龙上之龙

龙架构新旧世界兼容层 libLoL 的实现原理

Miao Wang

缘起

梦开始的地方

白铭骢 in Loongson

3A6000 正式版主板 (XA61200) 订购信息

近日龙芯武汉通知我可以订购 3A6000 正式版主板 (XA61200) 且可以分享此消息，所以在这里扩散一下：

– 板型为 DTX（可理解为窄版 mATX，203mm × 244mm）
– 桥片依然为 7A2000
– 扩展槽有：2 * DDR4 UDIMM, 1 * PCIe 3.0 x16 (x8), 1* PCIe 3.0 x8 (x8), 1* PCIe 3.0 x4 (x4), 1 * m.2 E.Key (Wi-Fi), 1* mPCIe (PCIe 2.0 x1), 1 * NVMe (PCIe 3.0 x4)
– 主板自带处理器及桥片散热

报价为 1499 元人民币（可同时订购配套内存），从龙芯直接发货并提供质保和技术支持（包括固件更新）；有意者请加微信 18929770995 联系销售

t.me/loongson_users/72574

Oct 23, 2023 at 12:39

错过了 RISC-V，这次总得给龙做点啥

“听说龙芯有个新旧世界的事情，
要不就搞它了！”

啥是新旧世界

“旧世界是指最早在龙芯中科内部适配的、随着 LoongArch 公开一并发布的那个 LoongArch 软件生态。 新世界是指龙芯中科与社区同仁一道，以典型开源社区协作模式打造的，完全开源的 LoongArch 软件生态。”

——xen0n 等.旧世界与新世界[R/OL].https://areweloongyet.com/docs/old-and-new-worlds/

- 龙芯中科对 LoongArch 采取了秘密开发、突然全盘推出的商业策略
  - 商业软件跟进迁移适配
- 由于未能预见到这一版工作有些地方不得不做不兼容修改……
  - （社区：你们整这 BogoLOONGARCH，搁这扯犊子呐？）
  - 所以修改后整出来的这个就是新世界
- ……而使客户和自身不得不面对的无奈后果
  - 商业软件：我是谁？我在哪？
- 按照目前的趋势和一些公开消息，未来旧世界将逐渐消亡

新旧世界有啥不一样

听说各种东西都不一样

遇事不决，先搞内核
——毕竟我们能 chroot

内核有啥不一样

内核有啥不一样

其实也没啥不一样

arch/loongarch/kernel/scall64-64.S:85

```
syscall_common:
        /* Check to make sure we don't jump to a bogus syscall number. */
        li.w t0, __NR_syscalls
        bgeu a7, t0, illegal_syscall

        /* Syscall number held in a7 */
        slli.d  t0, a7, 3               # offset into table
        la      t2, sys_call_table
        add.d   t0, t2, t0
        ld.d    t2, t0, 0               #syscall routine
        beqz    t2, illegal_syscall

        jalr    t2                      # Do The Real Thing (TM)

        ld.d    t1, sp, PT_R11          # syscall number
        addi.d  t1, t1, 1               # +1 for handle_signal
        st.d    t1, sp, PT_R0           # save it for syscall restarting
        st.d    v0, sp, PT_R4           # result
```

```
arch/loongarch/kernel/syscall.c:60:
void noinstr do_syscall(struct pt_regs *re
{
        unsigned long nr;
        sys_call_fn syscall_fn;

        nr = regs->regs[11];
        /* Set for syscall restarting */
        if (nr < NR_syscalls)
                regs->regs[0] = nr + 1;

        regs->csr_era += 4;
        regs->orig_a0 = regs->regs[4];
        regs->regs[4] = -ENOSYS;

        nr = syscall_enter_from_user_mode(regs, nr);

        if (nr < NR_syscalls) {
                syscall_fn = sys_call_table[nr];
                regs->regs[4] = syscall_fn(regs->orig_a0, regs->regs[5], regs->regs[6],
                                           regs->regs[7], regs->regs[8], regs->regs[9]);
        }

        syscall_exit_to_user_mode(regs);
}
```

Table 1. 通用寄存器使用约定

名称	别名	用途	在调用中是否保留
$r0	$zero	常数 0	（常数）
$r1	$ra	返回地址	否
$r2	$tp	线程指针	（不可分配）
$r3	$sp	栈指针	是
$r4 - $r5	$a0 - $a1	传参寄存器、返回值寄存器	否
$r6 - $r11	$a2 - $a7	传参寄存器	否
$r12 - $r20	$t0 - $t8	临时寄存器	否
$r21		保留	（不可分配）
$r22	$fp / $s9	栈帧指针 / 静态寄存器	是
$r23 - $r31	$s0 - $s8	静态寄存器	是

系统调用的传参方式是一致的

- 系统调用号：a7
- 系统调用参数：a0 ~ a5
- 返回值：a0

具体系统调用的区别

- 新世界缺了：
  - `newfstatat`
  - `fstat`
  - `getrlimit`
  - `setrlimit`
- 所以需要补到系统调用表里

如何用内核模块修改系统调用表

修改系统调用表只需三步！

- 找到系统调用表的地址
- 找到对应的实现函数的地址
- 把实现函数地址写到系统调用表中

如何用内核模块修改系统调用表

找实现函数

- 内核模块仅能连接内核中的导出（EXPORT）符号
- 可以用 kallsyms_lookup_name 通过符号名查找地址
- kallsyms_lookup_name 也没有导出
- 让用户帮忙找到后传给内核模块

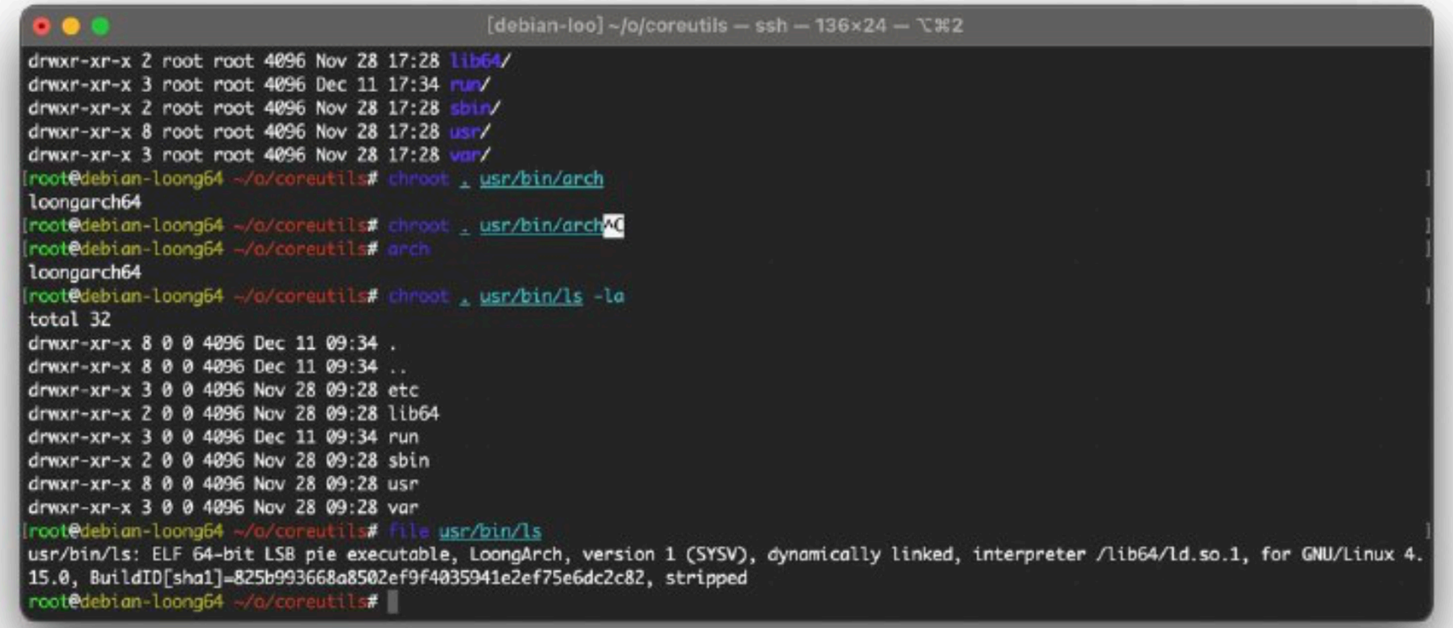
如何用内核模块修改系统调用表

找系统调用表

- kallsyms_lookup_name 不能用来找数据段的符号
- 观察内核导出的数据段符号，data 开头有一个 jiffies，bss 开头有一个 reboot_mode
- 系统调用表的地址一定在二者之间
- 还能找到已知的几个系统调用号的函数的地址
- 搜就行了！

让我们来试试看

Miao Wang in Loongson



t.me/loongson_users/88910

Dec 11, 2023 at 17:44

还有 NSIG

- NSIG: 指内核中允许的最多信号数目
  - 旧世界：128；MIPS：128；新世界：64；多数架构：64
- 会使 `sigset_t` 的大小发生变化
- 传入 `sigset_t` 的所有系统调用，都要显式传入 NSIG，以确保内核和用户态的数据结构一致
- 覆盖掉！
  - 只读的：允许传入 NSIG 为 128，使用时仅读取其前 64 位
  - 写入的：只写入其前 64 位，后续的清零

Miao Wang in Loongson

> 白铭骢
> 那作为下游功能好了（

我在写一些邪恶的代码

t.me/loongson_users/88935 Dec 11, 2023 at 18:37

白铭骢 in Loongson

> Miao Wang
> 我在写一些邪恶的代码

Uh oh，听起来是我喜欢的东西（x

t.me/loongson_users/88937 Dec 11, 2023 at 18:37

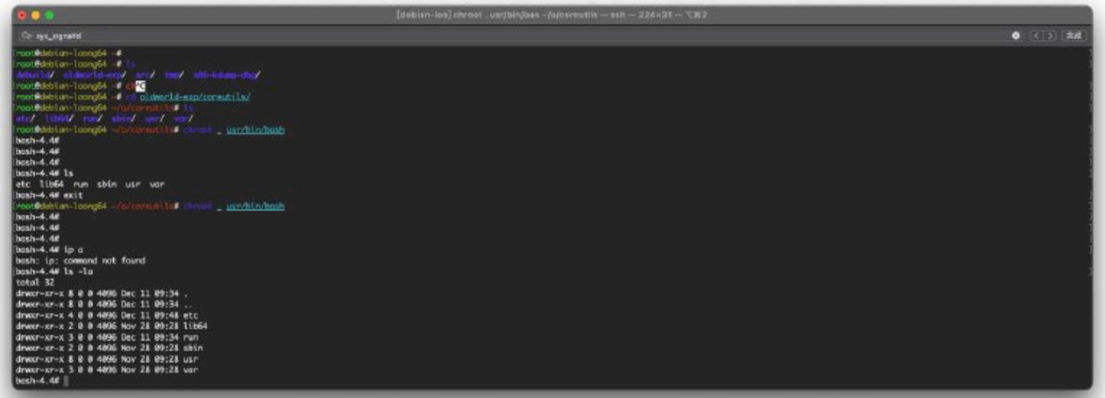
Miao Wang in Loongson

试图用个 kernel module 把缺的 syscall 补上

t.me/loongson_users/88938 Dec 11, 2023 at 18:37

Miao Wang in Loongson

> 白铭骢
> Uh oh，听起来是我喜欢的东西（x



t.me/loongson_users/89137 Dec 11, 2023 at 21:15

Miao Wang in Loongson

bash 也可用了

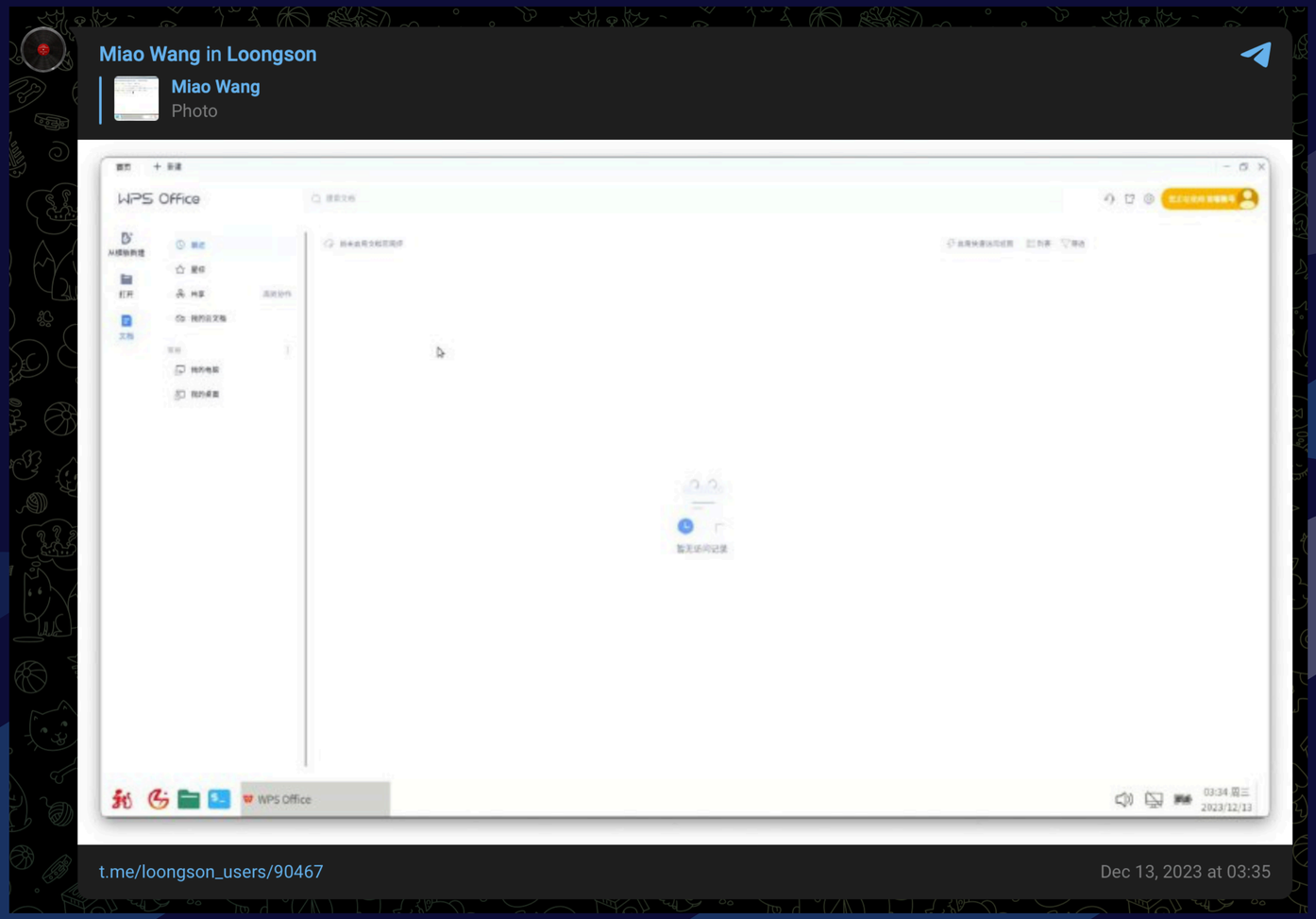
t.me/loongson_users/89138 Dec 11, 2023 at 21:15

后续

- DKMS 打包
- 使用 kprobe 等多种方法查找地址
  - 感谢刘子兴

效果


Miao Wang in Loongson
loongson@loongson-pc: ~/Desktop
文件(F) 编辑(E) 视图(V) 搜索(S) 终端(T) 帮助(H)
loongson@loongson-pc:~/Desktop$ uname -a
Linux loongson-pc 6.1.0-13-loong64 #1 SMP Debian 6.1.55-1 (2023-09-29)
loongarch64 loongarch64 loongarch64 GNU/Linux
loongson@loongson-pc:~/Desktop$
loongson@loongson-pc:···
02:43 周三
2023/12/13
t.me/loongson_users/90438
Dec 13, 2023 at 02:43


Miao Wang in Loongson
Miao Wang
Photo
WPS Office
WPS Office
03:34 周三
2023/12/13
t.me/loongson_users/90467
Dec 13, 2023 at 03:35

所以……用户该如何使用?

