**浙江大学实验报告**

**课程名称：** 操作系统

**实验项目名称：**lab0 RV64环境搭建和内核编译

**学生姓名：** **学号：**

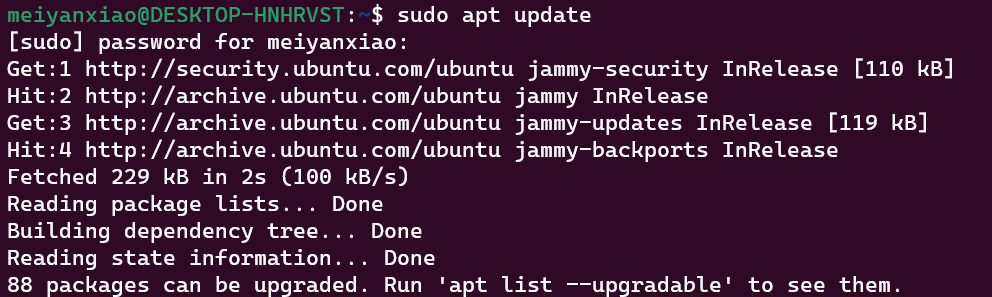
**电子邮件地址**：

**实验日期：** 2023 年 9 月 26 日

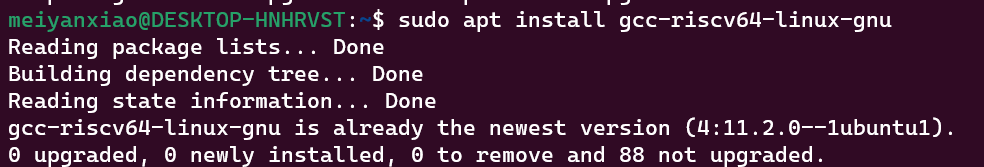
1. 实验内容

1.1搭建实验环境

首先，我们需要使用“$ sudo apt update”来更新软件包列表，检查可用的软件包更新，已使软件包达到最新状态。其中sudo表示linux的系统管理指令，以更高的权限运行对映的命令；而apt是linux系统中的软件包管理工具，用于管理linux中各软件的更新状态。

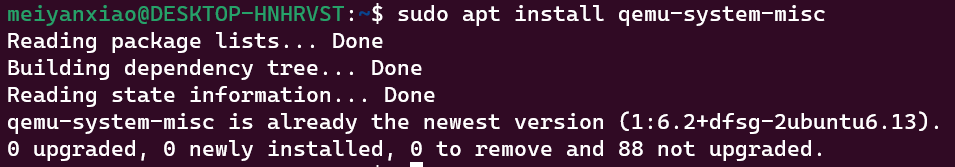
第二次使用update语句所得结果

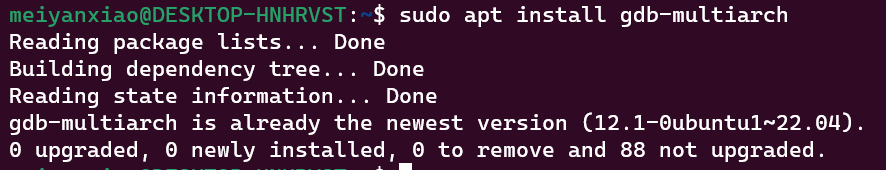
接着我们进行“$ sudo apt install gcc-riscv64-linux-gnu”操作，为linux系统安装合适的gcc编译器，当然，也要基于riscv64与linux的环境，便于后续程序编写的编译。

已安装有gcc编译器，得到以上结果

我们进行“$ sudo apt install autoconf automake autotools-dev curl libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool patchutils bc zlib1g-dev libexpat-dev git”命令安装依赖项。

接下来，我们需要使用“$ sudo apt install qemu-system-misc”指令安装用于启动 riscv64 平台上的内核的模拟器qemu；qemu能够完成用户程序模拟和系统虚拟化模拟，即qemu能够将为⼀个平台编译的二进制文件运行在另一个不同的平台。接着可以使用“$ sudo apt install gdb-multiarch”语句来安装gdb，gdb是GNU调试器的缩写，是一个由 GNU 开源组织发布的、在UNIX/LINUX 操作系统下的、基于命令行的、功能强大的程序调试工具；它可以在 qemu 上运⾏的 Linux 内核进⾏调试。

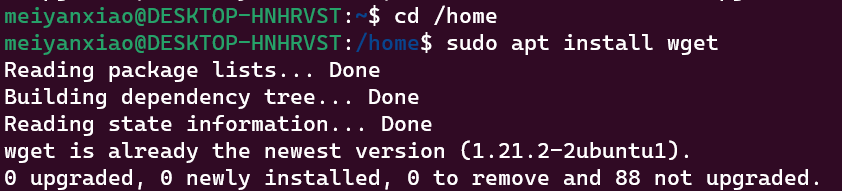
qemu模拟器的下载安装（图后补）

gdb调试器的下载安装

1.2获取 Linux 源码和已编译完成的文件系统

首先，我们需要进入指定的文件夹中“# cd /home”（在这里，我们可以使用“#pwd”来查看当前路径。

接着，我们需要使用命令“# apt install wget”来安装wget工具。wget 是一个从网络上自动下载文件的自由工具，支持通过[HTTP](https://baike.baidu.com/item/HTTP/243074?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/wget/_blank)、[HTTPS](https://baike.baidu.com/item/HTTPS/285356?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/wget/_blank)、FTP三个最常见的[TCP/IP协议](https://baike.baidu.com/item/TCP/IP%E5%8D%8F%E8%AE%AE?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/wget/_blank)[下载](https://baike.baidu.com/item/%E4%B8%8B%E8%BD%BD/2270927?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/wget/_blank)，并可以使用 HTTP 代理。

