

ARM 指令集编程调试实验教程

第一课

抗疫期间，大家短期没办法返回学校上课、到实验室做实验了，我们的网上mooc课程学到ARM指令集这一章，鼓励大家自己动手安装虚拟仿真环境，进行ARM指令集、汇编程序的学习。

1 准备工作

1.1 安装 ubuntu 以及 arm-linux-gcc

有条件的同学请使用电脑安装ubuntu操作系统，如果已经安装windows操作系统，也可以安装虚拟机(比如vmware)，如何安装ubuntu虚拟机请大家参照网络上的教程，自己动手安装。这个安装过程也是提高你们动手能力的一部分，实在不会安装请在qq群提问，有安装文档说明，涉及软件版权，只提供本次学习，请不要传播。一个士兵上战场不会装子弹，那是没办法打仗的哈:))。

为了运行ARM汇编代码，我们需要使用交叉编译器arm-linux-gcc对ARM汇编代码进行编译。请大家自行下载交叉编译器，注意对应可使用的ubuntu版本，安装完毕后，我们可以用arm-linux-gcc对ARM汇编代码进行编译。

1.2 安装 qemu 和 gdb

arm-linux-gcc编译ARM汇编程序，执行目标是ARM内核的处理器，我们的PC机不是ARM架构的计算机，如何运行这个程序呢？网上有很多仿真器，有纯软件的，也有IDE带ICE调试硬件的，目前这个状况，估计大家从网上也买不到，所以我这次给大家讲一下最简单的软件配置，qemu加gdb来进行调试。

Qemu的介绍，我从网上copy一段话，见图1。

QEMU本身是一个非常强大的虚拟机，甚至在Xen、KVM这些虚拟机产品中都少不了QEMU的身影。在QEMU的官方文档中也提到，QEMU可以利用Xen、KVM等技术来加速。为什么需要加速呢，那是因为如果单纯使用QEMU的时候，它自己模拟出了一个完整的个人电脑，它里面的CPU啊什么的都是模拟出来的，它甚至可以模拟不同架构的CPU，比如说在使用Intel X86的CPU的电脑中模拟出一个ARM的电脑或MIPS的电脑，这样模拟出的CPU的运行速度肯定赶不上物理CPU。使用加速以后呢，可以把客户操作系统的CPU指令直接转发到物理CPU，自然运行效率大增。

QEMU同时也是一个非常简单的虚拟机，给它一个硬盘镜像就可以启动一个虚拟机，如果想定制这个虚拟机的配置，比如用什么样的CPU啊、什么样的显卡啊、什么样的网络配置啊，指定相应的命令行参数就可以了。它支持许多格式的磁盘镜像，包括VirtualBox创建的磁盘镜像文件。它同时也提供一个创建和管理磁盘镜像的工具qemu-img。QEMU及其工具所使用的命令行参数，直接查看其文档即可。

图1 qemu 简介

Qemu是很好用的一个虚拟机，我在ubuntu底下用的比较顺手，也就介绍给大家用了。

GDB 是什么？我想爱好编程的同学应该比较熟悉吧，如果在 linux 底下干过活，学习过 C 语言，也都应该了解 gdb 吧，所以我也不会花篇幅去讲 gdb 如何调试程序。

gdb 简介

gdb 是 **UNIX** 及 **UNIX-like** 下的调试工具，在 Linux 下一般都直接在命令行中用 **gdb** 来调试程序，相比 Windows 上的集成开发环境 **IDE** 提供的图形界面调试，一开始使用 **gdb** 调试可能会让你感到生无可恋，但是只要熟悉了 **gdb** 调试的常用命令，调试出程序会很有成就感，一方面因为这些命令就类似图形界面调试按钮背后的逻辑，另一方面用命令行来调试程序，逼格瞬间就上了一个档次，这次就跟大家分享 **gdb** 调试的基本技术和 15 个常用调试命令。

图 2 gdb 简介

图 2 是我从知乎上随便找一个 **gdb** 的简单教程，实在不懂的同学应该很快就可入门。

接下来，我们编译安装 **qemu** 模拟器

为了 x86 的 Linux 系统内运行 **ARM** 体系结构的可执行程序，需要安装 **qemu** 模拟器。首先下载 **qemu** 源码，需要保证系统已经安装了 **flex** 和 **bison**。

编译安装 qemu：

```
./configure --prefix=/usr  
sudo make && make install
```

然后就可以使用 **qemu** 的 **ARM** 模拟器执行 **ARM** 程序。当然你们也可以不使用源码安装 **qemu**，也可以直接 **apt-get install** 安装。

编译安装 gdb：

为了调试 **ARM** 程序，需要使用 **gdb** 的源码编译生成 **arm-gdb**。

首先下载 **gdb** 源代码，编译安装：

```
./configure --target=arm-linux --prefix=/usr/local/arm-gdb -v  
sudo make && make install
```

