目录

1 移植问题

1.1 辅助

1.1.1 backtrace 调用栈无法使用的问题

现象

1. gdb 下调用栈最多显示两层
2. gdb 下 next 总是无法直接进入下一个表达式而是和 step 类似, 进入函数调用而非继续下一步

解决

1. 问题来自 LA 的调用栈格式和保存方式在 LLVM11 不被支持, 新工具链可能已经解决
2. 需要在 linker.ld 中加入 eh_frame 和 eh_frame_hdr(只加入任意一个都不行)
3. 之所以该部分是必要的，是因为 LoongArch 下的 GDB 调用栈格式依赖这两个存储，且不保证一定存 PC，SP 在栈上，所以 GDB 启发式搜索会失败。

1.1.2 uImage 的 BASE ADDR

加载和执行地址都是 0x0, 不采用 0x3 开头的

1.1.3 不重编译

1. Modification on compilation will NEVER trigger a rerun of compilation
2. 有时候虚拟机执行过慢会导致 expect 漏掉某些字符, 这时候建议清空缓存重新执行

1.1.4 base_address 无法设为 0

提示 PHDR 未配置, 最后加上链接选项-static

1.1.5 暂时没有任何的 NAND 驱动，改用别的方法测试 FAT32 文件系统

1. 没有使用 virtio, 一方面是因为这个虚拟机不支持 Virtio（好像开源上游的 qemu 在 LA target 下也不支持），另一方面之前的实验显示 VirtIO 在 512 下是失败的。
2. 改用下列命令:

```
SERIAL=2 DEBUG_GMAC_PHYAD=0 DEBUG_MYNAND=cs=0,id=0x2cda DEBUG_MYSPIFLASH=gd25q128 "$Q
        -M ls2k500 -drive if=pflash,file=$BIOS \
        -device loader,file="$FAT",addr=0x8000000 \
        -m 2048M \
        -D /tmp/qemu.log \
        -serial stdio \
        -drive if=mtd,file="$OS" \
        -net nic -net user,net=192.168.1.2/24,tftp=$SCRIPTPATH \
        -net nic -net user,net=10.0.3.0/24 -vnc 0.0.0.0:0 \
        -drive if=none,file="$DISK",id=udisk \
        -device usb-storage,drive=udisk -s $@\
```

将内存改为 2048M, 且 addr=0x8000000 保存镜像

1.1.6 bash 调用 sbrk

现象 调用 brk 多次, 无法修改分配到足够的页用于用户栈的拓展, assert 失败退出

解决 在 musl 代码中发现其实际上 14 年移除了对 sbrk 的支持, 只保留查询功能, 其他是伪实现 (return ENOMEM) 最后重新编译 bash, 除了开启 static 外, 还指定关闭 bash_malloc(见 lightlearning.org)

1.1.7 UART 换行错误问题

现象 板子上换行失效, 每行偏移一大段且最终超出列数而无法显示

原理 首先, 这个问题和龙芯关系不大, 如果斑竹认为不合适建议直接删帖, 实在抱歉太长不看版: minicom 在 Linux 下为了实现对 CRLF 的兼容, 保留其原来的语义, 使得如果要在其中换行, 需要 CRLF 一起发, 否则就要换 miniter.py

1.1.8 Rustc Cross Compilation

Note that the "target.x86_64-unknown-linux-gnu" is essentially the "HOST",

instead of the "TARGET". To cross compile the code, write the llvm-config to the formerly mentioned item.