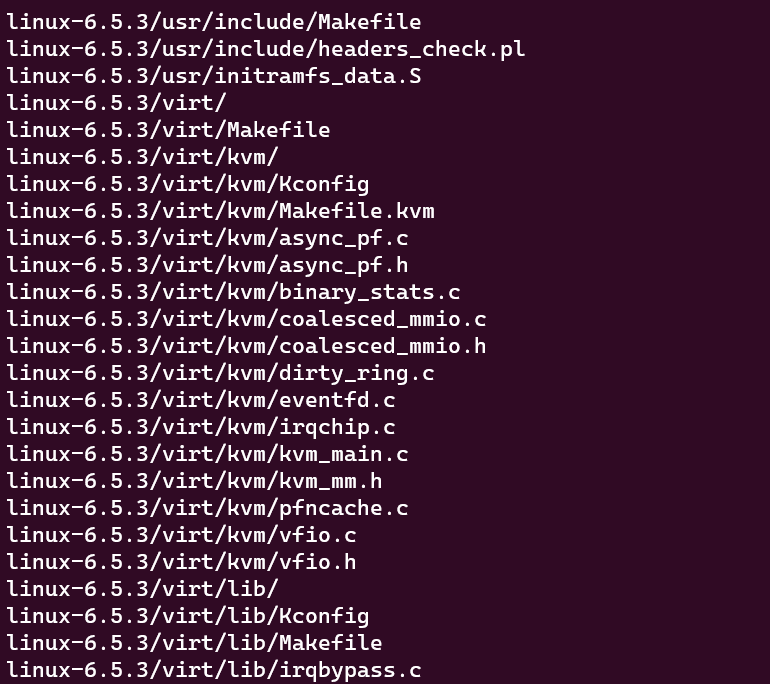
wget的安装/更新

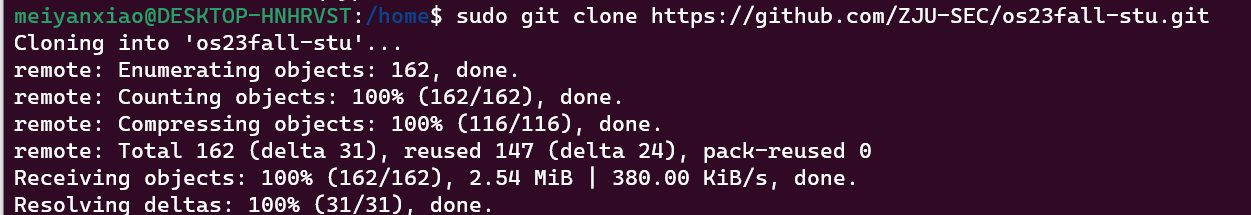
我们使用“wget <https://cdn.kernel.org/pub/linux/kernel>/v6.x/linux-6.5.3.tar.xz”命令，借助wget工具从特定网页下载linux-6.5.3的内核压缩包至/home文件夹。下载完成后，我们可以使用“$ls”命令在指定文件夹中查看下载的结果。

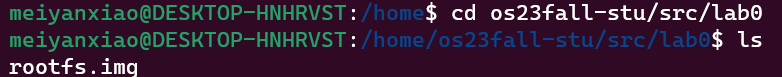
5738e817d5d122936ac08bc5805d88f已包含git步骤的额外结果

接着我们使用命令“# tar -xvf linux-6.5.3.tar.xz”解压Linux 源码包至/home目录下。然后，我们使用“git”命令https://github.com/ZJU-SEC/os23fall-stu

