

Linux Lab 龙芯实验手册 V0.2

泰晓科技 | TinyLab.org

September 12, 2020

目录

1. 手册说明	4
2. 名词解释	5
2.1 Linux 简介	5
2.2 龙芯处理器简介	5
2.2.1 龙芯处理器发展历史	5
2.2.2 Lemote 简介	6
2.3 Linux Lab 简介	7
3. Linux Lab 安装	9
3.1 软硬件要求	9
3.2 安装 Docker	9
3.3 选择工作目录	10
3.4 使用普通用户	11
3.5 下载实验环境	11
3.6 运行并登陆 Linux Lab	12
3.7 更新实验环境并重新运行	13
3.8 快速上手：启动一个开发板	13
4. Linux Lab 入门	14
4.1 选择开发板并下载板级资源	14
4.2 配置开发板	17
4.3 引导编译好的内核和文件系统	19
4.4 下载内核源代码	26
4.5 配置内核	27
4.6 修改内核	28
4.7 编译内核	28
4.8 引导新编译的内核	30

5. 用 Linux Lab 进行龙芯 Linux 内核开发	31
5.1 内核模块开发实验	32
5.1.1 撰写并运行内核模块	32
5.1.2 自动测试内核模块	33
5.2 内核开发实验	35
5.2.1 添加新的内核压缩算法	35
5.2.2 为内核添加新的系统调用	36
5.2.3 内核函数跟踪辅助工具	38
5.2.4 内核自动化调试	40
5.2.5 内核自动化测试	40
5.3 文件系统实验	42
5.3.1 更换文件系统格式	42
5.3.2 通过 Buildroot 构建根文件系统	42
5.3.3 通过 ls3a7a 启动 Debian 根文件系统	43
5.4 龙芯汇编语言实验	46
5.4.1 Hello World	46
5.4.2 memcpy & memset	47
5.5 龙芯 Qemu 实验	49
5.5.1 编译龙芯 Qemu 系统模拟器：qemu-system-mips64el	49
5.5.2 增大龙芯 Qemu 中传递内核参数的 Buffer	51
5.5.3 编译龙芯 Qemu 指令集翻译工具：qemu-mips64el	52
5.6 其他实验	53
5.6.1 使用龙芯中科提供的工具链	53
5.6.2 在主机和 Qemu 系统之间传输文件	54
5.6.3 用来开展《用“芯”探核》实验	55
6. 参考资料	57
6.1 Linux Lab 相关	57
6.2 龙芯相关	57
6.3 Linux 相关	57

1. 手册说明

版权声明

本手册由 **泰晓科技** 技术社区和相关贡献者联合撰写，手册主体基于 **Linux Lab** 开源项目中
文手册。

该手册遵守 Linux Lab 开源项目所采用的版权协议，采用 GPL v2 协议发布。未经授权，禁
止商业使用。

免责声明

该手册仅提供阶段性信息，所含内容可根据相关软硬件的实际情况随时更新，恕不另行通知。
如因手册使用不当造成的直接或间接损失，请自行承担。

泰晓科技

泰晓科技 技术社区创建于 2010 年，长期坚持 Linux 技术内容原创、交流与分享，有发起和
维护数个开源软件项目。

- 社区理念：聚焦 Linux —— 追本溯源，见微知著！
- 社区目标：致力于推动业界创造更极致的 Linux 产品。
- 微信公号：泰晓科技
- 联系微信：tinylab
- 联系邮箱：contact@tinylab.org

欢迎大家通过添加微信号 *tinylab* 为好友进入“Linux Lab 用户微信群”，一起交流和增补相
关的实验案例。也欢迎大家扫描下面的二维码关注“泰晓科技”公众号，及时了解 *Linux Lab* 项
目动态。



Figure 1: 泰晓科技 —— 微信公众号

2. 名词解释

2.1 Linux 简介

从狭义的角度看，Linux 是一套操作系统内核，由 Linus Torvalds 于 1991 年开发并发布。从广义的角度来看，Linux 代指所有基于 Linux 内核的操作系统，由于 Linux 发行版的核心软件主要源自 GNU 项目，所以这类系统又叫 GNU/Linux 系统。而发展到现在，除了传统的 Linux 发行版，也出现了面向各类小型终端的 Android/Linux 系统，而且规模上远远超过传统的 GNU/Linux 系统。

Linux 是一个开放源代码项目，到明年将满 30 周年。接近 30 年间，软件版本由当初的 v0.01 迭代到如今的 v5.8；代码规模由 Linux 0.11 的 1 万行左右，到 v5.8 时已接近 2000 万行；参与的工程师数量由最早的 1 人、数人到如今每个版本的贡献人数超过 1000 人；当前每个版本参与协作的企业达到 200 家左右；Linux 的应用领域也无比广泛，从航天器、汽车、服务器、台式机、手机、智能手表到机器人、儿童玩具，到处都有“Linux Inside”。Linux 是几乎所有重要领域最主流的操作系统之一。

- 官方网站：<https://www.kernel.org/>
- 代码仓库：<https://git.kernel.org/>



Figure 2: Linux 吉祥物

2.2 龙芯处理器简介

2.2.1 龙芯处理器发展历史

龙芯处理器最早由中科院计算所龙芯课题组设计与开发。

来自 [龙芯中科技术有限公司网站](#) 的数据显示，龙芯主要经历了两个发展阶段，一个是产品研发阶段、一个是产业化阶段。

产品研发阶段（2001-2009）：

- 2002 年成功流片我国首款通用处理器“龙芯 1 号”
- 2003 年成功流片我国首款 64 位通用处理器“龙芯 2B”
- 2007 年成功流片龙芯第一款产品级芯片“龙芯 2F”
- 2009 年成功流片首款四核处理器“龙芯 3A”

产业化阶段（2010-）

- 2010 年成立 [龙芯中科](#) 技术有限公司，龙芯正式从研发走向产业化。
- 2012 年成功流片八核 32 纳米龙芯 3B1500
- 2015 年发布龙芯新一代高性能处理器架构 GS464E
- 2015 年龙芯第二代高性能处理器产品龙芯 3A2000/3B2000 实现量产并推广应用
- 2017 年龙芯 3A3000/3B3000 实现量产并推广应用
- 2017 年成功流片龙芯 7A 桥片
- 2019 年成功推出第三代处理器产品 3A4000/3B4000

3A4000 主频提升到 1.8G-2G，支持 DDR4-2400，并通过配套桥片 7A1000 提供了 SATA II, PCIe 2.0, RGMII 千兆网等接口，结合 UOS、麒麟、Loongnix、Fedora、Debian 等系统，已然满足常规的办公场景。



Figure 3: 龙芯 3A4000 处理器

2.2.2 Lemote 简介

在龙芯的发展过程中，于 2004 年在常熟成立的 [龙芯产业化基地（Lemote）](#) 对龙芯桌面市场、技术和社区生态建设等方面做了很多前瞻性的探索和实践，为后来的进一步产业化打下了良好的基础。

比如，在社区生态建设方面，自由软件运动先驱 Richard Stallman 使用了很长时间的 YeeLoong 笔记本就由 Lemote 设计、生产和制造，其中运行的 Linux 内核就由 Lemote 开发，并由当年在 Lemote 实习的 Linux Lab 作者吴章金往官方 Linux 社区提交。实习之后，吴章金也利用业余时间和龙芯 Debian 早期维护者刘世伟、龙芯 Fulong2e Qemu 开发者周亚金博士一起发起了“龙芯 Linux 社区”并和多位来自全球各地的贡献者（例如 Alexandre Oliva、Zhang Le 等）一起持续免费维护了多年 [社区版龙芯 Linux 内核](#)。



Figure 4: Richard Stallman & Lemote YeeLoong Laptop

其后的龙芯 2K、3A 系列处理器的官方 Linux 内核支持也由 Lemote 主导，由后文提到的陈华才博士、杨嘉勋等同学提交和维护。

2.3 Linux Lab 简介

[Linux Lab](#) 提供了一套 Linux 内核即时开发环境，于 2016 年由吴章金（“泰晓科技”技术社区创始人）创建。该项目最早源自 2010 年左右的 [TinyLinux](#) 项目，最初目的是为了能够在多个架构上快速验证 [TinyLinux](#) 项目下的成果。

[Linux Lab](#) 也是一个开放源代码项目，于 2016 年发布到 [Github](#)，于 2019 年发布了第一个 [v0.1](#) 正式版，目前（2020 年 8 月）已经迭代到 [v0.5 rc2](#)。

当前版本已经支持 7 大处理器架构（X86、ARM、MIPS、PPC 以及 RISC-V、Loongson、CSKY），累计已支持 16 款开发板。

Linux Lab 集成了 Linux 内核开发必备的编辑器、编译器、调试器、模拟器、根文件系统等，可以用来高效地学习处理器架构、Linux 内核、嵌入式 Linux 系统、C 语言编程、Linux 汇编、Shell 编程等。另外，Linux Lab 也已经支持 Uboot、Bulldroot、Qemu 等嵌入式必备软件的编译和开发。

Linux Lab 作者吴章金是龙芯 Linux 内核的早期开发者和贡献者，于 2010 年左右往官方 Linux 社区提交了龙芯 2F 系列硬件支持以及其他诸多 MIPS 架构相关的基础特性，比如 MIPS Ftrace、MIPS 内核压缩、MIPS 实时抢占补丁等。

基于对龙芯处理器的长期关注和支持，Linux Lab 作者于 2019 年中促成龙芯开放全系处理器的 Qemu 模拟器源代码，并协同龙芯实验室开发了 Linux Lab 的龙芯开发板插件。通过最近一年的开发和迭代，双方通力合作，把龙芯支持进一步完善并合并到了 Linux Lab v0.5 版主线。

该文档旨在充分介绍和展示这一开发成果，让更多的开发者更容易上手国产龙芯 Linux 内核和系统的开发，进而不断壮大国产龙芯处理器的技术阵营。

- 项目主页：<http://tinylab.org/linux-lab>
- 代码仓库：
 - <http://gitee.com/tinylab/linux-lab>
 - <http://github.com/tinyclub/linux-lab>

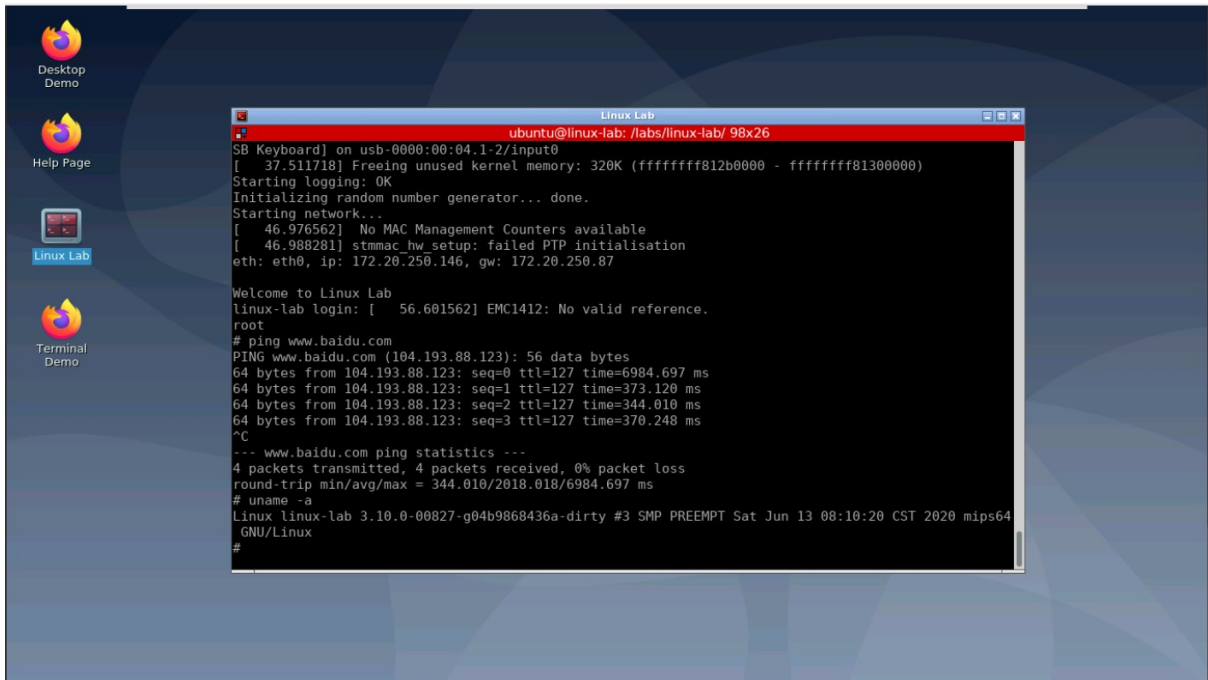


Figure 5: Linux Lab 运行龙芯 3A7A 开发板

3. Linux Lab 安装

3.1 软硬件要求

Linux Lab 是一套完备的 Linux 内核和嵌入式 Linux 系统开发环境，需要预留足够的算力和存储空间。

为了防止后续算力和存储空间不足，基本硬件配置建议如下：

硬件类型	要求	说明
处理器	X86_64, > 1.5GHz	创建虚拟机时也务必选择 64 位 X86 处理器
磁盘	>= 50G	系统 (25G), Docker 镜像 (~5G), Linux Lab(20G)
内存	>= 4G	过低的内存可能会导致各种卡顿以及异常缓慢

如果平时用的几率比较高，建议把磁盘空间提高到 100G ~ 200G 以上，内存可以提升到 8G 以上。

另外，Linux Lab 用户在持续分享自己 [成功运行过 Linux Lab 的系统信息](#)，截止到 2020 年 8 月 9 日，汇总如下，以便参考：

操作系统	系统版本	Docker 版本	内核版本
Ubuntu	16.04, 18.04, 20.04	18.09.4	Linux 4.15, 5.0, 5.3, 5.4
Debian	bullseye	19.03.7	Linux 5.4.42
Arch Linux		19.03.11	Linux 5.4.50, 5.7.4
CentOS	7.6, 7.7	19.03.8	Linux 3.10, 5.2.9
Deepin	15.11	18.09.6	Linux 4.15
Mac OS X	10.15.5	19.03.8	Darwin 19.5.0
Windows	10 PRO, WSL2	19.03.8	MINGW64_NT-10.0-17134

综上，Linux Lab 支持各大主流 Linux 发行版，还支持 Mac OS X 和较新版本的 Windows 10 PRO 以及 WSL2。由于 Windows 10 对 Docker 的支持还不尽完善，建议 Windows 用户优先通过虚拟机 + Ubuntu 的方式来运行 Linux Lab。

3.2 安装 Docker

运行 Linux Lab 需要基于 Docker，所以请务必先安装 Docker：

- Linux, Mac OS X, Windows 10 PRO

- 建议使用 [Docker CE](#)
- 更早的 Windows 版本（含大部分老版本 Windows 10）
 - 建议通过 Virtualbox 或 Vmware 安装 Ubuntu 后使用

在运行 Linux Lab 之前，请确保无需 sudo 权限也可以通过普通用户 <USER> 正常运行 docker：

```
1 $ sudo usermod -aG docker <USER>
2 $ newgrp docker
3 $ docker run hello-world
```

另外，在国内要正常使用 Docker，请务必配置好国内的 Docker 镜像加速服务：

- [阿里云 Docker 镜像使用文档](#)
- [USTC Docker 镜像使用文档](#)

Linux 发行版用户须知：

- 请参考安装手册 [doc/install](#)

Windows 用户须知：

- 请参考 [Docker 官方文档](#) 确保所用 Windows 版本支持 Docker
- Linux Lab 当前仅在 Git Bash 验证过，请务必配合 Git Bash 使用
 - 在安装完 [Git For Windows](#) 后，可通过鼠标右键使用“Git Bash Here”

3.3 选择工作目录

如果您是通过 Docker Toolbox 安装的，请在 Virtualbox 上进入 default 系统的 /mnt/sda1 目录。否则，关机后所有数据会丢失，因为缺省的 /root 目录是挂载在内存中的。

```
1 $ cd /mnt/sda1
```

对于 Linux 用户，可以简单地在 ~/Downloads 或 ~/Documents 下选择一个工作路径。

```
1 $ cd ~/Documents
```

对于 Windows 和 Mac OS X 用户，要正常编译 Linux，请先使能或创建一个能区分大小写的文件系统作为工作空间：

Windows：

```
1 (git bash) $ cd /path/to/cloud-lab
2 (git bash) $ fsutil file SetCaseSensitiveInfo ./ enable
```

Mac OS X:

```
1 $ hdiutil create -type SPARSE -size 60g -fs "Case-sensitive Journaled HFS+" -volname
   labspace labspace.dmg
2 $ hdiutil attach -mountpoint ~/Documents/labspace -nobrowse labspace.dmg.sparseimage
3 $ cd ~/Documents/labspace
```

对于 Windows 用户，在安装完 [Git For Windows](#) 后，可通过鼠标右键在选定的工作目录运行“Git Bash Here”。

3.4 使用普通用户

下载代码前，请**务必**切换到普通用户。Linux Lab 虽未禁用 root 帐号，但是不推荐使用 root 帐号，否则会有各种权限异常问题。

查看当前用户 ID，0 表示 root，非零表示普通用户：

```
1 $ id -u `whoami`
2 1000
```

如果当前为 root，需切换到普通用户，请替换 <USER> 为自己的帐号名，下同：

```
1 # id -u `whoami`
2 0
3 # sudo -su <USER>
```

