VMware Virtual S  
Total Disk: 53.47 G  
Disk Balance: 27.88 G  
Disk Usage(%): 52.14%  
-----Other Information-----  
Graphics Card Model: VMware SVGA II Adapter  
Mainboard Manufacturer: Intel Corporation  
Mainboard Model: 440BX Desktop Reference Platform  
BIOS Vendor: Phoenix Technologies LTD  
BIOS Version: 6.00  
BIOS Release Time: 07/22/2020  
Netcard Info: Intel Corporation 82545EM Gigabit Ethernet Controller (Copper) (rev 01)  
>>>>>Finished initializing...  
Log will be saved in ./Log
```

①硬件基本信息  
②内核符号表完成初始化  
③崩溃报告日志指定保存路径



## ➤ 操作系统比赛获奖作品：基于eBPF的Linux进程崩溃信息收集

实现基于eBPF的Linux进程崩溃信息收集，包括进程元信息、进程状态、进程退出信号、调用栈、依赖库等

```

CrashCrawler is a Linux crashed process capture program based on eBPF.

Adivsed by
  Prof. Pengfei Chen
  Shengjie Ji

Developed by
  Prof. Pengfei Chen
  HPC Students of SYSU in Grade 2020
  RuiPeng Hong
  Zhe Tang
  Hengyu Fu

=====
Basic System Information=====
--OS Information--
System Name:      Ubuntu 20.04.4
System Bits:       64
Kernel Version:   5.15.0-46-generic
--CPU Information--
Basic CPU Information: Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz
CPU_Architecture:  x86_64
Physical CPUs Number: 4
Cores Per CPU:    1
Logical CPUs Number: 4
L1d cache:        128 KiB
L1i cache:        128 KiB
L2 cache:         1 MiB
L3 cache:         24 MiB
--Memory Information--
Memory Information: 1*4096 MB
Total Memory:      3.80GB
Memory Balance:    440.60MB
Memory Usage(%):  88.67%
Virtual Memory Usage: 60.73MB
--Disk Information--
Disk Model:        VMware Virtual S
Total Disk:        53.47 G
Disk Balance:      28.97 G
Disk Usage(%):    54.18%
--Other Information--
Graphics Card Model: VMware SVGA II Adapter
Mainboard Manufacturer: Intel Corporation
Mainboard Model:    440BX Desktop Reference Platform
BIOS Vendor:        Phoenix Technologies LTD
BIOS Version:      6.00
BIOS Release Time: 07/22/2020
Netcard Info:       Intel Corporation 82545EM Gigabit Ethernet Controller (Copper) (rev 01)
>>>>>Finished initializing...
Log will be saved in: /var/log/crashlog

```

