2022操作系统功能挑战初赛报告文档

报告小组：QUINT

成员：清华大学 王之栋，项晨东，孙迅

目标描述

我们小组的选题为 **proj6-RV64N-user-level-interrupt** 。

我们希望探索运用新一代 Intel 硬件特性**用户态中断**（User Interrupt），搭建运行环境，设计新的 IPC 场景和框架，不断与传统 IPC 及 XPC、underbridge、skybridge 等新 IPC 进行性能比较并优化。

即选题仓库中第三题、第四题和第六题所描述的大致工作方向。一方面，我们希望测试用户态中断是否更快，理解它快在哪里，相比之下传统方式的性能瓶颈在哪里。另一方面，我们也希望把这个特性用起来，扩展操作系统内核，搭建我们设计的 IPC 框架，并找出合适的应用场景，说明它优化的有效性。

更多具体内容见文档后续部分。

比赛题目分析和相关资料调研

微内核正在变得越来越有影响力，相比之下宏内核，各模块之间的协同由函数调用的形式变成了进程间通信（IPC）的形式，因此如何优化IPC的性能变成了一个关键问题。

近几年，陈海波老师团队提出了诸如 XPC 、skybridge 、underbridge 等优化 IPC 问题的方式，但都各有一些缺陷——如 XPC 对硬件和内核的修改过大，underbridge 在安全上总让人感到担忧。

Intel 推出的新一届硬件特性中，有一种名为用户态中断的机制（在Intel的手硬件规范手册中进行了介绍），可以在不陷入内核的情况下，将中断发到用户空间中处理。Linux应用该特性已经实现了一版内核，并进行了简单的性能测试，说明了 uintr 在 IPC 上的性能优势。

但是 Linux 的工作并没有深入，只是提供了一个工作基础。一方面，uintr本身是一种通知机制，不具备信息传递的功能，因此为了实现 IPC 的需求，uintr需要与信息传递结合在一起，并通过系统调用的形式以统一接口为用户呈现。另一方面，关于 uintr 的潜力如何，有哪些高效的应用场景，也仍是等待发掘和设计。

因此，设计 IPC 框架，拓展内核实现，并测试性能和应用场景，便是我们小组的工作目标。

我们已进行了一系列的充分调研：

阅读了 XPC 、 underbridge 、skybridge 论文。

阅读与整理了 Intel Architecture Instruction Set Extensions and Future Features 中与用户态中断相关的硬件规范，主要是第2章与第11章。详见 `ppt/uintr-intel-linux.pptx` 。

阅读与整理了 uintr-linux-kernel （基于 linux 运用 uintr 特性的内核版本）仓库中的 commits 。详见 `ppt/uintr-intel-kernel commit summary.pptx` 。

对之后希望实现的 IPC 场景与框架进行了简单的调研与设计，受限于当前工作重心，该部分暂没有成熟的想法。阶段性想法产出如下：

对 XPC 和 uintr 进行了比较分析，提出 uintr+shmem 实现的 IPC 框架，详见 `ppt/初步设想.pdf` 。

对 uintr 的实际应用场景和会遇到的问题进行了简单分析，详见 `ppt/应用场景.md` 。

开发计划

第一步

受限于疫情影响，我们没办法拿到有 uintr 特性的物理机。为了方便后续工作开展，第一步我们将基于 qemu 实现支持 uintr 的模拟器。

预期产出：能在我们的 qemu 上跑 uintr-linux-kernel ，并正确通过 linux 实现的简单功能测例 uipi_sample.c 。

第二步

进行性能测试，基于 Linux RFC 报告使用的 ipc-bench 作测试，比较 uintr 与 pipe等传统 IPC 方式的性能异同。

预期产出：复现 Linux RFC 的性能测试结果。

第三步

自行编写 uintr+shmem 或其他运用 uintr 形式的用户测例，与传统 IPC 方式作比较。

预期产出：比 Linux RFC 更丰富的性能测试结果，验证 IPC 框架设计的可行性。

第四步

修改 Linux 内核，为 IPC 框架提供更通用的系统调用接口，并通过测试。

预期产出：一个初步成型的原型系统。

第五步

寻找能发挥我们系统优势的有意义应用场景，在调度、负载均衡等方面继续改进 IPC 框架和内核。

比赛过程中的重要进展

我们的项目进展可在主仓库的 `README.md` 中看到，其同时链接了我们得到相应进展时的工作记录文档或报告 ppt 。

在6月4日，我们基本完成了开发计划中前两步的内容，即完成了支持 uintr 的 qemu 的开发，并复现了 Linux RFC 的结果。

系统测试情况

第一步

用我们更改的 `qemu-uintr` （仓库地址：https://github.com/OS-F-4/qemu-uintr）运行 Linux 开发的使用 uintr 特性的内核 uintr-linux-kernel（仓库地址：https://github.com/intel/uintr-linux-kernel/），并用该内核运行功能测例 uipi_sample.c （位于 `uintr-linux-kernel/tools/uintr/sample/` 下），可看到运行成功，能看到输出了测例中输出的 `success` ，并且可以正常返回到内核。

```
/ # nosleep
[   46.352405] uintr_register_handler called
[   46.352914] recv: register handler task=85 flags 0 handler 401de5 ret 0
[   46.353651] uintr_create_fd called
[   46.354076] recv: Alloc vector success uintrfd 3 uvec 0 for task=85
Receiver enabled interrupts
[   46.356494] uintr_register_sender called
[   46.357888] send: register sender task=86 flags 0 ret(uipi_id)=0
Sending IPI from sender thread
direct sending
-|-| perv: 3
[   46.360847] uintr_unregister_sender called
        -- User Interrupt handler --
[   46.362767] send: unregister sender uintrfd 3 for task=86 ret 0
[   46.364495] recv: Release uintrfd for r_task 85 uvec 0
[   46.365305] uintr_unregister_handler called
[   46.365728] recv: unregister handler task=85 flags 0 ret 0
Success
```

