Linux 5.0 奔跑吧实验平台使用说明

注意：本实验建立在了安装 Ubuntu 18.04.2 版本的 x86_64 台式机或者 PC 机上。若使用其他版本的 ubuntu 或者发行版，遇到问题请自行解决。特别需要注意 qemu 的版本不能太老。

```
root@ubuntu:/home/runninglinuxkernel-5.0# ./run_debian_arm64.sh run
WARNING: Image format was not specified for 'rootfs_debian_arm64.ext4' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
qemu-system-aarch64: -device virtio-9p-pci,fsdev=kmod_dev,mount_tag=kmod_mount: 'virtio-9p-pci' is not a valid device model name
```

有小伙伴反馈，使用 ubuntu 18.04 版本的 ubuntu 会出现 Qemu 不能运行的问题，但是使用 Ubuntu 18.0.4.2 就没问题。因此，建议大家升级 ubuntu 18.04 到最新版本，或者直接下载 ubuntu 18.0.4.2 的 image 来安装。

```
figo@figo-OptiPlex-9020:~$ lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.2 LTS
Release:        18.04
Codename:       bionic
figo@figo-OptiPlex-9020:~$
```

使用 busybox 工具制作的最小文件系统，该最小系统仅仅包含了 Linux 系统最常用的命令，如 ls，top 等命令。如果要在此最小系统中进行 systemtap 以及 kdump 等试验的话，我们需要手动编译和安装这些工具，这个过程是相当复杂和繁琐的。为此，我们尝试使用 Debian 的根文件系统来构造一个小巧而且好用的实验平台。在这个实验平台中，读者可以在线安装丰富的软件包，比如 kdump，crash，systemtap 等工具。这个实验平台具有如下特点：

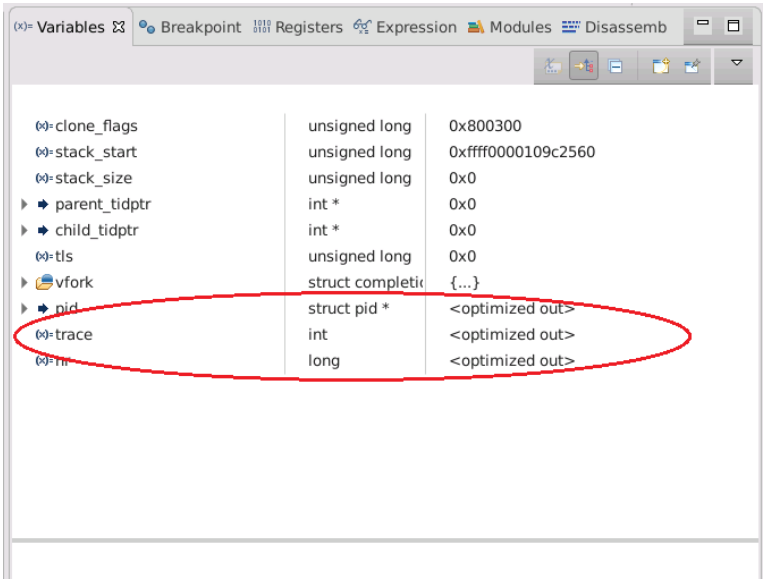
- **使用“O0”来编译内核**
- 在主机 Host 中编译内核
- 使用 QEMU 来加载系统
- 支持 GDB 单步调试内核以及 Debian 系统
- 使用 ARM64 版本的 Debian 系统的根文件系统
- 在线安装 Debian 软件包
- 支持在虚拟机里动态编译内核模块
- 支持 Host 主机和虚拟机共享文件

1. 使用 O0 编译内核的好处

这个 runninglinuxkernel 内核默认使用 GCC 的“O0”优化等级来编译的。***读者可能发现 gdb 在单步调试内核时会出现光标乱跳并且无法打印有些变量的值（例如出现<optimized out>）等问题***，其实这不是 gdb 或 QEMU 的问题。是因为内核编译的默认优化选项是 O2，因此如果不希望光标乱跳，可以尝试把 linux-5.0 根目录 Makefile 中的 O2 改成 O0，但是这样编译时有问题，作者为此做了一些修改。最后需要特别说明

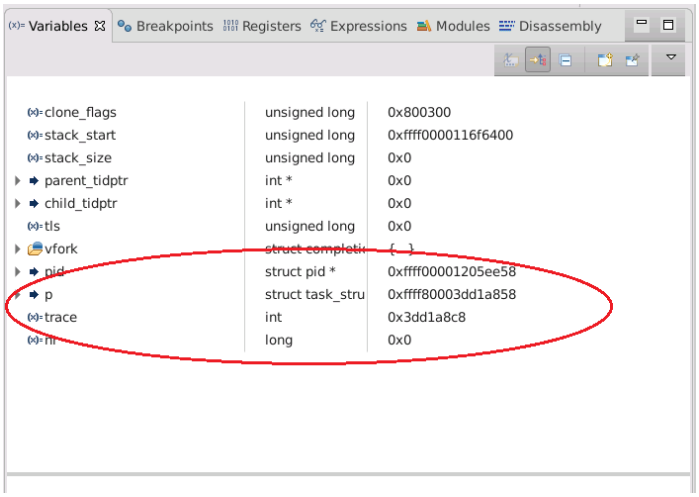
一下，使用 GCC 的“O0”优化等级编译内核会导致内核运行性能下降，因此我们仅仅是为了方便单步调试内核。