用户态有啥不一样

- 动态连接程序：需要 dynamic interpreter
- 新旧世界的 dynamic interpreter 不一致
  - 新世界：`/lib64/ld-linux-loongarch-lp64d.so.1`
  - 旧世界：`/lib64/ld.so.1`
  - MIPS64el：`/lib64/ld.so.1`

齿轮继续转动

13 December 2023

白 **白铭骢** 03:05

不过倒是，我在想就算 WPS 能装上，得怎么利用那个运行时（

除非说 ld-linux 名字不一样? 03:05

MW **Miao Wang** 03:11

是的，不一样

白 **白铭骢** 03:11

噢，那反而好办

白 **白铭骢** 03:14

In reply to this message

嗯，不过只要 ld-linux 能区分，就可以在 /opt 里放一个运行时环境了

我们现在 x86_64 上的 optenv32 就是这么做的 03:14

MW **Miao Wang** 03:14

唔

那就是你自己提供一个旧世界的 ld-linux 03:14

然后后者的 search 地方不一样? 03:14

白 **白铭骢** 03:14

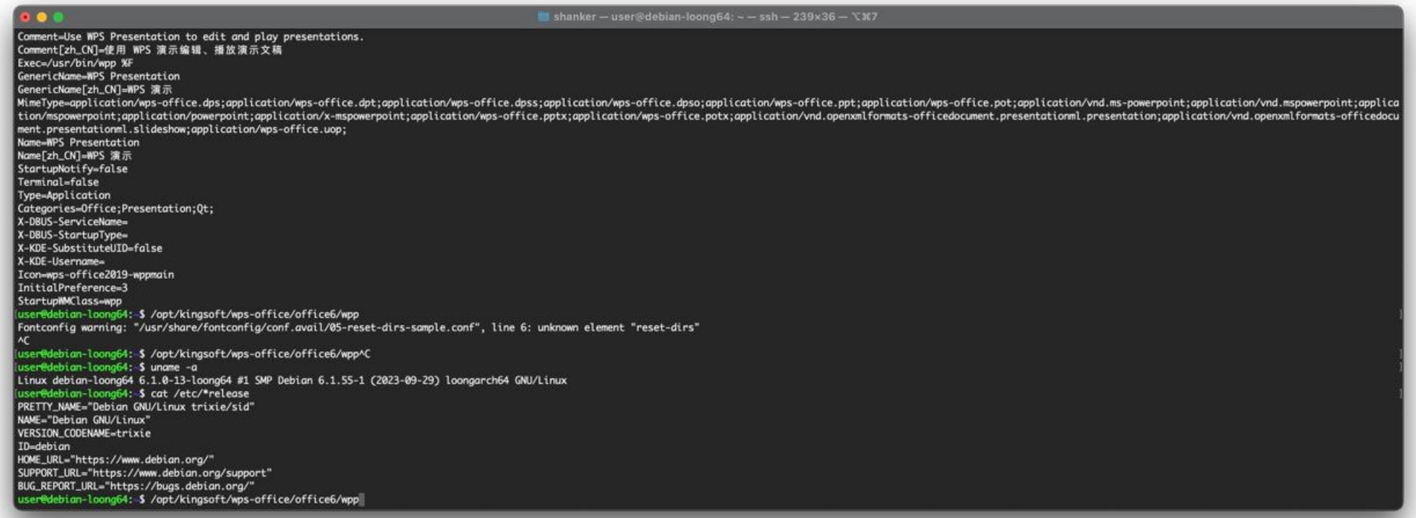
对

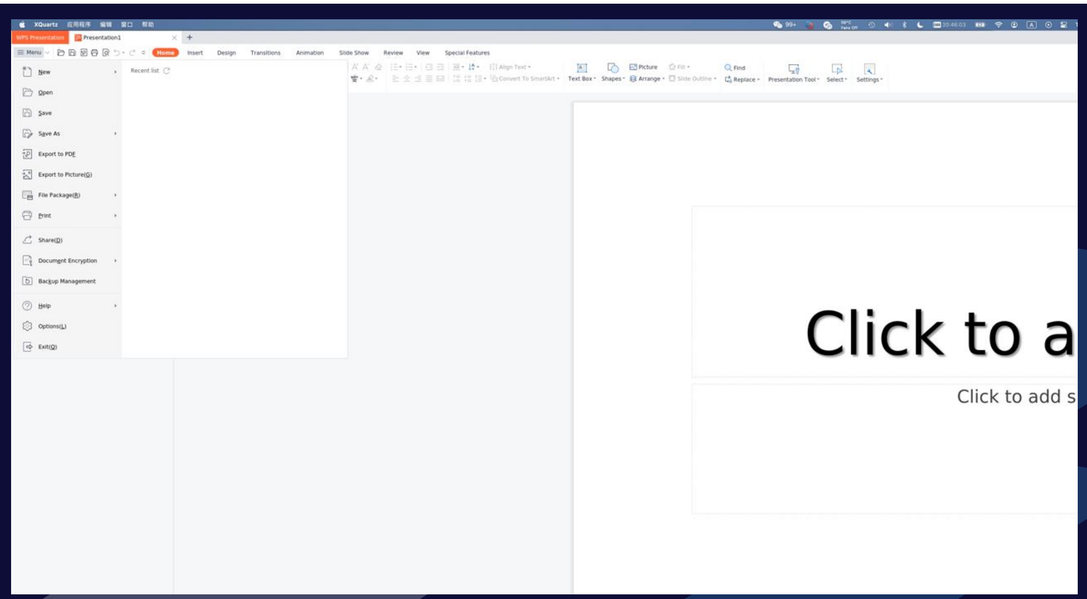
MW **Miao Wang** 03:14

好

- 重新编译一份旧世界的 glibc
- 重新设定其默认的共享库搜索路径，使其不搜索系统默认的存放路径
- 将旧世界程序工作需要的库放在那个位置
- 新旧世界井水不犯河水

说干就干





15 December 2023

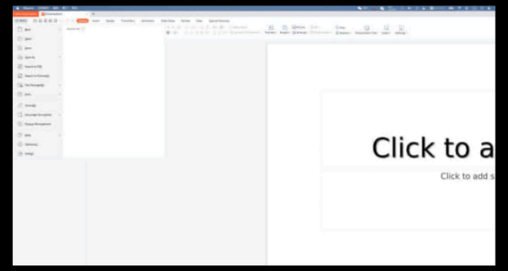
Miao Wang 06:10

我来整个活 06:11

白铭骢 06:11

嗯？咋整的，还是那个模块？

Miao Wang 06:15



In reply to this message 06:15

对

我重新掏出来 loongix 的源代码，在新世界重新编译了一份旧世界的 libc 06:15

剩下的 so 就直接原样偷 06:15

白铭骢 06:15

然后改了 LD PATH?

Miao Wang 06:16

对，重新编译的 glibc 的搜索路径和系统里的不一样，所以不会误载入新世界的 so

疯狂的打包之旅

启动！

- 编译旧世界 binutils
- 编译旧世界 gcc-8 至 xgcc
- 编译新世界 glibc 2.37 至 ldconfig
- 编译 make-4.3
- 偷 loongnix linux headers 和 libgcc 包进编译环境
- 编译旧世界 glibc，并安装 ld.so.1 和 libc.so.6
- 偷 loongnix `libblkid1`、`libbsd0`、`libexpat1`、`libffi7`、`libfontconfig1`、`libfreetype6`、`libgcc1`、`libgl1`、`libglib2.0-0`、`libglvnd0`、`libglx0`、`libice6`、`liblzma5`、`libmount1`、`libpcre3`、`libpng16-16`、`libselinux1`、`libsm6`、`libsqlite3-0`、`libuuid1`、`libx11-6`、`libx11-xcb1`、`libxau6`、`libxcb-xkb1`、`libxcb1`、`libxcomposite1`、`libxdmcp6`、`libxext6`、`libxi6`、`libxkbcommon-x11-0`、`libxkbcommon0`、`libxml2`、`libxrender1`、`libxtst6`、`zlib1g`、`libtinfo6`、`libc6` 包并安装
- 重新生成各种缓存

20 December 2023

MW **Miao Wang** 05:59

喵

你闲着吗 05:59

我的旧世界兼容的用户态的 lib （基本上）搞好了 05:59

你要不要康康 05:59

MW **Miao Wang** 06:17

https://github.com/shankerwangmiao/aosc-os-abbs/tree/feat-liblol/app-emulation/liblol

我放在这里了 06:17

白 **白铭骢** 06:35

牛逼，我晚上看看

MW **Miao Wang** 06:38

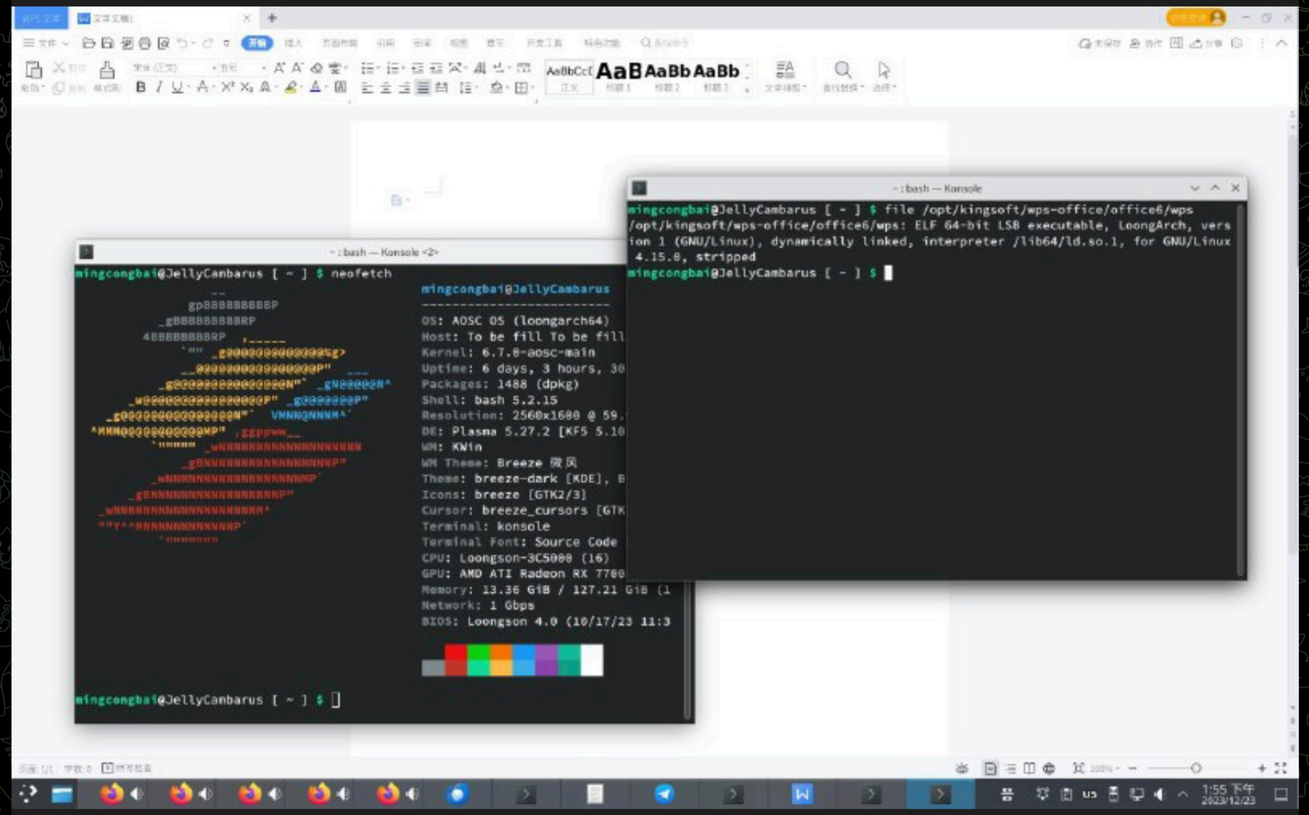
这个我测试是可以原样启动 wps 的

06:39

liblol_0.0.1-0_loongarch64.deb

6.0 MB

白铭骢 in Loongson
Forwarded from 白铭骢



t.me/loongson_users/92019 Dec 24, 2023 at 05:58

白铭骢 in Loongson

liblol 阶段性成果

t.me/loongson_users/92020 Dec 24, 2023 at 05:58

疯狂的打包之旅

再加！

- 选择 WPS 作为第一个样例是偶然的
  - 也是幸运的，因为 WPS 的依赖少
- 发现问题：
  - lbrowser 无法启动
  - 基于 Electron (Chromium) 打包的程序无法启动
- 共同点：都是浏览器

