JudgePenguin

基于Linux的应用程序稳态测试系统

中期汇报

致理-信计01 单敬博 2020012711

2023/4/16

目录

- 项目分析
  - 回顾：需求与技术选型
  - 工作流程：暂停、评测、恢复
  - 关键技术/主要挑战
- 上周进展
- 真机演示
- 下周计划

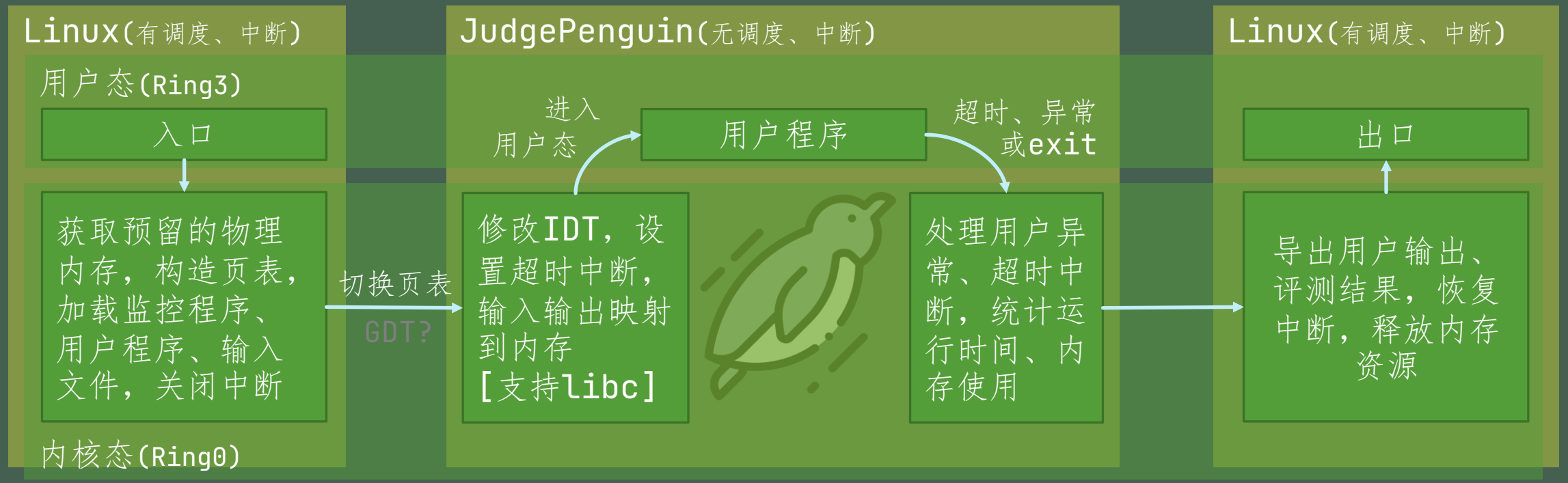
回顾：需求与技术选型

- **需求**：对给定用户程序进行**准确、稳定、安全**的运行时空测量
- 现有项目/框架因不稳定、无防护、强依赖等原因效果不佳

- **技术选型**：Linux Kernel Module
  - 架构：x86_64 Linux (Ubuntu 20.04, kernel 5.4.0)
  - 语言：C/C++ 内核模块入口暂时只能用C实现