如果目标机器上仅有 root 帐号，则**必须**新建一个普通用户帐号，假设取名为 laber：

```
1 $ sudo useradd --create-home --shell /bin/bash --user-group --groups adm,sudo laber
2 $ sudo passwd laber
3 $ sudo -su laber
4 $ whoami
5 laber
```

3.5 下载实验环境

下载 Cloud Lab，然后再选择 linux-lab 仓库：

```
1 $ git clone https://gitee.com/tinylab/cloud-lab.git
2 $ cd cloud-lab/ && tools/docker/choose linux-lab
```

如果错误使用了 root 帐号来 clone 代码，下载后请**务必**切换到普通用户，并把属主改为普通用户：

```
1 $ sudo -su <USER>
2 $ sudo chown -R <USER>:<USER> -R cloud-lab/{*,.git}
```

3.6 运行并登陆 Linux Lab

启动 Linux Lab 并根据控制台上打印的用户名和密码登录实验环境：

```
1 $ tools/docker/run linux-lab
```

通过 Bash 直接登陆：

```
1 $ tools/docker/bash
```

通过 Web 浏览器登录：

```
1 $ tools/docker/webvnc
```

其他登录方式：

```
1 $ tools/docker/vnc
2 $ tools/docker/ssh
3 $ tools/docker/webssh
```

选择某种登陆方式：

```
1 $ tools/docker/login list # 列出并选择，并且记住
2 $ tools/docker/login vnc # 直接选择一种并记住
```

登录方式汇总：

登录方法	描述	缺省用户	登录所在地
bash	docker bash	ubuntu	本地主机
ssh	普通 ssh	ubuntu	本地主机
vnc	普通桌面	ubuntu	本地主机 + VNC client
webvnc	web 桌面	ubuntu	互联网在线即可
webssh	web ssh	ubuntu	互联网在线即可

由于普通的 vnc 客户端五花八门，所以当前建议采用 webvnc，确保可以在各个平台能自动登陆。

如果想使用本地的 vnc 客户端，请先提前安装好客户端，Linux Lab 推荐使用 vinagre。其他的客户端请通过如下方式指定：

```
1 $ tools/docker/vnc vinagre
```

如果上述命令不能正常工作，请根据上述命令打印出来的 VNC 服务器信息，自行配置所用客户端。

3.7 更新实验环境并重新运行

为了更新 Linux Lab 的版本，首先 **必须** 备份所有的本地修改，比如固化容器：

```
1 $ tools/docker/commit linux-lab
2 $ git checkout -- configs/linux-lab/docker/name
```

然后就可以执行更新了：

```
1 $ tools/docker/update linux-lab
```

如果更新失败，请尝试清理当前运行的容器：

```
1 $ tools/docker/rm-all
```

如果有必要的话清理整个环境：

```
1 $ tools/docker/clean-all
```

之后重新运行 Linux Lab：

```
1 $ tools/docker/rerun linux-lab
```

3.8 快速上手：启动一个开发板

进入实验环境，切换目录：

```
1 $ cd /labs/linux-lab
```

输入如下命令，在缺省的 arm/vexpress-a9 开发板上启动预置的内核和根文件系统：

```
1 $ make boot
```

使用 root 帐号登录，不需要输入密码（密码为空），只需要输入 root 然后输入回车即可：

```
1 Welcome to Linux Lab
2
3 linux-lab login: root
4
5 # uname -a
6 Linux linux-lab 5.1.0 #3 SMP Thu May 30 08:44:37 UTC 2019 armv7l GNU/Linux
7 #
8 # poweroff
9 #
```

键入 poweroff 命令即可关闭板子。如果部分开发板的关机功能不完善，可通过 CTRL+a x（依次按下 CTRL 和 a，同时释放，再单独按下 x）来退出 Qemu。当然，也可以另开一个控制台，通过 kill 或 pkill 退出 Qemu 进程。

4. Linux Lab 入门

4.1 选择开发板并下载板级资源

Linux Lab 已经内置支持 16 款开发板，可以这样列出来：

```

1  $ make list
2  [ aarch64/raspi3 ]:
3      ARCH      = arm64
4      CPU       ?= cortex-a53
5      LINUX    ?= v5.1
6      ROOTDEV_LIST := /dev/mmcblk0 /dev/ram0
7      ROOTDEV ?= /dev/mmcblk0
8  [ aarch64/virt ]:
9      ARCH      = arm64
10     CPU       ?= cortex-a57
11     LINUX    ?= v5.1
12     ROOTDEV_LIST := /dev/sda /dev/vda /dev/ram0 /dev/nfs
13     ROOTDEV ?= /dev/vda
14 [ arm/mcimx6ul-evk ]:
15     ARCH      = arm
16     CPU       ?= cortex-a9
17     LINUX    ?= v5.4
18     ROOTDEV_LIST := /dev/mmcblk0 /dev/ram0 /dev/nfs
19     ROOTDEV ?= /dev/mmcblk0
20 [ arm/versatilepb ]:
21     ARCH      = arm
22     CPU       ?= arm926t
23     LINUX    ?= v5.1
24     ROOTDEV_LIST := /dev/sda /dev/ram0 /dev/nfs
25     ROOTDEV ?= /dev/ram0
26 [ arm/vexpress-a9 ]:
27     ARCH      = arm
28     CPU       ?= cortex-a9
29     LINUX    ?= v5.1
30     ROOTDEV_LIST := /dev/mmcblk0 /dev/ram0 /dev/nfs
31     ROOTDEV ?= /dev/ram0
32 [ i386/pc ]:
33     ARCH      = x86
34     CPU       ?= qemu32
35     LINUX    ?= v5.1
36     ROOTDEV_LIST ?= /dev/hda /dev/ram0 /dev/nfs
37     ROOTDEV_LIST[LINUX_v2.6.34.9] ?= /dev/sda /dev/ram0 /dev/nfs
38     ROOTDEV ?= /dev/hda
39 [ mips64el/ls2k ]:
40     ARCH      = mips
41     CPU       ?= mips64r2
42     LINUX    ?= loongnix-release-1903
43     LINUX[LINUX_loongnix-release-1903] := 04b98684
44     ROOTDEV_LIST := /dev/sda /dev/ram0 /dev/nfs
45     ROOTDEV ?= /dev/ram0
46 [ mips64el/ls3a7a ]:
47     ARCH      = mips
48     CPU       ?= mips64r2
49     LINUX    ?= loongnix-release-1903
50     LINUX[LINUX_loongnix-release-1903] := 04b98684
51     ROOTDEV_LIST ?= /dev/sda /dev/ram0 /dev/nfs
52     ROOTDEV ?= /dev/ram0
53 [ mipsel/ls1b ]:

```

```

54 ARCH = mips
55 CPU  ?= mips32r2
56 LINUX  ?= v5.2
57 ROOTDEV_LIST ?= /dev/ram0 /dev/nfs
58 ROOTDEV ?= /dev/ram0
59 [ mipsel/ls232 ]:
60 ARCH = mips
61 CPU  ?= mips32r2
62 LINUX  ?= v2.6.32-r190726
63 ROOTDEV_LIST := /dev/ram0 /dev/nfs
64 ROOTDEV ?= /dev/ram0
65 [ mipsel/malta ]:
66 ARCH = mips
67 CPU  ?= mips32r2
68 LINUX  ?= v5.1
69 ROOTDEV_LIST := /dev/hda /dev/ram0 /dev/nfs
70 ROOTDEV ?= /dev/ram0
71 [ ppc/g3beige ]:
72 ARCH = powerpc
73 CPU  ?= generic
74 LINUX  ?= v5.1
75 ROOTDEV_LIST := /dev/hda /dev/ram0 /dev/nfs
76 ROOTDEV ?= /dev/ram0
77 [ riscv32/virt ]:
78 ARCH = riscv
79 CPU  ?= any
80 LINUX  ?= v5.0.13
81 ROOTDEV_LIST := /dev/vda /dev/ram0 /dev/nfs
82 ROOTDEV ?= /dev/vda
83 [ riscv64/virt ]:
84 ARCH = riscv
85 CPU  ?= any
86 LINUX  ?= v5.1
87 ROOTDEV_LIST := /dev/vda /dev/ram0 /dev/nfs
88 ROOTDEV ?= /dev/vda
89 [ x86_64/pc ]:
90 ARCH = x86
91 CPU  ?= qemu64
92 LINUX  ?= v5.1
93 ROOTDEV_LIST := /dev/hda /dev/ram0 /dev/nfs
94 ROOTDEV_LIST[LINUX_v3.2] := /dev/sda /dev/ram0 /dev/nfs
95 ROOTDEV ?= /dev/ram0
96 [ csky/virt ]:
97 ARCH = csky
98 CPU  ?= ck810
99 LINUX  ?= v4.9.56
100 ROOTDEV ?= /dev/nfs

```

仅列出龙芯的板子：

```

1 $ make list FILTER=ls
2 [ mips64e1/ls2k ]:
3 ARCH = mips
4 CPU  ?= mips64r2
5 LINUX  ?= loongnix-release-1903
6 LINUX[LINUX_loongnix-release-1903] := 04b98684
7 ROOTDEV_LIST := /dev/sda /dev/ram0 /dev/nfs
8 ROOTDEV ?= /dev/ram0
9 [ mips64e1/ls3a7a ]:
10 ARCH = mips
11 CPU  ?= mips64r2

```

```

12     LINUX    ?= loongnix-release-1903
13     LINUX[LINUX_loongnix-release-1903] := 04b98684
14     ROOTDEV_LIST ?= /dev/sda /dev/ram0 /dev/nfs
15     ROOTDEV ?= /dev/ram0
16 [ mipsel/ls1b ]:
17     ARCH     = mips
18     CPU      ?= mips32r2
19     LINUX    ?= v5.2
20     ROOTDEV_LIST ?= /dev/ram0 /dev/nfs
21     ROOTDEV ?= /dev/ram0
22 [ mipsel/ls232 ]:
23     ARCH     = mips
24     CPU      ?= mips32r2
25     LINUX    ?= v2.6.32-r190726
26     ROOTDEV_LIST := /dev/ram0 /dev/nfs
27     ROOTDEV ?= /dev/ram0

```

选择某款开发板用于 Linux 内核开发，以龙芯 3A 为例：

```

1  $ make BOARD=mips64el/ls3a7a
2  [ mips64el/ls3a7a ]:
3     ARCH = mips
4     XARCH = mips64el
5     CPU = mips64r2
6     BUILDROOT = 2016.05
7     MEM = 1024M
8     SERIAL = ttyS0
9     NETDEV_LIST = synopgmac rtl8139
10    NETDEV = synopgmac
11    LINUX = loongnix-release-1903
12    LINUX[LINUX_loongnix-release-1903] = 04b98684
13    KERNEL_GIT[LINUX_loongnix-release-1903] = git://cgit.loongnix.org/kernel/linux-3.10.
    git
14    KERNEL_SRC[LINUX_loongnix-release-1903] = loongnix-linux-3.10
15    ORIIMG = vmlinuz
16    KIMAGE = /labs/linux-lab/output/mips64el/linux-loongnix-release-1903-ls3a7a/vmlinuz
17    DTS[LINUX_loongnix-release-1903] = loongnix-linux-3.10/arch/mips/loongson/loongson
    -3/dts/loongson3_ls7a.dts
18    DTB[LINUX_loongnix-release-1903] = /labs/linux-lab/boards/mips64el/ls3a7a/bsp/kernel
    /loongnix-release-1903/loongson3_ls7a.dtb
19    ROOTDEV_LIST = /dev/sda /dev/ram0 /dev/nfs
20    ROOTDEV = /dev/ram0
21    FSTYPE = ext2
22    ROOTFS = /labs/linux-lab/boards/mips64el/ls3a7a/bsp/root/2016.05/rootfs.cpio.gz
23    HROOTFS = /labs/linux-lab/boards/mips64el/ls3a7a/bsp/root/2016.05/rootfs.ext2
24    QEMU_GIT = https://gitee.com/loongsonlab/qemu
25    QEMU_SRC = loongsonlab-qemu
26    QEMU = loongson-v1.0
27    QTOOL = /labs/linux-lab/boards/mips64el/ls3a7a/bsp/qemu/loongson-v1.0/bin/qemu-
    system-mips64el
28    XOPTS = -device usb-mouse -device usb-kbd -show-cursor
29    XENVS = INITRD_OFFSET=0x04000000
30    XKCLI = maxcpus=1 nomsi nmsix
31    CCORI = internal

```

首次使用会自动下载当前所用板子的最新 BSP 代码仓库：

```

1  Downloading bsp source ...
2
3  Initialized empty Git repository in /labs/linux-lab/boards/mips64el/ls3a7a/bsp/.git/

```



```
4 remote: Enumerating objects: 140, done.
5 remote: Counting objects: 100% (140/140), done.
6 remote: Compressing objects: 100% (108/108), done.
7 remote: Total 140 (delta 26), reused 118 (delta 18), pack-reused 0
8 Receiving objects: 100% (140/140), 25.59 MiB | 1.89 MiB/s, done.
9 Resolving deltas: 100% (26/26), done.
10 From https://gitee.com/tinylab/qemu-mipsel-ls3a7a
11 * [new branch]      master    -> origin/master
12 Note: switching to 'FETCH_HEAD'.
```

bsp 仓库中包含了 Linux 内核与 Buildroot 配置文件、预编译好的内核版本、预编译好的文件系统等板级相关信息。

```
1 $ tree -L 3 boards/mips64el/ls3a7a/bsp
2 boards/mips64el/ls3a7a/bsp
3   |-- boot.sh
4   |-- configs
5   |-- buildroot_2016.05_defconfig
6   |-- linux_loongnix-release-1903_defconfig
7   |-- linux_v5.7_defconfig
8   |-- COPYING
9   |-- kernel
10  |-- loongnix-release-1903
11  |-- loongson3_ls7a.dtb
12  |-- vmlinux
13  |-- v5.7
14  |-- vmlinux
15  |-- patch
16  |-- linux
17  |-- loongnix-release-1903
18  |-- v5.7
19  |-- qemu
20  |-- loongson-v1.0
21  |-- bin
22  |-- libexec
23  |-- share
24  |-- README.md
25  |-- root
26  |-- 2016.05
27  |-- rootfs.cpio.gz
```

4.2 配置开发板

开发板的配置信息存放在 boards/ 目录下，例如 mips64el/ls3a7a 的板级配置为：

```
1 boards/mips64el/ls3a7a/Makefile
```

相关信息可以通过编辑器直接编辑修改，也可以通过如下方式调整：

```
1 // 本地方式
2 $ make local-edit
3
4 // 如果希望发送 Pull Request
5 $ make board-edit
```

本地方式不会直接修改板级配置文件，而是修改本地配置文件：

```
1 boards/mip64sel/ls3a7a/.labconfig
```

还可以通过 local-config, board-config 直接修改某个变量：

```
1 // 本地方式
2 $ make local-config MEM=1280M
3
4 // 如果希望发送 Pull Request
5 $ make board-config MEM=1280M
```

相关变量设定跟当前 Qemu 的模拟支持程度密切相关，请勿随意改动，避免出现未知异常。

常用配置信息如下表：

变量	描述
ARCH	处理器指令集架构
XARCH	架构版本，主要用于 qemu-system-
CPU	处理器型号
BUILDROOT	Buildroot 版本号
MEM	内存大小
SERIAL	串口
NETDEV_LIST	验证过的网络设备
NETDEV	当前在用的网络设备
LINUX	Linux 版本号
KERNEL_GIT	Linux 内核源码 GIT 仓库地址
KERNEL_SRC	Linux 内核源码本地存放路径
ORIIMG	Linux 内核映像相对内核根目录的路径
KIMAGE	Linux 内核映像 BSP 代码仓库中的存放路径
DTS	所用的 DTS 路径
DTB	所生成的 DTB 在 BSP 代码仓库中的存放路径
ROOTDEV_LIST	验证过的根文件系统设备
ROOTDEV	当前在用的根文件系统设备
FSTYPE	根文件系统类型
ROOTFS	根文件系统路径
QEMU_GIT	QEMU 源码 GIT 仓库地址
QEMU_SRC	QEMU 源码本地存放路径
QEMU	QEMU 版本号

变量	描述
QTOOL	qemu-system- 工具在 BSP 代码仓库路径
XOPTS	QEMU 额外的参数
XENVS	QEMU 额外的环境变量
XKCLI	Linux 内核额外的启动参数
CCORI	交叉工具链版本，可通过 make gcc-list 查看

4.3 引导编译好的内核和文件系统

Linux Lab 相比于 Buildroot 和 Yocto 之类的构建系统，我们的突出优势是：

- 不仅能够用于编译 Linux 内核、根文件系统、Qemu 等软件包
- 而且已经提供了预先编译好可以立即使用的二进制版本
- 并且提供了可以立即执行的 Qemu 启动脚本

