

Linux Plumbers Conference 2022

>> Dublin, Ireland / September 12-14, 2022



LoongArch: 我们接下来干啥



- 我们是谁
- 我们做了啥
- 我们接下来干啥
- 问答时间



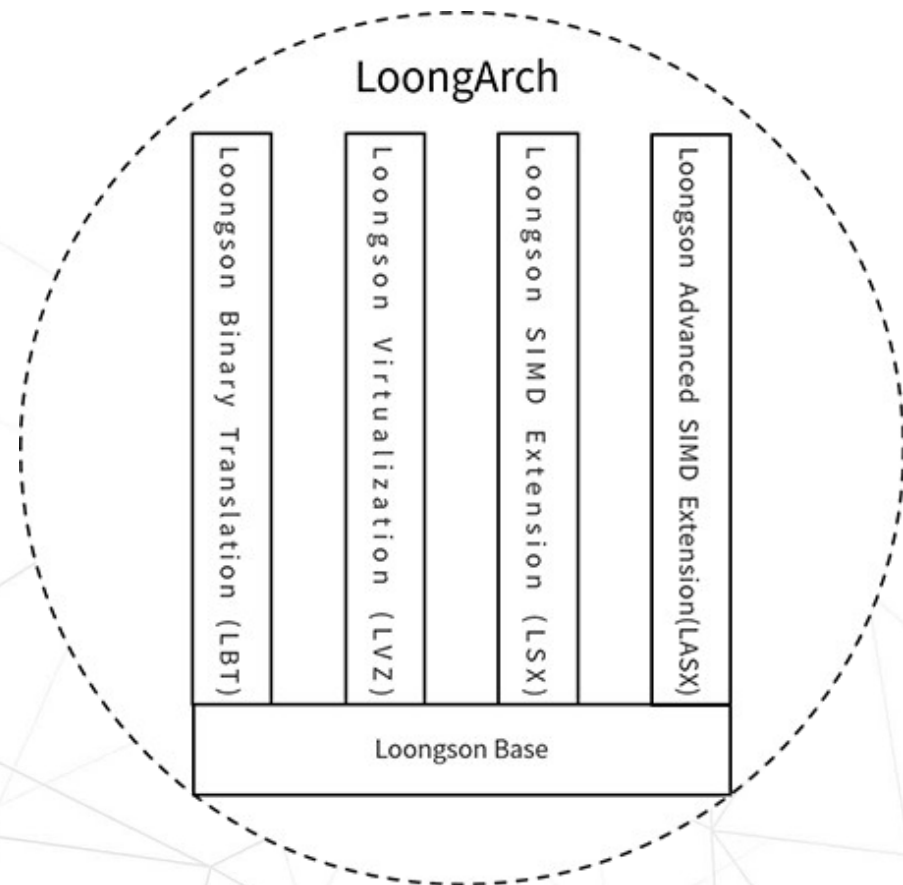
我们是谁

- 陈华才 @chenhuacai
 - arch/loongarch 维护者
- 王雪瑞 @xen0n
 - Gentoo 开发者、 arch/loongarch 代码审查者，在无数项目打杂
 - 很荣幸能作为“爱好者”成功“混”入现场！



LoongArch 是啥？

- “a new RISC ISA, a bit like MIPS or RISC-V”
- 一些数字
 - 3个ISA子集(LA32R、LA32S、LA64)
 - 4个权限级别(PLV0 ~ PLV3)
 - 32个通用寄存器，32个浮点/向量寄存器，8个浮点条件寄存器
- 型号
 - 龙芯3A5000、3C5000(L)、2K1000LA、2K0500等等
- 更多信息
 - 请看[官方文档](#)
 - 也别错过 [@xen0n](#) 的非官方FAQ





Linux
Plumbers
Conference 2022

>> Dublin, Ireland / September 12-14, 2022

我们做了啥

- 上游状态概览
- 上游内核现状



上游状态概览

- 基本支持基本都上游了
 - 完成: binutils, gcc, linux, glibc, go, libffi, libunwind, systemd 等等
 - 移植中 / 等代码审查: LLVM, Rust, musl, libseccomp 等等
- ELF psABI 最近稍微改了一版不太兼容的
- 整体 ABI 稳定了, 已经有很多发行版可用了
 - Gentoo
 - Arch Linux (非官方, 有两个项目在同时推进) 由 @yetist 与 @shipujin 提供
 - Slackware (非官方) 由 @shipujin 提供
 - CLFS (非官方) 由 @sunhaiyong1978 提供
 - 我现在应该就在在一台 LoongArch 笔记本上做这展示!