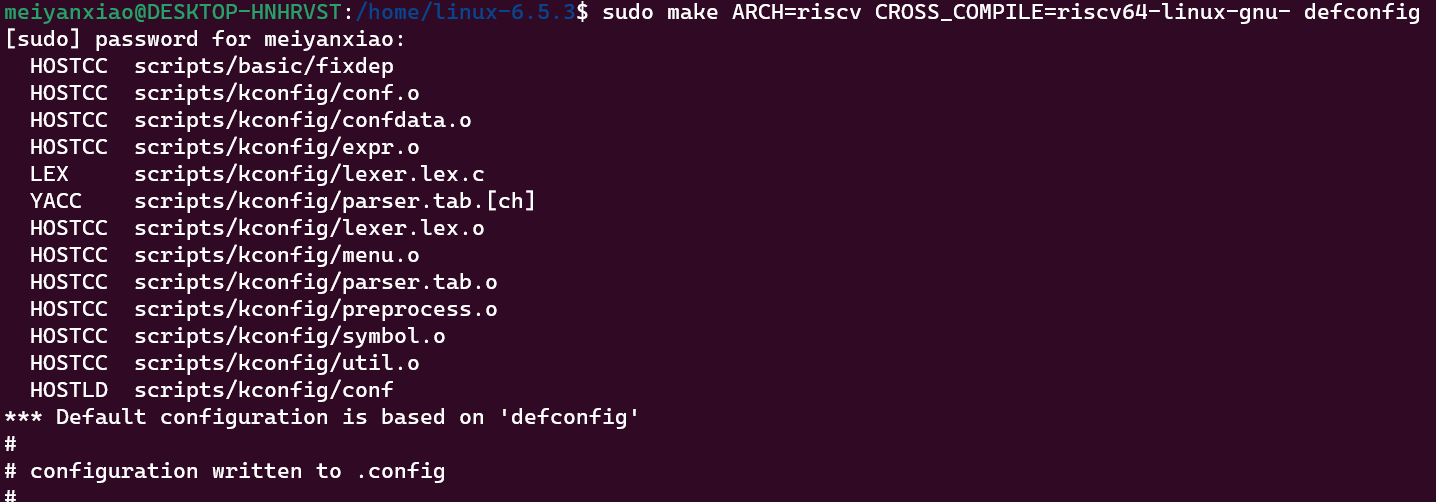
.git仓库中，获取根文件系统的镜像；并使用“cd os23fall-stu/src/lab0”与“ls”命令切换到指定文件夹，查看clone的结果。

解压linux信息概览（部分）

二次clone git仓库的结果

进入列表并查看文件

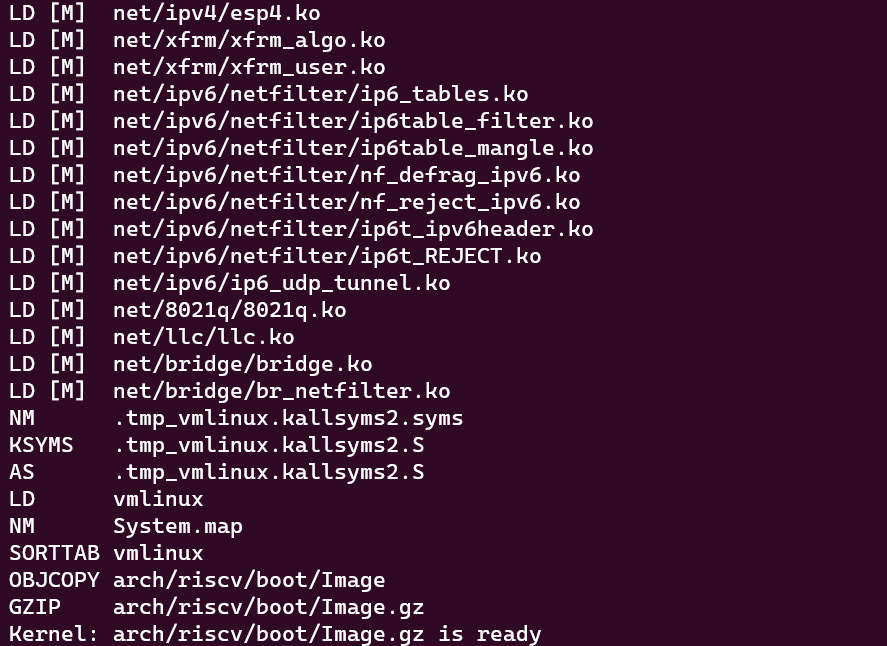
1.3 编译linux内核

继续，我们可以使用命令“$ cd /home/linux-6.5.3”切换至对映linux文件夹；再使用“$ make ARCH=riscv CROSS\_COMPILE=riscv64-linux-gnu- defconfig”指令使用默认配置。

默认配置完成后得到的结果显示

随着默认配置的完成，我们可以开始进行linux内核的编译；进行linux内核的编译需要以下的命令“$ make ARCH=riscv CROSS\_COMPILE=riscv64-

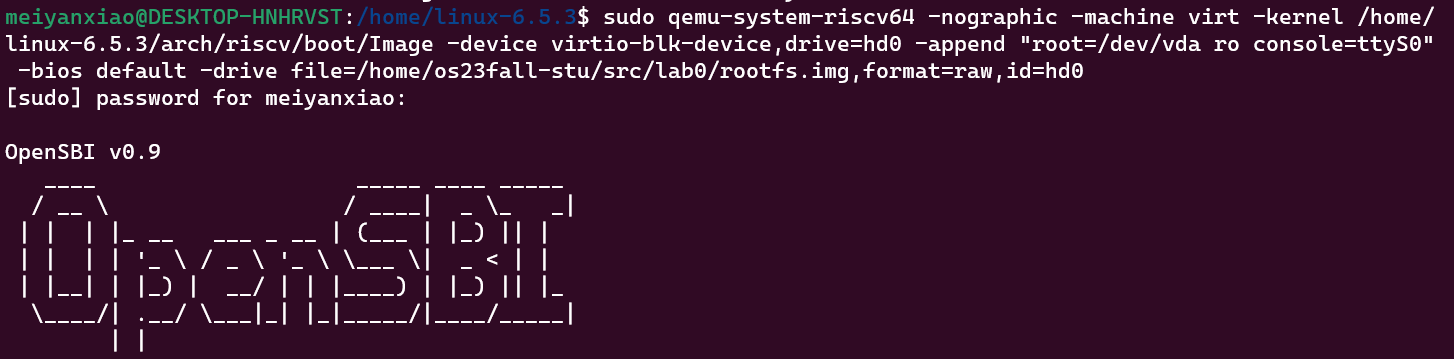
linux-gnu- -j$(nproc) ”；其中-j表示此过程可以使用多线程进行，而$（nproc）则表示使用最大可用的线程数量；需要特别注意，使用多线程进行编译的操作可能会触发“out of memory”的报错，导致编译过程异常中断。