使用 O1 或者 O2 编译的内核，使用 gdb 进行单步调试时查看变量会出现大量的<optimized out>。



使用O2编译的内核

使用 O0 来编译内核，在使用 gdb 调试时就不会出现<optimized out>问题。



使用00编译内核

2. Qemu+debian 系统

（1）安装工具

首先在 Ubuntu Linux 18.04 中安装如下工具。

```
$ sudo apt-get install qemu libncurses5-dev gcc-aarch64-linux-gnu build-essential git bison flex libssl-dev
```

（3）编译内核以及制作文件系统

在 runninglinuxkernel 目录下面有一个 rootfs_debian.tar.xz 文件，这个是基于 ARM64 版本的 Debian 系统的根文件系统。但是这个根文件系统还只是一个半成品，我们还需要根据编译好的内核来安装内核镜像和内核模块。整个过程比较复杂：

- 编译内核
- 编译内核模块
- 安装内核模块
- 安装内核头文件
- 安装编译内核模块必须依赖文件
- 制作 ext4 根文件系统

这个过程比较繁琐，作者制作了一个脚本来简化上述过程。

注意，该脚本会使用 dd 命令来生成一个 8GB 大小的镜像文件，因此主机系统需要保证至少需要 10 个 GB 的空余磁盘空间。若读者需要生成一个更大的根文件系统镜像，可以修该 run_debian_arm64.sh 这个脚本文件。

首先编译内核。

```
$ cd runninglinuxkernel-5.0
$ ./run_debian_arm64.sh build_kernel
```

执行上述脚本需要几十分钟，依赖于主机的计算能力。

然后编译文件系统。

```
$ cd runninglinuxkernel-5.0
$ sudo ./run_debian_arm64.sh build_rootfs
```

注意编译文件系统需要 root 权限，编译内核不需要。

（3）运行刚才编译好的 ARM64 版本的 Debian 系统。

运行 run_debian_arm64.sh 脚本，输入 run 参数即可。

```
$ ./run_debian_arm64.sh run
```

运行的结果如下：

```
$ ./run_debian_arm64.sh run
EFI stub: Booting Linux Kernel...
EFI stub: EFI_RNG_PROTOCOL unavailable, no randomness supplied
EFI stub: Using DTB from configuration table
EFI stub: Exiting boot services and installing virtual address map...
[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x411fd070]
[    0.000000] Linux version 5.0.0+ (root@figo-OptiPlex-9020) (gcc version
5.5.0 20171010 (Ubuntu/Linaro 5.5.0-12ubuntu1)) #1 SMP Mon Apr 22
05:40:30 CST 2019
[    0.000000] Machine model: linux,dummy-virt
[    0.000000] efi: Getting EFI parameters from FDT:
[    0.000000] efi: EFI v2.60 by EDK II
[    0.000000] efi:  SMBIOS 3.0=0xbbeb0000  ACPI=0xbc030000  ACPI
2.0=0xbc030014  MEMATTR=0xbd8ca018  MEMRESERVE=0xbbd5f018
[    0.000000] crashkernel reserved: 0x000000009c800000 -
0x00000000bbc00000 (500 MB)
[    0.000000] cma: Reserved 64 MiB at 0x0000000098800000
[    0.000000] NUMA: No NUMA configuration found
[    0.000000] NUMA: Faking a node at [mem 0x0000000040000000-
```

```
0x00000000bfffffff]
[    0.000000] NUMA: NODE_DATA [mem 0xbfbf2840-0xbfbf3fff]
[    0.000000] Zone ranges:
[    0.000000]   DMA32    [mem 0x0000000040000000-0x00000000bfffffff]
[    0.000000]   Normal   empty
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x0000000040000000-0x00000000bbd5ffff]
[    0.000000]   node   0: [mem 0x00000000bbd60000-0x00000000bbffffff]
[    0.000000]   node   0: [mem 0x00000000bc000000-0x00000000bc03ffff]
[    0.000000]   node   0: [mem 0x00000000bc040000-0x00000000bc1d3fff]
[    0.000000]   node   0: [mem 0x00000000bc1d4000-0x00000000bf4affff]
[    0.000000]   node   0: [mem 0x00000000bf4b0000-0x00000000bf53ffff]
[    0.000000]   node   0: [mem 0x00000000bf540000-0x00000000bf54ffff]
[    0.000000]   node   0: [mem 0x00000000bf550000-0x00000000bf66ffff]
[    0.000000]   node   0: [mem 0x00000000bf670000-0x00000000bfffffff]
[    0.000000] Zeroed struct page in unavailable ranges: 884 pages
[    0.000000] Initmem setup node 0 [mem 0x0000000040000000-
0x00000000bfffffff]
Welcome to Debian GNU/Linux buster/sid!
[  OK  ] Reached target Network is Online.
[  OK  ] Started LSB: Load kernel image with kexec.
[  OK  ] Started Permit User Sessions.
[  OK  ] Started Serial Getty on ttyAMA0.
[  OK  ] Started Getty on tty1.
[  OK  ] Reached target Login Prompts.
[  OK  ] Started DHCP Client Daemon.
[  OK  ] Started Online ext4 Metadata Check for All Filesystems.
[   13.406564] kdump-tools[330]: Starting kdump-tools: Creating symlink
/var/lib/kdump/vmlinuz.
[   13.454300] kdump-tools[330]: Creating symlink
/var/lib/kdump/initrd.img.
[   15.721642] kdump-tools[330]: loaded kdump kernel.
[  OK  ] Started Kernel crash dump capture service.

Debian GNU/Linux buster/sid benshushu ttyAMA0

benshushu login:
```