```

[ex]-[tz@tangzhe]--[~/os/competition/OScomp/crashtest/segfault]
$ ./write_read_only_mem.bin

```



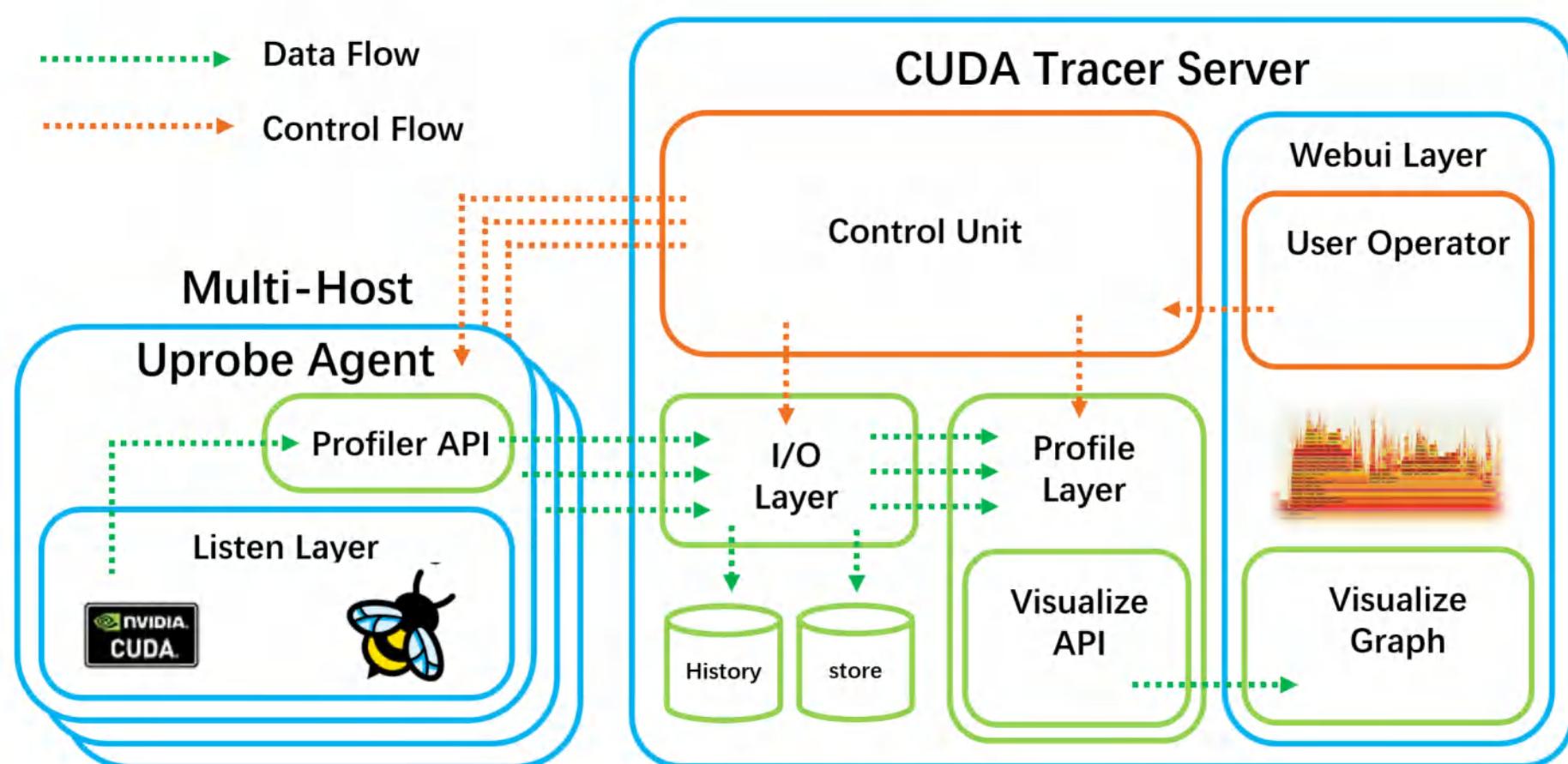
➤ 操作系统比赛获奖作品：基于eBPF的Linux进程崩溃信息收集  
获得全国操作系统大赛三等奖，并且获得华为2023“鲲鹏应用创新大赛”高校组银奖；





## ➤ 操作系统比赛作品：基于eBPF的大模型全栈可观测性

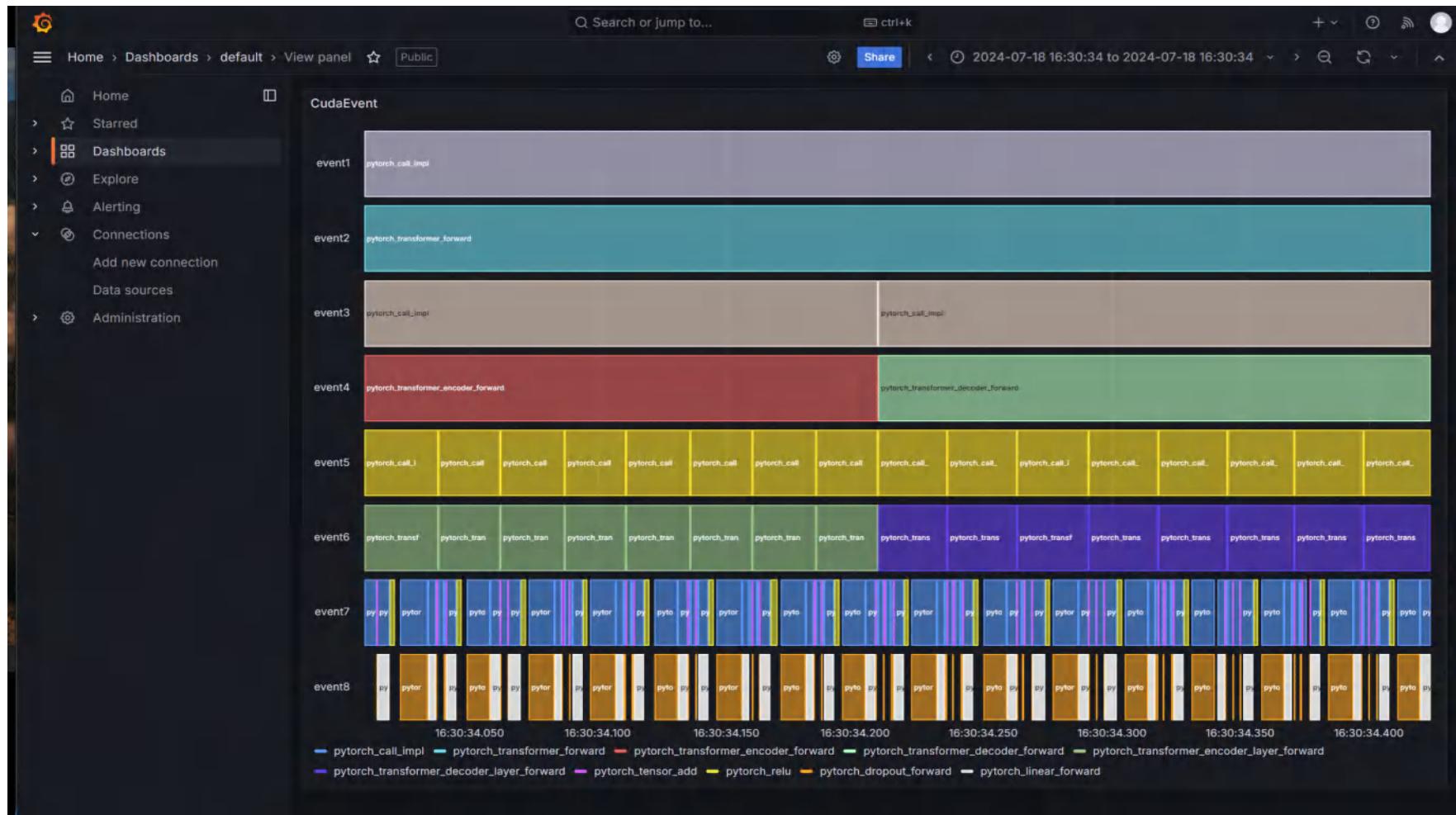
基于eBPF，实现 C-CUDA 层、Torch 算子层、Python 层进行事件追踪，帮助 AI 开发人员识别瓶颈并优化性能；





## ➤ 操作系统比赛作品：基于eBPF的大模型全栈可观测性

基于eBPF，实现 C-CUDA 层、Torch 算子层、Python 层进行事件追踪，帮助 AI 开发人员识别瓶颈并优化性能；





# 从零构建操作系统的“乐”之教学论文

基于操作系统实验教学内容，在中国高校计算机教育大会2022发表论文2篇，并获优秀论文二等奖；

## YatSen OS: 一个支点撬动操作系统大山

### —教学操作系统及实验设计

张韵宇, 陈鹏飞, 凌应标

中山大学计算机学院

#### 摘要

操作系统是计算机类学生培养的基础和核心课程之一，教学效果决定了学生对计算机系统的理解深度。为了强化和巩固学生对操作系统原理知识的理解，多数高校往往开设操作系统实验课程。但是，实验课程的内容、难度、质量、知识覆盖面等参差不齐。为操作系统教学改革带来一些不确定因素，传统的操作系统实验教学多基于现代操作系统以高级语言的形式实现操作系统原理知识。但是，随着 ARM、Risc-v 和国产芯片等新型 CPU 汇入以及新型编程语言如 Rust 的兴起和流行给原有的操作系统方案带来了极大的挑战，亟需提出一种和操作系统发展趋势同步的新方案。因此，本项目提出了“以点带面”的实现方案。首先，在使用 C/C++ 与 x86 组合的实验模式下实现了一种教学操作系统及实验方案。以此为支点，可以同时开展“64 位操作系统”、“Rust + ARM + 树莓派”、“Rust + Risc-v”等实验方案的探索，从而全面推动操作系统改革。在完成“支点”的过程中，本项目提出了一种简明、全面、易用。渐进的实验方案设计原则，编写了一套以递进演进方式叙述的实验教材——《操作系统实验简明教程》，自底向上介绍了如何从零开始编写操作系统。同时，该方案已经应用到中山大学 2021、2022 年春季操作系统实验课程中，并收获学生的一致好评。目前，相关文档和源码已在 github 开源，收获 star 数量 400 有余，相较于其他方案（16 位模式操作系统）和使用 ucore 的方案，学生对操作系统概念、操作系统开发和操作系统设计等重要知识的理解程度均有明显提高。通过提出新方案完成了操作系统实验课的初步改造方案。“给一个支点，可以撬动操作系统这座大山”。操作系统的构建不再是遥不可及的事情。

关键词：操作系统实验，保护模式，实验教材，Risc-v