上游内核现状

- 支持 UEFI+ACPI 系统
- 时间线
 - v5.19: 架构支持、用户态 API
 - v6.0: irqchip、PCI、临时的 ACPI 定义
 - 以及 vDSO getcpu 等等
 - v6.1 基本开箱即用了!
 - 正式的 ACPI 定义、合理的 EFI 引导支持、eBPF JIT、qspinlock、性能监测事件
 - 更多支持：挂起 / 唤醒、LS7A 集成声卡、……



我们接下来干啥

- “旧世界” 问题
- 其他引导协议支持
- ~~EFI 压缩映像引导流程的前进方向~~
 - 搞定了，感谢 @ardb!



“旧世界”问题

- 背景知识：“双界记”
- 不兼容点
 - psABI
 - 固件 & 引导协议
 - Linux 用户态 API
 - 用户态 (libc 符号版本等等)
- 前路何方？

双界记

- 最早的 LoongArch 移植基本是 MIPS 代码的批量复制粘贴、字符串替换
 - 非技术原因，赶工赶出来的
 - 一些名场面比方说 BogoLOONGARCH 和 LBT_LOONGARCH
 - 这样下去肯定不行的……
- 新 ABI 基本照着 RISC-V 搞的
 - ELF psABI 和过程调用约定 * 基本 * 没受影响 (很幸运)
 - 其他部分就没这么幸运了，**每一层**都存在不兼容

不兼容点- psABI

- 重定位记录
 - 旧世界使用基于栈机模型的重定位记录，类似 r178 和 rx 架构的做法
 - 新世界使用经典风格的重定位记录；迁移工作快完成了
- ELF e_flags[7:6]
 - 非常新的新世界工具链产生的目标文件的此位域为 0x1, 旧世界则为 0x0
- 这意味着什么
 - 上游 LLVM/mold 不懂怎么处理栈机重定位记录，也没法让它们懂
 - 若干下游项目要改



重定位记录：前后对比

```
gen2-sysroot/usr/lib64/crt1.o: file format elf64-loongarch

Disassembly of section .text:

0000000000000000 <_start>:
 0: 00150089 move $a5, $a0

0000000000000004 <L0^A>:
 4: 1c000004 pcaddu12i $a0, 0 4: R_LARCH_SOP_PUSH_PCREL _GLOBAL_OFFSET_TABLE_
    4: R_LARCH_SOP_PUSH_GPREL main
    4: R_LARCH_SOP_ADD *ABS*
    4: R_LARCH_SOP_PUSH_PCREL _GLOBAL_OFFSET_TABLE_+0x80000000
    4: R_LARCH_SOP_PUSH_GPREL main
    4: R_LARCH_SOP_ADD *ABS*
    4: R_LARCH_SOP_PUSH_ABSOLUTE *ABS**+0x20
    4: R_LARCH_SOP_SR *ABS*
    4: R_LARCH_SOP_PUSH_ABSOLUTE *ABS**+0x20
    4: R_LARCH_SOP_SL *ABS*
    4: R_LARCH_SOP_SUB *ABS*
    4: R_LARCH_SOP_PUSH_ABSOLUTE *ABS**+0x20
    4: R_LARCH_SOP_SL *ABS*
    4: R_LARCH_SOP_PUSH_ABSOLUTE *ABS**+0x2c
    4: R_LARCH_SOP_SR *ABS*
    4: R_LARCH_SOP_POP_32_S_5_20 *ABS*
 8: 0380000c ori $t0, $zero, 0x0 8: R_LARCH_SOP_PUSH_PCREL _GLOBAL_OFFSET_TABLE_+0x4
    8: R_LARCH_SOP_PUSH_GPREL main
    8: R_LARCH_SOP_ADD *ABS*
    8: R_LARCH_SOP_PUSH_PCREL _GLOBAL_OFFSET_TABLE_+0x80000004
    8: R_LARCH_SOP_PUSH_GPREL main
    8: R_LARCH_SOP_ADD *ABS*
    8: R_LARCH_SOP_PUSH_ABSOLUTE *ABS**+0x20
    8: R_LARCH_SOP_SR *ABS*
    8: R_LARCH_SOP_PUSH_ABSOLUTE *ABS**+0x20
    8: R_LARCH_SOP_SL *ABS*
    8: R_LARCH_SOP_SUB *ABS*
    8: R_LARCH_SOP_PUSH_ABSOLUTE *ABS**+0xffff
    8: R_LARCH_SOP_AND *ABS*
    8: R_LARCH_SOP_POP_32_U_10_12 *ABS*
```