登录 Debian 系统：

- 用户名：root 或者 benshushu
- 密码：123

（4）在线安装软件包。

QEMU 虚拟机可以通过 VirtIO-NET 技术来生成一个虚拟的网卡，并且通过 NAT 网络桥接技术和主机进行网络共享。首先使用 ifconfig 命令来检查网络配置。

```
root@benshushu:~# ifconfig
enp0s1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fec0::ce16:adb:3e70:3e71  prefixlen 64  scopeid 0x40<site>
        inet6 fe80::c86e:28c4:625b:2767  prefixlen 64  scopeid 0x20<link>
        ether 52:54:00:12:34:56  txqueuelen 1000  (Ethernet)
        RX packets 23217  bytes 33246898 (31.7 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4740  bytes 267860 (261.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
```

```
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 2  bytes 78 (78.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 2  bytes 78 (78.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

可以看到生成了一个名为 enp0s1 的网卡设备，分配的 IP 地址为：10.0.2.15。

通过 apt update 命令来更新 Debian 系统的软件仓库。

```
$ apt update
```

如果更新失败，有可能是系统时间比较旧了，可以使用 date 命令来设置日期。

```
root@benshushu:~# date -s 2019-04-25 #假设最新日期是2019年4月25日
Thu Apr 25 00:00:00 UTC 2019
```

使用 apt install 命令来安装软件包。比如，可以在线安装 gcc。

```
root@benshushu:~# apt install gcc
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  cpp cpp-8 gcc-8 libasan5 libatomic1 libc-dev-bin libc6-dev libcc1-0
  libgcc-8-dev libgomp1 libisl19 libitm1 liblsan0 libmpc3 libmpfr6
libtsan0
  libubsan1 linux-libc-dev manpages manpages-dev
Suggested packages:
  cpp-doc gcc-8-locales gcc-multilib make autoconf automake libtool flex
bison
  gdb gcc-doc gcc-8-doc libgcc1-dbg libgomp1-dbg libitm1-dbg libatomic1-
dbg
  libasan5-dbg liblsan0-dbg libtsan0-dbg libubsan1-dbg libmpx2-dbg
  libquadmath0-dbg glibc-doc man-browser
The following NEW packages will be installed:
  cpp cpp-8 gcc gcc-8 libasan5 libatomic1 libc-dev-bin libc6-dev libcc1-0
  libgcc-8-dev libgomp1 libisl19 libitm1 liblsan0 libmpc3 libmpfr6
libtsan0
  libubsan1 linux-libc-dev manpages manpages-dev
0 upgraded, 21 newly installed, 0 to remove and 17 not upgraded.
Need to get 25.6 MB of archives.
After this operation, 86.4 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

（4）主机和 QEMU 虚拟机之间共享文件。

主机和 QEMU 虚拟机可以通过 NET_9P 技术进行文件共享，这个需要 QEMU 虚拟机的 Linux 内核使能 NET_9P 的内核模块。本实验平台已经支持主机和 QEMU 虚拟机的共享文件，可以通过如下简单方法来测试。

复制一个文件到 runninglinuxkernel-5.0/kmodules 目录下面。

```
$cp test.c  runninglinuxkernel-5.0/kmodules
```

启动 QEMU 虚拟机之后，首先检查一下/mnt 目录是否有 test.c 文件。

```
/ # cd /mnt/
/mnt # ls
README    test.c
/mnt #
```

我们在后续的实验中会经常利用这个特性，比如把编译好的内核模块或者内核模块源代码放入 QEMU 虚拟机。

（5） 在主机上交叉编译内核模块。

在本书中，常常需要编译内核模块然后放入到 QEMU 虚拟机中加载内核模块。我们这里提供两种编译内核模块的方法，一种是在主机上通过交叉编译，然后共享到 QEMU 虚拟机，另外一个方法是在 QEMU 虚拟机里进行本地编译。

读者可以编写一个最简单的“hello world”的内核模块，也可以参考本章的其他内核模块的例子，我们在这里简单介绍主机交叉编译内核模块的方法。

```
$ cd hello_world  #进入内核模块代码目录
$ export ARCH=arm64
$ export CROSS_COMPILE=aarch64-linux-gnu-
```