前面下载的板级 BSP 仓库包含了这些：

```

1 // 内核
2 $ ls -R boards/mips64el/ls3a7a/bsp/kernel
3 boards/mips64el/ls3a7a/bsp/kernel:
4 loongnix-release-1903 v5.7
5
6 boards/mips64el/ls3a7a/bsp/kernel/loongnix-release-1903:
7 loongson3_ls7a.dtb vmlinux
8
9 boards/mips64el/ls3a7a/bsp/kernel/v5.7:
10 vmlinux
11
12 // 根文件系统
13 $ ls -R boards/mips64el/ls3a7a/bsp/root
14 boards/mips64el/ls3a7a/bsp/root:
15 2016.05
16
17 boards/mips64el/ls3a7a/bsp/root/2016.05:
18 rootfs.cpio.gz
19
20 // Qemu 引导脚本
21 $ ls -R boards/mips64el/ls3a7a/bsp/boot.sh
22 boards/mips64el/ls3a7a/bsp/boot.sh

```

直接引导（传统方式）：

```

1 $ cd boards/mips64el/ls3a7a/bsp/
2
3 $ ./boot.sh
4 memory_offset = 0x78;cpu_offset = 0xc88; system_offset = 0xce8; irq_offset = 0x3058;
   interface_offset = 0x30b8;
5 param len=35408

```

```
6 env f000080
7 iomem=0x5650268c56c0
8 ram=0x5650268c56c0
9 ram=0x5650268c2b30
10 audio: Could not init `oss' audio driver
11 zimage at:      81310AEO 81A26C19
12 Uncompressing Linux at load address 80200000
13 Now, booting the kernel...
14 SET HT_DMA CACHED
15 [ 0.000000] Initializing cgroup subsys cpu
16 [ 0.000000] Initializing cgroup subsys cpucct
17 [ 0.000000] Linux version 3.10.0 (ubuntu@linux-lab) (gcc version 4.4.7 20120313 (Red
    Hat 4.4.7-3.1) (GCC) ) #57 SMP PREEMPT Tue Jul 23 04:49:46 UTC 2019
18 [ 0.000000] BIOS configured I/O coherency: ON
19 [ 0.000000] The BIOS Version: Loongson-PMON-V3.3.0
20 [ 0.000000] Shutdown Addr: ffffffffbc00000 Reset Addr: ffffffffbc00000
21 [ 0.000000] CpuClock = 200000000
22 [ 0.000000] CPO_Config3: CPO 16.3 (0x80)
23 [ 0.000000] CPO_PageGrain: CPO 5.1 (0x20000000)
24 [ 0.000000] NUMA: Discovered 1 cpus on 1 nodes
25 [ 0.000000] Debug: node_id:0, mem_type:1, mem_start:0x200000, mem_size:0xee MB
26 [ 0.000000]      start_pfn:0x80, end_pfn:0x3c00, num_physpages:0x3b80
27 [ 0.000000] Debug: node_id:0, mem_type:2, mem_start:0x90000000, mem_size:0x100 MB
28 [ 0.000000]      start_pfn:0x24000, end_pfn:0x28000, num_physpages:0x7b80
29 [ 0.000000] node0's addrspace_offset is 0x0
30 [ 0.000000] Node0's start_pfn is 0x80, end_pfn is 0x28000, freepfn is 0x914
31 [ 0.000000] NUMA: set cpumask cpu 0 on node 0
32 [ 0.000000] bootconsole [early0] enabled
33 [ 0.000000] CPU revision is: 00006305 (ICT Loongson-3)
34 [ 0.000000] FPU revision is: 007f0000
35 [ 0.000000] Checking for the multiply/shift bug... no.
36 [ 0.000000] Checking for the daddiu bug... no.
37 [ 0.000000] MIPS: machine is loongson,generic
38 [ 0.000000] Determined physical RAM map:
39 [ 0.000000]   memory: 00000000ee0000 @ 000000000200000 (usable)
40 [ 0.000000]   memory: 000000000002000 @ 00000000fffe000 (reserved)
41 [ 0.000000]   memory: 0000000010000000 @ 0000000090000000 (usable)
42 [ 0.000000] Initial ramdisk at: 0x980000000400000 (4187816 bytes)
43 [ 0.000000] software IO TLB [mem 0x04400000-0x08400000] (64MB) mapped at
    [9800000004400000-98000000083ffff]
44 [ 0.000000] Zone ranges:
45 [ 0.000000]   DMA32    [mem 0x00200000-0xfffffff]
46 [ 0.000000]   Normal    empty
47 [ 0.000000] Movable zone start for each node
48 [ 0.000000] Early memory node ranges
49 [ 0.000000]   node 0: [mem 0x00200000-0x0efffff]
50 [ 0.000000]   node 0: [mem 0x90000000-0x9ffffff]
51 [ 0.000000] Initmem setup node 0 [mem 0x00200000-0x9ffffff]
52 [ 0.000000] Detected 4 available CPU(s)
53 [ 0.000000] Primary instruction cache 64kB, VIPT, 4-way, linesize 32 bytes.
54 [ 0.000000] Primary data cache 64kB, 4-way, VIPT, no aliases, linesize 32 bytes
55 [ 0.000000] Unified victim cache 0kB direct mapped, linesize 0 bytes.
56 [ 0.000000] Unified secondary cache 4096kB 4-way, linesize 32 bytes.
57 [ 0.000000] PERCPU: Embedded 4 pages/cpu @9800000008478000 s15616 r8192 d41728 u65536
58 [ 0.000000] Built 1 zonelists in Node order, mobility grouping on. Total pages: 31492
59 [ 0.000000] Policy zone: DMA32
60 [ 0.000000] Kernel command line: rd_start=0xffffffff84000000 rd_size=4187816 route
    =172.17.0.5 root=/dev/ram0 console=ttyS0 maxcpus=1 e1000e.InterruptThrottleRate
    =4,4,4,4
61 [ 0.000000] PID hash table entries: 2048 (order: 0, 16384 bytes)
62 [ 0.000000] total ram pages initialed 0
```

```
63 [ 0.000000] total ram pages are 24889
64 [ 0.000000] Memory: 398208k/505856k available (6099k kernel code, 107648k reserved,
    11028k data, 336k init, 0k highmem)
65 [ 0.000000] Preemptible hierarchical RCU implementation.
66 [ 0.000000] Dump stacks of tasks blocking RCU-preempt GP.
67 [ 0.000000] RCU restricting CPUs from NR_CPUS=16 to nr_cpu_ids=4.
68 [ 0.000000] NR_IRQS:352
69 [ 0.000000] Supports HT MSI interrupt, enabling LS7A MSI Interrupt.
70 [ 0.000000] set clock event to periodic mode!
71 [ 0.000000] hpet clock event device register
72 [ 0.011718] Console: colour dummy device 80x25
73 [ 0.011718] Calibrating delay loop... 89.62 BogoMIPS (lpj=174080)
74 [ 0.070312] pid_max: default: 32768 minimum: 301
75 [ 0.082031] Dentry cache hash table entries: 65536 (order: 5, 524288 bytes)
76 [ 0.093750] Inode-cache hash table entries: 32768 (order: 4, 262144 bytes)
77 [ 0.101562] Mount-cache hash table entries: 2048 (order: 0, 16384 bytes)
78 [ 0.101562] Mountpoint-cache hash table entries: 2048 (order: 0, 16384 bytes)
79 [ 0.125000] Checking for the daddi bug... no.
80 [ 0.167968] Brought up 1 CPUs
81 [ 0.199218] devtmpfs: initialized
82 [ 0.285156] NET: Registered protocol family 16
83 [ 0.769531] vgaarb: loaded
84 [ 0.773437] SCSI subsystem initialized
85 [ 0.785156] usbcore: registered new interface driver usbfs
86 [ 0.785156] usbcore: registered new interface driver hub
87 [ 0.789062] usbcore: registered new device driver usb
88 [ 0.792968] pps_core: LinuxPPS API ver. 1 registered
89 [ 0.796875] pps_core: Software ver. 5.3.6 - Copyright 2005-2007 Rodolfo Giometti <
    giometti@linux.it>
90 [ 0.800781] PTP clock support registered
91 [ 0.808593] PCI host bridge to bus 0000:00
92 [ 0.808593] pci_bus 0000:00: root bus resource [mem 0x40000000-0x7fffffff]
93 [ 0.812500] pci_bus 0000:00: root bus resource [io 0x4000-0xffff]
94 [ 0.812500] pci_bus 0000:00: root bus resource [??? 0x00000000 flags 0x0]
95 [ 0.812500] pci_bus 0000:00: No busn resource found for root bus, will use [bus 00-ff]
96 [ 0.847656] pci 0000:00:09.0: bridge configuration invalid ([bus 00-00]),
    reconfiguring
97 [ 0.847656] pci 0000:00:0a.0: bridge configuration invalid ([bus 00-00]),
    reconfiguring
98 [ 0.867187] pci 0000:00:06.0: BAR 2: assigned [mem 0x40000000-0x47ffffff]
99 [ 0.867187] pci 0000:00:0a.0: BAR 8: assigned [mem 0x48000000-0x480ffffff]
100 [ 0.867187] pci 0000:00:06.0: BAR 6: assigned [mem 0x48100000-0x4810ffff pref]
101 [ 0.871093] pci 0000:00:06.1: BAR 0: assigned [mem 0x48110000-0x4811ffff]
102 [ 0.871093] pci 0000:00:15.0: BAR 6: assigned [mem 0x48120000-0x4812ffff pref]
103 [ 0.871093] pci 0000:00:06.0: BAR 0: assigned [mem 0x48130000-0x48137fff]
104 [ 0.871093] pci 0000:00:15.0: BAR 0: assigned [mem 0x48138000-0x4813ffff]
105 [ 0.871093] pci 0000:00:07.0: BAR 0: assigned [mem 0x48140000-0x48143fff]
106 [ 0.871093] pci 0000:00:03.0: BAR 0: assigned [mem 0x48144000-0x48145fff]
107 [ 0.875000] pci 0000:00:03.1: BAR 0: assigned [mem 0x48146000-0x48147fff]
108 [ 0.875000] pci 0000:00:04.1: BAR 0: assigned [mem 0x48148000-0x48148fff]
109 [ 0.875000] pci 0000:00:08.0: BAR 0: assigned [mem 0x48149000-0x48149fff]
110 [ 0.875000] pci 0000:00:0a.0: BAR 7: assigned [io 0x4000-0x4fff]
111 [ 0.875000] pci 0000:00:04.0: BAR 0: assigned [mem 0x4814a000-0x4814a0ff]
112 [ 0.878906] pci 0000:00:16.0: BAR 0: assigned [mem 0x4814a100-0x4814a1ff]
113 [ 0.878906] pci 0000:00:09.0: PCI bridge to [bus 01]
114 [ 0.882812] pci 0000:02:00.0: BAR 0: assigned [mem 0x48000000-0x4801ffff]
115 [ 0.882812] pci 0000:02:00.0: BAR 1: assigned [mem 0x48020000-0x4803ffff]
116 [ 0.886718] pci 0000:02:00.0: BAR 3: assigned [mem 0x48040000-0x48043fff]
117 [ 0.886718] pci 0000:02:00.0: BAR 2: assigned [io 0x4000-0x401f]
118 [ 0.886718] pci 0000:00:0a.0: PCI bridge to [bus 02]
119 [ 0.886718] pci 0000:00:0a.0: bridge window [io 0x4000-0x4fff]
```

```

120 [ 0.890625] pci 0000:00:0a.0: bridge window [mem 0x48000000-0x480fffff]
121 [ 0.894531] PCI: Enabling device 0000:00:16.0 (0000 -> 0002)
122 [ 0.902343] ls-spi ls-spi.0: master is unqueued, this is deprecated
123 [ 0.953125] Switched to clocksource hpet
124 [ 0.972656] FS-Cache: Loaded
125 [ 0.984375] CacheFiles: Loaded
126 [ 1.066406] random: fast init done
127 [ 1.675781] NET: Registered protocol family 2
128 [ 1.707031] TCP established hash table entries: 4096 (order: 1, 32768 bytes)
129 [ 1.714843] TCP bind hash table entries: 4096 (order: 2, 65536 bytes)
130 [ 1.714843] TCP: Hash tables configured (established 4096 bind 4096)
131 [ 1.718750] TCP: reno registered
132 [ 1.722656] UDP hash table entries: 256 (order: 0, 16384 bytes)
133 [ 1.722656] UDP-Lite hash table entries: 256 (order: 0, 16384 bytes)
134 [ 1.726562] NET: Registered protocol family 1
135 [ 1.738281] RPC: Registered named UNIX socket transport module.
136 [ 1.746093] RPC: Registered udp transport module.
137 [ 1.746093] RPC: Registered tcp transport module.
138 [ 1.750000] RPC: Registered tcp NFSv4.1 backchannel transport module.
139 [ 1.753906] PCI: Enabling device 0000:00:04.0 (0000 -> 0002)
140 [ 1.761718] PCI: Enabling device 0000:00:04.1 (0000 -> 0002)
141 [ 1.769531] Trying to unpack rootfs image as initramfs...
142 [ 8.457031] Freeing initrd memory: 4080K (9800000004000000 - 98000000043fc000)
143 [ 8.621093] i2c-gpio platform:i2c-gpio@0: using pins 73 (SDA) and 74 (SCL)
144 [ 8.628906] i2c-gpio platform:i2c-gpio@1: using pins 75 (SDA) and 76 (SCL)
145 [ 8.710937] futex hash table entries: 1024 (order: 2, 65536 bytes)
146 [ 8.765625] NFS: Registering the id_resolver key type
147 [ 8.765625] Key type id_resolver registered
148 [ 8.765625] Key type id_legacy registered
149 [ 8.773437] nfs4filelayout_init: NFSv4 File Layout Driver Registering...
150 [ 8.781250] msgmni has been set to 785
151 [ 8.820312] Block layer SCSI generic (bsg) driver version 0.4 loaded (major 251)
152 [ 8.820312] io scheduler noop registered
153 [ 8.820312] io scheduler cfq registered (default)
154 [ 8.835937] pci_hotplug: PCI Hot Plug PCI Core version: 0.5
155 [ 8.851562] PCI: Enabling device 0000:00:06.0 (0000 -> 0002)
156 [ 8.859375] lsgpu_hw_coherent = 1
157 [ 8.867187] res->start is 0xa0000000, res->end is 0xbfffffff
158 [ 8.867187] contiguousSize is 1f000000
159 [ 8.867187] all reserved_size is 20000000
160 [ 8.867187] get VRAM set from UEFI,device_addr = 0x20000000,bus_addr = 0xa0000000,
vram_addr_offset=0x80000000,all_reserved_size=0x20000000
161 [ 8.867187] contiguousSize is 1f000000
162 [ 8.867187] all reserved_size is 20000000
163 [ 8.867187] Galcore Version 4.6.9.6622
164 [ 8.867187] Galcore Options:
165 [ 8.867187] irqLine = 93
166 [ 8.867187] registerMemBase = 0x48130000
167 [ 8.867187] registerMemSize = 0x00008000
168 [ 8.867187] contiguousSize = 520093696
169 [ 8.867187] contiguousBase = 0x00000000
170 [ 8.867187] bankSize = 0x00000000
171 [ 8.867187] fastClear = -1
172 [ 8.867187] compression = -1
173 [ 8.867187] signal = 48
174 [ 8.867187] baseAddress = 0x00000000
175 [ 8.867187] physSize = 0x80000000
176 [ 8.867187] logFileSize = 0 KB
177 [ 8.867187] powerManagement = 1
178 [ 11.265625] Serial: 8250/16550 driver, 16 ports, IRQ sharing enabled
179 [ 11.382812] serial8250.0: ttyS0 at MMIO 0x1fe001e0 (irq = 58) is a 16550A

