

# Build Linux System

Based on Ubuntu 18.04 & linux-5.0.1

孟宁



关注孟宁

# 下载Linux内核源代码

- <https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.0.1.tar.xz>
- `xz -d linux-5.0.1.tar.xz`
- `tar -xvf linux-5.0.1.tar`
- `cd linux-5.0.1`



linux-5.0.1.tar.xz下载地址

# 安装内核编译工具

- `sudo apt install build-essential flex bison libssl-dev libelf-dev libncurses-dev`

# 配置编译内核

- make defconfig #按照默认值生成.config
- make i386\_defconfig #生成32位x86的配置文件, x86\_64\_defconfig为64为配置
- make config #遍历选择编译内核功能
- make allyesconfig #启用内核全部功能
- make allnoconfig #内核功能选项全部为否
- make menuconfig #开启文本菜单选项, 对窗口有限制, 尽量调大窗口
- make 或 make -j\* # \*为cpu核心数

# 升级当前系统内核

- `sudo make modules_install` # ⚠️安装前通过系统快照备份系统，以防出现故障前功尽弃
- `sudo make install`
- `sudo update-grub`
- `reboot`
- `uname -a`
  - Linux ubuntu 5.0.1 #1 SMP Wed Mar 13 14:19:31 CST 2019 x86\_64 x86\_64 x86\_64 GNU/Linux

# 通过QEMU虚拟机加载内核

- `sudo apt install qemu`
- `qemu-system-i386 -kernel linux-5.0.1/arch/x86/boot/bzImage`
- `qemu-system-x86_64 -kernel linux-5.0.1/arch/x86/boot/bzImage`

# 构造MenuOS

- `git clone https://github.com/mengning/menu.git`
- `cd menu`
- `sudo apt-get install libc6-dev-i386` # 在64位环境下编译32位需安装
- `make rootfs`
- `cd ..`
- `qemu-system-i386 -kernel linux-5.0.1/arch/x86/boot/bzImage -initrd rootfs.img` # 配置内核  
`make i386_defconfig`





# 基于BusyBox构造Linux系统

- `wget https://busybox.net/downloads/busybox-1.30.1.tar.bz2`
- `tar -xvf busybox-1.30.1.tar.bz2`
- `make help`可以得到一些编译busybox的帮助信息
- `make defconfig`
- `make menuconfig`修改如下配置：
- enable: Settings → build options → build busybox as a static binary (no share libs)
- `make`

# 基于BusyBox构造Linux系统

- 准备根目录映像，并安装busybox到根目录映像中
- `dd if=/dev/zero of=rootfs.img bs=4096 count=1024`
- `mkfs.ext3 rootfs.img`
- `mkdir rootfs`
- `sudo mount -o loop rootfs.img rootfs`
- 在busybox目录下
- `sudo make CONFIG_PREFIX=../rootfs/ install`
- `sudo umount rootfs`
- `qemu-system-x86_64 -kernel linux-5.0.1/arch/x86/boot/bzImage -initrd rootfs.img -append "root=/dev/ram init=/bin/ash"`

# 构建Linux内核的gdb调试环境

- 重新配置编译内核使之携带调试信息
- 在qemu中启动gdb server
- 建立gdb与gdbserver之间的连接
- 加载vmlinux中的符号表，设置断点

# 重新配置编译内核使之携带调试信息

- make defconfig
- make menuconfig
  - Kernel hacking—>[\*] Kernel debugging
- make重新编译（时间较长）

# 在qemu中启动gdb server

- ◆ `qemu-system-x86_64 -kernel linux-5.0.1/arch/x86/boot/bzImage -initrd rootfs.img -append "root=/dev/ram init=/bin/ash" -s -S`
- ◆ 可以看到在新打开的qemu虚拟机上，整个是一个黑屏，此时qemu在等待gdb的连接
- ◆ 关于-s和-S选项的说明
  - -S freeze CPU at startup (use 'c' to start execution)
  - -s shorthand for -gdb tcp::1234 若不想使用1234端口，则可以使用-gdb tcp:xxxx来取代-s选项

# 建立gdb与gdbserver之间的连接

- 在另外一个终端运行gdb，然后在gdb界面中运行如下命令
- `target remote:1234` #则可以建立gdb和gdbserver之间的连接
- 按c 让qemu上的Linux继续运行
- 假如在前面使用`-gdb tcp::xxxx`，则这里的1234也要修改为对应的端口xxxx
- 问题：此时没有加载符号表，无法根据符号设置断点

# 加载vmlinux中的符号表， 设置断点

- 在gdb界面中targe remote之前加载符号表
- file linux-5.0.1/vmlinux
- 在gdb界面中设置断点
- break start\_kernel #断点的设置可以在target remote之前， 也可以在之后
- 在设置好start\_kernel处断点并且target remote之后可以继续运行， 则在运行到start\_kernel的时候会停下来， 等待gdb调试命令的输入， 可以使用list来显示断点处相关的源代码
- 此后可以继续设置新的断点， ...