工作流程：暂停、评测、恢复

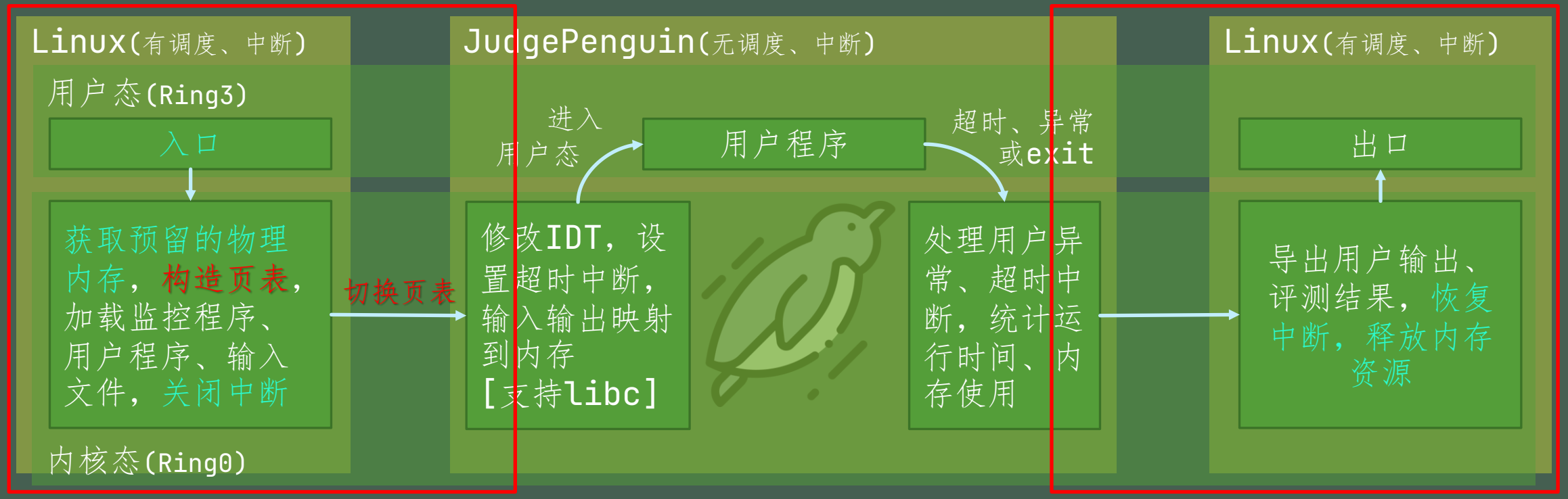
- 三个阶段：暂停Linux，进行评测，恢复Linux



关键技术/主要挑战

- **评测**阶段：已经在JudgeDuck中有较完善的实现
  - 主要的工作：熟悉x86-64架构细节，逐步完成功能迁移
- **暂停/恢复**阶段：项目的关键技术与主要挑战所在
  - 评测需要获取系统的完整控制权，可能会破坏一些Linux建立好的环境
    - GDT / 页表 / IDT
  - Linux内核码量巨大，逻辑复杂，对其进行操作比较困难 -> ioremap
  - 需要完整保存Linux内核状态，使完成恢复后内核正常运行 -> watchdog
  - 已经遇到了不同版本内核的接口变动问题，尝试提升兼容性

关键技术/主要挑战


Linux(有调度、中断)
用户态(Ring3)
入口
获取预留的物理内存，构造页表，加载监控程序、用户程序、输入文件，关闭中断
内核态(Ring0)
切换页表
JudgePenguin(无调度、中断)
进入用户态
修改IDT，设置超时中断，输入输出映射到内存
[支持libc]
用户程序
超时、异常或exit
处理用户异常、超时中断，统计运行时间、内存使用
Linux(有调度、中断)
出口
导出用户输出、评测结果，恢复中断，释放内存资源

（上周的）下周计划

- 进入内核模块后暂停其他核上正在执行的任务
- 排查未关闭的中断，使rdtsc取得稳定的间隔
- 在内核模块中获取预留的物理内存
- 学习x86页表结构，尝试在内核模块中使用预留内存

rdtsc测试

- rdtsc测试
  - 使用rdtsc指令连续多次读取CPU时间戳，记录最大间隔
  - 若没有中断影响，最大值应稳定取得较小的值(≤100)
- 在未关闭中断情况下运行：$t_{max}>10^4$ ($>10^3$ in 10^8)
- 上周关闭中断(?)后运行：
  - 大部分情况$t_{max}<100$
  - 偶尔仍出现$t_{max}>10^4$ (<5 in 10^8)　怀疑其他核仍在执行！

暂停其他核上的任务

- 使用on_each_cpu在所有核上执行相同的内容（底层使用IPI实现）
- 使用atomic_t原子量使所有核关闭中断后同步到相同位置
- 接下来主核独自执行后续任务，其他核使用cpu_relax忙等

- 现象：系统立即卡住，但rdtsc间隔仍偶尔出现t_{max}>1000（=1）

未关闭的中断：Lockup Watchdog

- 连续多次运行测试(耗时超过20s)，在内核日志中发现了：

  `NMI watchdog: Watchdog detected hard LOCKUP on cpu *`

  这是由Hard Lockup Watchdog发出的Non-Maskable中断
- 解决方法：在内核模块中定期touch_nmi_watchdog(喂狗?)；
  使用`sysctl -w kernel.watchdog=0`关闭watchdog~~(杀狗)~~
- 处理好watchdog后，rdtsc测量间隔稳定地取得t_{max}<100