为了和系统的 **gdb** 避免冲突，我们将 **gdb** 的安装目录安装到 **/usr/local**，然后建立软链接即可。

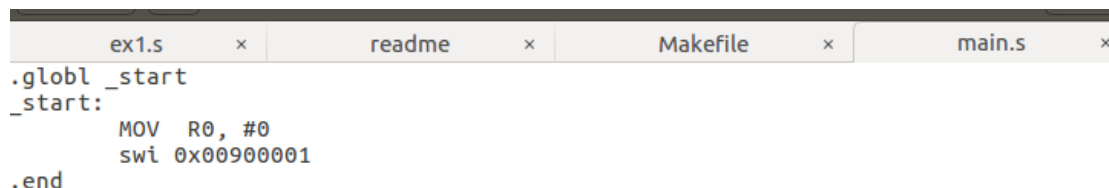
```
ln -s /usr/local/gdb/gdb /usr/bin/arm-gdb
```

之后我们便可以使用 **arm-gdb** 命令调试 **ARM** 程序了。

2 ARM 汇编

请大家先学习网络教学 ARM 指令集这一章的内容, 课本或者这个网络的 mooc 课程, 它只是单纯讲解指令集, 没有告诉大家程序设计如何进行。这里我们也不是程序设计课程, 所以我们也没有课时去讲一个完整的 ARM 汇编程序如何设计。

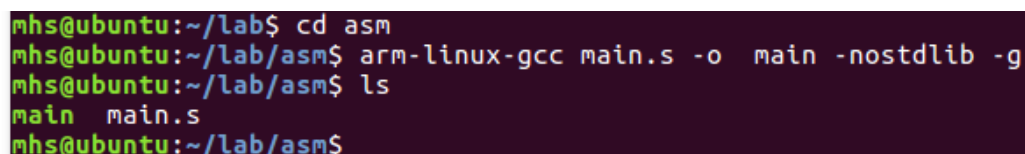
这里我们从一个最简单的汇编程序开始, 代码见图 3。



```
ex1.s  x  readme  x  Makefile  x  main.s  x
.globl _start
_start:
    MOV    R0, #0
    swi    0x00900001
.end
```

图 3 第一个最简单汇编程序

上面这是一个最简单的汇编程序, 只声明了一个程序入口和结束指令 end, 使用汇编指令完成了 0 号系统调用 exit 的调用。mov 指令将系统调用号传入寄存器 R0, 然后使用 0x00900001 软中断陷入系统调用。



```
mhs@ubuntu:~/lab$ cd asm
mhs@ubuntu:~/lab/asm$ arm-linux-gcc main.s -o main -nostdlib -g
mhs@ubuntu:~/lab/asm$ ls
main  main.s
mhs@ubuntu:~/lab/asm$
```

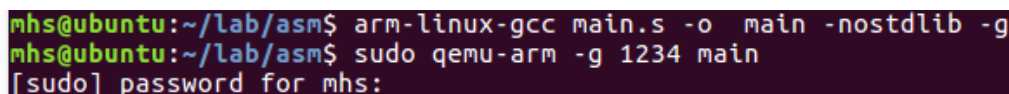
图 4 ARM 程序交叉编译

编好这个程序后, 我们保存好程序代码, 然后我们使用交叉编译器 arm-linux-gcc 编译这个程序, 大家在上图中可以看到编译成功后生成了可执行文件 main, 编译选项 “-nostdlib” 表示不使用任何运行时库文件, 编译生成的可执行文件 main 只能在 ARM 体系结构的系统上运行。

3 调试运行 ARM 程序

前面我们已经安装好了 qemu 和 gdb, 那么这个时候, 我们可以用这两个小工具进行我们的 ARM 程序设计代码调试了, 实际上 gdb 也可以调试 C 语言的哈:)。