Linux-6.5.3内核的编译结果

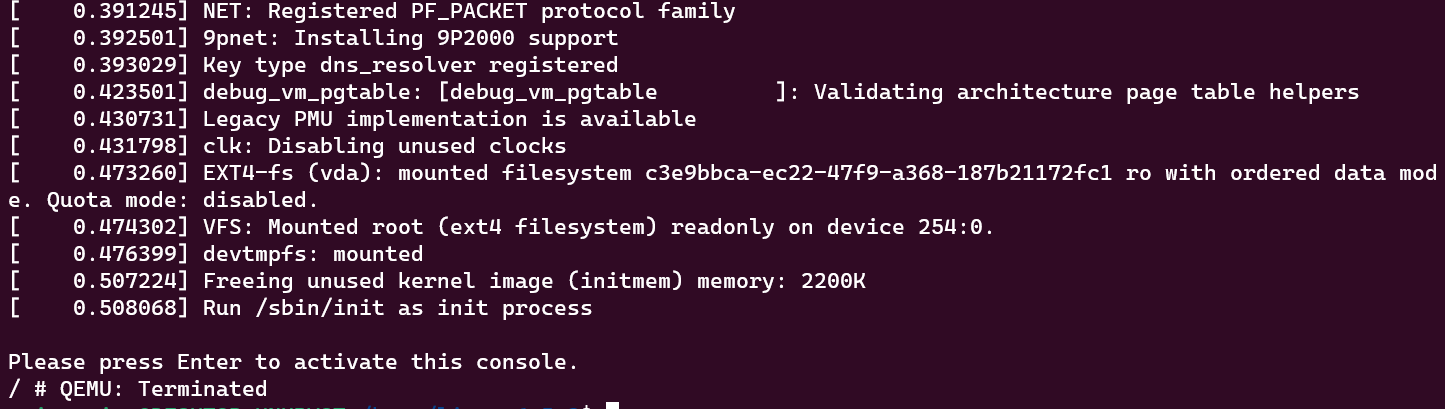
1.4使用qemu运行内核及使用gdb对内核进行调试

完成了内核编译之后，我们就获得了linux-6.5.3内核的可执行文件，我们可以使用命令“$sudo qemu-system-riscv64 -nographic -machine virt -kernel /home/linux-6.5.3/arch/riscv/boot/Image -device virtio-blk-device,drive=hd0 -append "root=/dev/vda ro console=ttyS0" -bios default -drive file=/home/os2023fall-stu

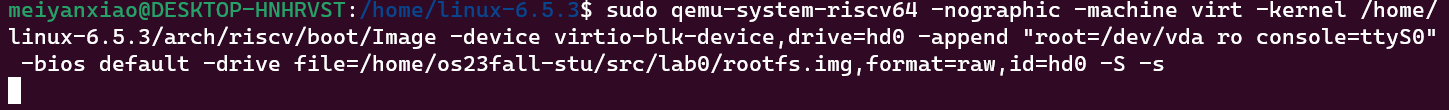
/src/lab0/rootfs.img,format=raw,id=hd0”来使用qemu工具，运行我们已经编译好的linux内核。