编译内核模块。

```
$ make
```

把内核模块 ko 文件拷贝到 runninglinuxkernel-5.0/kmodules 目录下面。

```
$cp test.ko  runninglinuxkernel-5.0/kmodules
```

在 QEMU 虚拟机里的 mnt 目录可以看到这个 test.ko 模块。加载该内核模块。

```
$ insmod test.ko
```

（6） 在 QEMU 虚拟机上本地编译内核模块。

在 QEMU 虚拟机中安装必要的软件包。

```
root@benshushu: # apt install build-essential
```

编译内核模块。

```
root@benshushu:/mnt/hello_world# make
make -C /lib/modules/5.0.0+/build M=/mnt/hello_world modules;
make[1]: Entering directory '/usr/src/linux'
  CC [M]  /mnt/hello_world/test-1.o
  LD [M]  /mnt/hello_world/test.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /mnt/hello_world/test.mod.o
  LD [M]  /mnt/hello_world /test.ko
make[1]: Leaving directory '/usr/src/linux'
root@benshushu: /mnt/hello_world#
```

加载内核模块。

```
root@benshushu:/mnt/hello_world# insmod test.ko
```

3. kdump 使用

1. 在 QEMU+Debian 实验平台中，作者已经把 kdump 服务配置在 Debian 的根文件系统中，读者只需要按照第 6.1.3 章的步骤来构建一个文件系统镜像即可。
2. 在 QEMU 虚拟机中检查 kdump 服务是否开启。
   使用 systemctl status kdump-tools 命令来查看 kdump 服务是否正常工作。

```
root@benshushu:~# systemctl status kdump-tools
● kdump-tools.service - Kernel crash dump capture service
   Loaded: loaded (/lib/systemd/system/kdump-tools.service; enabled; vendor pres
   Active: active (exited) since Wed 2019-05-29 14:25:20 UTC; 1min 12s ago
  Process: 283 ExecStart=/etc/init.d/kdump-tools start (code=exited, status=0/SU
 Main PID: 283 (code=exited, status=0/SUCCESS)

May 29 14:25:12 benshushu systemd[1]: Starting Kernel crash dump capture service
May 29 14:25:16 benshushu kdump-tools[283]: Starting kdump-tools: Creating symli
May 29 14:25:16 benshushu kdump-tools[283]: Creating symlink /var/lib/kdump/init
May 29 14:25:19 benshushu kdump-tools[283]: loaded kdump kernel.
May 29 14:25:20 benshushu systemd[1]: Started Kernel crash dump capture service.
```

图6.x 检查kdump服务

3. 编译一个简单的死机例子。加载模块的时候会触发重启，进入捕获内核，打印一句“Starting Crashdump kernel…”，或者输入”echo c > /proc/sysrq-trigger”来触发一个内核异常。

```
[  466.804320] Process insmod (pid: 3958, stack limit = 0x00000000e2020a2e)
[  466.804726] Call trace:
[  466.804956]  create_oops+0x20/0x4c [oops]
[  466.805156]  my_oops_init+0xa0/0x1000 [oops]
[  466.805681]  do_one_initcall+0x54/0x1d8
[  466.805867]  do_init_module+0x60/0x1f0
[  466.806042]  load_module.isra.34+0x1be4/0x1e20
[  466.806236]  __se_sys_finit_module+0xa0/0xf8
[  466.806431]  __arm64_sys_finit_module+0x24/0x30
[  466.806681]  el0_svc_common+0x94/0x108
[  466.806918]  el0_svc_handler+0x38/0x78
[  466.807113]  el0_svc+0x8/0xc
[  466.807466] Code: f9000be1 aa0203e0 d503201f f9400fe0 (f9402800)
[  466.808507] SMP: stopping secondary CPUs
[  466.809596] Starting crashdump kernel...
[  466.809892] Bye!
```

进入捕获内核之后，会调用 makedumpfile 进行内核信息转存。转存完成之后，自动重启到生产内核。

```
[  OK  ] Started Raise network interfaces.
[  OK  ] Reached target Network.
[  OK  ] Reached target Network is Online.
         Starting Kernel crash dump capture service...
[   34.429764] kdump-tools[330]: Starting kdump-tools: running makedumpfile -c -d 31 /proc/vmcore /var/crash/201903270619/d
ump-incomplete.
Copying data                                : [ 25.9 %] /         eta: 63s
```

