关于 `cp` 时发现的的离奇 bug

背景

我在按照 nextcloud 中的文档使用 qemu 启动 linux kernel，其中编译 busybox 部分步骤如下：

> 生成文件在`./_install/bin`目录下，查看busybox是否为riscv格式使用以下指令：
>
> ```
> file ./_install/bin/busybox
> ```
>
> 如出现以下输出，则编译正确：
>
> ```
> ./_install/bin/busybox: ELF 64-bit LSB executable, UCB RISC-V, RVC, double-float ABI, version 1 (SYSV), statically linked, BuildID[sha1]=461ff297e613b1624b8edb3ea22cf9474dfdeae7, for GNU/Linux 4.15.0, stripped
> ```
>
> 拷贝`busybox`到工作目录：
>
> ```
> cp _install/bin/busybox ../linux-riscv/
> ```

我在启动 kernel 的时候一直因为 init 的问题 panic：

```
 ......
[    3.078232] Kernel panic - not syncing: No working init found.  Try passing init= option to kernel. See Linux Documentation/admin-guide/init.rst for guidance.
[    3.080163] CPU: 3 UID: 0 PID: 1 Comm: swapper/0 Not tainted 6.12.0-gadc218676eef #7
[    3.081094] Hardware name: riscv-virtio,qemu (DT)
[    3.081774] Call Trace:
[    3.082422] [<ffffffff8000628a>] dump_backtrace+0x1c/0x24
[    3.084245] [<ffffffff80961428>] show_stack+0x2c/0x38
[    3.084677] [<ffffffff8096df70>] dump_stack_lvl+0x50/0x72
[    3.085093] [<ffffffff8096dfa6>] dump_stack+0x14/0x1c
[    3.085499] [<ffffffff809619ae>] panic+0x108/0x304
[    3.085872] [<ffffffff8096fb18>] _cpu_down+0x0/0x3f0
[    3.086258] [<ffffffff80977ee2>] ret_from_fork+0xe/0x18
[    3.087027] SMP: stopping secondary CPUs
[    3.089681] ---[ end Kernel panic - not syncing: No working init found.  Try passing init= option to kernel. See Linux Documentation/admin-guide/init.rst for guidance. ]---
```

debug 过程

排查了一段时间之后，我发现我文件系统中的 busybox 架构居然不对：

```
sazikk@ubuntu2:~/linux-riscv/initramfs/bin$ file ../../../busybox/_install/bin/busybox
../../../busybox/_install/bin/busybox: ELF 64-bit LSB executable, UCB RISC-V, RVC, double-float ABI, version 1 (SYSV), statically linked, BuildID[sha1]=18a85b59e40b79594882b7f3ba40d31f0942b7b0, for GNU/Linux 4.15.0, stripped
```

```
sazikk@ubuntu2:~/linux-riscv/initramfs/bin$ file ./busybox
./busybox: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), for GNU/Linux 3.2.0, BuildID[sha1]=a5b
db209387e06cba305d4d5db76c52b7cb6ea26, dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, no sec
tion header
```

可以看到编译出来的是 RISC-V 的，工作目录里变成 x86 了！

问题是我严格遵守了文档的每一步，以及我从来没有编译过 x86 的版本，那这个 x86 的busybox 是哪里来的?

控制了一下变量发现，好好的编译出来的的 RISC-V busybox，`cp` 过来之后变成 x86 busybox 了！！

```
sazikk@ubuntu2:~/busybox$ file ./_install/bin/busybox
./_install/bin/busybox: ELF 64-bit LSB executable, UCB RISC-V, RVC, double-float ABI, version 1 (SYSV), st
atically linked, BuildID[sha1]=18a85b59e40b7959488[illegible]42b7b0, for GNU/Linux 4.15.0, stripped
sazikk@ubuntu2:~/busybox$ cp _install/bin/busybox ../linux-riscv/
sazikk@ubuntu2:~/busybox$ cd ..
sazikk@ubuntu2:~$ cd linux-riscv/
sazikk@ubuntu2:~/linux-riscv$ ls
build  busybox  initramfs  kernel-6.12  kernel-64k  Makefile
sazikk@ubuntu2:~/linux-riscv$ cd ..
sazikk@ubuntu2:~$ cd linux-riscv/
sazikk@ubuntu2:~/linux-riscv$ cd build/
sazikk@ubuntu2:~/linux-riscv/build$ cd ..
sazikk@ubuntu2:~/linux-riscv$ cd initramfs/
sazikk@ubuntu2:~/linux-riscv/initramfs$ mv ../busybox ./bin
sazikk@ubuntu2:~/linux-riscv/initramfs$ file ./bin/
./bin/: directory
sazikk@ubuntu2:~/linux-riscv/initramfs$ file ./bin/busybox
./bin/busybox: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), for GNU/Linux 3.2.0, BuildID[sha1]
=a5bdb209387e06cba305d4d5db76c52b7cb6ea26, dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, no
 section header
```