* each of these RELA records
takes up 24 bytes

* 每条 RELA 记录占 24 字节

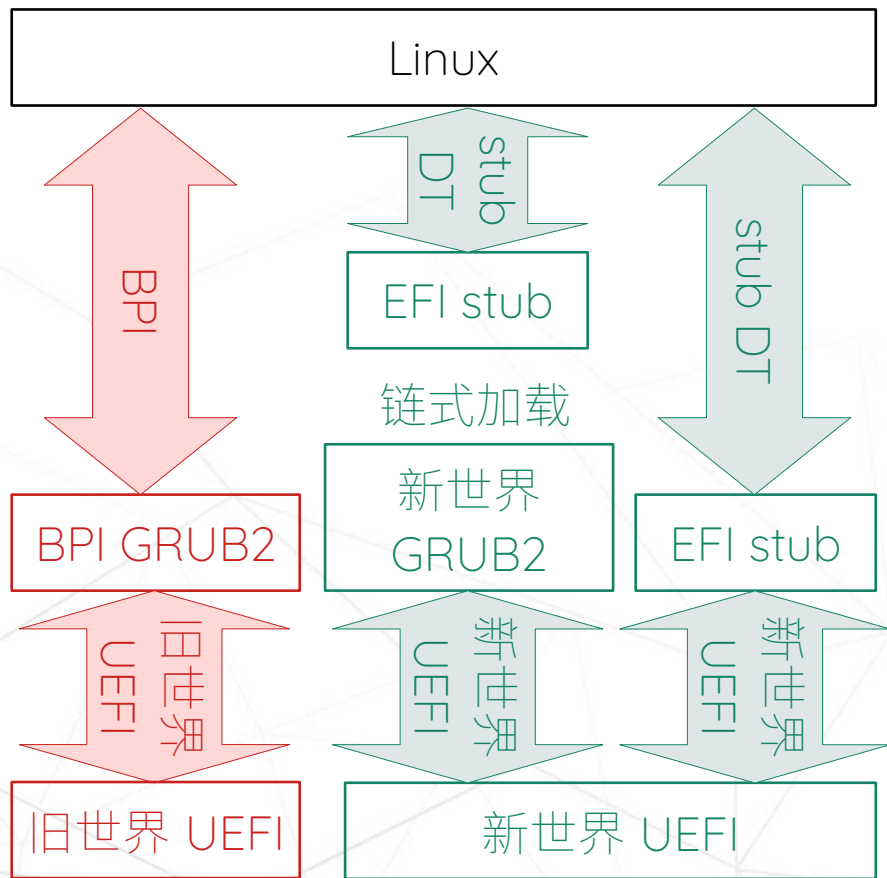
```
/usr/lib64/crt1.o: file format elf64-loongarch

Disassembly of section .text:

0000000000000000 <_start>:
 0: 00150089 move $a5, $a0
 4: 1a000004 pcalau12i $a0, 0 4: R_LARCH_GOT_PC_HI20 main
 8: 02c0000c addi.d $t0, $zero, 0 8: R_LARCH_GOT_PC_LO12 main
 c: 1600000c lu32i.d $t0, 0 c: R_LARCH_GOT64_PC_LO20 main
10: 0300018c lu52i.d $t0, $t0, 0 10: R_LARCH_GOT64_PC_HI12 main
14: 380c3084 ldx.d $a0, $a0, $t0
18: 28c00065 ld.d $a1, $sp, 0
1c: 02c02066 addi.d $a2, $sp, 8(0x8)
20: 00830003 bstrins.d $sp, $zero, 0x3, 0x0
24: 00150007 move $a3, $zero
28: 00150008 move $a4, $zero
2c: 0015006a move $a6, $sp
30: 1a000001 pcalau12i $ra, 0 30: R_LARCH_GOT_PC_HI20 __libc_start_main
34: 02c0000c addi.d $t0, $zero, 0 34: R_LARCH_GOT_PC_LO12 __libc_start_main
38: 1600000c lu32i.d $t0, 0 38: R_LARCH_GOT64_PC_LO20 __libc_start_main
3c: 0300018c lu52i.d $t0, $t0, 0 3c: R_LARCH_GOT64_PC_HI12 __libc_start_main
40: 380c3021 ldx.d $ra, $ra, $t0
44: 4c000021 jirl $ra, $ra, 0
48: 1a000001 pcalau12i $ra, 0 48: R_LARCH_GOT_PC_HI20 abort
4c: 02c0000c addi.d $t0, $zero, 0 4c: R_LARCH_GOT_PC_LO12 abort
50: 1600000c lu32i.d $t0, 0 50: R_LARCH_GOT64_PC_LO20 abort
54: 0300018c lu52i.d $t0, $t0, 0 54: R_LARCH_GOT64_PC_HI12 abort
58: 380c3021 ldx.d $ra, $ra, $t0
5c: 4c000021 jirl $ra, $ra, 0
```