首先我们开一个终端窗口, 按快捷键 Ctrl+Alt+T 即可, 操作示范见图 5。



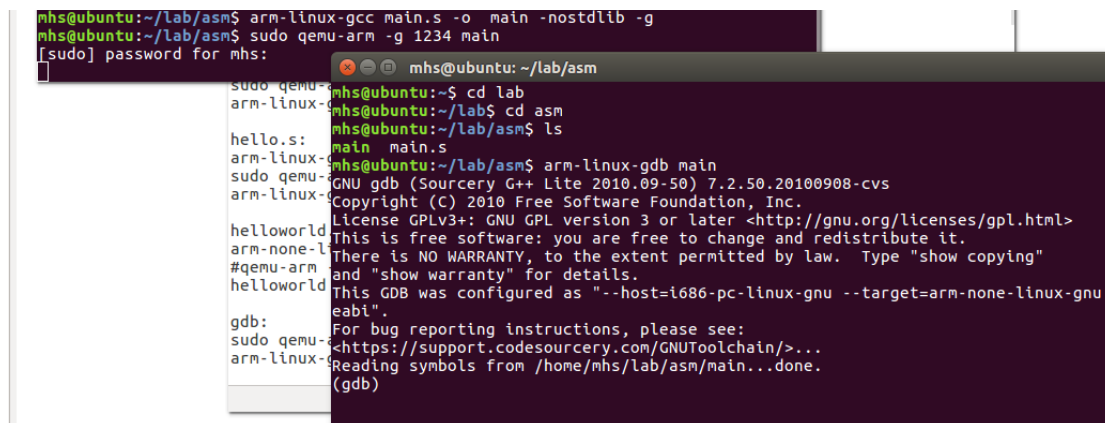
```
mhs@ubuntu:~/lab/asm$ arm-linux-gcc main.s -o main -nostdlib -g
mhs@ubuntu:~/lab/asm$ sudo qemu-arm -g 1234 main
[sudo] password for mhs:
```

图 5 模拟运行 arm 程序进行调试

-g 参数表示是在内存中调试运行, 相当于大家习惯在编程 IDE 里面点 debug 按钮, 1234 是给 gdb 的远程调试端口号, 调试采用的是 socket 通信机制。相当

于开了一个模拟 ARM 指令运行的服务器，它在运行 main 这个程序。

接下来我们 Ctrl+Alt+T 另开一个终端窗口，进入到 main 这个程序目录下，运行演示如图 6。

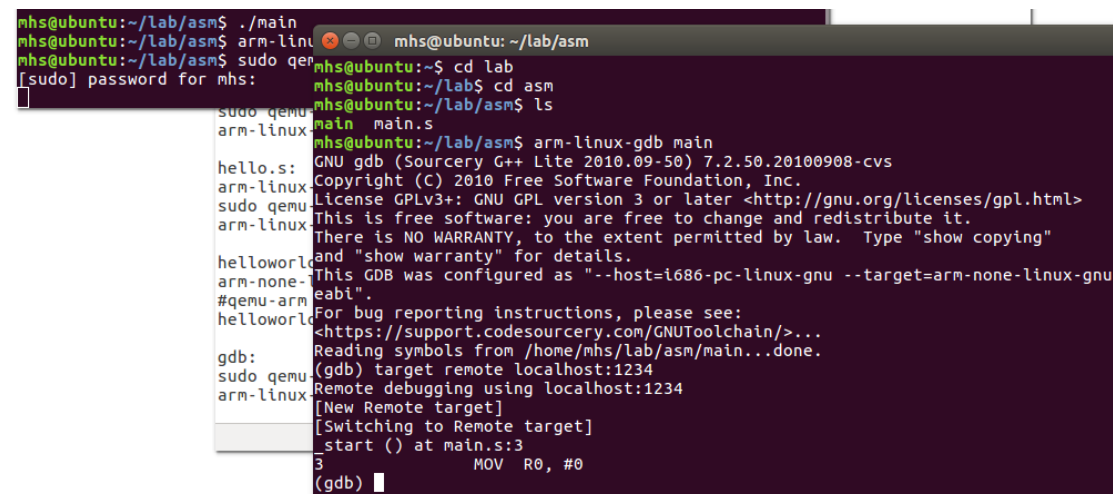


```
mhs@ubuntu:~/lab/asm$ arm-linux-gcc main.s -o main -nostdlib -g
mhs@ubuntu:~/lab/asm$ sudo qemu-arm -g 1234 main
[sudo] password for mhs:
mhs@ubuntu:~/lab/asm$ cd lab
mhs@ubuntu:~/lab$ cd asm
mhs@ubuntu:~/lab/asm$ ls
main main.s
mhs@ubuntu:~/lab/asm$ arm-linux-gdb main
GNU gdb (Sourcery G++ Lite 2010.09-50) 7.2.50.20100908-cv5
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=arm-none-linux-gnu
eabi".
For bug reporting instructions, please see:
<https://support.codesourcery.com/GNUToolchain/>...
Reading symbols from /home/mhs/lab/asm/main...done.
(gdb)
```

图 6 启动 gdb 调试

大家可以看到 gdb 找到了 main 程序的入口符号，然后停留在(gdb)后面等待你键入 gdb 指令。

首先我们需要连接 qemu 的服务器，使用 target remote localhost:1234 指令进行连接。见图 7。



```
mhs@ubuntu:~/lab/asm$ ./main
mhs@ubuntu:~/lab/asm$ arm-linux-gdb main
mhs@ubuntu:~/lab/asm$ sudo qemu-arm -g 1234 main
[sudo] password for mhs:
mhs@ubuntu:~/lab/asm$ cd lab
mhs@ubuntu:~/lab$ cd asm
mhs@ubuntu:~/lab/asm$ ls
main main.s
mhs@ubuntu:~/lab/asm$ arm-linux-gdb main
GNU gdb (Sourcery G++ Lite 2010.09-50) 7.2.50.20100908-cv5
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=arm-none-linux-gnu
eabi".
For bug reporting instructions, please see:
<https://support.codesourcery.com/GNUToolchain/>...
Reading symbols from /home/mhs/lab/asm/main...done.
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
[New Remote target]
[Switching to Remote target]
start () at main.s:3
3      MOV    R0, #0
(gdb)
```