4. 在主机 Linux 中，拷贝带调试符号信息的 vmlinux 文件到共享文件夹 kmodules 目录。
   在 QEMU 虚拟机的 mnt 目录可以访问到该文件。
5. 在 QEMU 虚拟机中，启动 crash 工具进行分析。
   进入/var/crash/目录。转存的目录是以日期来命名，这一点和 Centos 系统略有不同。
   使用 crash 命令来加载内核转存文件。

```
root@benshushu:/var/crash# ls
201904221429  kexec_cmd
root@benshushu:/var/crash#

root@benshushu:/var/crash/201904221429# crash dump.201904221429
/mnt/vmlinux
```

```
          KERNEL: /mnt/vmlinux
        DUMPFILE: dump.201904221429  [PARTIAL DUMP]
            CPUS: 4
            DATE: Mon Apr 22 14:28:49 2019
          UPTIME: 00:00:13
    LOAD AVERAGE: 0.47, 0.31, 0.13
           TASKS: 87
        NODENAME: benshushu
         RELEASE: 5.0.0+
         VERSION: #1 SMP Mon Apr 22 05:40:30 CST 2019
         MACHINE: aarch64  (unknown Mhz)
          MEMORY: 2 GB
           PANIC: "Unable to handle kernel NULL pointer dereference at
virtual address 0000000000000050"
             PID: 1243
         COMMAND: "insmod"
            TASK: ffff800052d0c600  [THREAD_INFO: ffff800052d0c600]
             CPU: 0
           STATE: TASK_RUNNING (PANIC)

    crash>
```

4. QEMU 调试 ARM Linux 内核

由于 Ubuntu 18.04 里没有 arm64 版本的 GBD 工具，因此我们需要下载 GDB 源代码进行编译。

安装必要的软件依赖包。

```
sudo apt build-dep gdb
```

下载 GDB 源代码：gdb-8.2.tar.xz，并解压。

```
   $ xz -d gdb-8.2.tar.xz
   $ tar xf gdb-8.2.tar
```

编译 GDB。

```
   $ cd gdb-8.2
   $ ./configure --target=aarch64-linux-gnu
   $ make
   $ sudo make install
```

在超级终端中输入如下内容。

```
$ ./run_debian_arm64.sh run debug
   Enable qemu debug server
```

然后在另外一个超级终端中启动 ARM GDB。

```
   $ cd runninglinuxkenrel-5.0
   $ aarch64-linux-gnu-gdb --tui vmlinux
   (gdb) target remote localhost:1234      <= 通过1234端口远程连接到QEMU平台
   (gdb) b start_kernel                    <= 在内核的start_kernel处设置断点
   (gdb) c
```

如图 6.1 所示，GDB 开始接管 Linux 内核运行，并且到断点中暂停，这时即可使用 GDB 命令来调试内核。

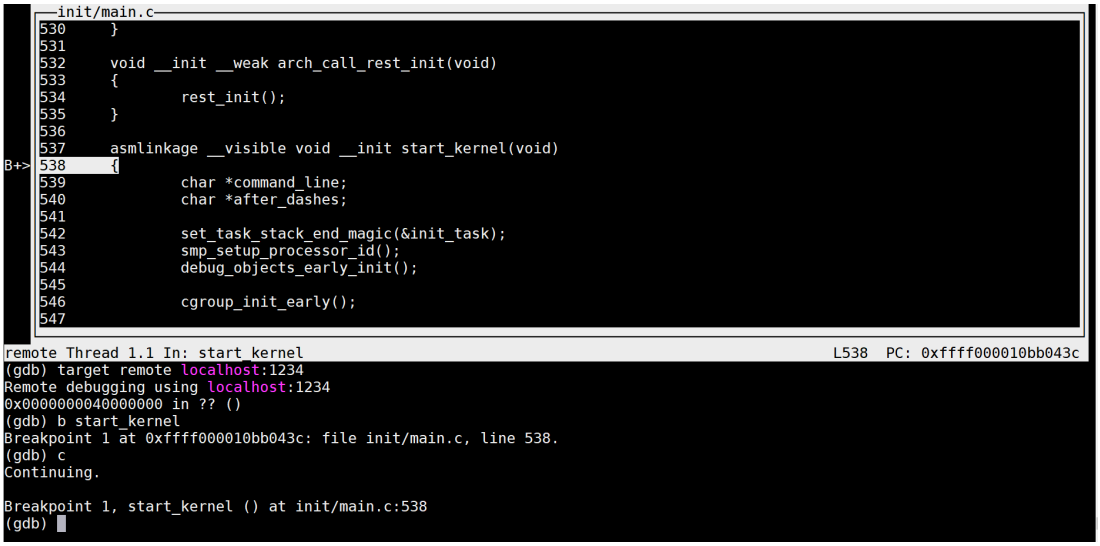


图6.1 gdb调试内核

5. 图形化单步调试内核

前文中介绍了如何使用 gdb 和 QEMU 调试 Linux 内核源代码。由于 gdb 是命令行的方式，可能有些读者希望在 Linux 中能有类似 Virtual C++图形化的开发工具，这里介绍使用 Eclipse 工具来调试内核。Eclipse 是著名的跨平台的开源集成开发环境（IDE），最初主要用于 JAVA 语言开发，目前可以支持 C/C++、Python 等多种开发语言。Eclipse 最初由 IBM 公司开发，2001 年贡献给开源社区，目前有很多集成开发环境都是基于 Eclipse 完成的。

（1）在主机上安装 Eclipse-CDT 软件

Eclipse-CDT 是 Eclipse 的一个插件，可以提供强大的 C/C++编译和编辑功能。

```
$ sudo apt install openjdk-11-jdk eclipse-cdt[1]
```