疯狂的打包之旅

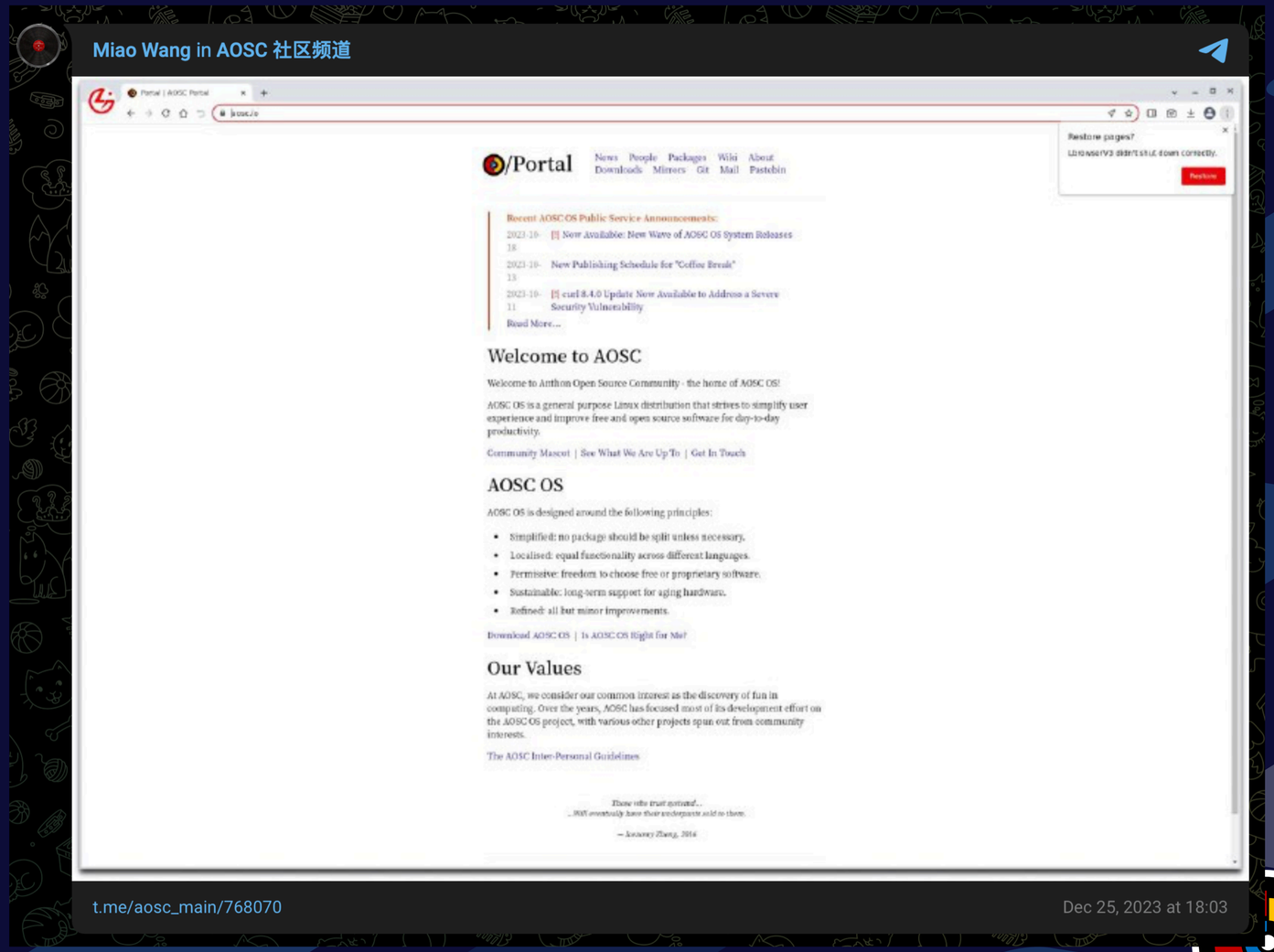
浏览器有啥问题

- 缺库
  - 补上！
- sandbox 不正常
  - 先绕过

疯狂的打包之旅

浏览器有啥问题

- 缺库
  - 补上！
- sandbox 不正常
  - 先绕过

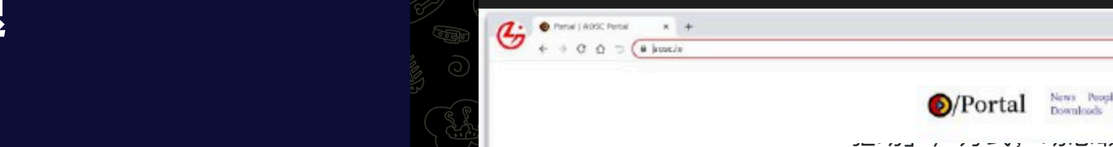


疯狂的打包之旅

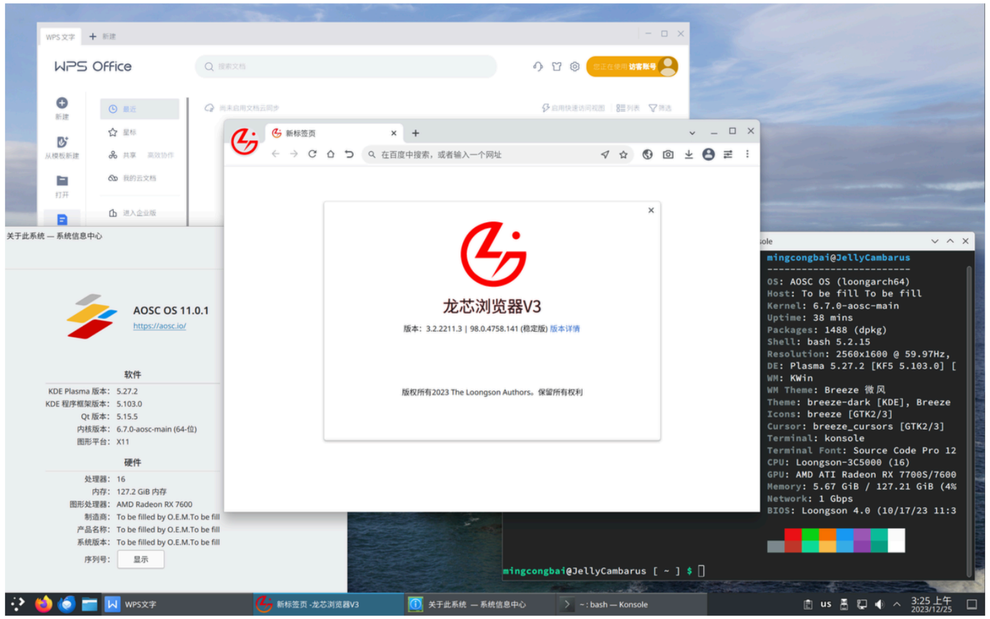
浏览器有啥问题

- 缺库
  - 补上！
- sandbox 不正常
  - 先绕过

Miao Wang in AOSC 社区频道



户态的运行时部分：打包了旧世界参考发行版 Loongnix 的核心运行时库， 使旧世界应用能够有「回家的感觉」从而正常启动。 目前 AOSC OS 已经能够运行 WPS Office 与龙芯浏览器等标志性的旧世界应用， 如图所示（点击即看高清大图）：



后续，在完成配套内核模块的载入向导后，使用龙架构设备的 AOSC OS 用户便可按需启用此兼容方案了。AOSC 同仁们也欢迎其他新世界发行版维护者考察 `libLoL`，并视自身情况和需求集成此项目的成果。

t.me/aosc_main/768070

疯狂的打包之旅

浏览器有啥问题

- Chromium 的 sandbox
  - 原理：使用 seccomp 拦截“不安全”的系统调用
    - 拦下来之后：发出 sigsys 信号
      - 在信号处理函数中获取系统调用的具体信息，从而或是拒绝或者重写
- 问题：如何在信号处理函数中获得系统调用的具体信息?

疯狂的打包之旅

浏览器有啥问题

- 信号处理函数的第三个参数：
  - 传入了发生信号时的用户态现场
  - 当信号处理函数结束时，内核负责恢复该现场

The siginfo_t argument to a SA_SIGINFO handler

When the **SA_SIGINFO** flag is specified in *act.sa_flags*, the signal handler address is passed via the *act.sa_sigaction* field. This handler takes three arguments, as follows:

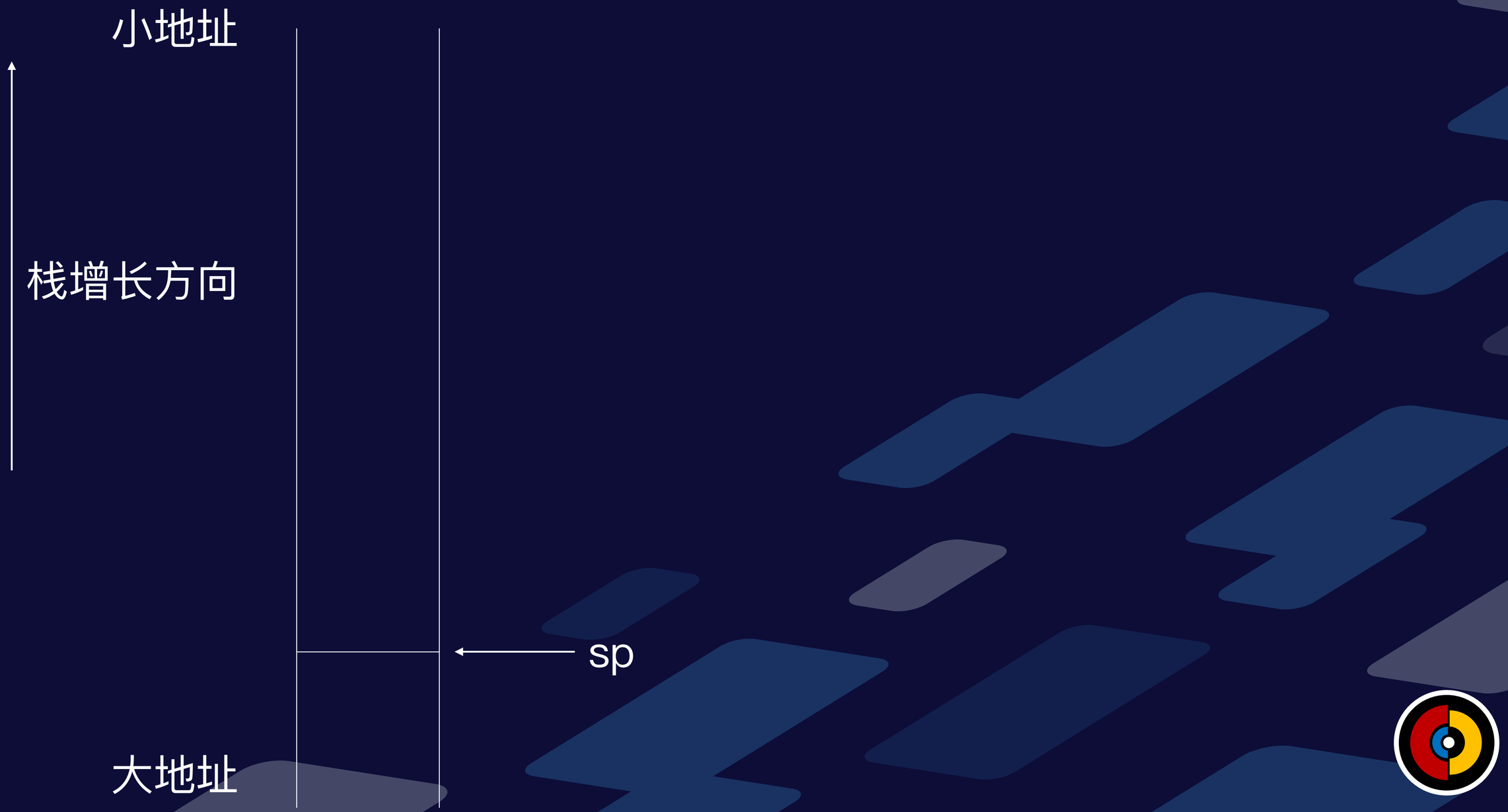
```
void
handler(int sig, siginfo_t *info, void *ucontext)
{
    ...
}
```

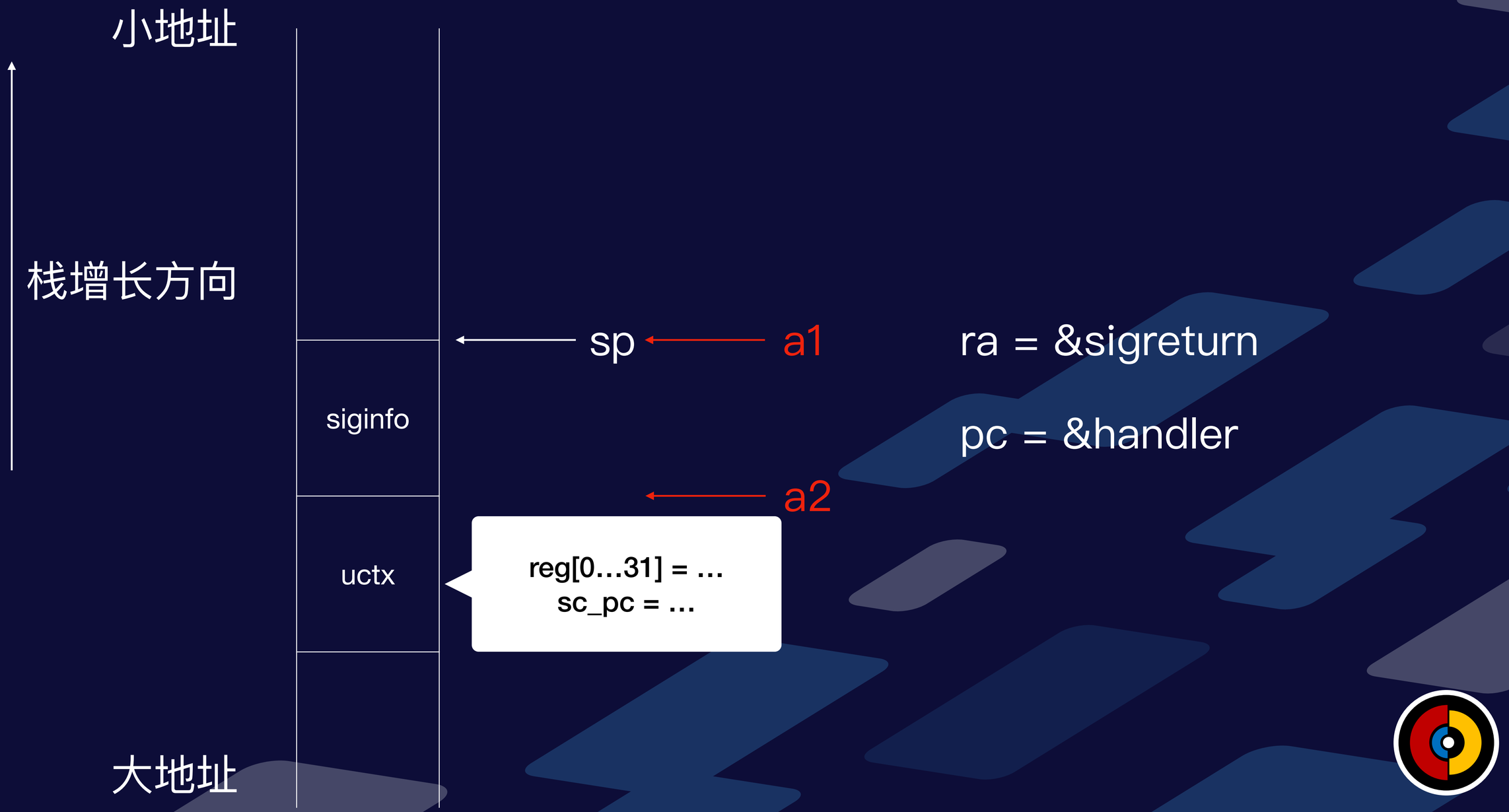
These three arguments are as follows

sig The number of the signal that caused invocation of the handler.

info A pointer to a *siginfo_t*, which is a structure containing further information about the signal, as described below.

ucontext
This is a pointer to a *ucontext_t* structure, cast to *void **. The structure pointed to by this field contains signal context information that was saved on the user-space stack by the kernel; for details, see sigreturn(2). Further information about the *ucontext_t* structure can be found in getcontext(3) and signal(7). Commonly, the handler function doesn't make any use of the third argument.