图 7 连接 gdb 调试

键入刚才的连接命令后，大家可以看到连接成功后，gdb 会显示汇编程序，停留在第一句，也就是源代码的第 3 行：MOV R0, #0

接下来我们就可以使用 gdb 的各种调试命令，进行程序运行调试了。

```
mhs@ubuntu:~/lab/asm$ ./main
mhs@ubuntu:~/lab/asm$ arm-linux-gnueabihf-gdb ./main
mhs@ubuntu:~/lab/asm$ sudo qemu-system-arm -M arm-linux-gnueabihf -m 1024 -s -S
[sudo] password for mhs:
Remote debugging using localhost:1234
[New Remote target]
[Switching to Remote target]
start () at main.s:3
3      MOV R0, #0
(gdb) info register
r0             0x00000000      0
r1             0xf6ff90c      -150996724
r2             0x00000000      0
r3             0x00000000      0
r4             0x00000000      0
r5             0x00000000      0
r6             0x00000000      0
r7             0x00000000      0
r8             0x00000000      0
r9             0x00000000      0
r10            0x805c         32860
r11            0x00000000      0
r12            0x00000000      0
sp             0xf6fff810      0xf6fff810
lr             0x00000000      0
pc             0x8054         0x8054 <_start>
cpsr           0x10         16
(gdb)
```

图 8 gdb 调试命令示范

这里简单介绍几个常用的 gdb 指令：

- (gdb) disassemble // 查看反汇编
- (gdb) x /8xw 0x0000808e // 查看内存
- (gdb) info register // 查看寄存器
- (gdb) continue // 继续执行
- (gdb) stepi // 汇编级逐过程
- (gdb) nexti // 汇编级逐语句

请大家自行查询学习 **GDB 跳转执行命令**、**GDB 调试输出命令**、以及如何设置断点的命令，这里特别需要大家使用的是寄存器查看命令：info register，以及内存查看命令：x 命令。

其中查看内存的命令解释如下：

使用 examine 命令（简写是 x）来查看内存地址中的值。x 命令的语法如下所示：

X /nfu startaddress

n、f、u 是可选的参数。

n 是一个正整数，表示显示内存的长度，也就是说从当前地址向后显示几个地址的内容。

f 表示显示的格式，大致格式如下：

x 按十六进制格式显示变量。

- d 按十进制格式显示变量。
- u 按十六进制格式显示无符号整型。
- o 按八进制格式显示变量。
- t 按二进制格式显示变量。
- a 按十六进制格式显示变量。
- c 按字符格式显示变量。
- f 按浮点数格式显示变量。

如果地址所指的是字符串，那么格式可以是 s，如果地址是指令地址，那么格式可以是 i。

u 表示从当前地址往后请求的字节数，如果不指定的话，GDB 默认是 4 个 bytes。u 参数选择如下：

- b 表示单字节
- h 表示双字节
- w 表示四字节
- g 表示八字节

当我们指定了字节长度后，GDB 会从指定内存地址开始，读取相应内容，以相应格式显示出来。

n/f/u 三个参数可以一起使用。例如图 9 操作所示。

```

[sudo] password for mhs:
mhs@ubuntu:~/lab/asm$ sudo qemu-system-arm -M arm-linux-gnueabihf -m 128M -nographic
[sudo] password for mhs:
gdb:
sudo qemu-system-arm -M arm-linux-gnueabihf -m 128M -nographic
arm-linux-gnueabihf (gdb) c
Continuing.

dataStruct: Program exited normally.
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
[New Remote target]
[Switching to Remote target]
_start () at main.s:3
3      MOV    R0, #0
(gdb) n
4      swi    0x00900001
(gdb) x /8xw 0x0000808e
0x808e: 0x80540000  0x00080000  0x00000000  0x00000000
0x809e: 0x003e0000  0x00020000  0x00000000  0x00000104
(gdb)
  
```

图 9 内存查看命令

命令：x /8xw 0x0000808e，打印地址 0x0000808e 起始的内存内容，连续 8 乘 4 字节，以 16 进制 4 个字节格式输出。

4 作业布置：

请大家掌握后，对课本和网课所讲的程序示例逐一进行运行调试，加深理解。

注：本实验教程为武汉科技大学机器人与智能系统研究院闵华松老师的网络课程教学文档，可以复制，不做商业用途。