## 1 引言

一直以来，从零开始渐进实现操作系统是中山大学操作系统课程的一大特色。但是，随着操作系统的发展，Rust、Risc-v 和 ARM 等新技术开始不断涌现。在深入调查了操作系统的发展现状后，本文认为需要革新原有的操作系统实验方案，使方案能够保持和操作系统发展趋势相一致。改革并不是一蹴而就，需要的工作量巨大。同时，改革并不是对旧方案的全盘否定，而是继承和发展。在分析和比较旧方案和操作系统现状后，本文采取了一种折中方案。首先，使用 C/C++ 高级语言，在 x86 的实验模式下提出一种新方案。新方案保留了中山大学一以贯之的特色——从零开始编写操作系统。但在此基础上，本文提出了一种简明、全面的实验方案设计原则，独立自主地编写了 32 位保护模式操作系统代码。其中，架构设计如图 1 所示，内存规划如图 2 所示。基于本项目的代码，编写了一套以递进演进方式推进的实验教材——《操作系统实

## 操作系统实验课程教学的原型演化方法

陈鹏飞, 陈贵锋, 凌应标

《中山大学计算机学院, 广东 广州 510275》

**摘要：**在操作系统原理实验课程的多年教学实践中，该文作者进行了教学方法改革和创新，提出操作系统原理实验课程的原型演化方法，并在教学实践工作中不断完善，积累了一些实践经验和教训。本文全面介绍了该课程的教学改革和创新工作。从制定课程教学目标开始，说明该课程教学目标提高和设计实现完善需要作系统原型的必要性。进而介绍提出的该课程原型演化教学方法，展示相关的教学实践和取得的成效。多年的教学实践表明，原型演化方法具有较多优势，有助于该课程教学目标的实现。该文提升了学生的专业核心能力，更加适合新工科规划的计算机专业的本科人才培养。

**关键词：**新工科、计算机专业核心能力、操作系统、操作系统实验课、原型演化教学方法

## Prototype Evolution Method for Teaching Operating System Lab Course

CHEN peng\_fei, ZHENG gu\_feng, LING ying\_biao

(School of Computer Science and Technology, SUN YAT-SEN University, Guangzhou, Guangdong, China)

**Abstract:** In the teaching practice of the operating system principle course for years, we have carried out reform and innovation in teaching. We explored and continuously improved the prototype evolution method of the operating system principle lab course in our teaching practice, accumulating some practical experience and lessons. This paper provides a comprehensive