打开 Eclipse 菜单，选择“Help”→“About Eclipse”，可以看到当前软件的版本，如图 3.6 所示。

1 截至 2019 年 4 月，Ubuntu 18.04 系统默认安装的 Eclipse CDT 工具不能运行，读者可以到 Eclipse CDT 官网上直接下载 CDT 9.7.0 版本 x86_64 的 Linux 版本压缩包，解压并打开二进制文件即可。

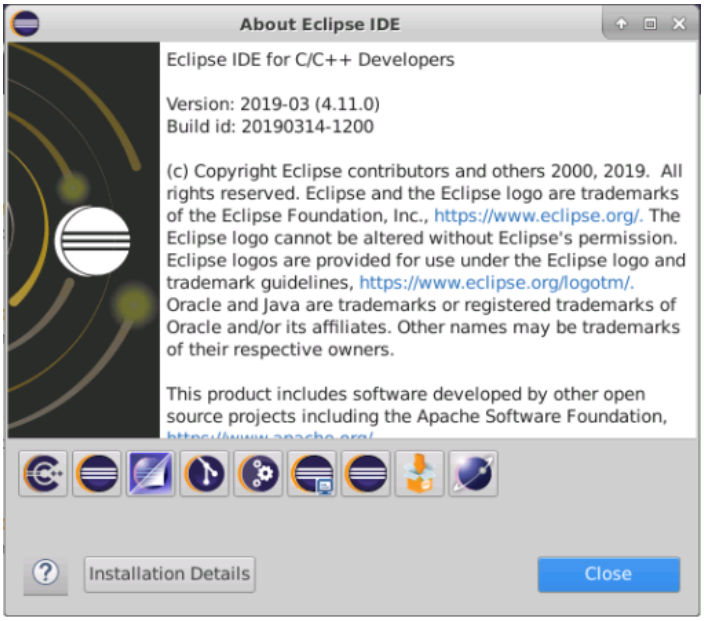


图3.6　Eclipse CDT 9.7.0版本

（2）创建工程

打开 Eclipse 菜单，选择“Window”→“Open Perspective”→“C/C++”。新建一个 C/C++ 的 Makefile 工程，在“File”→“New”→“Project”中选择“Makefile Project with Exiting Code”，创建一个新的工程，如图 3.7 所示。

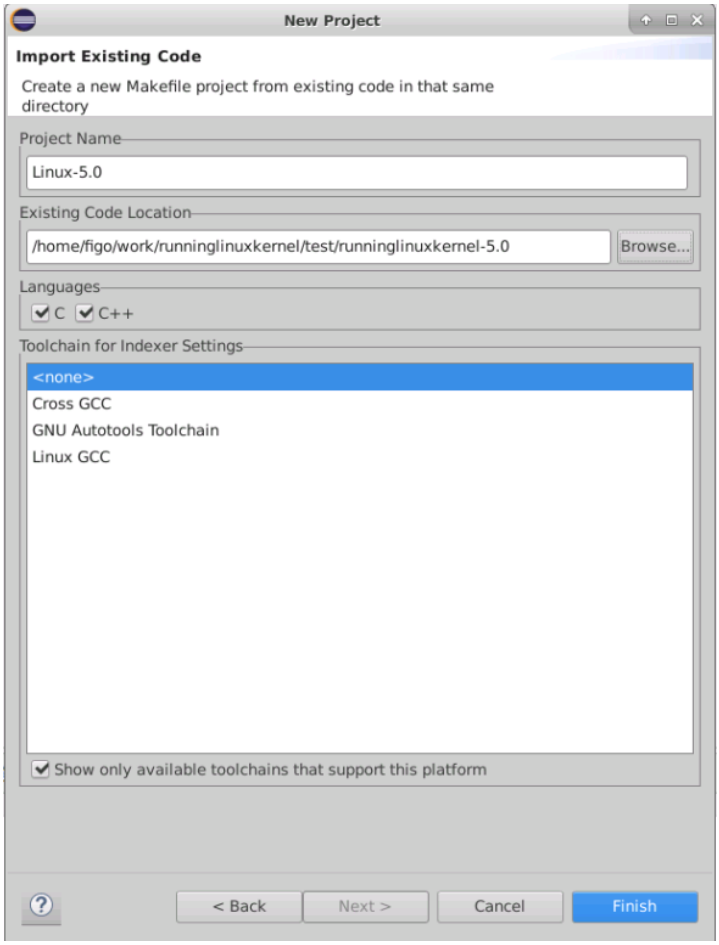


图3.7　创建工程

接下来配置调试选项。选择 Eclipse 菜单中的“Run”→“Debug Configurations”选项，创建一个“C/C++ Attach to Application”调试选项。

在 main 标签页里：

- Project：选择刚才创建的工程。
- C/C++ Appliction：选择编译 Linux 内核带符号表信息的 vmlinux 文件。
- Build befor launching：选择“Disable auto build”，如图 3.8 所示。