```

```

180 [ 11.390625] console [ttyS0] enabled, bootconsole disabled
181 [ 11.390625] console [ttyS0] enabled, bootconsole disabled
182 [ 11.417968] 10080000.serial: ttyS1 at MMIO 0x10080000 (irq = 72) is a 16550A
183 [ 11.441406] 10080100.serial: ttyS2 at MMIO 0x10080100 (irq = 72) is a 16550A
184 [ 11.453125] 10080200.serial: ttyS3 at MMIO 0x10080200 (irq = 72) is a 16550A
185 [ 11.468750] 10080300.serial: ttyS4 at MMIO 0x10080300 (irq = 72) is a 16550A
186 [ 11.500000] [drm] Initialized
187 [ 11.500000] loongson_vga_pci_register
188 [ 11.511718] PCI: Enabling device 0000:00:06.1 (0000 -> 0002)
189 [ 11.531250] [drm] Read VBIOS data from spi flash.
190 [ 11.984375] [drm:loongson_vbios_crc_check] *ERROR* VBIOS crc check is wrong,use
    default setting!
191 [ 11.992187] [drm] =====LOONGSON VBIOS INFO
    =====
192 [ 11.992187] [drm] title is Loongson-VBIOS
193 [ 11.996093] [drm] loongson vbios version:0.1
194 [ 11.996093] [drm] vbios information:
195 [ 11.996093] [drm] crtc num:2
196 [ 12.000000] [drm] connector num:2
197 [ 12.000000] [drm] phy num:2
198 [ 12.000000] [drm] =====CRTC INFO
    =====
199 [ 12.000000] [drm] CRTC-0:max_weight=2048,max_height=2048
200 [ 12.000000] [drm] CRTC-0 type is default version
201 [ 12.007812] [drm] Bind connector id is 0
202 [ 12.007812] [drm] Bind phy number is 1
203 [ 12.007812] [drm] Bind phy[0] ID is:0
204 [ 12.007812] [drm] CRTC-1:max_weight=2048,max_height=2048
205 [ 12.007812] [drm] CRTC-1 type is default version
206 [ 12.007812] [drm] Bind connector id is 1
207 [ 12.007812] [drm] Bind phy number is 1
208 [ 12.007812] [drm] Bind phy[0] ID is:1
209 [ 12.007812] [drm] =====CONNECTOR INFO
    =====
210 [ 12.007812] [drm] connector-0:No EDID
211 [ 12.007812] [drm] connector-1:No EDID
212 [ 12.007812] [drm] =====PHY INFO
    =====
213 [ 12.007812] [drm] phy-0:NONE or TRANSPARENT PHY
214 [ 12.007812] [drm] phy-0:bind with CRTC 0
215 [ 12.007812] [drm] phy-0:bind with connector 0
216 [ 12.015625] [drm] phy-1:NONE or TRANSPARENT PHY
217 [ 12.019531] [drm] phy-1:bind with CRTC 1
218 [ 12.019531] [drm] phy-1:bind with connector 1
219 [ 12.019531] [drm] =====END
    =====
220 [ 12.019531] [drm] ldev->rmmio_base = 0x48110000,ldev->rmmio_size = 0x10000
221 [ 13.632812] [drm] loongson vram test success.
222 [ 13.640625] [TTM] Zone kernel: Available graphics memory: 201144 kiB
223 [ 13.640625] [TTM] Initializing pool allocator
224 [ 13.640625] [TTM] Initializing DMA pool allocator
225 [ 14.332031] Console: switching to colour frame buffer device 128x48
226 [ 14.851562] loongson-vga-drm 0000:00:06.1: fb0: loongsondrmfb frame buffer device
227 [ 14.882812] [drm] Initialized loongson-vga-drm 0.1.0 20180328 for 0000:00:06.1 on
    minor 0
228 [ 14.902343] brd: module loaded
229 [ 15.011718] loop: module loaded
230 [ 15.105468] PCI: Enabling device 0000:00:08.0 (0000 -> 0002)
231 [ 15.121093] ahci 0000:00:08.0: AHCI 0001.0000 32 slots 6 ports 1.5 Gbps 0x3f impl SATA
    mode
232 [ 15.121093] ahci 0000:00:08.0: flags: 64bit ncq only

```

```
233 [ 15.464843] scsi host0: ahci
234 [ 15.480468] scsi host1: ahci
235 [ 15.488281] scsi host2: ahci
236 [ 15.496093] scsi host3: ahci
237 [ 15.519531] scsi host4: ahci
238 [ 15.539062] scsi host5: ahci
239 [ 15.558593] ata1: SATA max UDMA/133 abar m4096@0x48149000 port 0x48149100 irq 80
240 [ 15.562500] ata2: SATA max UDMA/133 abar m4096@0x48149000 port 0x48149180 irq 80
241 [ 15.562500] ata3: SATA max UDMA/133 abar m4096@0x48149000 port 0x48149200 irq 80
242 [ 15.566406] ata4: SATA max UDMA/133 abar m4096@0x48149000 port 0x48149280 irq 80
243 [ 15.566406] ata5: SATA max UDMA/133 abar m4096@0x48149000 port 0x48149300 irq 80
244 [ 15.570312] ata6: SATA max UDMA/133 abar m4096@0x48149000 port 0x48149380 irq 80
245 [ 15.796875] PCI: Enabling device 0000:00:03.0 (0000 -> 0002)
246 [ 15.796875] Enhanced/Alternate descriptors
247 [ 15.804687] Extended descriptors not supported
248 [ 15.812500] Ring mode enabled
249 [ 15.812500] No HW DMA feature register supported
250 [ 15.812500] Enable GMAC 32bit DMA
251 [ 15.820312] eth%d: device MAC address 0e:c7:7a:c9:41:d0
252 [ 16.113281] ata4: SATA link down (SStatus 0 SControl 300)
253 [ 16.128906] ata6: SATA link down (SStatus 0 SControl 300)
254 [ 16.148437] ata5: SATA link down (SStatus 0 SControl 300)
255 [ 16.156250] ata1: SATA link down (SStatus 0 SControl 300)
256 [ 16.289062] ata2: SATA link down (SStatus 0 SControl 300)
257 [ 16.292968] ata3: SATA link up 1.5 Gbps (SStatus 113 SControl 300)
258 [ 16.312500] ata3.00: ATAPI: QEMU DVD-ROM, 2.5+, max UDMA/100
259 [ 16.316406] ata3.00: applying bridge limits
260 [ 16.332031] ata3.00: configured for UDMA/100
261 [ 16.472656] scsi 2:0:0:0: CD-ROM QEMU QEMU DVD-ROM 2.5+ PQ: 0 ANSI: 5
262 [ 16.554687] scsi 2:0:0:0: Attached scsi generic sg0 type 5
263 [ 16.652343] libphy: stmmac: probed
264 [ 16.652343] eth0: PHY ID 00120012 at 1 IRQ 0 (stmmac-1:01) active
265 [ 16.660156] PCI: Enabling device 0000:00:03.1 (0000 -> 0002)
266 [ 16.660156] Enhanced/Alternate descriptors
267 [ 16.660156] Extended descriptors not supported
268 [ 16.667968] Ring mode enabled
269 [ 16.667968] No HW DMA feature register supported
270 [ 16.667968] Enable GMAC 32bit DMA
271 [ 16.675781] eth%d: device MAC address 1e:84:1e:42:a9:59
272 [ 16.882812] libphy: stmmac: probed
273 [ 16.890625] eth1: PHY ID 00120012 at 1 IRQ 0 (stmmac-2:01) active
274 [ 16.898437] ehci_hcd: USB 2.0 'Enhanced' Host Controller (EHCI) Driver
275 [ 16.914062] ehci-pci: EHCI PCI platform driver
276 [ 16.921875] ehci-pci 0000:00:04.1: EHCI Host Controller
277 [ 16.933593] ehci-pci 0000:00:04.1: new USB bus registered, assigned bus number 1
278 [ 16.949218] ehci-pci 0000:00:04.1: irq 112, io mem 0x48148000
279 [ 16.964843] ehci-pci 0000:00:04.1: USB 2.0 started, EHCI 1.00
280 [ 16.988281] hub 1-0:1.0: USB hub found
281 [ 17.046875] hub 1-0:1.0: 6 ports detected
282 [ 17.078125] ehci-platform: EHCI generic platform driver
283 [ 17.085937] ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
284 [ 17.093750] ohci-pci: OHCI PCI platform driver
285 [ 17.093750] ohci-pci 0000:00:04.0: OHCI PCI host controller
286 [ 17.093750] ohci-pci 0000:00:04.0: new USB bus registered, assigned bus number 2
287 [ 17.101562] ohci-pci 0000:00:04.0: irq 113, io mem 0x4814a000
288 [ 17.210937] hub 2-0:1.0: USB hub found
289 [ 17.218750] hub 2-0:1.0: 3 ports detected
290 [ 17.238281] ohci-platform: OHCI generic platform driver
291 [ 17.257812] usbcore: registered new interface driver usb-storage
292 [ 17.265625] usbcore: registered new interface driver ums-realtek
293 [ 18.148437] i8042: i8042 controller selftest timeout
```



```

294 [ 18.160156] mousedev: PS/2 mouse device common for all mice
295 [ 18.183593] ls-rtc 100d0100.rtc: rtc core: registered 100d0100.rtc as rtc0
296 [ 18.187500] i2c /dev entries driver
297 [ 18.214843] hidraw: raw HID events driver (C) Jiri Kosina
298 [ 18.230468] usbcore: registered new interface driver usbhid
299 [ 18.230468] usbhid: USB HID core driver
300 [ 18.238281] no options.
301 [ 18.238281] Loongson Hwmon Enter...
302 [ 18.253906] usb 1-1: new high-speed USB device number 2 using ehci-pci
303 [ 18.281250] input: ACPI Power Button as /devices/virtual/input/input0
304 [ 18.289062] ACPI Power Button Driver: Init successful!
305 [ 18.296875] TCP: cubic registered
306 [ 18.304687] Initializing XFRM netlink socket
307 [ 18.304687] NET: Registered protocol family 17
308 [ 18.304687] NET: Registered protocol family 15
309 [ 18.312500] Key type dns_resolver registered
310 [ 18.343750] cpufreq: Loongson-3 CPU autoplug driver.
311 [ 18.398437] ls-rtc 100d0100.rtc: setting system clock to 2020-07-08 02:52:54 UTC
(1594176774)
312 [ 18.683593] input: QEMU QEMU USB Mouse as /devices/pci0000:00/0000:00:04.1/usb1
/1-1/1-1:1.0/input/input1
313 [ 18.710937] hid-generic 0003:0627:0001.0001: input,hidraw0: USB HID v0.01 Mouse [QEMU
QEMU USB Mouse] on usb-0000:00:04.1-1/input0
314 [ 18.765625] Freeing unused kernel memory: 336K (ffff812bc000 - ffffffff81310000)
315 [ 18.863281] usb 1-2: new high-speed USB device number 3 using ehci-pci
316 [ 19.281250] input: QEMU QEMU USB Keyboard as /devices/pci0000:00/0000:00:04.1/usb1
/1-2/1-2:1.0/input/input2
317 [ 19.363281] hid-generic 0003:0627:0001.0002: input,hidraw1: USB HID v1.11 Keyboard [
QEMU QEMU USB Keyboard] on usb-0000:00:04.1-2/input0
318 Starting logging: OK
319 Initializing random number generator... done.
320 Starting network...
321 [ 30.859375] No MAC Management Counters available
322 [ 30.867187] stmmac_hw_setup: failed PTP initialisation
323 eth: eth0, ip: 172.17.0.144, gw: 172.17.0.5
324 [ 38.250000] EMC1412: No valid reference.
325
326 Welcome to Linux Lab
327 linux-lab login:

```

Linux Lab 提供了非常“现代”的启动方式：

```

1 $ make boot
2 sudo env PATH=/labs/linux-lab/boards/mips64el/ls3a7a/bsp/qemu/loongson-v1.0/bin:/usr/
local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin INITRD_OFFSET=0x04000000 qemu
-system-mips64el -M ls3a7a -m 1280M -net nic,model=synopgmac -net tap -smp 1 -kernel
/labs/linux-lab/output/mips64el/linux-loongnix-release-1903-ls3a7a/vmlinuz -no-
reboot -initrd /labs/linux-lab/boards/mips64el/ls3a7a/bsp/root/2016.05/rootfs.cpio.
gz -dtb /labs/linux-lab/boards/mips64el/ls3a7a/bsp/kernel/loongnix-release-1903/
loongson3_ls7a.dtb -nographic -device usb-mouse -device usb-kbd -show-cursor -append
'route=172.20.135.223 iface=eth0 root=/dev/ram0 maxcpus=1 nomsi nmsix console=ttyS0'
3 memory_offset = 0x78;cpu_offset = 0xc88; system_offset = 0xce8; irq_offset = 0x3058;
interface_offset = 0x30b8;
4 param len=35440
5 env f0000a0
6 iomem=0x564482939e80
7 ram=0x564482939e80
8 ram=0x56448293a800
9 audio: Could not init `oss' audio driver
10 qemu-system-mips64el: warning: nic pci-synopgmac.1 has no peer
11 qemu-system-mips64el: warning: nic e1000e.0 has no peer

```

```
12 zimage at:      81300A30 81A14EA4
13 Uncompressing Linux at load address 80200000
14 ...
```

启动完的效果跟传统的方式完全一致，但是可以灵活调整各类 Qemu 选项和内核参数，前面介绍到的各类板级配置基本都可以通过命令行设定。

```
1 $ make boot LINUX=v5.7
```

上面在引导时可以直接通过 LINUX 变量切换 Linux 版本号，非常灵活。

4.4 下载内核源代码

龙芯 1 系、2 系早期处理器 (2E/2F) 的官方 Linux 社区支持比较完善；龙芯 2K、龙芯 3A 的官方 Linux 社区支持还不完整，目前主要是缺乏 7A1000 的支持，导致网卡、磁盘这些目前都不能运行，预计要到 v5.9 左右才能完整支持，目前正由前文提到的陈华才博士、杨嘉勋同学等不断提交和完善。

当前龙芯 Qemu 模拟的板子如下：

```
1 $ ./boards/mips64el/ls3a7a/bsp/qemu/loongson-v1.0/bin/qemu-system-mips64el -M ? | egrep "
    ls|long"
2 fulong2e          Fulong 2e mini pc
3 ls1a             mips ls1a platform
4 ls1b             mips ls1b platform
5 ls1c             mips ls1c platform
6 ls232            mips ls232 platform
7 ls2f1a           mips ls2f1a platform
8 ls2h             mips ls2h platform
9 ls2k             mips ls2k platform
10 ls3a            mips ls3a platform
11 ls3a2h          mips ls3a2h platform
12 ls3a7a          mips ls3a7a platform
```

虽然龙芯 Qemu 也提供了 2E/2F 的模拟，但是并未支持主线 Linux 内核中的板子。

目前 Linux Lab 验证了其中 4 款开发板，每个处理器系列支持一款，当前支持的内核源代码路径如下：

板子	内核源代码
ls1b	https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/
ls232	https://gitee.com/loongsonlab/linux-2.6.32.git
ls2k	git://cgit.loongnix.org/kernel/linux-3.10.git
ls3a7a	git://cgit.loongnix.org/kernel/linux-3.10.git

不过，为了确保相关功能完备，Linux Lab 在上述代码仓库的基础上维护了相应的内核 patch：

```
1  $$ ls -R boards/mips64el/ls3a7a/bsp/patch/linux/
2  boards/mips64el/ls3a7a/bsp/patch/linux/:
3  loongnix-release-1903  v5.7
4
5  boards/mips64el/ls3a7a/bsp/patch/linux/loongnix-release-1903:
6  0001-add-option.patch
7  0002-don-t-use-software-page-coloring-for-loongson2k1000-.patch
8  0003-debugfs_create_file_unsafe-is-not-defined.-changed-t.patch
9  0004-modify-binder-code-to-support-32bit-user-load-from-w.patch
10 0005-mips-zboot-add-missing-memcpy-and-memset.patch
11 0006-ls2k-add-zboot-and-zboot-debug-support.patch
12 0007-ls3a-add-zboot-debug-support.patch
13 0008-ls3a-fix-up-building-failure-for-kexec.patch
14 0009-noinird-skip-builtin-initramfs-too.patch
15 0010-log2.h-fix-up-conflicts-between-const-and-noreturn.patch
16 0011-gcc-9-fix-up-conflicts-between-mloongson-mmi-and-mso.patch
17 0012-gcc-9-fix-up-stack-detect-errors.patch
18
19 boards/mips64el/ls3a7a/bsp/patch/linux/v5.7:
20 0001-ls3a7a-fix-up-zboot-uart-support.patch
21 0002-stmmac-set-dma_mask-to-DMA_BIT_MASK-32.patch
```

如果要更换内核版本，需要相应地做内核 patch 移植。

上述源码和 patch，在 Linux Lab 下都是“透明”管理的，无需直接手动 clone 和逐个打 patch，仅需使用这样的命令：

```
1 // 下载内核源码
2 $ make kernel-download
3
4 // 检出相应的版本，即 LINUX 变量指定的版本号
5 $ make kernel-checkout
6
7 // 应用相应的 patch
8 $ make kernel-patch
```

如果 checkout 时提示有修改没有保存，请检查修改是否重要并确保做好备份，之后再执行：

```
1 // cleanup 历史修改
2 $ make kernel-cleanup
3
4 // 再次 checkout 版本
5 $ make kernel-checkout
```

4.5 配置内核

上面下载的板级 BSP 资源包同样提供了预先验证过的内核配置文件：

```
1 $ ls boards/mips64el/ls3a7a/bsp/configs/
2 buildroot_2016.05_defconfig      linux_v5.7_defconfig
3 linux_loongnix-release-1903_defconfig
```

默认配置的是基于 Linux v3.10 代码仓库的 linux_loongnix-release-1903_defconfig 配置文件。

使用该文件默认配置内核，仅需：

```
1 $ make kernel-defconfig
```

4.6 修改内核

在编译内核之前，简单添加一点修改，模拟内核开发的必要过程。

首先，查看本地内核源码路径，该路径在板级配置文件中设定：

```
1 $ make | grep KERNEL_SRC
2 KERNEL_SRC[LINUX_loongnix-release-1903] = loongnix-linux-3.10
```

然后进到源码路径：

```
1 $ cd loongnix-linux-3.10/
```

接着做一些修改，例如在 `kernel_init()` 中打印一点内容：

```
1 $ git diff init/main.c
2 diff --git a/init/main.c b/init/main.c
3 index d7bc443..0a189f9 100644
4 --- a/init/main.c
5 +++ b/init/main.c
6 @@ -901,6 +901,8 @@ static int __ref kernel_init(void *unused)
7
8     flush_delayed_fput();
9
10 +     printk(KERN_INFO "Hello, Linux Lab\n");
11 +
12     if (ramdisk_execute_command) {
13         if (!run_init_process(ramdisk_execute_command))
14             return 0;
```