若希望复现该部分工作，可参考 `ppt/qemu-worklinglog.md` 中的"编译intel实现的linux内核"、"编译测试程序"、"编译qemu"几个部分。

第二步

用我们更改的 `qemu-uintr` 运行 Linux 开发的使用 uintr 特性的内核 uintr-linux-kernel，并用该内核运行 Linux RFC 报告（https://lore.kernel.org/lkml/20210913200132.3396598-1-sohil.mehta@intel.com/）使用的 ipc-bench 性能测试框架（仓库地址：https://github.com/intel/uintr-ipc-bench/tree/linux-rfc-v1/），对 RFC 报告提到的测试结果进行复现和验证。

RFC中测试结果：

```
Why care about this? - Micro benchmark performance
================================================
There is a ~9x or higher performance improvement using User IPI over other IPC
mechanisms for event signaling.

Below is the average normalized latency for a 1M ping-pong IPC notifications
with message size=1.

+------------+-------------------------+
| IPC type   |   Relative Latency      |
|            |(normalized to User IPI) |
+------------+-------------------------+
| User IPI   |                     1.0 |
| Signal     |                    14.8 |
| Eventfd    |                     9.7 |
| Pipe       |                    16.3 |
| Domain     |                    17.3 |
+------------+-------------------------+

Results have been estimated based on tests on internal hardware with Linux
v5.14 + User IPI patches.

Original benchmark: https://github.com/goldsborough/ipc-bench
Updated benchmark: https://github.com/intel/uintr-ipc-bench/tree/linux-rfc-v1

*Performance varies by use, configuration and other factors.
```

RFC里限制传递消息大小为 1，比较 uintr 与传统 IPC 方式的延迟，说明其的有效性。

我们也能运行其框架并复现这一结果，以 uintr 与 pipe 的测试为例：

- 取得了初步的性能测试结果

```
/ # ./uintrfd-uni
[   15.479844] uintr_register_handler called
[   15.481033] recv: register handler task=78 flags 0 handler 401700 ret 0
[   15.481970] uintr_create_fd called
[   15.483236] recv: Alloc vector success uintrfd 3 uvec 0 for task=78
[   15.487026] uintr_register_sender called
[   15.488232] send: register sender task=78 flags 0 ret(uipi_id)=0
--uif not zero, return
--uif not zero, return
[   15.490148] sending self ipi.

============ RESULTS ================
Message size:       1
Message count:      1000
Total duration:     34.149  ms
Average duration:   25.829  us
Minimum duration:   7.424   us
Maximum duration:   7092.480    us
Standard deviation: 305.858 us
Message rate:       29283   msg/s
====================================
[   15.513506] sending self ipi.
[   15.540198] recv: Release uintrfd for r_task 78 uvec 0
```

```
/ # ./pipe

============ RESULTS ================
Message size:       4096
Message count:      1000
Total duration:     338.916 ms
Average duration:   327.621 us
Minimum duration:   151.040 us
Maximum duration:   6475.008    us
Standard deviation: 234.942 us
Message rate:       2950    msg/s
====================================
```

遇到的主要问题和解决方法

最大的问题是受限于疫情，无法获得有 uintr 硬件的物理机。因此我们必须将工作从自行开发功能环境（即 qemu-uintr）开始。

开发过程中，我们对 qemu 也不熟悉，很多地方不知如何动手。除了问及校内各位学长以外，还通过 qemu 官网提供的邮件列表，对社区成员进行了邮件询问，其中比较幸运有一位来着台湾的朋友，回复比较积极，也给我们提供了不少的帮助。

其他各种开发细节问题，可以查看我们仓库 `README/md` 以及 `ppt/` 下对应的文档来了解。遇到的每一个问题，我们都留下了解决过程的文档记录，如 `ppt/调度问题5-28.md` 、 `ppt/debug-log-5-5.md` 等，都是我们开发中遇到的一些困难的思考和解决过程。

分工和协作

王之栋：前期调研，外部沟通，项目主导，qemu调试

项晨东：qemu开发

孙迅：qemu开发调试，性能测试

提交仓库目录和文件描述

我们提交的仓库是项目的主仓库，基本没有项目代码，项目代码通过链接的形式放在主仓库根目录的 `README.md` 中。当前，项目代码主要包括 `qemu-uintr` 。

仓库根目录的文件夹 `ppt/` 存放我们在项目进展过程产出的报告和文档，可配合 `README.md` 中的链接定位。

比赛收获

通过这次比赛，增进了知识，开阔了眼界，也收获了一段探索的经历，希望能进入决赛，继续完成后半段的探索。