1.2 内存

1.2.1 某些时候

现象 mmap 一个 area 后没有 frame_allocate 就直接 SPF, 地址错误然后 assertion failed 失败退出

解决 正常时候应当触发 fault, 但没有, 可能是 TLB 有错

发现是因为 LA 下 csrwr 是 swap 语义, 如果是需要真正写入相同内容, 需要在下一次执行之前重写一次

1.2.2 在加载重映射时出现冲突

原理 虽然头部不相同，但是其重填/加载的时候是用的旧的页表，所以我们至少

解决 改用 0 段作为直接映射窗口，1 扩展段作为内核页表。

1.2.3 无 sv39 导致的问题

fs/cache 问题 (insert_program_area 初次崩溃)

原因 发现是将 0x9 地址映射到页表, 导致直接翻译给窗口, 最终失败

处理 将 MMAP_BASE 改为 0x0 地址初始

后续 发现如果改为 0x0, 可能某些地方会出错导致其被视为 NULL, 遂改为 0x1000_0000,

某些地址 map 出现 already mapped 原理: 地址空间过小 (1GB), 两个跳板地址和页表冲突 (后缀相同)

可以用 show_areas 尝试打印信息

发现最后一页似乎会被映射两次，进而 assertion fail.

过程

1. 注意到，如果设置断点为 laflex.rs:295, 则在第一次 c 后，c 1772 为崩溃临界点
2. Overlapped at the very beginning.

```
[90mkernel: [frame_alloc] FrameTracker:PPN=0x9000000000370
[0m[34mkernel: root ppn:FrameTracker:PPN=0x9000000000370
[0m[34mkernel: [find_pte_create] vpn: VPN:0x90000000007ff, pte:0x900000000037001
[0m[90mkernel: [frame_alloc] FrameTracker:PPN=0x9000000000371
[0m[34mkernel: [find_pte_create] vpn: VPN:0x90000000007ff, pte:0x371ff8, i:1
[0m[34mkernel: [laflex::map] vpn: VPN:0x90000000007ff
[0m.text [0x9000000000000000, 0x90000000000f3000)
.rodata [0x90000000000f3000, 0x9000000000113000)
.data [0x9000000000113000, 0x900000000011f000)
.bss [0x900000000011f000, 0x9000000000370000)
```

结果 Finally, realized that is was the map_trampoline and it resulted from the wrong config constant change.

```
pub const TRAMPOLINE: usize = MEMORY_END - PAGE_SIZE + 1;
```

1.2.4 UART 不输出

虚拟机上 地址写错，注意 2k500 的 UART 地址和 3A5000 不同，低 4 位非 0!!!

板子上 发现板子必须访问 0x8 段内存 (MAT: SUC), 否则缓存会让 UART 的 MMIO 失效

1.2.5 在内核态出现直接映射地址被 TLBRFill

原因 从 map, find_pte_create 中产生的 deref 得到的是没有 0x9 高位的，所以解引用会出错

Workaround 考虑到每次读写内存都进行一次 or 会增大开销，改用在 DMW0.vesg:=0, 增加 0 位访问从而减小问题。

1.2.6 需要去掉的直接映射检查

1. remap test

1.2.7 直接映射 new_kernel 阶段大量出现 vec 申请行为，直到掏空所有 frame_allocator

第一次发现

```
*pte = LAFlexPageTableEntry::new(frame.ppn, LAPTEFlagBits::V & LAPTEFlagBits::P);
```

中的"&" 导致其 flags 为 0, 实际上改为"|" 即可

第二次发现

1. 实际涉及的部位只有 MapArea::map_one_unchecked(MapArea::map 通过这个接口间接访问), 对直接映射改成空实现
2. 修改后, 仍有大量的 pte 访问, 发现是 is_mapped 进行的, 遂对直接映射特殊判断, 不进行访问直接已经映射

1.2.8 随机地址错误与大量 reset

错误解决

现象 曾经出现过各种奇怪的访存错误，且位置随机

意外 (思路?)

现象 发现如果只是删除清零部分, 仍然可用最后发现只是因为 set_timer_interrupt 的某一行改了导致可以稳定运行, 实际上还是内存错误 (伪解决) 但为后来提供了一个稳定的平台: 之前是随机错误, 现在是稳定 panic, 也不错 (出现过两种稳定的 panic, ADE 和现在的地址错误导致的 panic!)