不兼容点 - 固件

- UEFI 表
 - 旧世界表中的指针是虚拟地址
 - 可能的原因: 这样就和 arch/loongarch 预期的 DMW 配置一致
 - 新世界则是物理地址和所有别的架构一样
- ACPI 表
 - 布局不一样, 不兼容
- 引导协议
 - `struct bootparamsinterface` (“BPI”)
 - 通过特殊的 GRUB 转换, 用来配合旧世界和早期新世界的内核
 - 新世界采用 EFI stub





不兼容点 - Linux 用户态 API

- `_NSIG`
 - 旧世界为 128 (跟 MIPS 一样), 新世界为 64
- 系统调用
 - `{get,set}rlimit` → `prlimit64`
 - `fstat, newfstatat` → `statx`
- `ptrace, sigcontext` 的差异



不兼容点 - 用户态

- libc 符号版本
 - 旧世界取 `GLIBC_2.27` (你一定猜到原因了)
 - 新世界取 `GLIBC_2.36`
- ld.so 路径
 - 旧世界取 `/lib64/ld.so.1` (同上)
 - 新世界取 `/lib64/ld-linux-loongarch-lp64d.so.1`

怎么统一两个世界？

- 目标：数字遗产保护 (Digital preservation), 一种可能的思路是允许在**新世界内核**上运行**旧世界二进制**
 - 我们真要这么干吗？
 - 如果真这么干，应该分层解决这件事情
 - 否则为了人们的精神健康，应该采取类似 WINE 的实现思路
- 固件 ↔ 内核：在**旧世界**或者**新世界**固件上跑**新世界内核**
 - 意味着在上游支持 BPI
 - 有些早期 3A5000 系统可能永远不会得到固件升级，我们还管它们吗？
- 内核、用户态 ABI
 - 为了精神健康，很可能需要单独的 chroot/sysroot，但用户体验可能受损
 - 剩下的工作，交给用户态兼容层还是内核处理？



要做的话怎么搞？

- 分界线画在哪
 - 系统调用边界，还是内核内的机制？
 - 我们怎么知道一个进程在使用旧世界 ABI？
通过检查 `e_flags`，还是看第一次 `sigprocmask` 调用时隐含的 `_NSIG`？
 - 怎么解决 `exec` 异世界程序的情况？
- 入口点
 - 通过 `binfmt_misc`：怎么识别旧世界二进制？
 - 以 `ld.so` 替代品的形式
 - `libc` 符号版本处理的 hacks - 估计没法推上游
 - 静态链接的二进制咋办？
- 兼容层
 - ABI 风味标记： `ptrace` 还是 `personality`？