小地址
栈增长方向
sp
大地址


小地址
栈增长方向
大地址
siginfo
uctx
sp
a1
a2
ra = &sigreturn
pc = &handler
reg[0…31] = …
sc_pc = …

表格 1

	新世界			旧世界		
偏移量	成员名	长度	备注	成员名	长度	备注
0	uc_flags	8		uc_flags	8	
8	uc_link	8		uc_link	8	
16	uc_stack	24		uc_stack	24	
40	uc_sigmask	8		（间隙）	24	
48	unused	120				
64				uc_mcontext.sc_pc	8	
72				uc_mcontext.sc_regs[0…11]	32 × 8	32 个通用寄存器
168	（间隙）	8		uc_mcontext.sc_regs[12]		
176	uc_mcontext.sc_pc	8		uc_mcontext.sc_regs[13]		
184	uc_mcontext.sc_regs[0…17]	32 × 8	32 个通用寄存器	uc_mcontext.sc_regs[14…31]		
328	uc_mcontext.sc_regs[18]			uc_mcontext.sc_flags	4	
332				uc_mcontext.sc_fcsr	4	
336	uc_mcontext.sc_regs[19]			uc_mcontext.sc_none	4	
				（间隙）	4	
344	uc_mcontext.sc_regs[20]			uc_mcontext.sc_fcc	8	
352	uc_mcontext.sc_regs[21…24]			uc_mcontext.sc_scr[4]	4 × 8	4 个 LBT 寄存器
384	uc_mcontext.sc_regs[25]			uc_mcontext.sc_fpregs[32]	32 × 32	32 个浮点寄存器，对齐到 32 字节
440	uc_mcontext.sc_flags	4				
444	（间隙）	4				
448	uc_mcontext.sc_extcontext					
1408				uc_mcontext.sc_reserved[4]	4	对齐到 16 字节，前 4 字节实际存储 LBT 的 eflags
1412				uc_mcontext.sc_reserved[4092]	4092	
5504				uc_sigmask	16	
5520				__unused	112	
5632						

```c
int sigaction(int sig, const struct sigaction *sa, struct sigaction *osa){
    return syscall(sigaction, sig, sa, osa);
}
```

```c
static void *handlers[NSIG];

int sigaction(int sig, const struct sigaction *sa, struct sigaction *osa){
    old_handler = handlers[sig];
    handlers[sig] = sa->sa_handler;
    sa->sa_handler = &shim_handler;
    int rc = syscall(sigaction, sig, sa, osa);
    if(rc >= 0){
        osa->sa_handler = old_handler;
    } else {
        handlers[sig] = old_handler;
    }
    return rc;
}
void shim_handler(int sig, siginfo_t *info, nw_ucontext_t *ctx){
    ucontext_t ow_ctx;
    // convert nw_ucontext_t to oldworld_ucontext_t
    // ow_ctx = covert(*ctx);
    handlers[sig](sig, info, &ow_ctx);
    // convert oldworld_ucontext_t to ucontext_t
    // *ctx = covert(ow_ctx);
}
```

保存新 handler，备份旧 handler

向内核注册我们的 shim

如果注册失败，恢复原 handler

```c
static void *handlers[NSIG];

int sigaction(int sig, const struct sigaction *sa, struct sigaction *osa){
    old_handler = handlers[sig];
    handlers[sig] = sa->sa_handler;
    sa->sa_handler = &shim_handler;
    int rc = syscall(sigaction, sig, sa, osa);
    if(rc >= 0){
        osa->sa_handler = old_handler;
    } else {
        handlers[sig] = old_handler;
    }
    return rc;
}

void shim_handler(int sig, siginfo_t *info, nw_ucontext_t *ctx){
    ucontext_t ow_ctx;
    // ow_ctx = covert(*ctx);
    handlers[sig](sig, info, &ow_ctx);
    // convert oldworld_ucontext_t to ucontext_t
    // *ctx = covert(ow_ctx);
}
```

保存新 handler，备份旧 handler

向内核注册我们的 shim

如果注册失败，恢复原 handler

问题：

- 如果注册失败，新的 handler 存在短暂的时间窗口可能被调用

```c
static void *handlers[NSIG];

int sigaction(int sig, const struct sigaction *sa, struct sigaction *osa){
    real_handler = sa->sa_handler;
    sa->sa_handler = &shim_handler;
    int rc = syscall(sigaction, sig, sa, osa);
    if(rc >= 0){
        old_handler = handlers[sig];
        handlers[sig] = real_handler;
        osa->sa_handler = old_handler;
    } else {

    }
    return rc;
}
void shim_handler(int sig, siginfo_t *info, nw_ucontext_t *ctx){
    ucontext_t ow_ctx;
    // convert nw_ucontext_t to oldworld_ucontext_t
    // ow_ctx = covert(*ctx);
    handlers[sig](sig, info, &ow_ctx);
    // convert oldworld_ucontext_t to ucontext_t
    // *ctx = covert(ow_ctx);
}
```

向内核注册我们的 shim

保存新 handler，备份旧 handler

如果注册失败，什么都不做

```
static void *handlers[NSIG];

int sigaction(int sig, const struct sigaction *sa, struct sigaction *osa){
    real_handler = sa->sa_handler;
    sa->sa_handler = &shim_handler;
    int rc = syscall(sigaction, sig, sa, osa);
    if(rc >= 0){
        old_handler = handlers[sig];
        handlers[sig] = real_handler;
        osa->sa_handler = old_handler;
    } else {

    }
    return rc;
}
void shim_handler(int sig, siginfo_t *info, nw_ucontext_t *ctx){
    oldworld_ucontext_t ow_ctx;
    // convert nw_ucontext_t to oldworld_ucontext_t
    // ow_ctx = covert(*ctx);
    handlers[sig](sig, info, &ow_ctx);
    // convert oldworld_ucontext_t to ucontext_t
    // *ctx = covert(ow_ctx);
}
```

向内核注册我们的 shim

保存新 handler，备份旧 handler

如果注册失败，什么都不做

问题：

- 如果注册成功后，保存新 handler 前有信号到来，会丢失信号

疯狂的打包之旅

其它问题

- 输入法不工作：
  - 补上输入法相关动态库
- VSCode 打开目录按钮点击后崩溃
  - 重新编译 gtk2、gtk3、glib2，以修改 gio 模块搜索路径
- 部分图标不能正常显示
  - 重新编译 gdk-pixbuf，以修改渲染不同格式的模块搜索路径

疯狂的打包之旅

启动！

- 编译旧世界 binutils
- 编译旧世界 gcc-8 至 xgcc
- 编译新世界 glibc 2.37 至 ldconfig
- 编译 make-4.3
- 偷 loongnix linux headers 和 libgcc 包进编译环境
- 编译旧世界 glibc，并安装 ld.so.1 和 libc.so.6
- 偷 loongnix libblkid1、libbsd0、libexpat1、libffi7、libfontconfig1、libfreetype6、libgcc1、libgl1、libglib2.0-0、libglvnd0、libglx0、libice6、liblzma5、libmount1、libpcre3、libpng16-16、libselinux1、libsm6、libuuid1、libx11-6、libx11-xcb1、libxau6、libxcb1、libxcomposite1、libxdmcp6、libxext6、libxi6、libxkbcommon0、libxml2、libxrender1、libxtst6、zlib1g、libc6、libatk1.0-0、libatk-bridge2.0-0、libatspi2.0-0、libavahi-client3、libavahi-common3、libcairo2、libcairo-gobject2、libcom-err2、libcups2、libdatrie1、libdbus-1-3、libdrm2、libepoxy0、libfribidi0、libgcrypt20、libgdk-pixbuf2.0-0、libgmp10、libgnutls30、libgpg-error0、libgraphite2-3、libgssapi-krb5-2、libgtk-3-0、libharfbuzz0b、libhogweed4、libidn2-0、libk5crypto3、libkeyutils1、libkrb5-3、libkrb5support0、liblz4-1、libnettle6、libp11-kit0、libpango-1.0-0、libpangocairo-1.0-0、libpangoft2-1.0-0、libpixman-1-0、

libsystemd0、libtasn1-6、libthai0、libunistring2、libwayland-client0、libwayland-cursor0、libwayland-egl1、libwayland-server0、libxcb-render0、libxcb-shm0、libxcursor1、libxdamage1、libxfixes3、libxinerama1、libxrandr2、libgtk2.0-0、libjpeg62-turbo、libtiff5、libjbig0、libcairo-script-interpreter2、libcolord2、libcupsimage2、libdrm-amdgpu1、libdrm-etnaviv1、libdrm-gsgpu1、libdrm-nouveau2、libdrm-radeon1、libdrm-utils、libegl1、libgirepository-1.0-1、libgles1、libgles2、libharfbuzz-gobject0、libharfbuzz-icu0、libicu67、libjson-glib-1.0-0、libopengl0、libpangoxft-1.0-0、libpcre16-3、libpcre32-3、librest-0.7-0、libsoup2.4-1、libtiffxx5、libwayland-bin、libxcb-dri2-0、libxcb-dri3-0、libxcb-glx0、libxcb-present0、libxcb-randr0、libxcb-shape0、libxcb-sync1、libxcb-xfixes0、libxft2、libxkbfile1、libxshmfence1、libxt6、libxxf86vm1、libgirepository1.0-dev、libglib2.0-dev、libjpeg62-turbo-dev、libpng-dev、libtiff5-dev、libtiff-dev、libx11-dev、libpcre3-dev、libffi-dev、libmount-dev、libblkid-dev、libselinux1-dev、libsepol1、libsepol1-dev、zlib1g-dev、libc6-dev、libgcc-8-dev、libxcb1-dev、libxdmcp-dev、libxau-dev、x11proto-dev、libpthread-stubs0-dev、libatk-bridge2.0-dev、libatk1.0-dev、libatspi2.0-dev、libcairo2-dev、libcolord-dev、libcups2-dev、libcupsimage2-dev、libdbus-1-dev、libdrm-dev、libegl1-mesa-dev、libepoxy-dev、libexpat1-dev、libfontconfig1-dev、libfreetype6-dev、libfribidi-dev、libgdk-pixbuf2.0-dev、libgl1-mesa-dev、libglvnd-core-dev、libglvnd-dev、libgraphite2-dev、libharfbuzz-dev、libice-dev、libicu-dev、libjbig-dev、libjpeg-dev、libjson-glib-dev、liblzma-dev、libpango1.0-dev、libpixman-1-dev、librest-dev、libsm-dev、libsoup2.4-dev、libwayland-dev、libx11-xcb-dev、libxcb-dri2-0-dev、libxcb-dri3-dev、libxcb-glx0-dev、libxcb-present-dev、libxcb-randr0-dev、libxcb-render0-dev、libxcb-shape0-dev、libxcb-shm0-dev、libxcb-sync-dev、libxcb-xfixes0-dev、libxcomposite-dev、libxcursor-dev、libxdamage-dev、libxext-dev、libxfixes-dev、libxft-dev、libxi-dev、libxinerama-dev、libxkbcommon-dev、libxkbfile-dev、libxml2-dev、libxrandr-dev、libxrender-dev、libxshmfence-dev、libxt-dev、libxtst-dev、libxxf86vm-dev、mesa-common-dev、uuid-dev、x11proto-composite-dev、x11proto-core-dev、x11proto-damage-dev、x11proto-fixes-dev、x11proto-input-dev、x11proto-randr-dev、x11proto-record-dev、x11proto-xext-dev、x11proto-xf86vidmode-dev、x11proto-xinerama-dev、xtrans-dev 包进编译环境

- 编译 gdk-pixbuf、gtk2、gtk3、glib2 并安装
- 偷 loongnix libblkid1、libbsd0、libexpat1、libffi7、libfontconfig1、libfreetype6、libgcc1、libgl1、libglib2.0-0、libglvnd0、libglx0、libice6、liblzma5、libmount1、libpcre3、libpng16-16、libselinux1、libsm6、libsqlite3-0、libuuid1、libx11-6、libx11-xcb1、libxau6、libxcb-xkb1、libxcb1、libxcomposite1、libxdmcp6、libxext6、libxi6、libxkbcommon-x11-0、libxkbcommon0、libxml2、libxrender1、libxtst6、zlib1g、libtinfo6、libc6、gcc-8-base、libasound2、libatk1.0-0、libatk-bridge2.0-0、libatspi2.0-0、libavahi-client3、libavahi-common3、libcairo2、libcairo-gobject2、libcom-err2、libcups2、libdatrie1、libdbus-1-3、libdrm2、libepoxy0、libfribidi0、libgbm1、libgcrypt20、libgdk-pixbuf2.0-0、libgmp10、libgnutls30、libgpg-error0、libgraphite2-3、libgssapi-krb5-2、libkrb5-dbg、libgtk-3-0、libharfbuzz0b、libhogweed4、libidn2-0、libk5crypto3、libkeyutils1、libkrb5-3、libkrb5support0、liblz4-1、libnettle6、libnspr4、libnss3、libp11-kit0、libpango-1.0-0、libpangocairo-1.0-0、libpangoft2-1.0-0、libpixman-1-0、libsystemd0、libtasn1-6、libthai0、libunistring2、libwayland-client0、libwayland-cursor0、libwayland-egl1、libwayland-server0、libxcb-render0、libxcb-shm0、libxcursor1、libxdamage1、libxfixes3、libxinerama1、libxrandr2、libasyncns0、libcap2、libflac8、libgtk2.0-0、libogg0、libogg-dbg、libpulse0、libsndfile1、libvorbis0a、libvorbisenc2、loonggl、libwrap0、libpci3、libudev1、libstdc++6、libstdc++6-8-dbg、libjpeg62-turbo、libtiff5、libjbig0、libzstd1、libwebp6、libcairo-script-interpreter2、libcolord2、libcupsimage2、libdrm-amdgpu1、libdrm-etnaviv1、libdrm-gsgpu1、libdrm-nouveau2、libdrm-radeon1、libdrm-utils、libegl1、libgirepository-1.0-1、libgles1、libgles2、libharfbuzz-gobject0、libharfbuzz-icu0、libicu67、libjson-glib-1.0-0、libopengl0、libpangoxft-1.0-0、libpcre16-3、libpcre32-3、librest-0.7-0、libsoup2.4-1、libtiffxx5、libwayland-bin、libxcb-dri2-0、libxcb-dri3-0、libxcb-glx0、libxcb-present0、libxcb-randr0、libxcb-shape0、libxcb-sync1、libxcb-xfixes0、libxft2、libxkbfile1、libxshmfence1、libxt6、libxxf86vm1、libsepol1、librsvg2-common、librsvg2-2、libicu63、libcroco3、dconf-gsettings-backend、gvfs、glib-networking、libproxy1v5、gvfs-libs、ibus-gtk、ibus-gtk3、libibus-1.0-5、fcitx-frontend-gtk2、fcitx-frontend-gtk3、libfcitx-gclient1、libfcitx-utils0 包并安装

- 重新生成各种缓存

疯狂的打包之旅

还没完!