4.7 编译内核

交叉编译内核通常是比较繁琐的，需要指定交叉编译器、内核源码路径、内核构建路径等等，Linux Lab 把这些都简化了。

在 Linux Lab 下，仅需一条命令就可直接编译内核：

```
1 $ make kernel
```

完整编译过程如下，不仅编译了内核，还编译了 DTB：

```
1 $ make kernel
2 Building dtb ...
3   DTS: loongnix-linux-3.10/arch/mips/loongson/loongson-3/dts/loongson3_ls7a.dts
4   DTB: /labs/linux-lab/boards/mips64el/ls3a7a/bsp/kernel/loongnix-release-1903/
        loongson3_ls7a.dtb
5 env PATH=/usr/bin/:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin make 0=/
        labs/linux-lab/output/mips64el/linux-loongnix-release-1903-ls3a7a -C loongnix-linux
        -3.10 ARCH=mips LOADADDR= CROSS_COMPILE=mipsel-linux-gnu- V= CONFIG_INITRAMFS_SOURCE
        = -j4 vmlinux
```

```

6 make[1]: Entering directory '/labs/linux-lab/loongnix-linux-3.10'
7 make -C /labs/linux-lab/output/mips64el/linux-loongnix-release-1903-1s3a7a \
8 KBUILD_SRC=/labs/linux-lab/loongnix-linux-3.10 \
9 KBUILD_EXTMOD="" -f /labs/linux-lab/loongnix-linux-3.10/Makefile \
10 vmlinux
11 GEN      ./Makefile
12 CHK      include/generated/uapi/linux/version.h
13 CHK      include/generated/qrwlock.h
14 CHK      include/generated/qrwlock_api_smp.h
15 CHK      include/generated/qrwlock_types.h
16 CHK      kernel/qrwlock_gen.c
17 CHK      lib/qrwlock_debug.c
18 CC       scripts/mod/devicetable-offsets.s
19 GEN      scripts/mod/devicetable-offsets.h
20 HOSTCC   scripts/mod/file2alias.o
21 Using /labs/linux-lab/loongnix-linux-3.10 as source for kernel
22 CHK      include/generated/utsrelease.h
23 Checking missing-syscalls for N32
24 CALL     /labs/linux-lab/loongnix-linux-3.10/scripts/checksyscalls.sh
25 HOSTLD   scripts/mod/modpost
26 Checking missing-syscalls for O32
27 CALL     /labs/linux-lab/loongnix-linux-3.10/scripts/checksyscalls.sh
28 CALL     /labs/linux-lab/loongnix-linux-3.10/scripts/checksyscalls.sh
29 CHK      include/generated/compile.h
30 DTC      arch/mips/loongson/loongson-3/dts/loongson3_ls7a.dtb
31 DTB      arch/mips/loongson/loongson-3/dts/loongson3_ls7a.dtb.S
32 AS       arch/mips/loongson/loongson-3/dts/loongson3_ls7a.dtb.o
33 LD       arch/mips/loongson/loongson-3/dts/built-in.o
34 rm arch/mips/loongson/loongson-3/dts/loongson3_ls7a.dtb.S arch/mips/loongson/loongson-3/
    dts/loongson3_ls7a.dtb
35 LD       arch/mips/loongson/loongson-3/built-in.o
36 LD       arch/mips/loongson/built-in.o
37 LD       arch/mips/built-in.o
38 CHK      kernel/config_data.h
39 LINK     vmlinux
40 LD       vmlinux.o
41 MODPOST  vmlinux.o
42 WARNING: modpost: Found 2 section mismatch(es).
43 To see full details build your kernel with:
44 'make CONFIG_DEBUG_SECTION_MISMATCH=y'
45 GEN      .version
46 CHK      include/generated/compile.h
47 UPD      include/generated/compile.h
48 CC       init/version.o
49 LD       init/built-in.o
50 KSYM     .tmp_kallsyms1.o
51 KSYM     .tmp_kallsyms2.o
52 LD       vmlinux
53 SORTEX   vmlinux
54 SYMAP    System.map
55 mipsel-linux-gnu-objcopy -O elf32-tradlittlemips --remove-section=.reginfo vmlinux
    vmlinux.32
56 CC       arch/mips/boot/compressed/dummy.o
57 OBJCOPY  arch/mips/boot/compressed/vmlinux.bin
58 LZO      arch/mips/boot/compressed/vmlinux.bin.z
59 OBJCOPY  arch/mips/boot/compressed/piggy.o
60 LD       vmlinux
61 STRIP    vmlinux

```

4.8 引导新编译的内核

当新的内核顺利编译完成后，`make boot` 会默认引导新内核，反之，就会使用预先编译好的内核映像。

```

1 $ make boot
2 ...
3 [ 53.632812] ls-rtc 100d0100.rtc: setting system clock to 2020-07-08 04:57:13 UTC
   (1594184233)
4 [ 53.785156] Freeing unused kernel memory: 320K (ffffff812b0000 - fffffff81300000)
5 [ 53.785156] Hello, Linux Lab
6 [ 54.601562] input: QEMU QEMU USB Mouse as /devices/pci0000:00/0000:00:04.1/usb1
   /1-1/1-1:1.0/input/input1
7 [ 54.625000] hid-generic 0003:0627:0001.0001: input,hidraw0: USB HID v0.01 Mouse [QEMU
   QEMU USB Mouse] on usb-0000:00:04.1-1/input0
8 [ 55.500000] usb 1-2: new high-speed USB device number 3 using ehci-pci
9 [ 55.945312] input: QEMU QEMU USB Keyboard as /devices/pci0000:00/0000:00:04.1/usb1
   /1-2/1-2:1.0/input/input2
10 [ 56.246093] hid-generic 0003:0627:0001.0002: input,hidraw1: USB HID v1.11 Keyboard [
   QEMU QEMU USB Keyboard] on usb-0000:00:04.1-2/input0
11 Starting logging: OK
12 Initializing random number generator... done.
13 Starting network...
14 [ 64.378906] No MAC Management Counters available
15 [ 64.386718] stmmac_hw_setup: failed PTP initialisation
16 eth: eth0, ip: 172.20.135.14, gw: 172.20.135.223
17
18 Welcome to Linux Lab
19 linux-lab login: root
20 #
21 # dmesg | grep -i hello
22 [ 53.785156] Hello, Linux Lab
    
```

当然，也可以通过参数明确指定启用哪个内核版本，以便比较差异或者排查新内核失败的原因：

```

1 // 明确启动预先编译的内核版本，可使用 old 或 prebuilt
2 $ make boot kernel=old
3 $ make boot kernel=prebuilt
4
5 // 明确启动新编译的内核版本，可使用 new 或 build
6 $ make boot kernel=new
7 $ make boot kernel=build
    
```

以上介绍了基本的开发过程，接下来再结合案例做深入的讲解。

5. 用 Linux Lab 进行龙芯 Linux 内核开发

本章结合实实在在的案例来演示如何通过 Linux Lab 开展龙芯 Linux 内核开发。

如果没有明确说明，以下演示所用开发板为 mips64el/ls3a7a，所用内核版本为默认采用的 loongnix-release-1903。

如果中途切换过其他开发板和配置，请记得用如下命令修改回来：

```
1 $ make BOARD=mips64el/ls3a7a
2 $ make local-config LINUX=loongnix-release-1903
```

5.1 内核模块开发实验

大部分时候其实是跟设备驱动打交道，而设备驱动则通常是以内核模块的方式编写的，本节首先介绍如何通过 Linux Lab 开展内核模块的实验。

5.1.1 撰写并运行内核模块

Linux Lab 下提供了 3 个非常基础的内核模块例子，下面先来实验最基础的 hello 模块。

```

1  $ ls modules/
2  exception hello ldt README.md
3
4  $ ls modules/hello/
5  hello.c Makefile
6
7  $ cat modules/hello/hello.c
8  #include <linux/kernel.h>
9  #include <linux/module.h>
10 #include <linux/init.h>
11
12 static int __init my_hello_init(void)
13 {
14     pr_info("hello module init\n");
15
16     return 0;
17 }
18
19 static void __exit my_hello_exit(void)
20 {
21     pr_info("hello module exit\n");
22 }
23
24 module_init(my_hello_init);
25 module_exit(my_hello_exit);
26
27 MODULE_DESCRIPTION("hello - Linux Lab module example");
28 MODULE_AUTHOR("Wu Zhangjin <wuzhangjin@gmail.com>");
29 MODULE_LICENSE("GPL");
30
31 $ cat modules/hello/Makefile
32 # ARCH = arm
33 # CROSS_COMPILE = arm-linux-gnueabi-
34
35 KERNEL_SRC ?= /lib/modules/`uname -r`/build
36
37 obj-m += hello.o
38
39 modules:
40     $(MAKE) -C $(KERNEL_SRC) M=$$PWD modules;
41
42 modules-install:
43     $(MAKE) -C $(KERNEL_SRC) M=$$PWD modules-install;
44
45 clean:
46     $(MAKE) -C $(KERNEL_SRC) M=$$PWD clean;
47     rm -f *.ko;

```

编译该内核模块：


```
1 $ make modules M=modules/hello
```

安装到根文件系统：

```
1 $ make modules-install M=modules/hello
```

更新 initrd 文件系统：

```
1 $ make root-rebuild
```

启动新的根文件系统并通过 `insmod`, `rmmmod` 和 `modprobe` 来演示如何 insert/remove 内核模块：

```
1 $ make boot
2 ...
3 Welcome to Linux Lab
4 linux-lab login: root
5
6 # insmod /lib/modules/3.10.0-00827-g04b9868436a-dirty/extra/hello.ko
7 [ 141.375000] hello: loading out-of-tree module taints kernel.
8 [ 141.425781] hello module init
9 # rmmmod [ 167.660156] random: crng init done
10 /lib/modules/3.10.0-00827-g04b9868436a-dirty/extra/hello.ko
11 [ 170.269531] hello module exit
12
13 # modprobe hello
14 [ 179.164062] hello module init
15 # modprobe -r hello
16 [ 202.519531] hello module exit
```

5.1.2 自动测试内核模块

上述步骤略微繁琐，Linux Lab 允许通过一条命令完成编译、安装、启动并自动 insert/remove 模块：

```
1 $ make test m=hello TEST_RD=ram0
2 Starting testing ...
3
4 Testing begin: Running "echo 'begin'" ...
5 begin
6 Testing feature (top part): boot
7 Testing feature (top part): module
8
9 module: Starting testing: hello ...
10 module: Starting testing module: hello
11
12 module: modprobe hello
13 [ 26.425781] hello: loading out-of-tree module taints kernel.
14 [ 26.500000] hello module init
15
16 module: lsmod hello
17 Module                Size  Used by    Tainted: G
18 hello                  1142  0
19
20 Testing feature (bottom part): boot
21 Testing feature (bottom part): module
22
```

```
23 module: rmmod hello
24 [ 29.625000] EMC1412: No valid reference.
25 [ 29.765625] hello module exit
26
27 module: Stoping testing module: hello
28 module: Stoping testing: hello ...
29
30 Testing end: Running "echo 'end'" ...
31 end
32 Testing finish: Running "poweroff" ...
33
34 Stop testing ...
35 OK
36 Stopping network...ifdown: interface lo not configured
37 Saving random seed... done.
38 Stopping logging: OK
39 The system is going down NOW!
40 Sent SIGTERM to all processes
41 Sent SIGKILL to all processes
42 Requesting system poweroff
43 [ 38.605468] Power down.
```

龙芯 Qemu 默认配置的内核参数 Buffer 只有 1024 字节，所以上面测试时通过 `TEST_RD=ram0` 选用了 `initrd` 作为根文件系统，`initrd` 的内核参数比 NFS 根文件系统要少一些，不会被截断。

如果想用 NFS 引导的话，需要稍微修改一下 Qemu，否则内核参数字符超出的部分会被截断。具体怎么修改 Qemu，后面的章节再来介绍。

另外，当前 `ls3a7a` 的 `poweroff` 功能不完整，执行完以后不能正常退出 Qemu，需要手动按下 `CTRL+a x` 退出 Qemu。

当前情况下，如果想在测试完成后自动退出 Qemu，可以临时通过 `TEST_TIMEOUT=30` 自动检测超时并退出。

```
1 $ make test m=hello TEST_RD=ram0 TEST_TIMEOUT=30
2 ...
3 qemu-system-mips64el: terminating on signal 15 from pid 65130 (timeout)
4 ERR: Boot timeout in 30.
```

该设定下的启动过程超过 30s 以后，会自动退出并打印出超时信息。如果启动和测试过程比较长，可以相应加大 `TEST_TIMEOUT`。

5.2 内核开发实验

下面通过几个例子简单介绍 Linux 内核部分的开发。

5.2.1 添加新的内核压缩算法

不同的内核压缩算法会带来不同的受益，从而可以满足不同场景的需求，比如说 xz 压缩率比较高，而 lzo 则解压比较快。

如果处理器性能不错，但是磁盘比较慢，就可以考虑压缩率比较高的，反之则可以用解压比较快的算法。当然，具体情况视目标需求而定。

龙芯 v3.10 内核已经支持了 gzip, bzip2, lzma 以及 lzo。下面简单介绍一下如何新增压缩率更高的 xz。

首先，在 arch/mips/boot/compressed 中新增 xz 支持：

```

1 $ git diff arch/mips/boot/compressed
2 diff --git a/arch/mips/boot/compressed/Makefile b/arch/mips/boot/compressed/Makefile
3 index bbaa1d4..cad3c26 100644
4 --- a/arch/mips/boot/compressed/Makefile
5 +++ b/arch/mips/boot/compressed/Makefile
6 @@ -43,6 +43,7 @@ $(obj)/vmlinux.bin: $(KBUILD_IMAGE) FORCE
7  tool_$(CONFIG_KERNEL_GZIP)      = gzip
8  tool_$(CONFIG_KERNEL_BZIP2)    = bzip2
9  tool_$(CONFIG_KERNEL_LZMA)     = lzma
10 +tool_$(CONFIG_KERNEL_XZ)       = xzkern
11  tool_$(CONFIG_KERNEL_LZO)     = lzo
12
13 diff --git a/arch/mips/boot/compressed/decompress.c b/arch/mips/boot/compressed/
14   decompress.c
15 index 6bfd0be..eb437e0 100644
16 --- a/arch/mips/boot/compressed/decompress.c
17 +++ b/arch/mips/boot/compressed/decompress.c
18 @@ -74,6 +76,10 @@ void *memset(void *s, int c, size_t n)
19  #include "../../../../../lib/decompress_unlzma.c"
20  #endif
21
22 +#ifdef CONFIG_KERNEL_XZ
23 +#include "../../../../../lib/decompress_unxz.c"
24 +#endif
25 +
26 #ifdef CONFIG_KERNEL_LZO
27 #include "../../../../../lib/decompress_unlzo.c"
28 #endif

```

如果有新的解压库，也可以类似新增到内核源码的 lib/decompress_XXX.c。

然后，告诉构建系统我们的支持状态：

```

1 $ git diff arch/mips/Kconfig
2 diff --git a/arch/mips/Kconfig b/arch/mips/Kconfig
3 index d9cc419..0589533 100644
4 --- a/arch/mips/Kconfig
5 +++ b/arch/mips/Kconfig
6

```

```
7 @@ -1572,6 +1574,7 @@ config SYS_SUPPORTS_ZBOOT
8     select HAVE_KERNEL_GZIP
9     select HAVE_KERNEL_BZIP2
10    select HAVE_KERNEL_LZMA
11 +   select HAVE_KERNEL_XZ
12     select HAVE_KERNEL_LZO
13
14 config SYS_SUPPORTS_ZBOOT_UART16550
```

接着，重新配置内核，选择 XZ 算法。

```
1 $ make kernel-menuconfig
2
3 General Setup -->
4   Kernel Compression Mode -->
5     ( ) Gzip
6     ( ) Bzip2
7     ( ) Lzma
8     (X) XZ
9     ( ) LZ0
```

最后，编译并验证结果：

```
1 $ make kernel
2
3 $ ls -lh output/mips64el/linux-loongnix-release-1903-1s3a7a/vmlinuz
4 -rwxr-xr-x 1 ubuntu ubuntu 3.4M  8月  9 19:00 output/mips64el/linux-loongnix-release
   -1903-1s3a7a/vmlinuz
```

作为比较，采用 lz0 压缩算法的情况如下，大了 1 倍多：

```
1 $ ls -lh output/mips64el/linux-loongnix-release-1903-1s3a7a/vmlinuz
2 -rwxr-xr-x 1 ubuntu ubuntu 7.2M  8月  9 19:03 output/mips64el/linux-loongnix-release
   -1903-1s3a7a/vmlinuz
3
4 $ echo "scale=3;7.2/3.4" | bc -l
5 2.117
```

大家也可以借鉴上述方法，新增 lz4。在上面的基础上，如果内核本身还没支持新算法的话，应该还需要在 `scripts/Makefile.lib` 中新增压缩指令，例如上面的 `xzkern` 是这么加的：

```
1 $ grep xzkern -rh loongnix-linux-3.10/scripts/
2 # Use xzkern to compress the kernel image and xzmisc to compress other things.
3 # xzkern uses a big LZMA2 dictionary since it doesn't increase memory usage
4 # the target architecture. xzkern also appends uncompressed size of the data
5 # files. xzmisc uses smaller LZMA2 dictionary than xzkern, because a very
6 quiet_cmd_xzkern = XZKERN  $@
7 cmd_xzkern = (cat $(filter-out FORCE,$^) | \
```