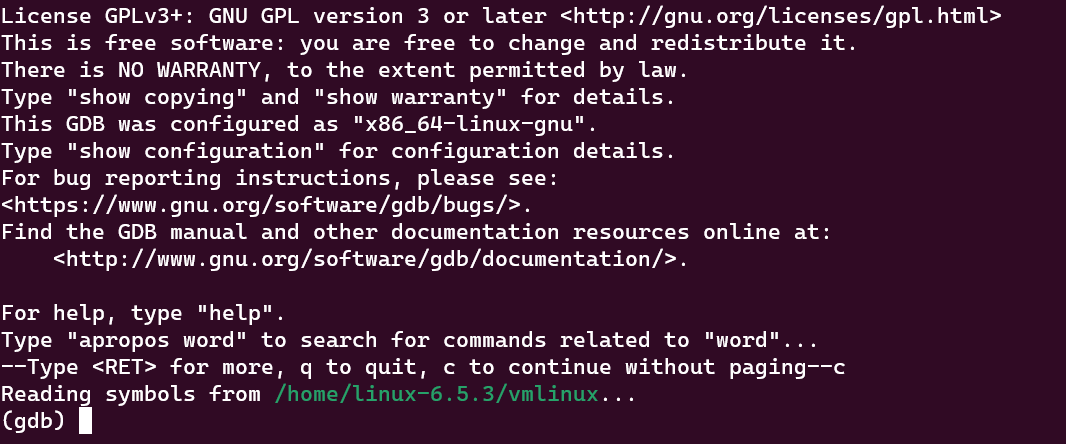
进入opensbi的界面截图结果

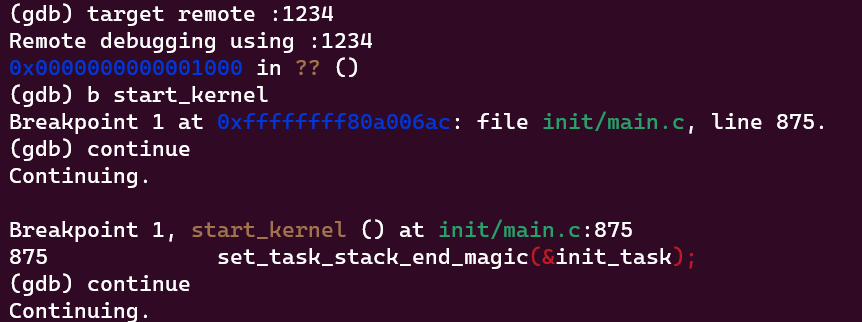
紧接着，我们进行退出qemu模拟器方法的尝试，我们先同时摁下“ctrl + a”按键，松开后再摁下“x”键，就可以起到快捷退出qemu模拟器的效果。

qemu模拟器成功运行提示及退出

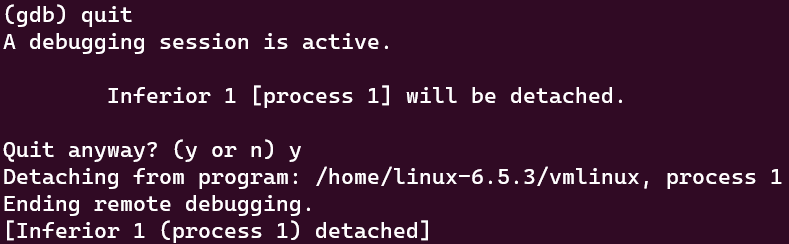
最后，我们还需要测试gdb调试工具的可用状态。在这里我们需要开启两个会话终端，一个终端借助qemu模拟器启用linux-6.5.3系统；另一个终端使用gdb与qemu模拟器进行远程通信；其中，对于gdb工具而言，我们基于“tcp::1234端口”进行调试。

第一个终端进入qemu模拟的linux系统并进行等待

 在qemu模拟linux系统等待终端基础上的gdb调试工具成功进入

当gdb调试系统正常启用后，我们便可以使用gdb相映的语句对linux内核代码进行调试。这里以连接qemu模拟器，设置断点位置，继续执行代码及退出gdb调试四个功能为例。

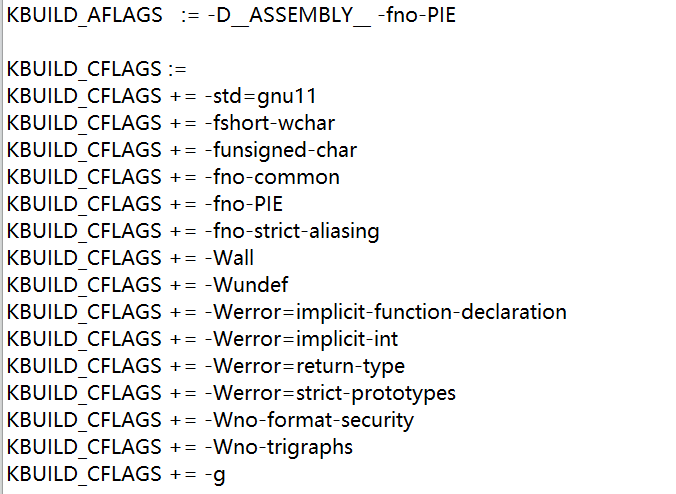
gdb工具基础功能的代码及调试

在gdb调试文件中，我们可以使用“target remote:1234”来指定需要远程连接的端口序号；接着，我们可以使用“b start\_kernel”命令来设置断点，在当前例子中，我们选择将断点设置在内核开始运行代码之前；在设置断点之后，我们可以使用“continue”命令执行编译好的文件，文件会在我们设置的第一个断点处暂停；最后，我们可以使用“quit”命令退出gdb调试工具。

退出gdb调试工具

对于上述结果截图中的问题：“a debugging session is active...”我们可以选择y来介导gdb调试程序的quit anyway；如果我们不需要这样的信息，我们也可以使用命令“set confirm off”来关闭这样的信息提示。

但在实际的操作中，我们却不能直接连接到gdb调试程序，gdb会呈现“Reading symbols from vmlinux...(No debugging symbols found in vmlinux)”的提示，我们需要在内核的 Makefile 文件中，找到 KBUILD\_CFLAGS 变量并添加 -g 选项，然后重新编译内核，才可以介导gdb调试程序的正确连接与运行。

可以添加修改信息的Makefile文件定位如上

1. 讨论、心得

在这里写：实验过程中遇到的问题及解决的办法，你做本实验的体会

实验问题与解决方案：

虽然“在内核的 Makefile 文件中，找到 KBUILD\_CFLAGS 变量并添加 -g 选项”，就可以使gdb工具进行正常的调试；但是，为KBUILD\_CFLAGS 变量添加 -g选项的过程也并非容易。

最直接的问题，权限问题；我尝试直接用txt的格式打开Makefile文件，并在对映变量的位置进行添加，没有权限；我尝试了使用vscode，进行wsl的虚拟机连接，再打开Makefile文件进行修改，没有权限。