- 0.0.3 发布后，lbrowser 和各种 electron 打包的应用可以正常运行
- 赶在了 2023 年最后一天

疯狂的打包之旅

启动！

- 编译旧世界 binutils
- 编译旧世界 gcc-8 至 xgcc
- 编译新世界 glibc 2.37 至 ldconfig
- 编译 make-4.3
- 偷 loongnix linux headers 和 libgcc 包进编译环境
- 编译旧世界 glibc，并安装 ld.so.1 和 libc.so.6
- 偷 loongnix libblkid1、libbsd0、libexpat1、libffi7、libfontconfig1、libfreetype6、libgcc1、libgl1、libglib2.0-0、libglvnd0、libglx0、libice6、liblzma5、libmount1、libpcre3、libpng16-16、libselinux1、libsm6、libuuid1、libx11-6、libx11-xcb1、libxau6、libxcb1、libxcomposite1、libxdmcp6、libxext6、libxi6、libxkbcommon0、libxml2、libxrender1、libxtst6、zlib1g、libc6、libatk1.0-0、libatk-bridge2.0-0、libatspi2.0-0、libavahi-client3、libavahi-common3、libcairo2、libcairo-gobject2、libcom-err2、libcups2、libdatrie1、libdbus-1-3、libdrm2、libepoxy0、libfribidi0、libgcrypt20、libgdk-pixbuf2.0-0、libgmp10、libgnutls30、libgpg-error0、libgraphite2-3、libgssapi-krb5-2、libgtk-3-0、libharfbuzz0b、libhogweed4、libidn2-0、libk5crypto3、libkeyutils1、libkrb5-3、libkrb5support0、liblz4-1、libnettle6、libp11-kit0、libpango-1.0-0、libpangocairo-1.0-0、libpangoft2-1.0-0、libpixman-1-0、libsystemd0、libtasn1-6、libthai0、libunistring2、libwayland-client0、libwayland-cursor0、libwayland-egl1、libwayland-server0、libxcb-render0、libxcb-shm0、libxcursor1、libxdamage1、libxfixes3、libxinerama1、libxrandr2、libgtk2.0-0、libjpeg62-turbo、libtiff5、libjbig0、libcairo-script-interpreter2、libcolord2、libcupsimage2、libdrm-amdgpu1、libdrm-etnaviv1、libdrm-gsgpu1、libdrm-nouveau2、libdrm-radeon1、libdrm-utils、libegl1、libgirepository-1.0-1、libgles1、libgles2、libharfbuzz-gobject0、libharfbuzz-icu0、libicu67、libjson-glib-1.0-0、libopengl0、libpangoxft-1.0-0、libpcre16-3、libpcre32-3、librest-0.7-0、libsoup2.4-1、libtiffxx5、libwayland-bin、libxcb-dri2-0、libxcb-dri3-0、libxcb-glx0、libxcb-present0、libxcb-randr0、libxcb-shape0、libxcb-sync1、libxcb-xfixes0、libxft2、libxkbfile1、libxshmfence1、libxt6、libxxf86vm1、libgirepository1.0-dev、libglib2.0-dev、libjpeg62-turbo-dev、libpng-dev、libtiff5-dev、libtiff-dev、libx11-dev、libpcre3-dev、libffi-dev、libmount-dev、libblkid-dev、libselinux1-dev、libsepol1、libsepol1-dev、zlib1g-dev、libc6-dev、libgcc-8-dev、libxcb1-dev、libxdmcp-dev、libxau-dev、x11proto-dev、libpthread-stubs0-dev、libatk-bridge2.0-dev、libatk1.0-dev、libatspi2.0-dev、libcairo2-dev、libcolord-dev、libcups2-dev、libcupsimage2-dev、libdbus-1-dev、libdrm-dev、libegl1-mesa-dev、libepoxy-dev、libexpat1-dev、libfontconfig1-dev、libfreetype6-dev、libfribidi-dev、libgdk-pixbuf2.0-dev、libgl1-mesa-dev、libglvnd-core-dev、libglvnd-dev、libgraphite2-dev、libharfbuzz-dev、libice-dev、libicu-dev、libjbig-dev、libjpeg-dev、libjson-glib-dev、liblzma-dev、libpango1.0-dev、libpixman-1-dev、librest-dev、libsm-dev、libsoup2.4-dev、libwayland-dev、libx11-xcb-dev、libxcb-dri2-0-dev、libxcb-dri3-dev、libxcb-glx0-dev、libxcb-present-dev、libxcb-randr0-dev、libxcb-render0-dev、libxcb-shape0-dev、libxcb-shm0-dev、libxcb-sync-dev、libxcb-xfixes0-dev、libxcomposite-dev、libxcursor-dev、libxdamage-dev、libxext-dev、libxfixes-dev、libxft-dev、libxi-dev、libxinerama-dev、libxkbcommon-dev、libxkbfile-dev、libxml2-dev、libxrandr-dev、libxrender-dev、libxshmfence-dev、libxt-dev、libxtst-dev、libxxf86vm-dev、mesa-common-dev、uuid-dev、x11proto-composite-dev、x11proto-core-dev、x11proto-damage-dev、x11proto-fixes-dev、x11proto-input-dev、x11proto-randr-dev、x11proto-record-dev、x11proto-xext-dev、x11proto-xf86vidmode-dev、x11proto-xinerama-dev、xtrans-dev 包进编译环境
- 编译 gdk-pixbuf、gtk2、gtk3、glib2、 libcanberra 并安装
- 偷 loongnix libblkid1、libbsd0、libexpat1、libffi7、libfontconfig1、libfreetype6、libgcc1、libgl1、libglib2.0-0、libglvnd0、libglx0、libice6、liblzma5、libmount1、libpcre3、libpng16-16、libselinux1、libsm6、libsqlite3-0、libuuid1、libx11-6、libx11-xcb1、libxau6、libxcb-xkb1、libxcb1、libxcomposite1、libxdmcp6、libxext6、libxi6、libxkbcommon-x11-0、libxkbcommon0、libxml2、libxrender1、libxtst6、zlib1g、libtinfo6、libc6、gcc-8-base、libasound2、libatk1.0-0、libatk-bridge2.0-0、libatspi2.0-0、libavahi-client3、libavahi-common3、libcairo2、libcairo-gobject2、libcom-err2、libcups2、libdatrie1、libdbus-1-3、libdrm2、libepoxy0、libfribidi0、libgbm1、libgcrypt20、libgdk-pixbuf2.0-0、libgmp10、libgnutls30、libgpg-error0、libgraphite2-3、libgssapi-krb5-2、libkrb5-dbg、libgtk-3-0、libharfbuzz0b、libhogweed4、libidn2-0、libk5crypto3、libkeyutils1、libkrb5-3、libkrb5support0、liblz4-1、libnettle6、libnspr4、libnss3、libp11-kit0、libpango-1.0-0、libpangocairo-1.0-0、libpangoft2-1.0-0、libpixman-1-0、libsystemd0、libtasn1-6、libthai0、libunistring2、libwayland-client0、libwayland-cursor0、libwayland-egl1、libwayland-server0、libxcb-render0、libxcb-shm0、libxcursor1、libxdamage1、libxfixes3、libxinerama1、libxrandr2、libasyncns0、libcap2、libflac8、libgtk2.0-0、libogg0、libogg-dbg、libpulse0、libsndfile1、libvorbis0a、libvorbisenc2、loonggl、libwrap0、libpci3、libudev1、libstdc++6、libstdc++6-8-dbg、libjpeg62-turbo、libtiff5、libjbig0、libzstd1、libwebp6、libcairo-script-interpreter2、libcolord2、libcupsimage2、libdrm-amdgpu1、libdrm-etnaviv1、libdrm-gsgpu1、libdrm-nouveau2、libdrm-radeon1、libdrm-utils、libegl1、libgirepository-1.0-1、libgles1、libgles2、libharfbuzz-gobject0、libharfbuzz-icu0、libicu67、libjson-glib-1.0-0、libopengl0、libpangoxft-1.0-0、libpcre16-3、libpcre32-3、librest-0.7-0、libsoup2.4-1、libtiffxx5、libwayland-bin、libxcb-dri2-0、libxcb-dri3-0、libxcb-glx0、libxcb-present0、libxcb-randr0、libxcb-shape0、libxcb-sync1、libxcb-xfixes0、libxft2、libxkbfile1、libxshmfence1、libxt6、libxxf86vm1、libsepol1、librsvg2-common、librsvg2-2、libicu63、libcroco3、dconf-gsettings-backend、gvfs、glib-networking、libproxy1v5、gvfs-libs、ibus-gtk、ibus-gtk3、libibus-1.0-5、fcitx-frontend-gtk2、fcitx-frontend-gtk3、libfcitx-gclient1、libfcitx-utils0 包并安装
- 重新生成各种缓存

巧夺麻袋……信号处理那里还有 race 呢?

信号处理程序

问题在哪

- 本质原因：保存的 handler 地址与向内核注册的 handler 地址不具有固结关系
  - shim handler 从 handlers[] 数组中间接获取保存的 handler 地址
- What if…
  - 我们为每个待注册的 handler 现场生成一份 shim?
  - 这样，shim 与 handler 就固结了起来！

```
int sigaction(int sig, const struct sigaction *sa, struct sigaction *osa){
    sa->sa_handler = gen_shim_for(sa->sa_handler);
    int rc = syscall(sigaction, sig, sa, osa);
    if(rc >= 0){
        osa->sa_handler = get_orig_handler_from_shim(osa->sa_handler);
    }
    return rc;
}
```

偏移	内容	长度
0	Magic	8
8	pcaddi $t0, 4 ld.d $a3, $t0, 0 ld.d $t0, $t0, 8 jirl $zero, $t0, 0	4 × 4
24	real_handler_addr	8
32	shim_handler_addr	8
40		

```c
void shim_handler(int sig, siginfo_t *info, nw_ucontext_t *ctx, void *real_handler){
    ucontext_t ow_ctx;
    // convert nw_ucontext_t to oldworld_ucontext_t
    // ow_ctx = covert(*ctx);
    real_handler(sig, info, &ow_ctx);
    // convert oldworld_ucontext_t to ucontext_t
    // *ctx = covert(ow_ctx);
}
```

```
Magic = U"开刀"

开: 00005f00  ext.w.b       $zero, $s1

刀: 00005200  bitrev.w      $zero, $t4
```

疯狂是怎么终结的?

疯狂的终结

- 强行偷包
  - 体积过大
  - 难以穷尽
  - 版权不明

疯狂的终结
用户态的区别

- 用户态的二进制调用约定是否发生变化?
  - 事实上，没有；
- 可以用新世界编译器编译旧世界程序吗?
  - 可以!

疯狂的终结

- ~~编译旧世界 binutils~~
- ~~编译旧世界 gcc-8 至 xgcc~~
- ~~编译新世界 glibc 2.37 至 ldconfig~~
- 编译 make-4.3
- 偷 loongnix linux headers ~~和 libgcc~~ 包进编译环境
- 编译旧世界 glibc，并安装 ld.so.1 和 libc.so.6
- 偷 loongnix libblkid1、libbsd0、libexpat1、libffi7、libfontconfig1、libfreetype6、libgcc1、libgl1、libglib2.0-0、libglvnd0、libglx0、libice6、liblzma5、libmount1、libpcre3、libpng16-16、libselinux1、libsm6、libuuid1、libx11-6、libx11-xcb1、libxau6、libxcb1、libxcomposite1、libxdmcp6、libxext6、libxi6、libxkbcommon0、libxml2、libxrender1、libxtst6、zlib1g、libc6、libatk1.0-0、libatk-bridge2.0-0、libatspi2.0-0、libavahi-client3、libavahi-common3、libcairo2、libcairo-gobject2、libcom-err2、libcups2、libdatrie1、libdbus-1-3、libdrm2、libepoxy0、libfribidi0、libgcrypt20、libgdk-pixbuf2.0-0、libgmp10、libgnutls30、libgpg-error0、libgraphite2-3、libgssapi-krb5-2、libgtk-3-0、libharfbuzz0b、libhogweed4、libidn2-0、libk5crypto3、libkeyutils1、libkrb5-3、libkrb5support0、liblz4-1、libnettle6、libp11-kit0、libpango-1.0-0、libpangocairo-1.0-0、libpangoft2-1.0-0、libpixman-1-0、libsystemd0、libtasn1-6、libthai0、libunistring2、libwayland-client0、libwayland-cursor0、libwayland-egl1、libwayland-server0、libxcb-render0、libxcb-shm0、libxcursor1、libxdamage1、libxfixes3、libxinerama1、libxrandr2、libgtk2.0-0、libjpeg62-turbo、libtiff5、libjbig0、libcairo-script-interpreter2、libcolord2、libcupsimage2、libdrm-amdgpu1、libdrm-etnaviv1、libdrm-gsgpu1、libdrm-nouveau2、libdrm-radeon1、libdrm-utils、libegl1、libgirepository-1.0-1、libgles1、libgles2、libharfbuzz-gobject0、libharfbuzz-icu0、libicu67、libjson-glib-1.0-0、libopengl0、libpangoxft-1.0-0、libpcre16-3、libpcre32-3、librest-0.7-0、libsoup2.4-1、libtiffxx5、libwayland-bin、libxcb-dri2-0、libxcb-dri3-0、libxcb-glx0、libxcb-present0、libxcb-randr0、libxcb-shape0、libxcb-sync1、libxcb-xfixes0、libxft2、libxkbfile1、libxshmfence1、libxt6、libxxf86vm1、libgirepository1.0-dev、libglib2.0-dev、libjpeg62-turbo-dev、libpng-dev、libtiff5-dev、libtiff-dev、libx11-dev、libpcre3-dev、libffi-dev、libmount-dev、libblkid-dev、libselinux1-dev、libsepol1、libsepol1-dev、zlib1g-dev、libc6-dev、libgcc-8-dev、libxcb1-dev、libxdmcp-dev、libxau-dev、x11proto-dev、libpthread-stubs0-dev、libatk-bridge2.0-dev、libatk1.0-dev、libatspi2.0-dev、libcairo2-dev、libcolord-dev、libcups2-dev、libcupsimage2-dev、libdbus-1-dev、libdrm-dev、libegl1-mesa-dev、libepoxy-dev、libexpat1-dev、libfontconfig1-dev、libfreetype6-dev、libfribidi-dev、libgdk-pixbuf2.0-dev、libgl1-mesa-dev、libglvnd-core-dev、libglvnd-dev、libgraphite2-dev、libharfbuzz-dev、libice-dev、libicu-dev、libjbig-dev、libjpeg-dev、libjson-glib-dev、liblzma-dev、libpango1.0-dev、libpixman-1-dev、librest-dev、libsm-dev、libsoup2.4-dev、libwayland-dev、libx11-xcb-dev、libxcb-dri2-0-dev、libxcb-dri3-dev、libxcb-glx0-dev、libxcb-present-dev、libxcb-randr0-dev、libxcb-render0-dev、libxcb-shape0-dev、libxcb-shm0-dev、libxcb-sync-dev、libxcb-xfixes0-dev、libxcomposite-dev、libxcursor-dev、libxdamage-dev、libxext-dev、libxfixes-dev、libxft-dev、libxi-dev、libxinerama-dev、libxkbcommon-dev、libxkbfile-dev、libxml2-dev、libxrandr-dev、libxrender-dev、libxshmfence-dev、libxt-dev、libxtst-dev、libxxf86vm-dev、mesa-common-dev、uuid-dev、x11proto-composite-dev、x11proto-core-dev、x11proto-damage-dev、x11proto-fixes-dev、x11proto-input-dev、x11proto-randr-dev、x11proto-record-dev、x11proto-xext-dev、x11proto-xf86vidmode-dev、x11proto-xinerama-dev、xtrans-dev 包进编译环境