5.2.2 为内核添加新的系统调用

本节演示如何为龙芯 Linux 内核新增一个系统调用并编写一个简单的测试程序。

先做内核修改，新增 `sys_linux_lab` 内核调用：

```
1 $ cd loongnix-linux-3.10/
```

接下来，新增系统调用实现：

```

1 $ cat arch/mips/kernel/linux-lab.c
2 /* sys_linux_lab() syscall demo */
3
4 #include <linux/syscalls.h>
5
6 SYSCALL_DEFINE1(linux_lab, unsigned int, flags)
7 {
8     printk(KERN_INFO "Hello, Linux Lab Syscall, flags is %d\n", flags);
9
10    return 0;
11 }
    
```

并在 `sys_call_table` 增加一个入口，MIPS 的 ABI 比较复杂，这里统一用 64 位内核 64 位用户态的版本：

```

1 $ diff --git a/arch/mips/kernel/scall64-64.S b/arch/mips/kernel/scall64-64.S
2 index 4842795..8e241ea 100644
3 --- a/arch/mips/kernel/scall64-64.S
4 +++ b/arch/mips/kernel/scall64-64.S
5 @@ -430,4 +430,5 @@ sys_call_table:
6     PTR        sys_seccomp
7     PTR        sys_getrandom
8     PTR        sys_memfd_create
9 +    PTR        sys_linux_lab                /* 5315 */
10    .size      sys_call_table,.-sys_call_table
    
```

然后在 `unistd.h` 相应增加该系统调用号并递增系统调用总数：

```

1 $ git diff arch/mips/include/uapi/
2 diff --git a/arch/mips/include/uapi/asm/unistd.h b/arch/mips/include/uapi/asm/unistd.h
3 index 8548fd2..a0944ec 100644
4 --- a/arch/mips/include/uapi/asm/unistd.h
5 +++ b/arch/mips/include/uapi/asm/unistd.h
6 @@ -708,16 +708,17 @@
7     #define __NR_seccomp                (__NR_Linux + 312)
8     #define __NR_getrandom              (__NR_Linux + 313)
9     #define __NR_memfd_create          (__NR_Linux + 314)
10    +#define __NR_linux_lab              (__NR_Linux + 315)
11
12    /*
13     * Offset of the last Linux 64-bit flavoured syscall
14     */
15    -#define __NR_Linux_syscalls          314
16    +#define __NR_Linux_syscalls          315
17
18    #endif /* Want N64 || _MIPS_SIM == _MIPS_SIM_ABI64 */
19
20    #define __NR_64_Linux                5000
21    -#define __NR_64_Linux_syscalls      314
22    +#define __NR_64_Linux_syscalls      315
23
24    #if (defined(__WANT_SYSCALL_NUMBERS) &&
25         (__WANT_SYSCALL_NUMBERS == _MIPS_SIM_NABI32)) || \
    
```

接着编写一个用户态的汇编语言程序来调用并编译。

```

1 $ pushd examples/assembly/mips64el/
2
    
```

```

3 $ vim mips64el-linux-lab.s
4 # text section
5 .text
6 .globl __start
7 __start:
8     .set noreorder
9     .cplod $gp
10    .set reorder
11
12    li $a0, 1      # first argument: the standard output, 1
13    li $v0, 5315   # sys_linux_lab: system call number, defined as __NR_linux_lab in /
14    syscall       # causes a system call trap.
15                # exit from this program via calling the sys_exit system call
16    move $a0, $0   # or "li $a0, 0", set the normal exit status as 0
17                # you can print the exit status with "echo $?" after executing this
18    program
19    li $v0, 5058   # 5058 is __NR_exit defined in /usr/include/asm/unistd.h
20    syscall
21 $ make
22 $ popd

```

复制到根文件系统并重新打包后启动新内核和新文件系统：

```

1 $ cp examples/assembly/mips64el/mips64el-linux-lab boards/mips64el/ls3a7a/bsp/root
2   /2016.05/rootfs/
3 $ make root-rebuild
4
5 $ make boot
6 Welcome to Linux Lab
7 linux-lab login: root
8 #
9 # ls
10 mips64el-linux-lab
11 # ./mips64el-linux-lab
12 [ 117.839843] Hello, Linux Lab Syscall, flags is 1

```

大功告成。

5.2.3 内核函数跟踪辅助工具

本节简单演示如何开启和使用 Ftrace。

由于 ls3a7a 所用的 v3.10 内核太老，这里直接用 ls3a7a v5.7 做实验。需要做如下切换：

```

1 $ make local-config LINUX=v5.7
2 $ make gcc-switch CCOR1=bootlin

```

接着打开内核配置并编译：

```

1 $ make kernel-menuconfig
2
3 Kernel Hacking -->
4     [*] Tracers -->
5         [*] Kernel Function Tracer
6         [*] Kernel Function Graph Tracer

```

```
7 [*] enable/disable function tracing dynamically
8
9 $ make kernel
```

安装最新的 Ftrace 跟踪脚本到 initrd (ls3a7a v5.7 目前仅支持 initrd):

```
1 $ ls system/tools/ftrace/trace.sh
2 $ make root-install
3 $ make root-rebuild
```

启动新内核:

```
1 $ make boot
2 ...
3 Welcome to Linux Lab
4 linux-lab login: root
5 #
```

使用 /tools/ftrace/trace.sh 脚本并启用默认的 function tracer:

```
1 # /tools/ftrace/trace.sh
2 [Available Tracers]
3
4 function_graph function nop
5
6 [Using tracer: function]
7 [Enabling tracing]
8 [Running command: ls]
9
10 mips64el-hello      tools
11 mips64el-hello.result  trace.log
12
13 [Disabling tracing]
14 [Recording tracing result from /sys/kernel/tracing/trace]
15
16 Tracing [ls] log with [function] saved in /root/trace.log
17
18 # cat /root/trace.log | head -20
19 # tracer: function
20 #
21 # entries-in-buffer/entries-written: 51478/119806  #P:1
22 #
23 #          _-----> irqs-off
24 #          / _-----> need-resched
25 #          | / _---> hardirq/softirq
26 #          || / _--> preempt-depth
27 #          ||| /      delay
28 #          TASK-PID  CPU#  ||||  TIMESTAMP  FUNCTION
29 #          | |      |  ||||  |             |
30          ls-166    [000] d.h4   440.623682:  __update_load_avg_se <-task_tick_fair
31          ls-166    [000] d.h4   440.623716:  __accumulate_pelt_segments <-
32          __update_load_avg_se
33          ls-166    [000] d.h4   440.623731:  __update_load_avg_cfs_rq <-task_tick_fair
34          ls-166    [000] d.h4   440.623759:  __accumulate_pelt_segments <-
35          __update_load_avg_cfs_rq
36          ls-166    [000] d.h4   440.623775:  update_cfs_group <-task_tick_fair
37          ls-166    [000] d.h4   440.623802:  reweight_entity <-task_tick_fair
38          ls-166    [000] d.h4   440.623817:  update_curr <-reweight_entity
39          ls-166    [000] d.h4   440.623855:  hrtimer_active <-task_tick_fair
40          ls-166    [000] d.h4   440.623871:  calc_global_load_tick <-scheduler_tick
```

更换为 function_graph tracer：

```

1 # /tools/ftrace/trace.sh function_graph ls
2
3 # head -50 /root/trace.log
4 # tracer: function_graph
5 #
6 # CPU DURATION          FUNCTION CALLS
7 # |   |   |              |   |   |   |
8 0) + 32.000 us |          __rcu_read_unlock();
9 0) + 73.310 us |          } /* __unlock_page_memcg */
10 0) ! 114.260 us |         } /* unlock_page_memcg */
11 0) ! 260.490 us |         } /* page_add_file_rmap */
12 0) + 21.700 us |          __update_cache();
13 0) + 23.770 us |          __update_tlb();
14 0) ! 434.130 us |         } /* alloc_set_pte */
15 0) + 20.710 us |          unlock_page();
16 0)              |          alloc_set_pte() {
17 0) + 20.520 us |              add_mm_counter_fast();
18 0)              |              page_add_file_rmap() {

```

5.2.4 内核自动化调试

往常要调试内核非常不方便，首先需要设法打开多个内核调试相关的配置选项，然后配置 Qemu，配置 gdb。Linux Lab 把这一切都简化了。

调试前准备：

```

1 $ make feature feature=debug
2 $ make kernel-oldefconfig
3 $ make kernel

```

自动化调试：

```

1 $ make kernel-debug

```

在 WebVNC 或者 VNC 登陆 Linux Lab 的情况下，会自动开两个窗口，一个显示原有内核引导过程并停在某个预设的位置，一个用来连接到 Qemu 并提供交互调试界面给用户。

5.2.5 内核自动化测试

以下实验，请调整回默认内核，因为可能会用到磁盘和网络：

```

1 $ make local-config LINUX=loongnix-release-1903

```

前文介绍到了自动化测试内核模块，其实该功能只是内核自动化测试的一个子集。

进行重启压力测试，检测到卡死后自动退出：

```

1 $ make test TEST_REBOOT=2 TEST_TIMEOUT=30s TEST_RD=ram0

```

测试过程中如果超时，继续执行后续测试，而不是直接终止：

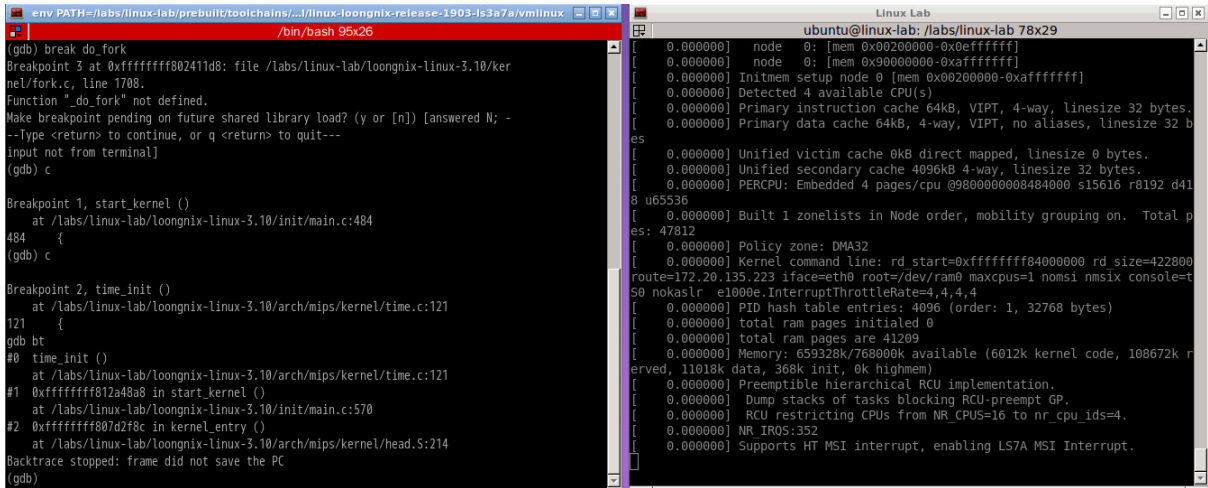


Figure 6: Linux Lab 调试内核

```
1 $ make test TEST_TIMEOUT=30s TIMEOUT_CONTINUE=1 TEST_RD=ram0
```

运行指定测试用例：

```
1 $ make test TEST_CASE='ls /root,echo hello world' TEST_RD=ram0
```

同时测试多个内核模块：

```
1 $ make test m=exception,hello TEST_RD=ram0
```

测试内核调试：

```
1 $ make test DEBUG=1 TEST_RD=ram0
```

5.3 文件系统实验

5.3.1 更换文件系统格式

前文已经介绍了可以从不同的存储介质加载文件系统，比如 ram0, nfs，还支持从磁盘加载：

```
1 $ make list rootdev
2 /dev/sda [/dev/ram0] /dev/nfs
3
4 $ make boot ROOTDEV=/dev/sda
```

默认使用的磁盘文件系统格式为 ext2，如果要换一种，可以设定 FSTYPE。

```
1 $ make boot ROOTDEV=/dev/sda FSTYPE=ext4
2 sudo env PATH=/labs/linux-lab/boards/mips64el/ls3a7a/bsp/qemu/loongson-v1.0/bin/:/usr/
   local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin INITRD_OFFSET=0x04000000 qemu
   -system-mips64el -M ls3a7a -m 1280M -net nic,model=synopgmac -net tap -smp 1 -kernel
   /labs/linux-lab/output/mips64el/linux-loongnix-release-1903-ls3a7a/vmlinuz -no-
   reboot -dtb /labs/linux-lab/boards/mips64el/ls3a7a/bsp/kernel/loongnix-release-1903/
   loongson3_ls7a.dtb -hda /labs/linux-lab/boards/mips64el/ls3a7a/bsp/root/2016.05/
   rootfs.ext4 -nographic -device usb-mouse -device usb-kbd -show-cursor -append 'route
   =172.20.135.223 iface=eth0 rw fsck.repair=yes rootwait root=/dev/sda maxcpus=1 nomsi
   nmsix console=ttyS0'
```

Linux Lab 会自动根据指定的格式创建相应的文件系统，原始系统是通过 Buildroot 构建的 rootfs.cpio.gz：

```
1 $ ls boards/mips64el/ls3a7a/bsp/root/2016.05/rootfs.cpio.gz
2 boards/mips64el/ls3a7a/bsp/root/2016.05/rootfs.cpio.gz
3
4 $ ls boards/mips64el/ls3a7a/bsp/root/2016.05/rootfs.ext*
5 boards/mips64el/ls3a7a/bsp/root/2016.05/rootfs.ext2 boards/mips64el/ls3a7a/bsp/root
   /2016.05/rootfs.ext4
```

如果不支持设定的格式，那么可能需要安装相应的 mkfs 工具并添加相应的内核支持。这里不做进一步介绍。

接下来，我们再看看不同规模和便利性的文件系统。

5.3.2 通过 Buildroot 构建根文件系统

Linux Lab 内置支持 Buildroot，当前 ls3a7a 用的是 2016.05，前述板级 BSP 资源包已经提供了验证过的配置文件，可以直接用来配置和编译 rootfs。

```
1 $ ls boards/mips64el/ls3a7a/bsp/configs/buildroot_2016.05_defconfig
2 boards/mips64el/ls3a7a/bsp/configs/buildroot_2016.05_defconfig
```

以下步骤可以完成源码下载、默认配置、交互配置以及文件系统编译：

```
1 $ make root-download
2 $ make root-defconfig
3 $ make root-menuconfig
4 $ make root
```

编译生成新的文件系统后，Linux Lab 同样会使用新编译的 rootfs。

这类文件系统适合小型嵌入式系统，需要新的工具得重新配置和编译，略微麻烦。



Figure 7: Buildroot

5.3.3 通过 ls3a7a 启动 Debian 根文件系统

下面介绍如何运行 Debian 发行版，Debian 发行版对 MIPS 架构的支持在所有发行版中是相对更为友好的，而且提供了非常丰富的软件仓库。

首先在主机系统中安装 qemu-user-static。

```
1 $ sudo apt-get install qemu-user-static
```

然后直接在主机上下载并导出 Docker 镜像：mips64le/debian

```
1 $ tools/root/docker/extract.sh mips64le/debian mips64le1
2 LOG: Pulling mips64le/debian
3 Using default tag: latest
4 latest: Pulling from mips64le/debian
5 91cc9e0aa8bc: Pulling fs layer
6 latest: Pulling from mips64le/debian
7 91cc9e0aa8bc: Pull complete
8 Digest: sha256:1bf10c81996745ddce4c972416318c3a46efd12258be2886001e0411d888c825
9 Status: Downloaded newer image for mips64le/debian:latest
10 docker.io/mips64le/debian:latest
11 LOG: Running mips64le/debian
12 LOG: Creating temporary rootdir: /media/falcon/develop/cloud-lab/labs/linux-lab/prebuilt/
    fullroot/tmp/mips64le-debian
13 LOG: Extract docker image to /media/falcon/develop/cloud-lab/labs/linux-lab/prebuilt/
    fullroot/tmp/mips64le-debian
```

该命令执行完以后会把镜像中的 Debian 解压到 prebuilt/fullroot/tmp/mips64le-debian。

然后安装一些必备工具并清空 root 帐号的密码：

```
1 $ tools/root/docker/chroot.sh mips64le/debian /bin/bash
2 root@ubuntu/# cat /etc/issue
3 Debian GNU/Linux 10 \n \l
4
5 root@ubuntu/# sed -ie "s/deb.debian.org/mirrors.163.com/g" /etc/apt/sources.list
6 root@ubuntu/# apt-get update
7 root@ubuntu/# apt-get install systemd net-tools
8
9 root@ubuntu/# passwd -d root
10
11 root@ubuntu/# hostname linux-lab
12 root@ubuntu/# echo linux-lab > /etc/hostname
13
14 root@ubuntu/# exit
```

接着，通过龙芯 ls3a7a 来启动：

```
1 $ make BOARD=ls3a7a
2 $ make boot ROOTFS=$PWD/prebuilt/fullroot/tmp/mips64le-debian ROOTDEV=nfs G=1
```