最后，我只能强制提权，使用“$ sudo chown -R meiyanxiao /home/linux-6.5.3

/Makefile”命令，赋予vscode修改此文件的权限，最终完成修改并保存。

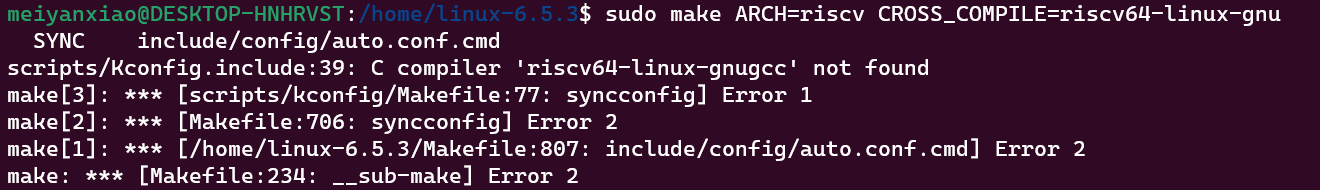
吐槽：

在进行实验的过程，还是存有一些有趣的小插曲，在这里浅浅地吐槽一下。

在搭建实验环境的第一步，我们需要使用“$ sudo apt install autoconf automake ...”长命令安装依赖项，其中有一个组成成分为“zlib1g”，在认真实验，不掺杂cv工程的code中，我将其中的1一直错输为l，导致linux系统一直报错，无法正确执行命令；直到最后才发现，“zlib1g”组成成分中，第一个是“l”，而第二个是“1”。

遗留的问题：

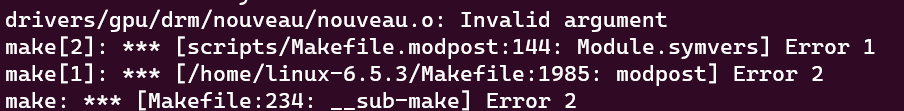
1. 对于搭建实验环境的第一步中，使用“$ sudo apt install autoconf automake ...”命令安装依赖项是什么操作，或者说有什么原理蕴含其中？
2. 在qemu模拟器上运行linux内核时，长段命令的输入，以及“-device”等的参数输入，含有什么特殊的限制含义？
3. 在使用vscode改写Makefile文件中的KBUILD\_CFLAGS变量（添加 -g指令）后，可能会导致编译无法正常进行。



一开始认为这一错误来源于“out of memory”,是因为编译过程中，线程使用数量过大，导致内存使用超过限制，从而编译失败，异常退出。

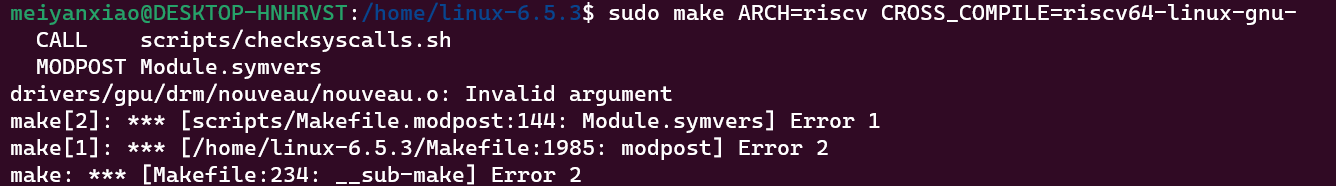
但是，我在修改编译指令，使用“$ make ARCH=riscv CROSS\_COMPILE=

riscv64-linux-gnu-”命令，用单线程对编译操作进行运行，程序依然出现异常退出的崩溃结果。



见此方法无效，说明编译错误很大概率不来源于内存问题。除非电脑配置过差，单线程也无法对单个文件进行编译；但是在没有添加“-g”指令时，本电脑使用12线程并行工作，依然可以完成编译的任务，因而单线程编译错误不可能为电脑内存不足的问题。

此时，我怀疑是在修改Makefile文件时，因修改失误导致了一些其他信息被修改，因而我选择先将“-g”命令删去；在删去后，重新运行linux内核的编译，发现运行成功，说明其他的相关文件大概率依然保持正确的信息。

到这里仍然没有有效地发掘问题，所以我选择采用重装的方式，重新配置linux-6.5.3的运行环境，并在第一次编译之前，便优先修改Makefile文件，但这次编译依然出现了异常退出的问题。

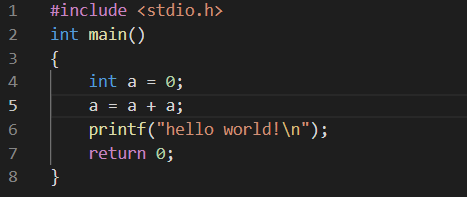
此时，在每次碰到异常退出问题后，我选择继续进行单线程的编译工作；一次一次编译运行，异常退出，编译继续运行，再次异常退出的摇摆之后；最终，突然成功，完成编译工作。