- 编译 gdk-pixbuf、gtk2、gtk3、glib2 并安装
- 偷 loongnix libblkid1、libbsd0、libexpat1、libffi7、libfontconfig1、libfreetype6、libgcc1、libgl1、libglib2.0-0、libglvnd0、libglx0、libice6、liblzma5、libmount1、libpcre3、libpng16-16、libselinux1、libsm6、libsqlite3-0、libuuid1、libx11-6、libx11-xcb1、libxau6、libxcb-xkb1、libxcb1、libxcomposite1、libxdmcp6、libxext6、libxi6、libxkbcommon-x11-0、libxkbcommon0、libxml2、libxrender1、libxtst6、zlib1g、libtinfo6、libc6、gcc-8-base、libasound2、libatk1.0-0、libatk-bridge2.0-0、libatspi2.0-0、libavahi-client3、libavahi-common3、libcairo2、libcairo-gobject2、libcom-err2、libcups2、libdatrie1、libdbus-1-3、libdrm2、libepoxy0、libfribidi0、libgbm1、libgcrypt20、libgdk-pixbuf2.0-0、libgmp10、libgnutls30、libgpg-error0、libgraphite2-3、libgssapi-krb5-2、libkrb5-dbg、libgtk-3-0、libharfbuzz0b、libhogweed4、libidn2-0、libk5crypto3、libkeyutils1、libkrb5-3、libkrb5support0、liblz4-1、libnettle6、libnspr4、libnss3、libp11-kit0、libpango-1.0-0、libpangocairo-1.0-0、libpangoft2-1.0-0、libpixman-1-0、libsystemd0、libtasn1-6、libthai0、libunistring2、libwayland-client0、libwayland-cursor0、libwayland-egl1、libwayland-server0、libxcb-render0、libxcb-shm0、libxcursor1、libxdamage1、libxfixes3、libxinerama1、libxrandr2、libasyncns0、libcap2、libflac8、libgtk2.0-0、libogg0、libogg-dbg、libpulse0、libsndfile1、libvorbis0a、libvorbisenc2、loonggl、libwrap0、libpci3、libudev1、libstdc++6、libstdc++6-8-dbg、libjpeg62-turbo、libtiff5、libjbig0、liblzma1、libwebp6、libcairo-script-interpreter2、libcolord2、libcupsimage2、libdrm-amdgpu1、libdrm-etnaviv1、libdrm-gsgpu1、libdrm-nouveau2、libdrm-radeon1、libdrm-utils、libegl1、libgirepository-1.0-1、libgles1、libgles2、libharfbuzz-gobject0、libharfbuzz-icu0、libicu67、libjson-glib-1.0-0、libopengl0、libpangoxft-1.0-0、libpcre16-3、libpcre32-3、librest-0.7-0、libsoup2.4-1、libtiffxx5、libwayland-bin、libxcb-dri2-0、libxcb-dri3-0、libxcb-glx0、libxcb-present0、libxcb-randr0、libxcb-shape0、libxcb-sync1、libxcb-xfixes0、libxft2、libxkbfile1、libxshmfence1、libxt6、libxxf86vm1、libsepol1、librsvg2-common、librsvg2-2、libicu63、libcroco3、dconf-gsettings-backend、gvfs、glib-networking、libproxy1v5、gvfs-libs、ibus-gtk、ibus-gtk3、libibus-1.0-5、fcitx-frontend-gtk2、fcitx-frontend-gtk3、libfcitx-gclient1、libfcitx-utils0 包并安装

- 重新生成各种缓存

疯狂的终结

用户态的区别

- 旧世界程序调用新世界的共享库?
  - 原理上可行!
- 阻碍因素?
  - 符号版本

何为符号版本

- 一个纯字符串
  - 与动态符号名相关联
  - 在执行动态连接时，匹配 <符号名，版本> 二元组
- 习惯上表示为：符号名@版本
  - 例如：memcpy@GLIBC_2.17
- 动态连接时，版本号不作顺序比较

```
[shanker@neomirrors ~> readelf --dyn-syms -W /lib/aarch64-linux-gnu/libc.so.6 ]

Symbol table '.dynsym' contains 2959 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000000273c0     0 SECTION LOCAL  DEFAULT   12 .text
     2: 000000000019cde8     0 SECTION LOCAL  DEFAULT   22 __libc_subfreeres
     3: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __tls_get_addr@GLIBC_2.17 (21)
     4: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND _dl_exception_create@GLIBC_PRIVATE (22)
     5: 0000000000000000     0 OBJECT  GLOBAL DEFAULT  UND _dl_argv@GLIBC_PRIVATE (22)
     6: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND _dl_find_dso_for_object@GLIBC_PRIVATE (22)
     7: 0000000000000000     0 OBJECT  GLOBAL DEFAULT  UND __pointer_chk_guard@GLIBC_PRIVATE (22)
     8: 0000000000000000     0 OBJECT  GLOBAL DEFAULT  UND __libc_enable_secure@GLIBC_PRIVATE (22)
     9: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND _dl_deallocate_tls@GLIBC_PRIVATE (22)
    10: 0000000000000000     0 OBJECT  GLOBAL DEFAULT  UND __stack_chk_guard@GLIBC_2.17 (21)
    11: 0000000000000000     0 OBJECT  GLOBAL DEFAULT  UND _rtld_global_ro@GLIBC_PRIVATE (22)
    12: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND _dl_fatal_printf@GLIBC_PRIVATE (22)
    13: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND _dl_audit_symbind_alt@GLIBC_PRIVATE (22)
    14: 0000000000000000     0 OBJECT  GLOBAL DEFAULT  UND __libc_stack_end@GLIBC_2.17 (21)
    15: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND _dl_rtld_di_serinfo@GLIBC_PRIVATE (22)
    16: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND _dl_allocate_tls@GLIBC_PRIVATE (22)
    17: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __tunable_get_val@GLIBC_PRIVATE (22)
    18: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND _dl_allocate_tls_init@GLIBC_PRIVATE (22)
    19: 0000000000000000     0 OBJECT  GLOBAL DEFAULT  UND _rtld_global@GLIBC_PRIVATE (22)
    20: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND __nptl_change_stack_perm@GLIBC_PRIVATE (22)
    21: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND _dl_audit_preinit@GLIBC_PRIVATE (22)
    22: 00000000000730b0   360 FUNC    WEAK   DEFAULT   12 fgetc@@GLIBC_2.17
    23: 000000000007cb70    48 FUNC    GLOBAL DEFAULT   12 pthread_attr_setscope@@GLIBC_2.17
    24: 000000000007c890   140 FUNC    GLOBAL DEFAULT   12 pthread_attr_getstacksize@GLIBC_2.17
    25: 00000000000928f0   136 FUNC    GLOBAL DEFAULT   12 envz_strip@@GLIBC_2.17
    26: 000000000007c890   140 FUNC    GLOBAL DEFAULT   12 pthread_attr_getstacksize@@GLIBC_2.34
    27: 00000000000fe380   180 FUNC    GLOBAL DEFAULT   12 iruserok_af@@GLIBC_2.17
    28: 0000000000117790   192 FUNC    GLOBAL DEFAULT   12 _nss_files_getpwent_r@@GLIBC_PRIVATE
    29: 0000000000085bb0   204 FUNC    WEAK   DEFAULT   12 pthread_setcancelstate@@GLIBC_2.17
    30: 00000000000df650    60 FUNC    GLOBAL DEFAULT   12 cfmakeraw@@GLIBC_2.17
    31: 0000000000108050   824 FUNC    GLOBAL DEFAULT   12 ns_name_pack@GLIBC_2.17
    32: 0000000000108050   824 FUNC    GLOBAL DEFAULT   12 ns_name_pack@@GLIBC_2.34
    33: 0000000000079cc4    12 FUNC    GLOBAL DEFAULT   12 _IO_iter_begin@@GLIBC_2.17
    34: 00000000000bd420    96 FUNC    GLOBAL DEFAULT   12 globfree@@GLIBC_2.17
    35: 000000000007a2c0    60 FUNC    GLOBAL DEFAULT   12 _IO_str_init_readonly@@GLIBC_2.17
    36: 00000000000f70b0    64 FUNC    GLOBAL DEFAULT   12 __vswprintf_chk@@GLIBC_2.17
    37: 00000000001a0268     4 OBJECT  GLOBAL DEFAULT   29 optind@@GLIBC_2.17
    38: 0000000000083ab0    24 FUNC    GLOBAL DEFAULT   12 pthread_mutexattr_getprotocol@@GLIBC_2.34
    39: 0000000000083ab0    24 FUNC    GLOBAL DEFAULT   12 pthread_mutexattr_getprotocol@GLIBC_2.17
    40: 000000000009c0a0     8 FUNC    WEAK   DEFAULT   12 wmemmove@@GLIBC_2.17
    41: 00000000001167b4   796 FUNC    GLOBAL DEFAULT   12 _nss_files_gethostbyname4_r@@GLIBC_PRIVATE
    42: 0000000000084700   676 FUNC    GLOBAL DEFAULT   12 __pthread_rwlock_rdlock@GLIBC_2.17
    43: 00000000000ead10    84 FUNC    GLOBAL DEFAULT   12 __cmsg_nxthdr@@GLIBC_2.17
    44: 00000000000f8f40  1176 FUNC    GLOBAL DEFAULT   12 gethostbyname_r@@GLIBC_2.17
    45: 0000000000126240   232 FUNC    GLOBAL DEFAULT   12 xdecrypt@GLIBC_2.17
    46: 00000000001163b0   200 FUNC    GLOBAL DEFAULT   12 _nss_files_gethostent_r@@GLIBC_PRIVATE
    47: 0000000000d9740    120 FUNC    WEAK   DEFAULT   12 statvfs64@@GLIBC_2.17
    48: 0000000000114fb4    16 FUNC    GLOBAL DEFAULT   12 _nss_files_setprotoent@@GLIBC_PRIVATE
    49: 000000000011c170   132 FUNC    GLOBAL DEFAULT   12 svcraw_create@GLIBC_2.17
```

glibc 符号版本

- glibc 新版本保持与旧版本兼容性
  - 和旧版本 glibc 连接生成的程序，与新版本 glibc 运行时搭配，一定能正常运行
- 为了保持该兼容性
  - 当引入新符号时
    - 新符号的版本被标注为引入时的 glibc 版本
  - 当符号发生 ABI 变化时
    - 新增同名符号，版本标注为发生变化时的 glibc 版本
    - 旧符号和版本保留，ABI 确保不变（但实现可以相应替换）

glibc 符号版本

- glibc 新版本保持与旧版本兼容性
  - 和旧版本 glibc 连接生成的程序，与新版本 glibc 运行时搭配，一定能正常运行
- 为了保持该兼容性
  - 当引入新符号时
    - 新符号的版本被标注为引入时的 glibc 版本
  - 当符号发生 ABI 变化时
    - 新增同名符号，版本标注为发生变化时的 glibc 版本
    - 旧符号和版本保留，ABI 确保不变（但实现可以相应替换）

glibc 符号版本

- glibc 新版本保持与旧版本兼容性
  - 和旧版本 glibc 连接生成的程序，与新版本 glibc 运行时搭配，一定能正常运行
- 为了保持该兼容性
  - 当引入新符号时
    - 新符号的版本被标注为引入时的 glibc 版本
  - 当符号发生 ABI 变化时
    - 新增同名符号，版本标注为发生变化时的 glibc 版本
    - 旧符号和版本保留，ABI 确保不变（但实现可以相应替换）


```
hello_foo.c:
    foo();
    bar();
```