在 Debugger 标签页里：

- Debugger：选择 gdbserver。
- GDB debugger：填入 aarch64-linux-gnu-gdb，如图 3.9 所示。

在 debugger 标签页的 connection 子标签页里：

- Host name or IP addrss：填入 localhost。
- Port number：填入 1234。

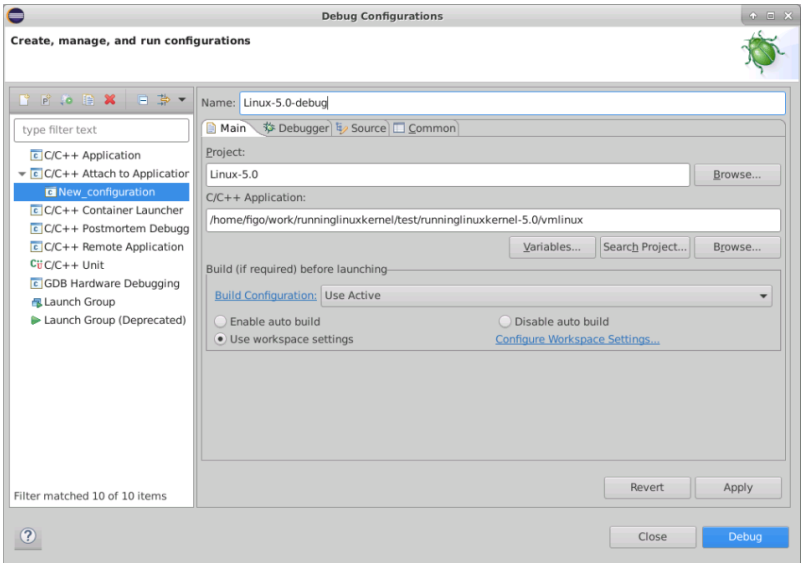


图3.8　debug配置选项

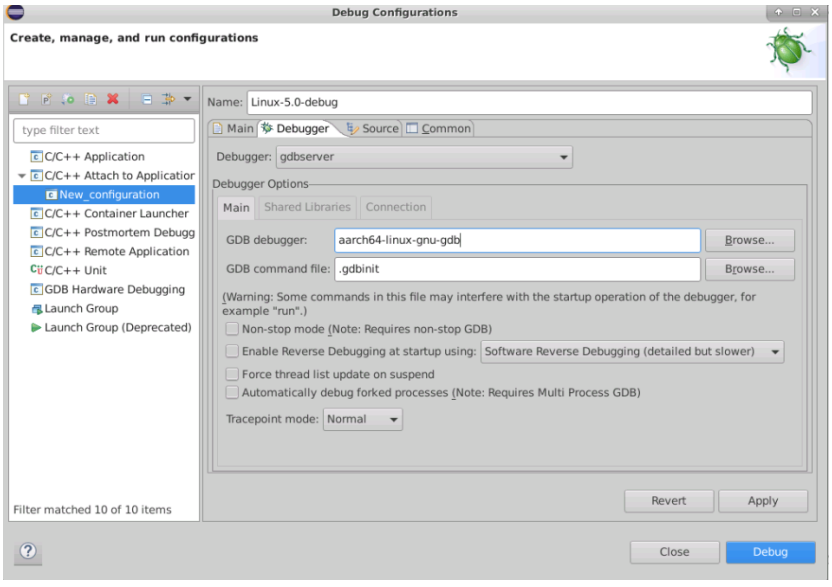


图3.9　debugger配置

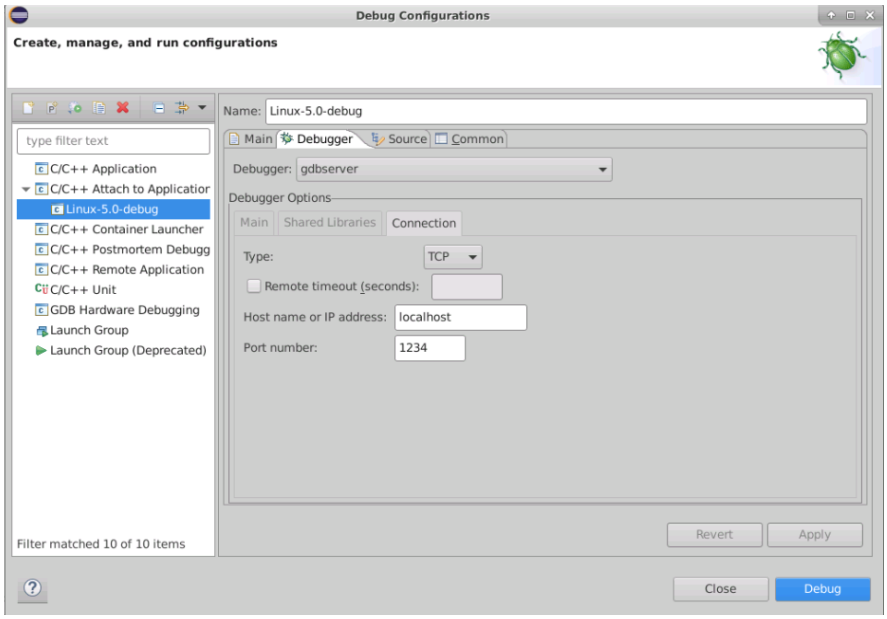


图6.x debugger连接选项

调试选项设置完成后，单击“Debug”按钮。

在 Linux 主机的一个终端中先打开 QEMU。为了调试方便，这里没有指定多个 CPU，而是单个 CPU。

```
$ cd runninglinuxkernel-5.0
$ ./run_debian_arm64.sh run debug
```