但总感觉完成编译内核的过程十分的奇怪。

1. 思考题

1.尝试使用 riscv64-linux-gnu-gcc 编译单个 .c 文件

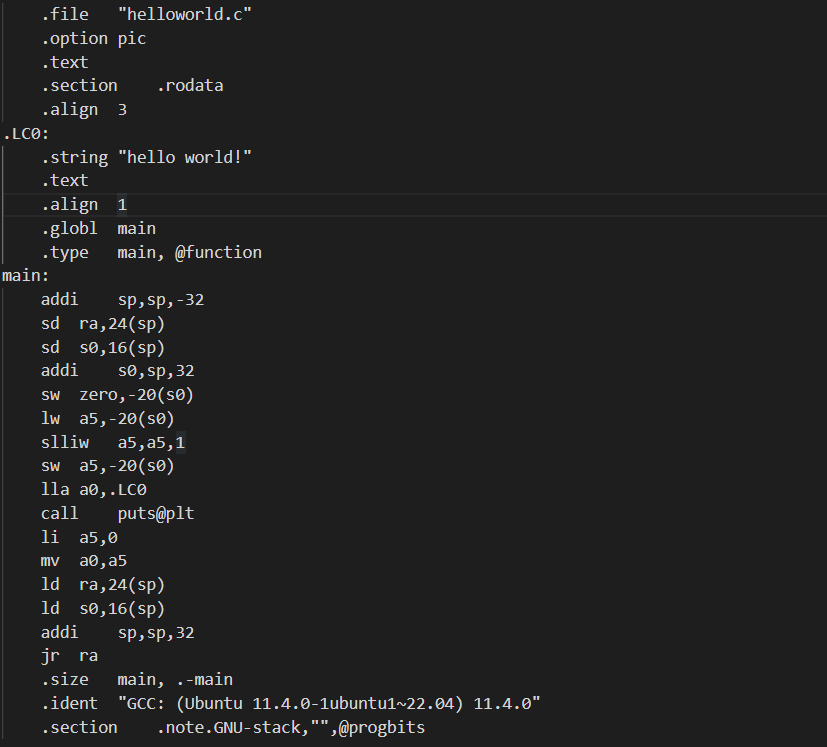
首先，我们使用C语言，编写了一个最为简单的文件，并取名为helloworld.c。

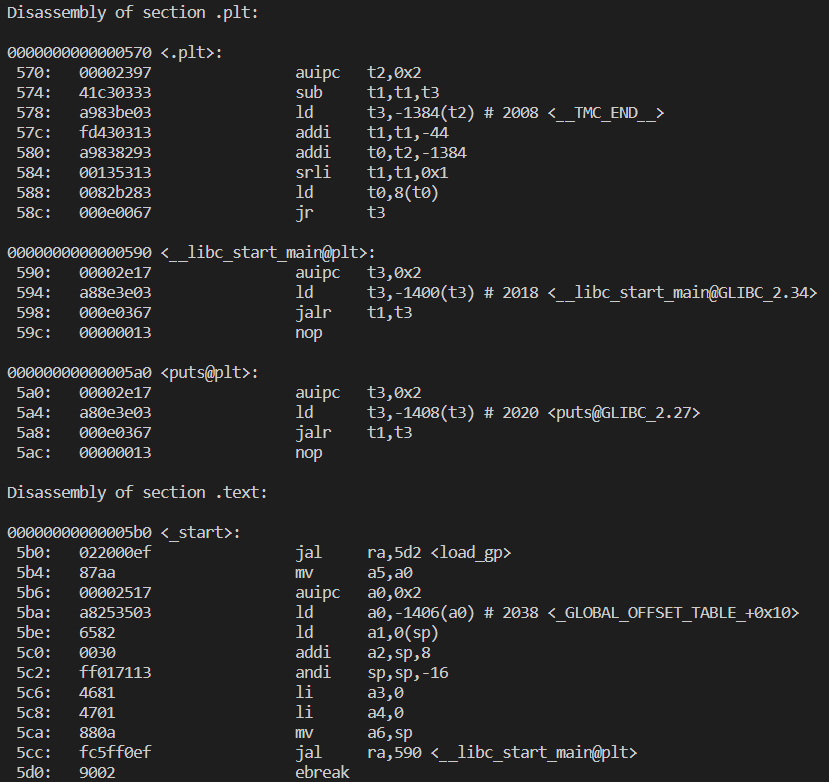


接着，我们使用riscv64-linux-gnu-gcc -S -o hello.s helloworld.c指令，用基于linux指令和riscv64架构的gcc编译器编译helloworld.c文件，并生成对映的汇编语言，保存于hello.s文件中。汇编结果见后。

2.使用 riscv64-linux-gnu-objdump 反汇编（1）中得到的编译产物

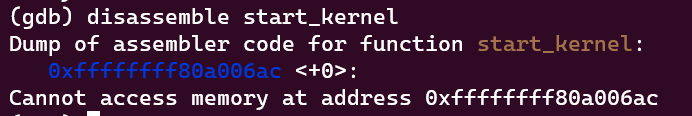
我们可以使用指令riscv64-linux-gnu-objdump -S -d hello.o > hello.o.txt反汇编（1）中得到的编译产物，并将对映的结果写入hello.o.txt文件中。

 helloworld.c文件对映的riscv汇编语言文件

hello.o文件对映的反汇编文件

3.使用gdb调试linux内核的样例

3.1 我们可以使用disassemble指令，来查看对映汇编语言代码。因为对映地址区域无法访问的原因，start\_kernel函数对映的汇编语言我们不能直接寻得。