glibc 符号版本

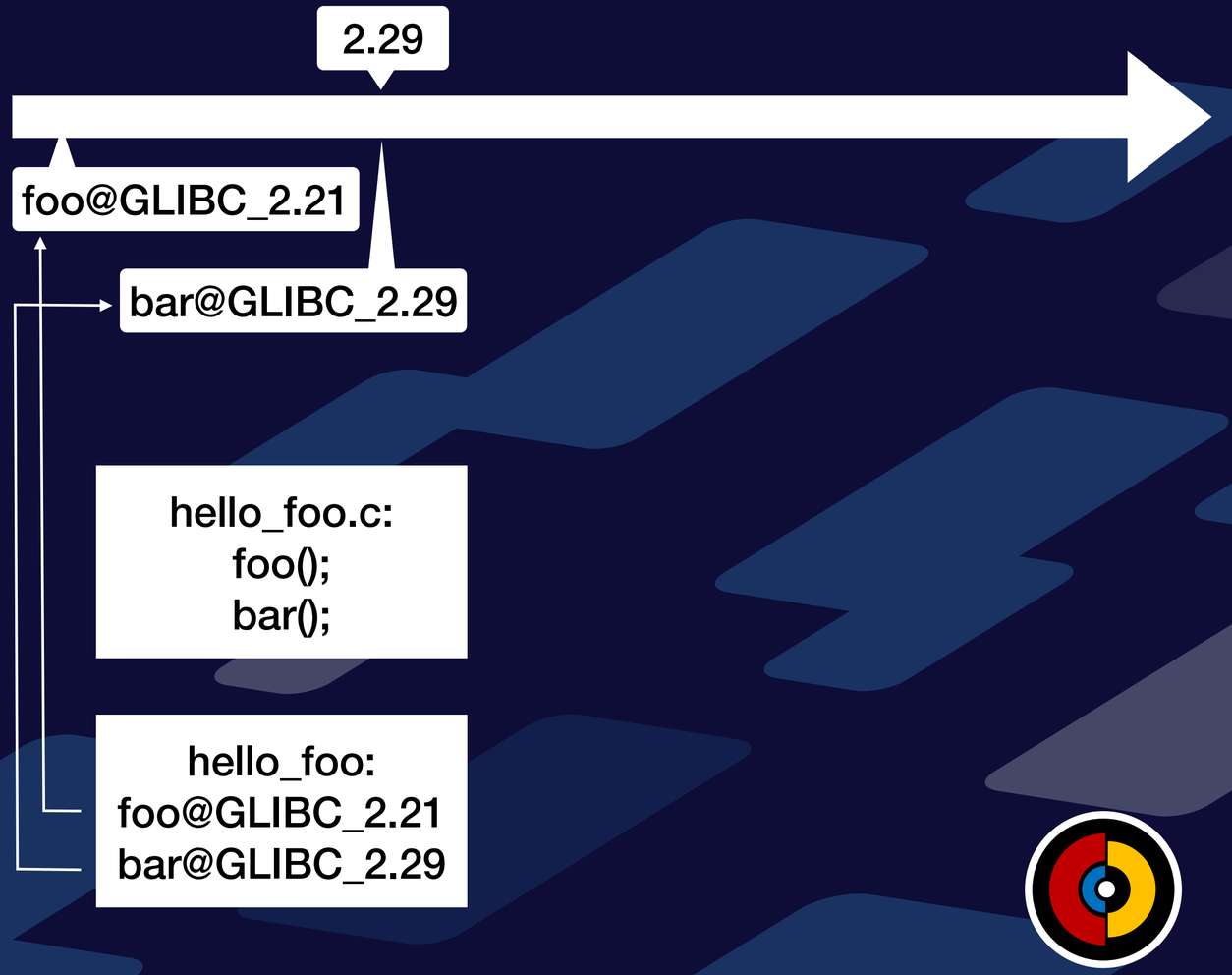
- glibc 新版本保持与旧版本兼容性
  - 和旧版本 glibc 连接生成的程序，与新版本 glibc 运行时搭配，一定能正常运行
- 为了保持该兼容性
  - 当引入新符号时
    - 新符号的版本被标注为引入时的 glibc 版本
  - 当符号发生 ABI 变化时
    - 新增同名符号，版本标注为发生变化时的 glibc 版本
    - 旧符号和版本保留，ABI 确保不变（但实现可以相应替换）


```
hello_foo.c:
foo();
bar();
```

```
hello_foo:
foo@GLIBC_2.21
bar@GLIBC_2.29
```

glibc 符号版本

- glibc 新版本保持与旧版本兼容性
  - 和旧版本 glibc 连接生成的程序，与新版本 glibc 运行时搭配，一定能正常运行
- 为了保持该兼容性
  - 当引入新符号时
    - 新符号的版本被标注为引入时的 glibc 版本
  - 当符号发生 ABI 变化时
    - 新增同名符号，版本标注为发生变化时的 glibc 版本
    - 旧符号和版本保留，ABI 确保不变（但实现可以相应替换）



glibc 符号版本

- glibc 新版本保持与旧版本兼容性
  - 和旧版本 glibc 连接生成的程序，与新版本 glibc 运行时搭配，一定能正常运行
- 为了保持该兼容性
  - 当引入新符号时
    - 新符号的版本被标注为引入时的 glibc 版本
  - 当符号发生 ABI 变化时
    - 新增同名符号，版本标注为发生变化时的 glibc 版本
    - 旧符号和版本保留，ABI 确保不变（但实现可以相应替换）


```
hello_foo.c:
foo();
bar();
```

```
hello_foo:
foo@GLIBC_2.21
bar@GLIBC_2.29
```

glibc 符号版本

- glibc 新版本保持与旧版本兼容性
  - 和旧版本 glibc 连接生成的程序，与新版本 glibc 运行时搭配，一定能正常运行
- 为了保持该兼容性
  - 当引入新符号时
    - 新符号的版本被标注为引入时的 glibc 版本
  - 当符号发生 ABI 变化时
    - 新增同名符号，版本标注为发生变化时的 glibc 版本
    - 旧符号和版本保留，ABI 确保不变（但实现可以相应替换）


```
hello_foo.c:
foo();
bar();
```

```
hello_foo:
foo@GLIBC_2.21
bar@GLIBC_2.29
```

glibc 符号版本

- glibc 新版本保持与旧版本兼容性
  - 和旧版本 glibc 连接生成的程序，与新版本 glibc 运行时搭配，一定能正常运行
- 为了保持该兼容性
  - 当引入新符号时
    - 新符号的版本被标注为引入时的 glibc 版本
  - 当符号发生 ABI 变化时
    - 新增同名符号，版本标注为发生变化时的 glibc 版本
    - 旧符号和版本保留，ABI 确保不变（但实现可以相应替换）

glibc 符号版本

- glibc 新版本保持与旧版本兼容性
  - 和旧版本 glibc 连接生成的程序，与新版本 glibc 运行时搭配，一定能正常运行
- 为了保持该兼容性
  - 当引入新符号时
    - 新符号的版本被标注为引入时的 glibc 版本
  - 当符号发生 ABI 变化时
    - 新增同名符号，版本标注为发生变化时的 glibc 版本
    - 旧符号和版本保留，ABI 确保不变（但实现可以相应替换）


```
hello_foo.c:
  foo();
  bar();
```

glibc 符号版本

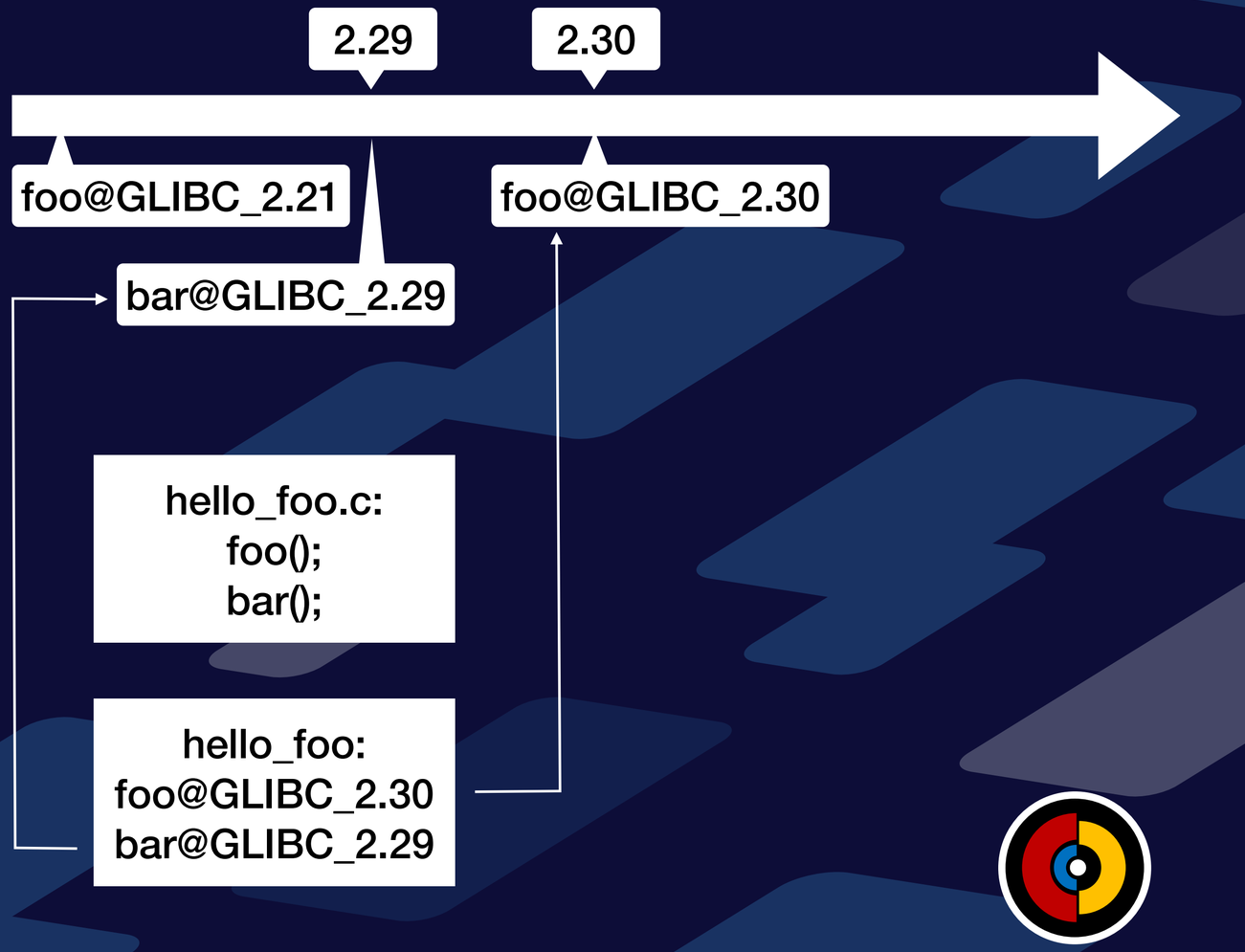
- glibc 新版本保持与旧版本兼容性
  - 和旧版本 glibc 连接生成的程序，与新版本 glibc 运行时搭配，一定能正常运行
- 为了保持该兼容性
  - 当引入新符号时
    - 新符号的版本被标注为引入时的 glibc 版本
  - 当符号发生 ABI 变化时
    - 新增同名符号，版本标注为发生变化时的 glibc 版本
    - 旧符号和版本保留，ABI 确保不变（但实现可以相应替换）


```
hello_foo.c:
foo();
bar();
```

```
hello_foo:
foo@GLIBC_2.30
bar@GLIBC_2.29
```

glibc 符号版本

- glibc 新版本保持与旧版本兼容性
  - 和旧版本 glibc 连接生成的程序，与新版本 glibc 运行时搭配，一定能正常运行
- 为了保持该兼容性
  - 当引入新符号时
    - 新符号的版本被标注为引入时的 glibc 版本
  - 当符号发生 ABI 变化时
    - 新增同名符号，版本标注为发生变化时的 glibc 版本
    - 旧符号和版本保留，ABI 确保不变（但实现可以相应替换）

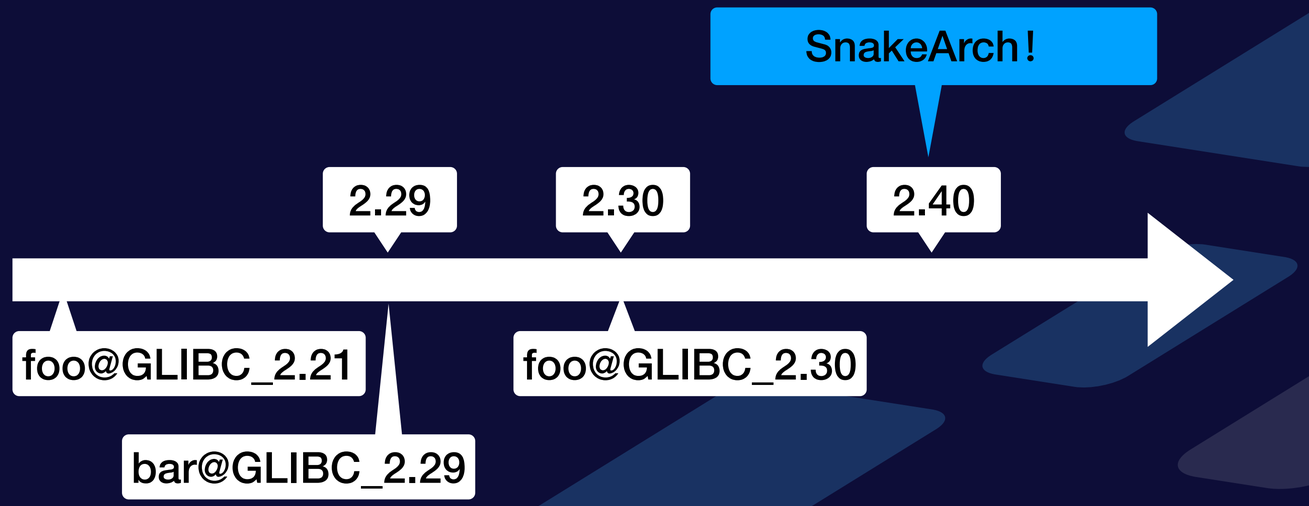


glibc 符号版本

- glibc 新引进架构时
  - 在该架构下的所有符号的版本确定为引进架构的版本
  - 此前更早的符号版本不再编译
- Loongarch 引入时 glibc 版本?
  - 华生，你发现了盲点!