其他引导协议支持

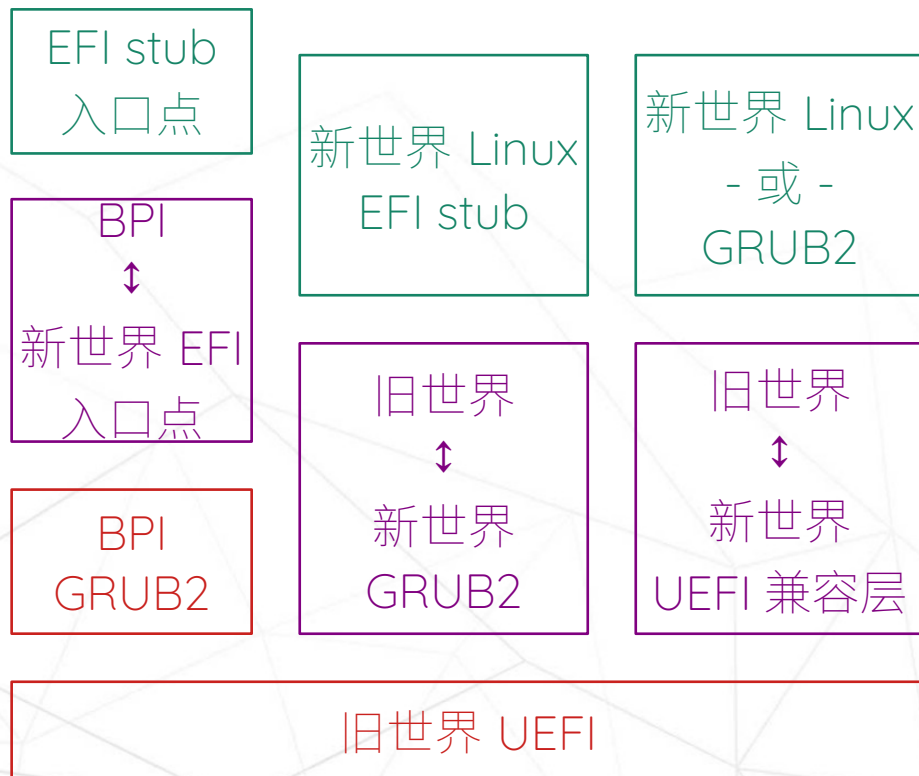
- 为何支持其他引导协议很重要
- 可能性： BPI 兼容性
- 可能性： 设备树引导

为何支持其他引导协议很重要？

- 旧世界 /BPI 兼容性
 - 一些古早硬件（尤指笔记本）可能永远不会有新世界固件
 - 用户不喜欢（半）计划报废
- 资源受限场景
 - 在完整 UEFI 太重的场合，用设备树引导
 - 我们想在这些设备上跑上游 Linux 吗？
- 自由软件固件 (coreboot 等等)
 - 项目、用户可能不想，或者不能支持 UEFI
 - 给人们以选择

可能性： BPI 兼容性

- BPI 引导过程长啥样？
 - 有 UEFI，但存放位置不同，表里的指针也是虚拟地址
 - 内存映射表的形状不同
- 兼容
 - 相同的问题：在哪一层做？
 - 如果在内核前，则链式引导未经修改的新世界内核
 - 如果在内核里，则是类似 EFI stub 的又一个入口点





可能性：设备树引导

- 估计能做，阻力不大（不像 BPI）
- DT 标准化
 - 据我所知两个龙芯演讲者都没在做设备树内核相关
 - 对在这个项目的同学们要说的话：
沟通，沟通，沟通！



致谢

- 感谢我的公司和龙芯
- 社区力量！
 - Gentoo 的 dilfridge 和 sam
 - @FlyGoat, @HougeLangley, @phorcys, @prcups, @Rabenda, @xry111 和 Telegram 龙芯用户群的群友们
 - 不计其数的其他同学们

Linux Plumbers Conference 2022

>> Dublin, Ireland / September 12-14, 2022



谢谢！

问答时间