overview of our teaching reform and innovation work of this course. Starting from the formulation of the teaching objectives of the course, explaining the need to improve the teaching objectives of the course to designing and implementing a complete operating system prototype, and then introducing the proposed prototype evolution teaching method of the course, showing the relevant teaching practice and effects achieved. Years of teaching practice have shown that the prototype evolution method has more advantages, which is conducive to the realization of the teaching objectives of the course, significantly improves the students' professional core ability, and is more suitable for the undergraduate student training of computer science majors in new engineering planning.

**Key words:** new engineering disciplines: professional core ability of computer science major: operating systems: operating system principle lab course: prototype evolution method

## 1 引言

近年来国内国际形势变化很大，党中央组织召开了全国教育大会和教育部召开了改革开放以来第一次新时代中国高等学校本科教育工作会议，提出了“着力提升学生解决复杂工程问题的能力。加大课程整合力度，推广实施案例教学、项目式教学等研究性教学方法，增强综合性项目训练”。在改革任务和重点举措方面的第二点提出“着力提升学生解决复杂工程问题的能力。加大课程整合力度，推广实施案例教学、项目式教学等研究性教学方法，增强综合性项目训练”。在改革任务和重点举措方面的第三点“创新创业”中，要求“给人才留空间、营造创新环境。”和“注重个性化培养，给学生提供自主选择导师、专业和课程的空间。开展研究性教学，鼓励学生参与科研项目训练，促进学生自主深度学习，建树知识体系，形成多种能力。”

还有，教育部对“双一流”高校建设的指导意见发布了《关于高等学校加快“双一流”建设的指导意见》[6]。在引导学生成长成才这一点上，也



# 感谢倾听！