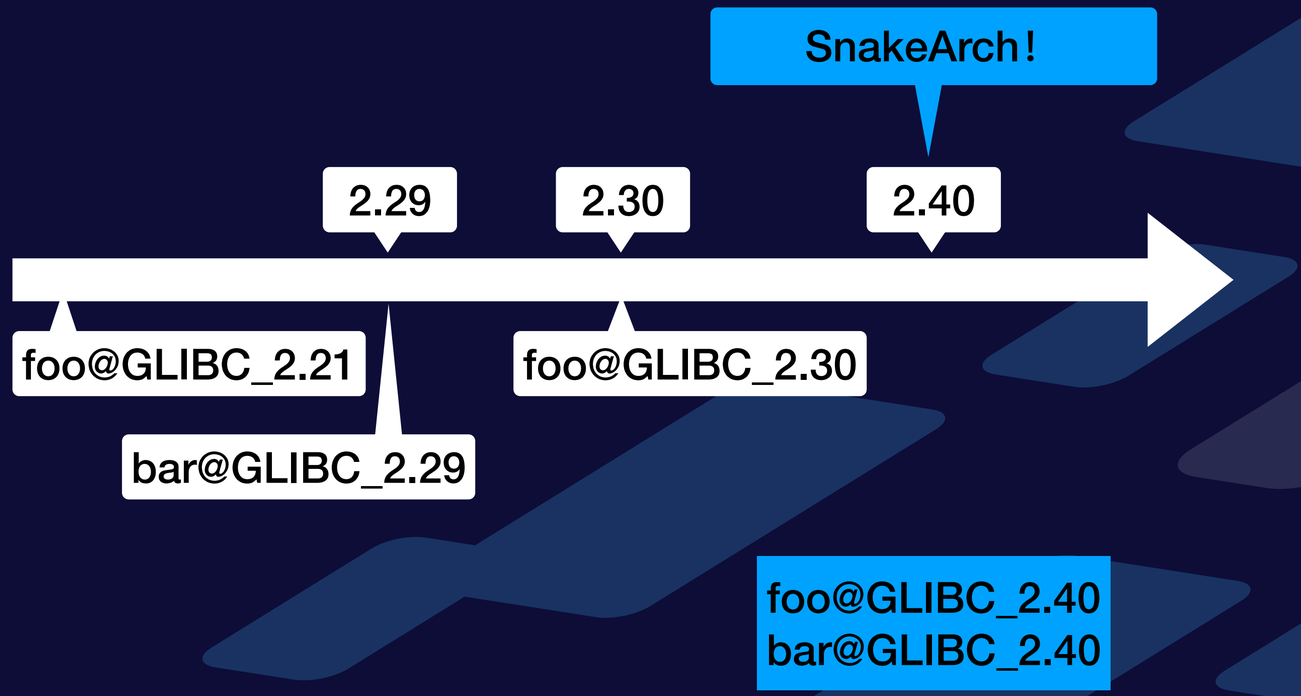
glibc 符号版本

- glibc 新引进架构时
  - 在该架构下的所有符号的版本确定为引进架构的版本
  - 此前更早的符号版本不再编译
- Loongarch 引入时 glibc 版本?
  - 华生，你发现了盲点!



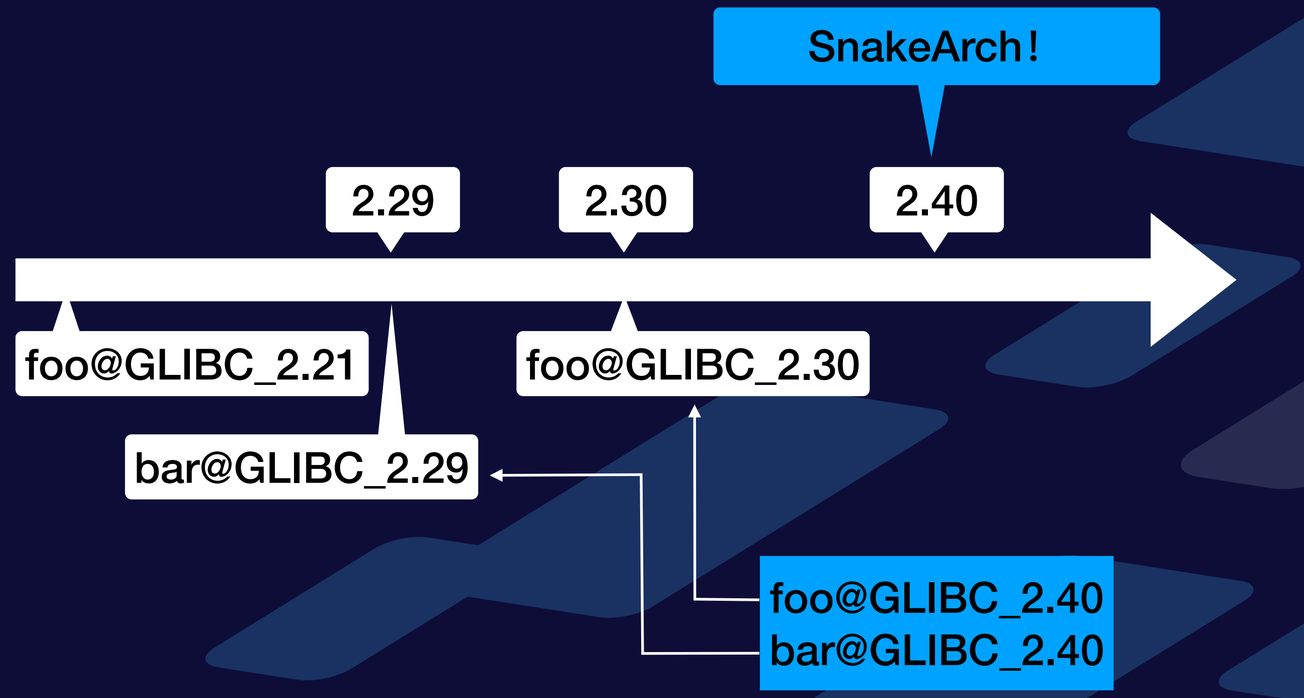
glibc 符号版本

- glibc 新引进架构时
  - 在该架构下的所有符号的版本确定为引进架构的版本
  - 此前更早的符号版本不再编译
- Loongarch 引入时 glibc 版本?
  - 华生，你发现了盲点!



glibc 符号版本

- glibc 新引进架构时
  - 在该架构下的所有符号的版本确定为引进架构的版本
  - 此前更早的符号版本不再编译
- Loongarch 引入时 glibc 版本?
  - 华生，你发现了盲点!



龙和龙
我们不一样

- glibc 引入版本：
  - 旧世界：2.27
  - 新世界：2.36
  - risc-v: 2.27

龙和龙

我们不一样

	旧世界	新世界
	libc.so.6: open@GLIBC_2.27 close@GLIBC_2.27	libc.so.6: open@GLIBC_2.36 close@GLIBC_2.36

hello_foo.c:
open();
libbar();
close();
	hello_foo:
	open@GLIBC_2.27
	libbar
	close@GLIBC_2.27
		hello_foo:
		open@GLIBC_2.36
		libbar
		close@GLIBC_2.36

libbar.c:
open();
close();
	libbar.so:
	open@GLIBC_2.27
	close@GLIBC_2.27
		libbar.so:
		open@GLIBC_2.36
		close@GLIBC_2.36

龙和龙

我们不一样

旧世界

新世界

libc.so.6:
open@GLIBC_2.27
close@GLIBC_2.27

libc.so.6:
open@GLIBC_2.36
close@GLIBC_2.36

hello_foo.c:
open();
libbar();
close();

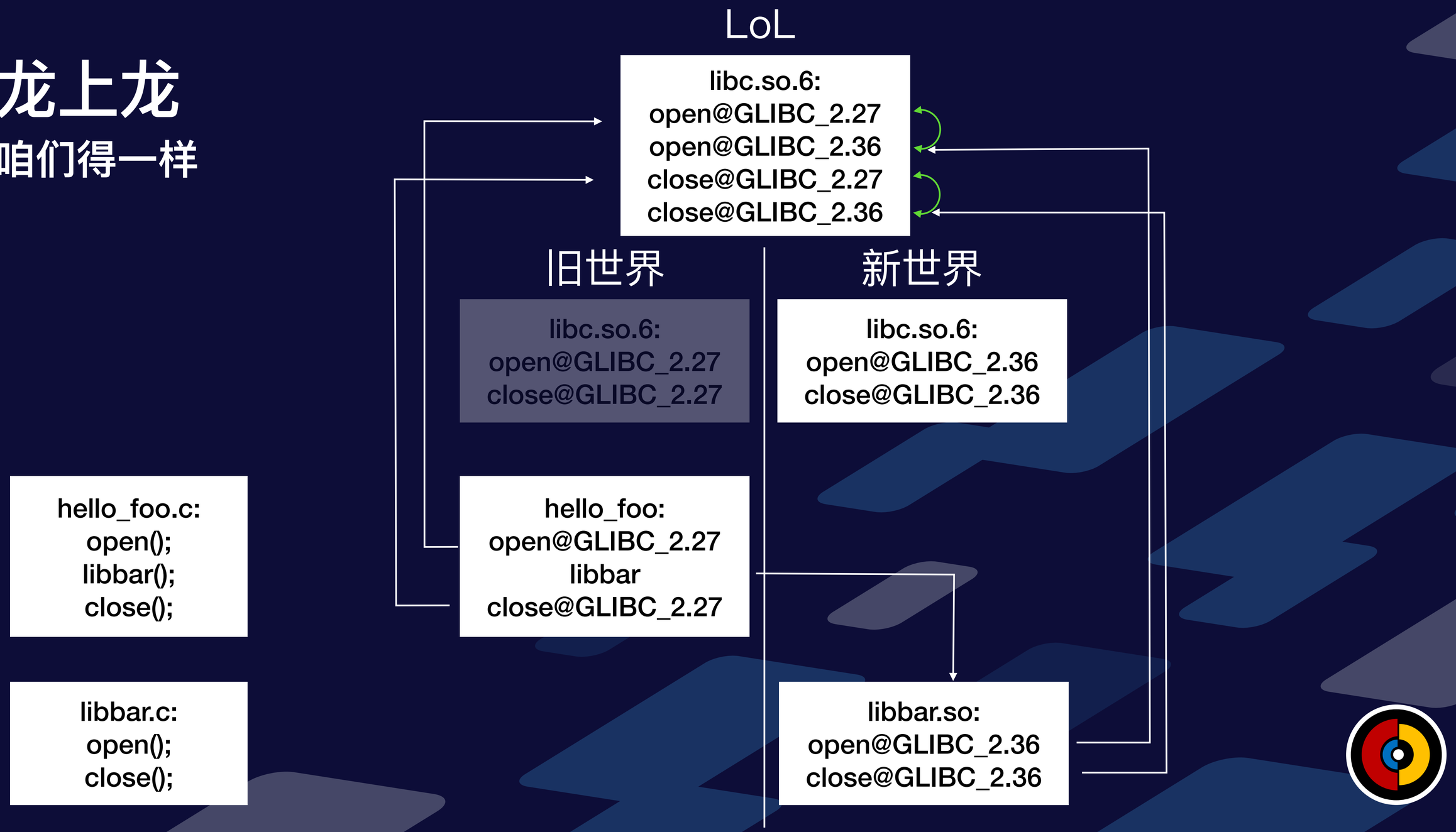
hello_foo:
open@GLIBC_2.27
libbar
close@GLIBC_2.27

libbar.c:
open();
close();

libbar.so:
open@GLIBC_2.36
close@GLIBC_2.36

龙上龙

咱们得一样


LoL
libc.so.6:
open@GLIBC_2.27
open@GLIBC_2.36
close@GLIBC_2.27
close@GLIBC_2.36
旧世界
新世界
libc.so.6:
open@GLIBC_2.27
close@GLIBC_2.27
libc.so.6:
open@GLIBC_2.36
close@GLIBC_2.36
hello_foo.c:
open();
libbar();
close();
hello_foo:
open@GLIBC_2.27
libbar
close@GLIBC_2.27
libbar.c:
open();
close();
libbar.so:
open@GLIBC_2.36
close@GLIBC_2.36

龙上龙
How to…

- 可否操作连接脚本?
  - 不行
- 可否修改一下 glibc 里面的宏?
  - 不行
- 可否所有符号逐一添加两个版本?
  - 可以！但是修改量太大

龙上龙
So...

- 使用 patchelf 后处理！
  - patchelf 似乎也不支持修改符号表
    - 我们可以 patch patchelf
- 编译流程
  - 修改新世界 glibc 的龙架构引入版本，降低至 2.27
  - 编译
  - 将所有 2.36 前的符号复制一份，修改版本号为 2.36

龙上龙
兼容性处理

- 行为有差异的符号：
  - 信号 （NSIG）相关、ucontext_t 相关
    - 旧版本（2.27）按旧世界行为；新版本（2.36）按新世界行为
  - sigaction
    - 假定使用旧版本符号注册的处理函数接收旧世界上下文；新版本符号注册的处理函数接收新世界上下文
    - 事实上：多数处理函数不处理上下文（第三参数）

Miao Wang in AOSC 社区频道

liblol_0.1.0-0~pre1_loongarch64.deb
979.5 KB

t.me/aosc_main/776978 Jan 14 at 17:54

Miao Wang in AOSC 社区频道

@JeffBai 求帮测试

t.me/aosc_main/776979 Jan 14 at 17:54

Miao Wang in AOSC 社区频道

装上有惊喜

t.me/aosc_main/776982 Jan 14 at 17:55

Cyano 🐏 in AOSC 社区频道

979KB?

t.me/aosc_main/776984 Jan 14 at 17:55

双倍多多🏀 | 冰utils in AOSC 社区频道

是不是共享系统 libc 的方案来了

t.me/aosc_main/776985 Jan 14 at 17:55

Miao Wang in AOSC 社区频道

> 双倍多多🏀 | 冰utils
> 是不是共享系统 libc 的方案来了

是的

t.me/aosc_main/776989 Jan 14 at 17:57

Miao Wang in AOSC 社区频道

第三世界凑合对付好了

t.me/aosc_main/776990 Jan 14 at 17:57

双倍多多🏀 | 冰utils in AOSC 社区频道

> Miao Wang
> 是的

看上去你现在 patch 了 glibc？

t.me/aosc_main/776992 Jan 14 at 17:57

Miao Wang in AOSC 社区频道

> 双倍多多🏀 | 冰utils
> 看上去你现在 patch 了 glibc？

是

t.me/aosc_main/776995 Jan 14 at 17:57

Miao Wang in AOSC 社区频道

现在只有 libc

t.me/aosc_main/776996 Jan 14 at 17:57

Miao Wang in AOSC 社区频道

肝了 21 个小时肝出来的

t.me/aosc_main/777000 Jan 14 at 17:58

新旧世界究竟有何差别?

新旧世界究竟有何差别?

- 内核态
  - 新世界缺少的系统调用
  - 用户态上下文对象
  - 信号数目（64 vs 128）
- 用户态
  - dynamic interpreter
  - glibc 符号版本
  - 信号数目
  - 用户态上下文对象

一些思考

- 旧世界商业应用的打包质量
  - 乱
- libLoL 是否已经完美兼容了旧世界应用
  - 多样性很贫乏，主要是基于 chromium 的 electron 打包
- libLoL 的现状和未来
  - 已经基本稳定，希望不会有未来

Acknowledgements

- 感谢龙芯公司
  - 带来的龙架构和龙芯处理器
- 感谢社区力量
- 感谢 AOSC 的小伙伴
- 感谢所有关注的用户

Q & A