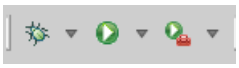


图3.10　“小昆虫”图标

在 Eclipse 菜单的“Run”→“Debug History”中选择刚才创建的调试选项，或在快捷菜单中单击“小昆虫”图标，如图 3.10 所示。

在 Eclipse 的 Debugger Console 控制台中输入“file vmlinux”命令，导入调试文件的符号表，如图 3.11 所示。

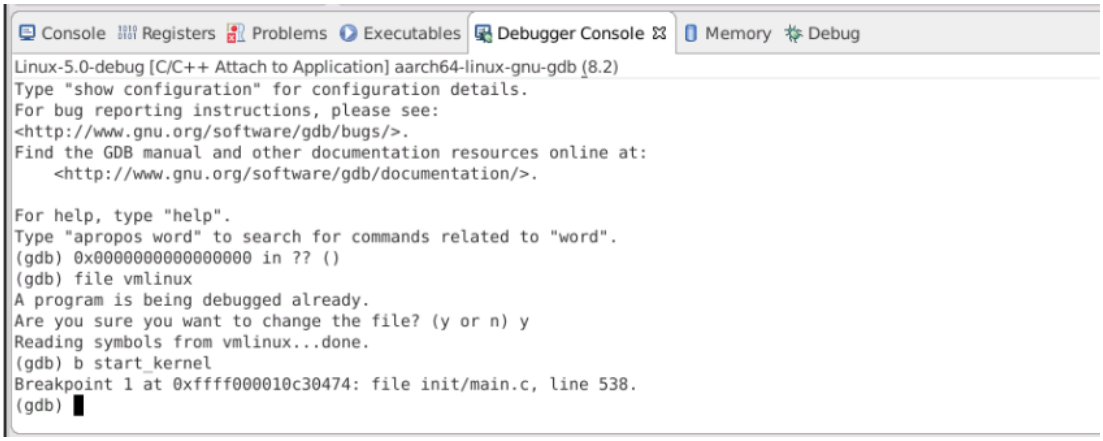


图3.11　Debugger Console控制台

在 Debugger Console 中输入“b start_kernel”，在 start_kernel 函数中设置一个断点。输入“c”命令，开始运行 QEMU 中的 Linux 内核，它会停在 start_kernel 函数中，如图 3.12 所示。

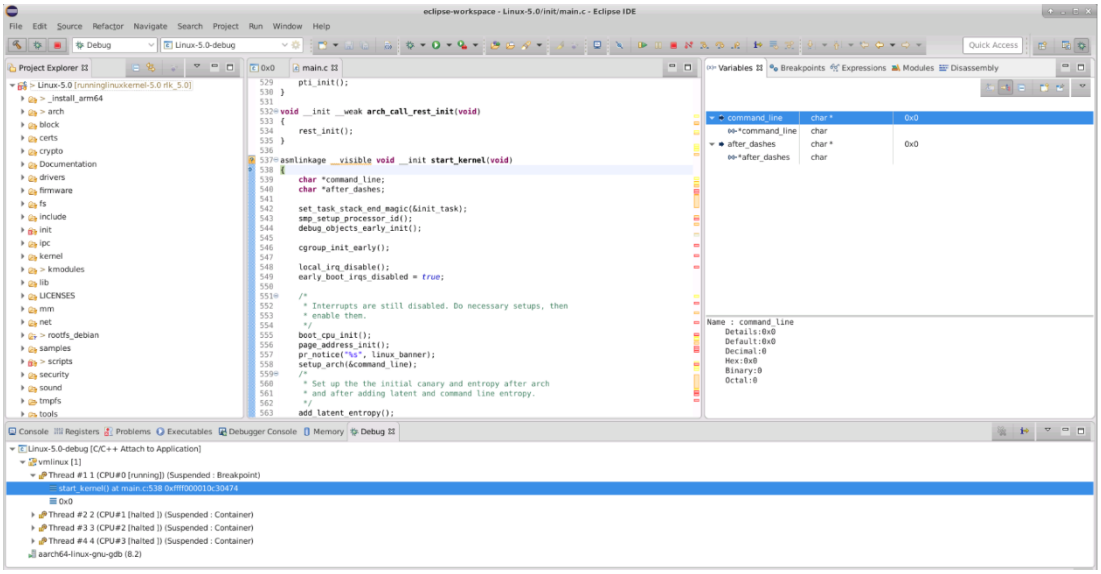


图3.12　Eclipse调试内核

Eclipse 调试内核比使用 GDB 命令要直观很多，例如参数、局部变量和数据结构的值都会自动显示在“Variables”标签卡上，不需要每次都使用 GDB 的打印命令才能看到变量的值。读者可以单步并且直观地调试内核。