G=1 表示图形化启动，会使用 tty1 作为登陆控制台。

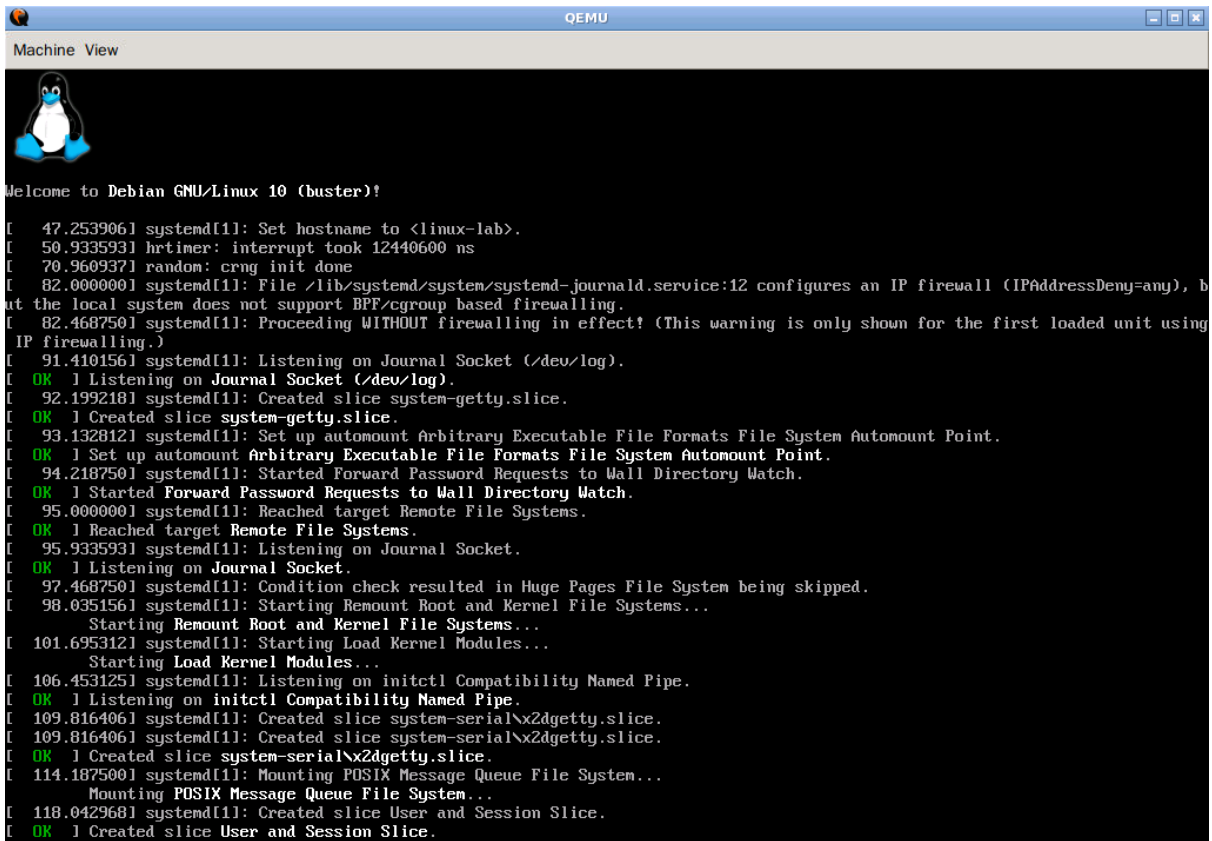
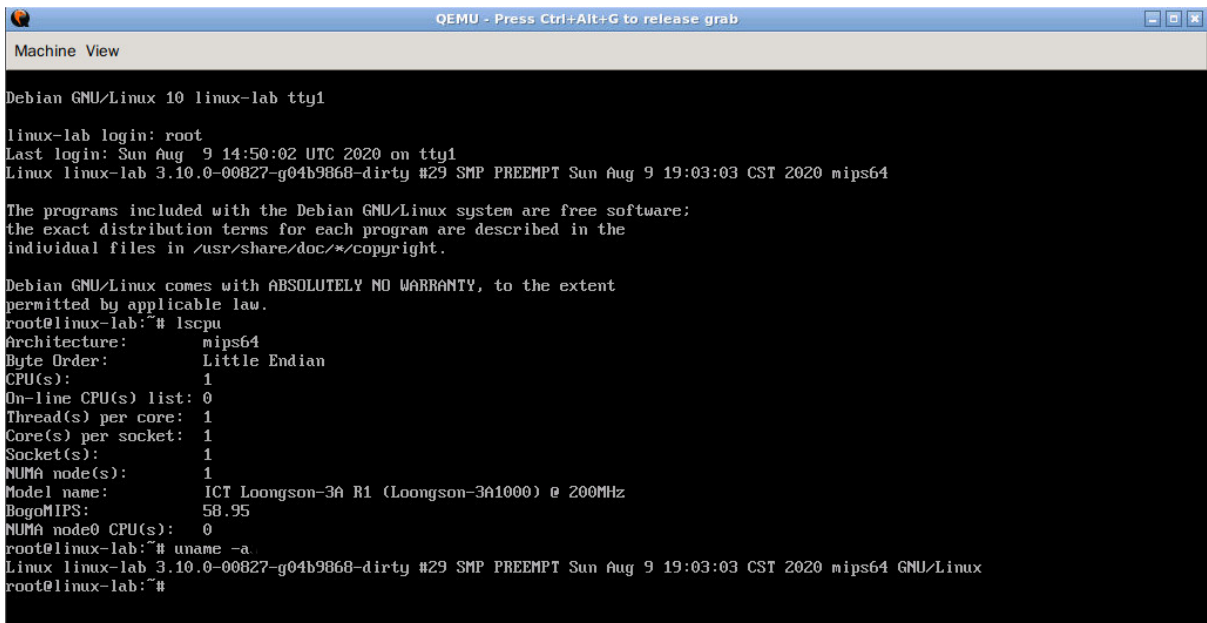


Figure 8: Linux Lab —— ls3a7a Debian 引导过程

如果 Qemu 窗口比 Webvnc 登陆的浏览器窗口要大，可以通过 CTRL+ALT+- 缩小 Qemu 窗口，也可以通过 View 菜单，选择 Zoom Out。

如果鼠标不小心点击进去了，可以根据 Qemu 窗口顶上的提示用 CTRL+ALT+G 切换出来。



```
QEMU - Press Ctrl+Alt+G to release grab
Machine View
Debian GNU/Linux 10 linux-lab tty1
linux-lab login: root
Last login: Sun Aug  9 14:50:02 UTC 2020 on tty1
Linux linux-lab 3.10.0-00827-g04b9868-dirty #29 SMP PREEMPT Sun Aug  9 19:03:03 CST 2020 mips64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@linux-lab:~# lscpu
Architecture:        mips64
Byte Order:          Little Endian
CPU(s):              1
On-line CPU(s) list: 0
Thread(s) per core: 1
Core(s) per socket: 1
Socket(s):           1
NUMA node(s):       1
Model name:          ICT Loongson-3A R1 (Loongson-3A1000) @ 200MHz
BogoMIPS:            58.95
NUMA node0 CPU(s):  0
root@linux-lab:~# uname -a
Linux linux-lab 3.10.0-00827-g04b9868-dirty #29 SMP PREEMPT Sun Aug  9 19:03:03 CST 2020 mips64 GNU/Linux
root@linux-lab:~#
```

Figure 9: Linux Lab —— ls3a7a Debian 登陆后

5.4 龙芯汇编语言实验

本节介绍如何开展龙芯汇编语言开发。龙芯基本兼容 MIPS 处理器架构，大部分指令都是兼容的。

5.4.1 Hello World

首先来看一个例子：

```

1  $ cd examples/assembly/mips64el
2
3  $ cat mips64el-hello.s
4  # File: hello.s -- "hello, world!" in MIPS Assembly Programming
5  # by falcon <wuzhangjin@gmail.com>, 2008/05/21
6  # refer to:
7  #   [*] http://www.tldp.org/HOWTO/Assembly-HOWTO/mips.html
8  #   [*] MIPS Assembly Language Programmer's Guide
9  #   [*] See MIPS Run Linux(second version)
10 # compile:
11 #   $ as -o hello.o hello.s
12 #   $ ld -e main -o hello hello.o
13
14 # data section
15 .rdata
16 hello: .asciiz "hello, world!\n"
17 length: .word . - hello          # length = current address - the string address
18
19 # text section
20 .text
21 .globl __start
22 __start:
23     # If compiled with gcc-4.2.3 in 2.6.18-6-qemu the following three statements are
24     # needed
25     # in compiling relocatable code, to follow the PIC-ABI calling conventions and other
26     # protocols.
27     .set noreorder
28     .cplod $gp
29     .set reorder
30
31     # There is no need to include regdef.h in gcc-4.2.3 in 2.6.18-6-qemu
32     # but you should use $a0, not a0, of course, you can use $4 directly
33     # print "hello, world!" with the sys_write system call,
34     # -- ssize_t write(int fd, const void *buf, size_t count);
35
36     li $a0, 1          # first argument: the standard output, 1
37     dla $a1, hello     # second argument: the string addr
38     lw $a2, length     # third argument: the string length
39     li $v0, 5001       # sys_write: system call number, defined as __NR_write in /usr/
40     include/asm/unistd.h
41     syscall           # causes a system call trap.
42     # exit from this program via calling the sys_exit system call
43     move $a0, $0       # or "li $a0, 0", set the normal exit status as 0
44     # you can print the exit status with "echo $?" after executing this
45     program
46     li $v0, 5058       # 4001 is __NR_exit defined in /usr/include/asm/unistd.h
47     syscall

```

代码中的注释已经比较清楚，直接来交叉编译运行：

```
1 $ make
2 mipsel-linux-gnu-as -mabi=64 -o mips64el-hello.o mips64el-hello.s
3 mipsel-linux-gnu-ld -m elf64ltsmip -o mips64el-hello mips64el-hello.o
4 # Need to register mips64el to /proc/sys/fs/binfmt_misc/register
5 # Based on /usr/share/binfmts/
6 # See the binfmt target, this must be run in host system, not guest
7 hello, world!
```

实际上，上面也可以通过 `qemu-mips64el` 来运行：

```
1 $ qemu-mips64el ./mips64el-hello
2 hello, world!
```

`qemu-mips64el` 是 `qemu-user` 的 `mips64el` 版本，可以用来直接翻译机器指令，把 `mips64el` 翻译为 `x86_64`。下一节会介绍如何编译 `qemu-system-mips64el` 和 `qemu-mips64el`。

当然，我们也可以把 `mips64el-hello` 拷贝到龙芯系统中运行：

```
1 $ cp /labs/linux-lab
2 $ cp examples/assembly/mips64el/mips64el-hello boards/mips64el/ls3a7a/bsp/root/2016.05/
   rootfs/root/
3
4 $ make boot ROOTDEV=nfs
5 ...
6 Welcome to Linux Lab
7 linux-lab login: root
8 #
9 # ls
10 mips64el-hello  tools
11 # ./mips64el-hello
12 hello, world!
13 #
```

5.4.2 memcpy & memset

上面介绍的是用户态的龙芯汇编语言例子，下面来看看内核里头的汇编。

内核中跟处理器相关的代码一般都放在 `arch/mips` 下面，初步看了下有 45 个汇编语言文件。

```
1 $ find loongnix-linux-3.10/arch/mips/ -name "*.S" | wc -l
2 45
```

而龙芯相关的定义放置在这里：

```
1 $ find loongnix-linux-3.10/arch/mips/loongson/ -name "*.S"
2 loongnix-linux-3.10/arch/mips/loongson/loongson-3/sleep.S
3 loongnix-linux-3.10/arch/mips/loongson/loongson-3/loongson3-memset.S
4 loongnix-linux-3.10/arch/mips/loongson/loongson-3/loongson3-memcpy.S
```

接下来，我们把上述针对龙芯加速过的 `memcpy()` 和 `memset()` 用到内核压缩代码中，稍作修改即可。

```
1 $ cd loongnix-linux-3.10/arch/mips/boot/compressed/
2 $ ln -sf ../../loongson/loongson-3/loongson3-memcpy.S loongson3-memcpy.S
3 $ ln -sf ../../loongson/loongson-3/loongson3-memset.S loongson3-memset.S
```

```
4
5 $ git diff
6 diff --git a/arch/mips/boot/compressed/Makefile b/arch/mips/boot/compressed/Makefile
7 index bbaa1d4..71f2d7c 100644
8 --- a/arch/mips/boot/compressed/Makefile
9 +++ b/arch/mips/boot/compressed/Makefile
10 @@ -30,6 +30,10 @@ targets := head.o decompress.o dbg.o uart-16550.o uart-alchemy.o
11 # decompressor objects (linked with vmlinux)
12 vmlinuxobjs-y := $(obj)/head.o $(obj)/decompress.o $(obj)/dbg.o
13
14 +ifdef CONFIG_CPU_LOONGSON3
15 +vmlinuxobjs-y += $(obj)/loongson3-memset.o $(obj)/loongson3-memcpy.o
16 +endif
17 +
```

重新编译内核并启动：

```
1 $ make kernel
2 $ make boot
3 ...
4 qemu-system-mips64el: warning: nic pci-synopgmac.1 has no peer
5 qemu-system-mips64el: warning: nic e1000e.0 has no peer
6 zimage at:      813019A0 81A0B029
7 Uncompressing Linux at load address 80200000
8 Now, booting the kernel...
9 SET HT_DMA CACHED
10 [ 0.000000] Initializing cgroup subsys cpu
11 [ 0.000000] Initializing cgroup subsys cpucct
12 ...
```

在真实的机器上，采用优化过的 `memcpy()` 和 `memset()` 之后，解压速度理论上应该会更好。

5.5 龙芯 Qemu 实验

龙芯已经开放了全系处理器的 Qemu 源代码，发布在：

- <https://gitee.com/loongsonlab/qemu>



Figure 10: Qemu

当前稳定分支为 loongson-v1.0，也是 Linux Lab 所采用的版本，不过同样地，Linux Lab 也添加了一些额外的补丁。

```
1 $ ls patch/qemu/loongson-v1.0/
2 0001-add-README.md.patch
3 0002-give-ls1x-clkreg-a-reasonable-default-to-avoid-kerne.patch
4 0003-ls3a2h-fix-axi-dma-and-pcie-dma-address-space.patch
5 0004-make-ls3a2h-default-blk-is-ide.patch
6 0005-ls3a7a-fix-ahci-irq.patch
7 0006-fix-ls3a-memenv.patch
8 0007-ls232-add-synopgmac-network-temporarily.patch
9 0008-enlarge-kernel-cmdline-size-from-256-bytes-to-1024-b.patch
10 0009-fix-warnings.patch
11 0010-comment-out-unused-variables.patch
12 0011-use-gitee-mirror-of-qemu-submodules.patch
13 0012-ls2k-fix-up-libfdt.h-include-issue.patch
```

5.5.1 编译龙芯 Qemu 系统模拟器：qemu-system-mips64el

传统方式下编译 Qemu 需要经过“千山万水”，各种配置、各种参数、各种库依赖、各种源代码下载不下来。

Linux Lab 下面仅需十几分钟就可以完成代码下载、配置、编译全过程，一条 `make qemu` 即可：

```
1 $ make qemu
2 Downloading qemu source ...
3
4 HEAD is now at 4475591... remove noused board ls1gp, ls1gpa.
5 /labs/linux-lab/boards/mips64el/ls3a7a/patch/qemu/loongson-v1/
6 /labs/linux-lab/boards/mips64el/ls3a7a/patch/qemu/loongson-v1.0/
7 /labs/linux-lab/boards/mips64el/ls3a7a/bsp/patch/qemu/loongson-v1/
8 /labs/linux-lab/boards/mips64el/ls3a7a/bsp/patch/qemu/loongson-v1.0/
```

```

 9 /labs/linux-lab/patch/qemu/loongson-v1/
10 /labs/linux-lab/patch/qemu/loongson-v1.0/
11 patching file README.md
12 patching file hw/mips/mips_ls1a.c
13 patching file hw/mips/mips_ls1b.c
14 patching file hw/mips/mips_ls1c.c
15 patching file hw/mips/mips_ls232.c
16 patching file hw/mips/mips_ls3a2h.c
17 patching file hw/mips/mips_ls3a2h.c
18 patching file hw/mips/mips_ls3a7a.c
19 patching file hw/mips/loongson_bootparam.c
20 ...
21 LDFLAGS      -Wl,--warn-common -Wl,-z,relro -Wl,-z,now -pie -m64 -g
22 QEMU_LDFLAGS -L$(BUILD_DIR)/dtc/libfdt
23 make         make
24 install      install
25 python       python -B
26 smbd         /usr/sbin/smbd
27 module support no
28 host CPU     x86_64
29 host big endian no
30 target list  mips64el-softmmu
31 gprof enabled no
32 sparse enabled no
33 ...
34 docker      no
35 tools/qemu/update-submodules.sh /labs/linux-lab/loongsonlab-qemu/.gitmodules
36 env PATH=/usr/bin/:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin make -C
   /labs/linux-lab/output/mips64el/qemu-loongson-v1.0-ls3a7a/ -j4 V=
37 make[1]: Entering directory `/labs/linux-lab/output/mips64el/qemu-loongson-v1.0-ls3a7a'
38 GEN         mips64el-softmmu/config-devices.mak.tmp
39 GEN         config-host.h
40 GIT         ui/keycodemapdb dtc capstone
41 GEN         mips64el-softmmu/config-devices.mak
42 GEN         qemu-options.def
43 GEN         qapi-gen
44 GEN         trace/generated-tcg-tracers.h
45 GEN         trace/generated-helpers-wrappers.h
46 ...
47 DEP /labs/linux-lab/loongsonlab-qemu/dtc/tests/asm_tree_dump.c
48 DEP /labs/linux-lab/loongsonlab-qemu/dtc/tests/truncated_property.c
49 DEP /labs/linux-lab/loongsonlab-qemu/dtc/tests/check_path.c
50 DEP /labs/linux-lab/loongsonlab-qemu/dtc/tests/overlay_bad_fixup.c
51 DEP /labs/linux-lab/loongsonlab-qemu/dtc/tests/overlay.c
52 DEP /labs/linux-lab/loongsonlab-qemu/dtc/tests/subnode_iterate.c
53 DEP /labs/linux-lab/loongsonlab-qemu/dtc/tests/property_iterate.c
54 ...
55 DEP dtc-parser.tab.c
56 DEP dtc-lexer.lex.c
57 ...
58 ar: creating /labs/linux-lab/output/mips64el/qemu-loongson-v1.0-ls3a7a/capstone/
   libcapstone.a
59 make[1]: Leaving directory `/labs/linux-lab/output/mips64el/qemu-loongson-v1.0-ls3a7a'
60 make[1]: Entering directory `/labs/linux-lab/output/mips64el/qemu-loongson-v1.0-ls3a7a'
61 GEN         qga/qapi-generated/qapi-gen
62 CC         qapi/qapi-builtin-types.o
63 CC         tests/qemu-iotests/socket_scm_helper.o
64 CC         qapi/qapi-types.o
65 CC         qapi/qapi-types-block-core.o
66 ...
67 CC         mips64el-softmmu/target/mips/cpu.o