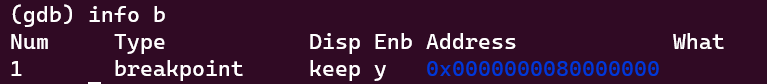


3.2 设定在区域设置断点并查看所有设置的断点

我们可以使用b \*address的方式在对映的默认文件中设置断点，其中address为需要设置断点位置的地址信息。

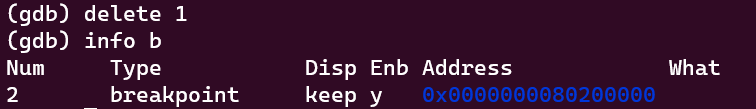
699ad53cfb4058c7e4e8682ede3ff27

进而，我们可以使用info b的指令查看所有设置的断点信息。



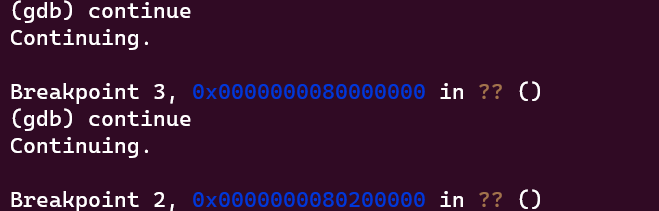
3.3设置新断点并删除已有断点

对于新断点的设置，我们可以效仿3.2中使用的方法。而为了删除已有断点，我们可以使用delete X的指令，其中X为需要删除断点对映的序号。我们可以使用info b查看所有设置的断点，来参照上述过程的正确性。由实验结果可以看出，当前仅有一个断点剩余。

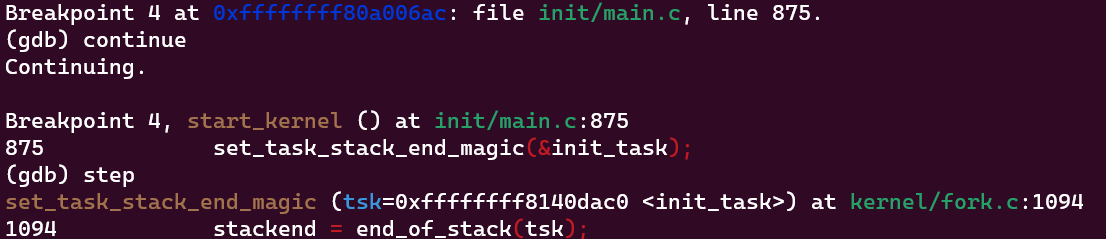


3.4 运行到断点并进行单步测试

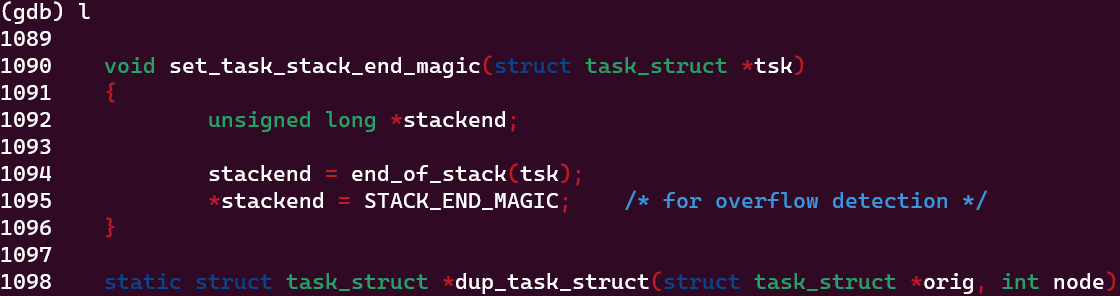
当设置完断点后，我们可以使用continue指令使程序继续运行，直到断点处停止。



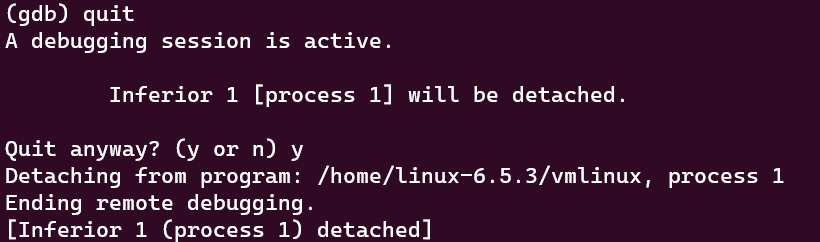
在此处，我们使用step指令进行单步的测试。



此时，我们可以使用l指令来查看当前运行/暂停语句附近的代码详情。



试验结束，退出gdb调试程序。



4.vmlinux 和 Image 的关系和区别是什么？

vmlinux是从源代码构建内核文件的过程中创建的文件，它以ELF格式保存，且未做任何压缩处理。vmlinux可以引导某些调试器（如gdb）等，将其用于调试目的。但是vmlinux不可以直接引导linux系统启动。

一般来说，系统在创建了vmlinux ELF格式文件的基础上，还会创建其他文件，如Image。Image文件是Linux内核编译时，使用objcopy处理vmlinux后生成的二进制内核映像。映像由vmlinux的一些部分和一些引导代码组成，因而其可以直接引导Linux系统的启动。