稳定错误条件下的 dbg 这时候没有竞争, gdb 下可以直接用某个常见函数 (如 trap_handler) 的次数定位某些错误, 例如我就使用了 break trap_handler 然后 continue 572

解决 注意到, 当错误稳定后 (发现错误/reset 是集中在 timer enabled 前后),

1. 用 occur(见 lightlearning) 数标志函数 (任选, 这里选 trap_handler) 出现次数, 为 572 次, 自定义命令次数-1, 重复运行寻找规律, 一般分为启动加载命令 (处于 gdb_script) 和热更新命令 (处于 new_cmd 中) 同时, 自动化命令一键抵达断点
2. 传统技能 asm-reg-mode 跟踪调试数据流

3. 观察寄存器转移, 发现问题来自 fp, 且是在内核 switch 之间寄存器变化 **这里有个败笔**, 其实不需要直接进行寄存器大规模转移检查, 直接检查寄存器取值表上循环之间的不正常值 (或者说初始值不同者即可定位)

4. 修改命令, 改为在 switch 附近继续 dbg

5. 观察汇编发现是 fp 存储的 ld.d $fp, $a1, 88 中 $a1

1.2.9 0 页赋值 (zero)

现象 给 0 地址后的 4096 字节赋值但失败

处理 从 0 开始的 8 字节, 手动 for 循环赋值, 其他使用 copy_nonoverlapping 即可。

1.2.10 LDDIR, LDPTE

现象 调用该指令却发生 INE

实质 实验发现, 2k500 虚拟机不支持 4 级页表, 且 lddir 最高级不能为 2

1.3 文件系统

1.3.1 bash 出现 illegal instruction

排除内存后, 发现 illegal instruction 从 0x800 开始, 注意到这是 BLOCK_SZ, 所以认为是 BLOCK_SZ 设置和读取的问题最后在 cache 中找到硬编码的 cache 数量, 改为从 BLOCK_SZ 计算

```
const PRIORITY_UPPERBOUND: usize = 1;
const BUFFER_SIZE: usize = BLOCK_SZ;
const PAGE_BUFFERS: usize = PAGE_SIZE / BUFFER_SIZE;
const CACHEPOOLSIZE: usize = 16 >> (BLOCK_SZ / 512).trailing_zeros();
const CACHEPOOLPAGE: usize = if (CACHEPOOLSIZE >> 3) > 1 {
```

```
    CACHEPOOLSIZE >> 3
} else {
    1
};
```

1.4 手册不符

1.4.1 musl 下 sigaction 结构体不一致

解决 对比 log, 注意到 LoongArch 下的 sigaction 地址实际上是不同于 RISC-V 的数据结构, 参考正确结构后修改为:

```
#[cfg(feature = "la64")]
#[derive(Clone, Copy)]
#[repr(C)]
pub struct SigAction {
    pub handler: SigHandler,
    pub flags: SigActionFlags,
    pub mask: Signals,
    pub restorer: usize,
}
```

1.4.2 TLBRFill 失败

原因

1. 虚拟机上不会在异常后将 MMU 模式转为 DA, 需要手动转 0x0 为 0xa8 如果不开, 可能会在物理地址上 (被认为是 PG 的"0 段") 重复进入 TLBRFill

2. 似乎 2k500 最高目录层为 dir3, dir4 不可用, 且只接受 dir3 从 pgd 读入, 其他层会 INE(Instruction Not defined Exception)

1.4.3 重填出现地址失败

最后发现是虚拟机上重填并不自动开启 DA 模式，需要手动设置，后面改为 0-H 映射后问题不再存在

1.4.4 refill 失败

现象 发现所有的 refill 在 ldpte 后出现错误

解决 发现 (包括来自北理工老哥的代码), 所有龙芯的非叶子节点页表都应当清 0, 否则就是纯粹的地址, 不应当有 V 位, 否则就会落空导致读取偏移