询问了 claude，得知可能是因为 `$PATH` 里有 `bin/busybox`，导致复制了错误的 busybox 版本

看了一眼，确实

```
.0, stripped
sazikk@ubuntu2:~/linux-riscv/initramfs/bin$ which busybox
/usr/bin/busybox
```

按照 claude 的方法，我重新复制了一份 busybox：

> 这个问题很有意思！这是因为你的系统中可能存在 `/bin/busybox` 的符号链接或者 PATH 中有其他 busybox，导致 cp 命令实际复制了错误的文件。
>
> 让我们用以下方法来确保复制正确的文件：
>
> 1. 使用绝对路径复制：
>
> ```
> cp -L ../../../busybox/_install/bin/busybox ./busybox
> ```

```
sazikk@ubuntu2:~/linux-riscv/initramfs/bin$ rm busybox
sazikk@ubuntu2:~/linux-riscv/initramfs/bin$ cp -L ../../../busybox/_install/bin/busybox ./busybox
sazikk@ubuntu2:~/linux-riscv/initramfs/bin$ file ./busybox
./busybox: ELF 64-bit LSB executable, UCB RISC-V, RVC, double-float ABI, version 1 (SYSV), statically link
ed, BuildID[sha1]=18a85b59e40b79594882b7f3ba40d31f0942b7b0, for GNU/Linux 4.15.0, stripped
sazikk@ubuntu2:~/linux-riscv/initramfs/bin$ ls
```

这回对了（此处伏笔），但是问题来了，为什么 -L 以及指定目标文件名字可以防止这种错误?

查了一下 -L 和 -P 参数（来自man）：

```
-L, --dereference
      always follow symbolic links in SOURCE
-P, --no-dereference
      never follow symbolic links in SOURCE
```

？？？，-L 指定使用符号链接不是反而会强制复制 `/usr/bin/busybox` 吗，理论上使用 -P 才会避免复制错

于是我测试了不同的指令，得到如下结果，均使用 `file ./busybox` 测试，以排除环境变量中 busybox，

测试流程：

```
rm busybox
cp xxxxxxx
ls
file ./busybox
```

结果如下：

命令	是否正确
`cp ../../../busybox/_install/bin/busybox .`	复制错误
`cp ../../../busybox/_install/bin/busybox ./busybox`	复制错误
`cp -L ../../../busybox/_install/bin/busybox .`	复制错误
`cp -L ../../../busybox/_install/bin/busybox ./busybox`	复制错误
`cp -P ../../../busybox/_install/bin/busybox .`	复制错误
`cp -P ../../../busybox/_install/bin/busybox ./busybox`	复制错误

？？？布什戈门，刚刚不是成功了吗

对比上下文，显然刚刚错误现在失败的关键在于多了一次 `ls`，我认为这应该是某种文件系统缓存或者某种类COW机制导致的，在 cp 时，实际上只是复制了一个源文件的链接，因此 `file ./busybox` 实际上是解析了源文件，结果自然正确，而 `ls` 之后，需要读当前文件夹下的文件，实际的复制发生，从而复制了错误的文件过来。值得注意的是，`ls` 的时候我会卡一下，这也印证了我的猜测。

我 strace 了一下 cp，不过输出的系统调用较为繁杂，而且涉及很多文件系统相关，没看懂（。因此我还是按照执行时间判断文件复制具体发生在哪一步，cp 命令极快而第一次 ls 会卡一下，所以我认为实际复制还是发生在了 ls 时

尝试了使用 mv，直接把源文件挪过去用，很不幸，也变成 x86 架构了

解决方案

使用 sftp，传回主机再传进目标文件夹

这里测试的时候发现哪怕把 /usr/bin/busybox 移除，which busybox 都找不到了，cp 还是会复制成那个 x86 的 busybox，麻了。

结论

到这一步仍然存在一堆疑问：

1. 究竟怎样 cp 才能在这种情况下正确做到复制
2. 为什么 x86 的 busybox 被我挪走了甚至删除了，cp 还是 cp 了x86 的 busybox（应该是某种文件系统缓存），以及为什么 mv 也受影响

3. 什么情况下 cp 会立刻复制，什么情况下会延迟实际复制

由于我不熟悉 cp 的底层机制，因此这些问题可能暂时无法得到解答，大家以我为戒。