关闭Linux调度、中断

- 简单的测试
  - ✓ 加载内核模块后，图形界面出现短暂卡顿
  - ✓ 关闭中断后执行死循环，系统最终卡死
  - × 前台进程仍在运行，发生系统调用后卡死
- 基于rdtsc的测试
  - 连续10^8次使用rdtsc获取系统时间戳，计算差分的最大值、平均值
  - × 均值约为20，但极大值可能高达10000+，怀疑某些中断(时钟中断)没有正确屏蔽
  - × 还发现了关闭中断后不同位置测量结果出现很大差异的现象，原因待排查

获取预留的物理内存

- 在grub中添加memmap=0x20000000\\\$0x40000000，预留512MiB物理内存
- 在内核模块中使用request_mem_region获取物理内存资源

```
40000000-5fffffff : Reserved
  40000000-5fffffff : JudgePenguin MEM
```

- 接下来应该寻找一段虚拟地址与物理地址建立映射，参考jailhouse的做法，先获取一块vm area，再进行ioremap

反复陷入Linux内核的大坑 (1/∞)

- 首先尝试使用__get_vm_area获取可用虚存片段
  - 运行后直接死机 
  - 排查许久后发现这个函数使用了一些与irq相关的函数
  - 意识到或许不应该在关完中断后使用它
  - 调整位置后恢复正常
- 排查过程中还发现了内核的正式接口get_vm_area 

反复陷入Linux内核的大坑 (2/∞)

- 接下来自然是想用printk(printf的内核版本,pr_info是其二次封装)来输出一下分配到的虚存地址:

```
pr_info("virt_addr = %p\n", vma->addr);
```

  - 结果发现 `virt_addr = 00000000cdc42852`，没有4KiB对齐🤔
  - 经过对get_vm_area的详细调查无果后，使用%llu输出强转为ull的地址，得到 `virt_addr=0xffff9b94a0000000`，竟然对齐了😅
  - 意识到可能是printk的%p有问题，查阅内核文档发现需要用%pK才能打印内核指针，~~调了一下午不存在的bug~~

反复陷入Linux内核的大坑 (3/∞)

- 下一步是把虚存映射到物理地址上去，参考jailhouse定义了：

```
static typeof(ioremap_page_range) *ioremap_page_range_sym;
```

  - 误以为这是某种内核特有的魔法，然后喜提 ~~空指针异常~~ 编译错误！ 
  - 仔细翻找jailhouse代码发现了如下真正的魔法：

反复陷入Linux内核的大坑 (3/∞)

Unfortunately, since Linux v5.7 kallsyms_lookup_name is also unexported,

```
#if defined(CONFIG_KALLSYMS_ALL) && LINUX_VERSION_CODE < KERNEL_VERSION(5, 7, 0)
#define __RESOLVE_EXTERNAL_SYMBOL(symbol)                                      \
  symbol##_sym = (void *)kallsyms_lookup_name(#symbol);                        \
  if (!symbol##_sym)                                                           \
  return -EINVAL
#else
#define __RESOLVE_EXTERNAL_SYMBOL(symbol) symbol##_sym = &symbol
#endif
#define RESOLVE_EXTERNAL_SYMBOL(symbol...) __RESOLVE_EXTERNAL_SYMBOL(symbol)

  RESOLVE_EXTERNAL_SYMBOL(ioremap_page_range);
```

反复陷入Linux内核的大坑 (3/∞)

- `uname -r`发现内核版本为5.15.0，反复尝试各种hack方法获取kallsyms_lookup_name与ioremap_page_range无果
- 于是暂时降级到内核版本5.4.0 (待主要功能完成后再考虑升级内核)
- 降级内核之后又修了好久grub，写出了grub特有的`GRUB_DEFAULT="1>4"`
- 至此终于顺利完成了内存映射

预留内存测试

- 完成了预留内存的虚存映射后，需要做一些测试来确认映射正常
  - 在虚存区域的头尾部各取一些地址，进行读取、自增、写入操作
  - 通过观察操作是否成功来确认是否成功映射
  - 通过观察重新加载模块 或 系统热重启后数据是否相同来推测是否映射到了被保留的物理内存

真机演示

下周计划

- 在预留内存中创建x86_64多级页表
- 设置跳板页，实现页表切换
- 参考JudgeDuck，调研监控程序的实现方式
  - 重点研究trap_handler、IDT配置、超时中断

感谢聆听 & 欢迎提问

致理-信计01 单敬博 2020012711