```

```
68 CC      mips64el-softmmu/target/mips/gdbstub.o
69 CC      mips64el-softmmu/target/mips/msa_helper.o
70 CC      mips64el-softmmu/target/mips/mips-semi.o
71 CC      mips64el-softmmu/target/mips/machine.o
72 CC      mips64el-softmmu/target/mips/cp0_timer.o
73 GEN     trace/generated-helpers.c
74 CC      mips64el-softmmu/trace/control-target.o
75 CC      mips64el-softmmu/trace/generated-helpers.o
76 LINK    mips64el-softmmu/qemu-system-mips64el
77 make[1]: Leaving directory `/labs/linux-lab/output/mips64el/qemu-loongson-v1.0-1s3a7a'
```

编译完成后，可以从下面找到新的 qemu-system-mips64el：

```
1 $ ls output/mips64el/qemu-loongson-v1.0-1s3a7a/mips64el-softmmu/qemu-system-mips64el
2 output/mips64el/qemu-loongson-v1.0-1s3a7a/mips64el-softmmu/qemu-system-mips64el
```

后续 make boot 会自动检测并使用新编译的 Qemu。

5.5.2 增大龙芯 Qemu 中传递内核参数的 Buffer

本章开头提到，龙芯 Qemu 默认设定的内核参数 Buffer 太小，这里介绍一下如何加大并再次验证通过 NFS 引导开展测试。

修改代码，加大内核参数的 Buffer：

```
1 $ make | grep QEMU_SRC
2     QEMU_SRC = loongsonlab-qemu
3 $ cd loongsonlab-qemu
4
5 $ git diff patch/qemu/
6 /* i8254 PIT is attached to the IRQ0 at PIC i8259 */
7 @@ -400,7 +400,8 @@ static int set_bootparam(ram_addr_t initrd_offset,long initrd_size)
8     int ret;
9
10     /* Store command line. */
11 -     params_size = 264;
12 +#define PBUF_SIZE 4096
13 +     params_size = PBUF_SIZE + 8;
14     params_buf = g_malloc(params_size);
```

重新编译：

```
1 $ make qemu
```

通过 NFS 引导根文件系统，开展自动化测试：

```
1 $ make test m=hello TEST_INIT=0
```

TEST_INIT=0 表示直接开展测试，不做前期的其他无关准备工作。

说明：上述修改已合并到 patch/qemu/loongson-v1.0/0008-enlarge-kernel-commandline-size-from-256-bytes-to-4096-b.patch

5.5.3 编译龙芯 Qemu 指令集翻译工具：qemu-mips64el

如果要编译用于指令集翻译的 Qemu 工具，在 Linux Lab 下面也很方便，稍微调整一个设定即可。

首先，清理一下历史编译文件：

```
1 $ make qemu-clean
```

然后把 QEMU_US 设置为 1，即编译 Static 版本的 qemu-user：

```
1 $ make qemu QEMU_US=1
2 ...
3 CC      mips64el-linux-user/target/mips/cpu.o
4 CC      mips64el-linux-user/target/mips/gdbstub.o
5 CC      mips64el-linux-user/target/mips/msa_helper.o
6 CC      mips64el-linux-user/target/mips/mips-semi.o
7 GEN     trace/generated-helpers.c
8 CC      mips64el-linux-user/trace/control-target.o
9 CC      mips64el-linux-user/trace/generated-helpers.o
10 LINK   mips64el-linux-user/qemu-mips64el
11
12 $ ls output/mips64el/qemu-loongson-v1.0-ls3a7a/mips64el-linux-user/qemu-mips64el
13 output/mips64el/qemu-loongson-v1.0-ls3a7a/mips64el-linux-user/qemu-mips64el
```

同样可以通过这个工具来运行用户态的程序，以上面编译的汇编语言例子演示：

```
1 $ output/mips64el/qemu-loongson-v1.0-ls3a7a/mips64el-linux-user/qemu-mips64el \
2   examples/assembly/mips64el/mips64el-hello
3 hello, world!
```

5.6 其他实验

5.6.1 使用龙芯中科提供的工具链

部分龙芯相关的编译选项在老版本的 gcc 中可能不支持，如果需要进行某些特定指令集的开发，可能需要用到龙芯中科的工具链，那么如何切换呢？



Figure 11: GCC

目前 Linux Lab 中已经验证过一款，配置在 `prebuilt/toolchains/mips64el/Makefile` (loongnix)，不过这款也比较老，更多新版本可以通过下面的链接查看：

- <http://ftp.loongnix.org/toolchain/gcc/release/>

请注意新工具链的目录结构，包名可能都有差异，建议新增一个 `CCORI`，比如 `loongnix-7.3` 这样，并相应地调整相关变量：

变量	说明
<code>CCPRE</code>	编译器前缀
<code>CCVER</code>	版本号
<code>CCBASE</code>	包名，不包括后缀
<code>CCPATH</code>	解压后的 bin 路径
<code>LLPATH</code>	解压后的 lib 路径
<code>CCURL</code>	远程下载地址

查看验证过的工具链：

```
1 $ make gcc-list
2 Listing prebuilt toolchain ...
3
4 [ internal gcc-4.7 ]:
5
6 Remote.:
```

```
7 Local...: /usr/bin/
8 Tool...: mipsel-linux-gnu-gcc
9 Version: mipsel-linux-gnu-gcc (Debian 4.7.2-4) 4.7.2
10 More...: /usr/bin/mipsel-linux-gnu-gcc-4.3 /usr/bin/mipsel-linux-gnu-gcc-4.7
11
12 [ loongnix 4.4.7 ]:
13
14 Remote..: http://ftp.loongnix.org/toolchain/gcc/release/gcc-4.4.7-7215-n64-loongson.tar.gz
15 Local...: /labs/linux-lab/prebuilt/toolchains/mips64el/gcc-4.4.7-7215-n64-loongson/usr/bin
16 Tool...: mips64el-redhat-linux-gcc
17 Version: mips64el-redhat-linux-gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-3.1)
18
19 [ bootlin 2018.11-1 ]:
20
21 Remote..: https://toolchains.bootlin.com/downloads/releases/toolchains/mips64el-n32/tarballs/mips64el-n32--uclibc--stable-2018.11-1.tar.bz2
22 Local...: /labs/linux-lab/prebuilt/toolchains/mips64el/mips64el-n32--uclibc--stable-2018.11-1/bin
23 Tool...: mips64el-linux-gcc
24 Version: mips64el-linux-gcc.br_real (Buildroot 2018.08.1-00003-g576b333) 7.3.0
25
26 [ lemote 10 ]:
27
28 Remote..: http://mirror.lemote.com:8000/loongson3-toolchain/binaries/v10/mips64el-toolchain-10.x64.tar.xz
29 Local...: /labs/linux-lab/prebuilt/toolchains/mips64el/opt/mips64el-toolchain/bin
30 Tool...: mips64el-unknown-linux-gnu-gcc
31 Version: mips64el-unknown-linux-gnu-gcc (GCC) 5.5.0
```

更换为 loongnix 的 4.4.7:

```
1 $ make gcc-switch CCOR=loonginx
```

查看是否切换成功:

```
1 $ make gcc
2 [ loongnix 4.4.7 ]:
3
4 Remote..: http://ftp.loongnix.org/toolchain/gcc/release/gcc-4.4.7-7215-n64-loongson.tar.gz
5 Local...: /labs/linux-lab/prebuilt/toolchains/mips64el/gcc-4.4.7-7215-n64-loongson/usr/bin
6 Tool...: mips64el-redhat-linux-gcc
7 Version: mips64el-redhat-linux-gcc (GCC) 4.4.7 20120313 (Red Hat 4.4.7-3.1)
```

通过新配置的 gcc 来编译内核:

```
1 // 清除老的编译记录
2 $ make kernel-clean
3
4 // 重新编译
5 $ make kernel
6
7 // 引导新编译的内核
8 $ make boot
```

5.6.2 在主机和 Qemu 系统之间传输文件

前文其实已经介绍到类似的用法，最简单的莫过于:

- 首先，启动板子时通过 NFS 引导根文件系统
- 然后，主机和 Qemu 可以完全对等地通过 NFS 根文件系统所在目录读、写文件

前文我们介绍了如何把 `mips64el-hello` 复制到 Qemu 所在的根文件系统中，接下来反过来：在 Qemu 系统中把 `mips64el-hello` 的执行结果保存到 `/root` 目录下，然后在主机上读取该文件。

控制台一：

```
1 $ make boot ROOTDEV=nfs
2 ...
3 Welcome to Linux Lab
4 linux-lab login: root
5 #
6 # ls
7 mips64el-hello  tools
8 #
9 # ./mips64el-hello > mips64el-hello.result
10 # ls
11 mips64el-hello mips64el-hello.result tools
```

控制台二：

```
1 $ cat boards/mips64el/ls3a7a/bsp/root/2016.05/rootfs/root/mips64el-hello.result
2 hello, world!
```

5.6.3 用来开展《用“芯”探核》实验

《用“芯”探核：基于龙芯的 Linux 内核探索解析》是业内第一本基于国产龙芯处理器的 Linux 内核图书，采用 v5.x 内核，由官方 Linux 社区贡献者陈华才博士撰写。陈华才博士是龙芯 3A 系列处理器官方 Linux 内核的主要提交者、贡献者和维护者。

本书主要是以源码解读的方式来讲解的，如果想在阅读书籍的同时，对相关源码进行修改，比如验证自己的思考、实验新的算法和策略，那么正好可以通过 Linux Lab 来做实验，从而加强学习的效果。

本书基于航天龙梦的 v5.4 内核，该内核版本与 Linux Lab 目前验证过的 v3.10 和官方 Linux v5.7 版本略有差异，大家开展实验时，请注意相关差异。

由于 v5.4 跟 v3.10 差异较大，实验时建议大家使用 Linux Lab 验证过的较为接近的 Linux v5.7，不过由于当前主线 v5.7 还缺少对龙芯 7A1000 桥片的支持，所以无法开展磁盘和网卡相关实验，如果需要开展相关实验，可以切换到 v3.10。预计到 v5.9 时会有完整支持。

由于 v5.9 比较临近了，加上本书所采用的 v5.4 版本托管在 github 上，下载体验无法保障，Linux Lab 暂时不考虑支持该内核版本。

书中所讲解的 Linux 内核源码发布在如下仓库：

- <http://dev.lemote.com:8000/cgit/linux-official.git> 的 ulek 分支
- <https://github.com/chenuacai/linux-uлек.git> 的 master 分支

本书已经在京东等渠道发售，如需了解详情，请查阅：

- 十年磨一剑，第一本龙芯平台的 Linux 内核书来了！

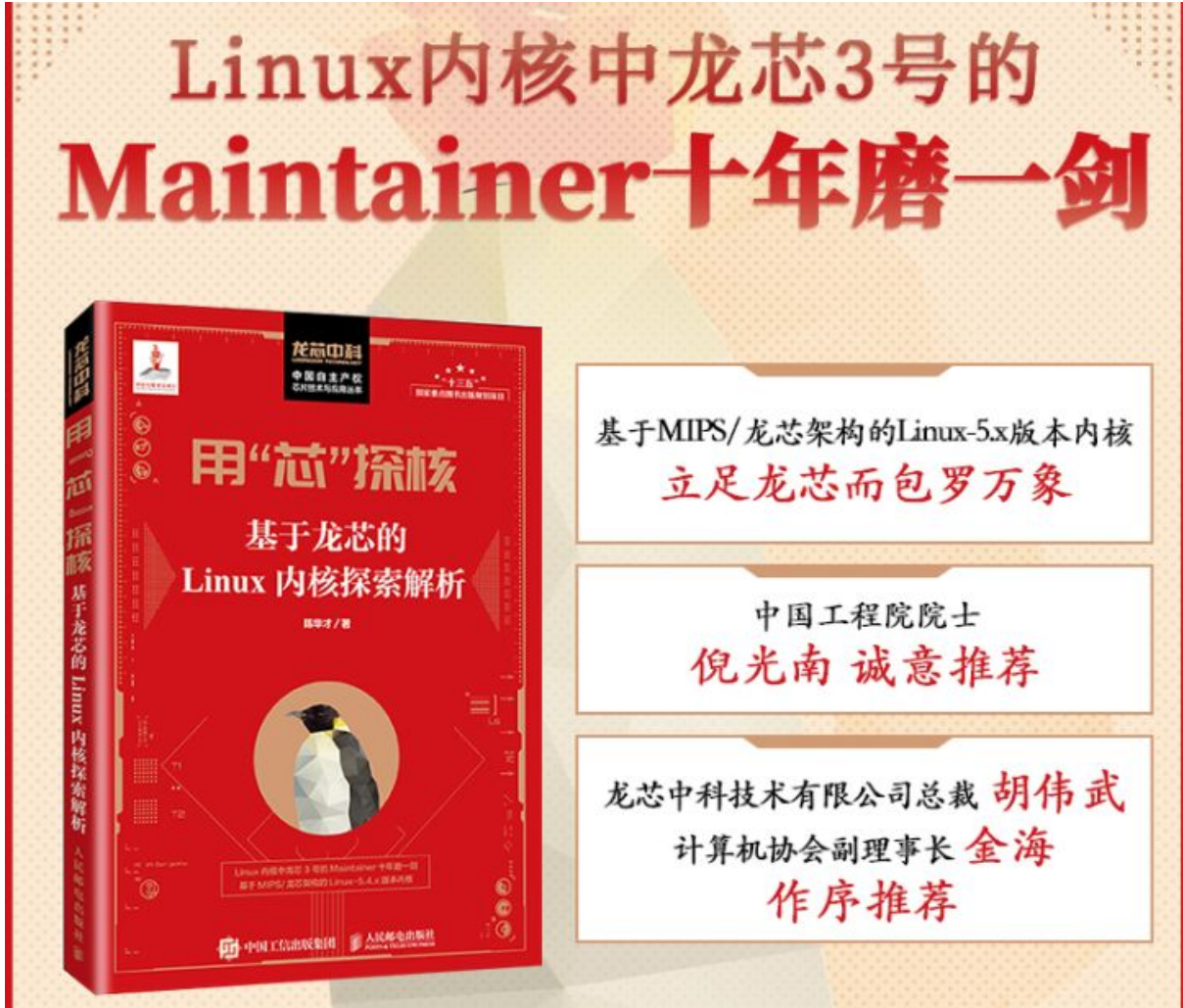


Figure 12: 用“芯”探核

6. 参考资料

6.1 Linux Lab 相关

- Linux Lab 中、英文文手册
 - [Linux Lab v0.5-rc2 中文手册](#)
 - [Linux Lab v0.5-rc2 英文手册](#)
 - [Linux Lab v0.4 中文手册](#)
 - [Linux Lab v0.4 英文手册](#)
- 使用 Linux Lab 的好处
 - [Linux Lab：难以抗拒的十大理由 v1.0](#)
 - [Linux Lab：难以抗拒的十大理由 v2.0](#)

6.2 龙芯相关

- 龙芯 3A4000 处理器手册
 - [数据手册](#)
 - [用户手册](#)
 - [向量指令软件开发手册](#)
- 龙芯 7A1000 桥片手册
 - [数据手册](#)
 - [用户手册](#)
- 龙芯 3A3000 处理器手册
 - [数据手册](#)
 - [用户手册（上）](#)
 - [用户手册（下）](#)
- 龙芯交叉工具链
 - <http://ftp.loongnix.org/toolchain/gcc/release/>

6.3 Linux 相关

- Linux 内核官方文档
 - <https://www.kernel.org/doc/html/latest/>
- Linux Weekly News

- <https://lwn.net>
- Linux Kernel Patchwork
 - <https://patchwork.kernel.org/>
- Linux Kernel Patch Statistic
 - http://www.remword.com/kps_result/