

# Linux\_0.11 调试环境搭建

组名:

组员:

## 目 录

插图或附表清单.....	1
简 述.....	2
1 下载所需文件.....	5
1.1 Linux0.11 源代码.....	5
1.2 Taglist 插件.....	5
1.3 Vim 配置文件.....	5
1.4 VMWare WorkStation 8.0 .....	5
1.5 Ubuntu 11.10.....	5
2 编译 linux-0.11.....	6
2.1 解压文件.....	6
2.2 安装编译所需工具 .....	6
2.3 编译 linux-0.11.....	6
3 运行 linux-0.11.....	8
3.1 安装 QEMU.....	8
3.2 运行 Linux-0.11 .....	8
4 Vim 阅读 Linux-0.11 源代码.....	9
4.1 安装 Vim 及所需工具 .....	10
4.1.1 安装 vim, cscope 及 ctags。 .....	10
4.1.2 配置.vimrc.....	10
4.1.3 安装 Taglist .....	10
4.2 生成代码阅读所需文件.....	10
4.2.1 生成 tags 文件.....	10
4.2.2 生成 cscope 所需文件.....	10

4.3 浏览代码.....	11
4.3.1 代码跳转.....	12
4.3.2 查找变量或函数.....	13
5 调试 linux-0.11.....	15
5.1 安装所需工具.....	15
5.2 调试 linux-0.11.....	15
5.2.1 简单调试命令.....	16
5.2.2 图形化数据显示.....	17
5.3 调试汇编程序.....	19

## 插图或附表清单

图 0.1vim 阅读 linux-0.11 .....	2
图 0.2 调试汇编 head.s .....	3
图 0.3 调试 c 代码:main 函数 .....	4
图 2.1linux-0.11.tar.gz 内容 .....	6
图 2.2 编译 linux-0.11 .....	7
图 3.1 运行 Linux-0.11 .....	8
图 4.1vim 阅读 linux-0.11 源代码 .....	9
图 4.2 代码浏览 .....	11
图 4.3 跳转至 main .....	13
图 4.4 所有调用 printf 的函数 .....	14
图 5.1 运行 ddd .....	16
图 5.2 设置断点 .....	17
图 5.3task_struct *p .....	18
图 5.4 图形化数据显示 1 .....	18
图 5.5 图形化数据显示 2 .....	19
图 5.6head.s 断点 .....	20
图 5.7 调试 head.s .....	20



## Linux\_0.11 调试环境搭建

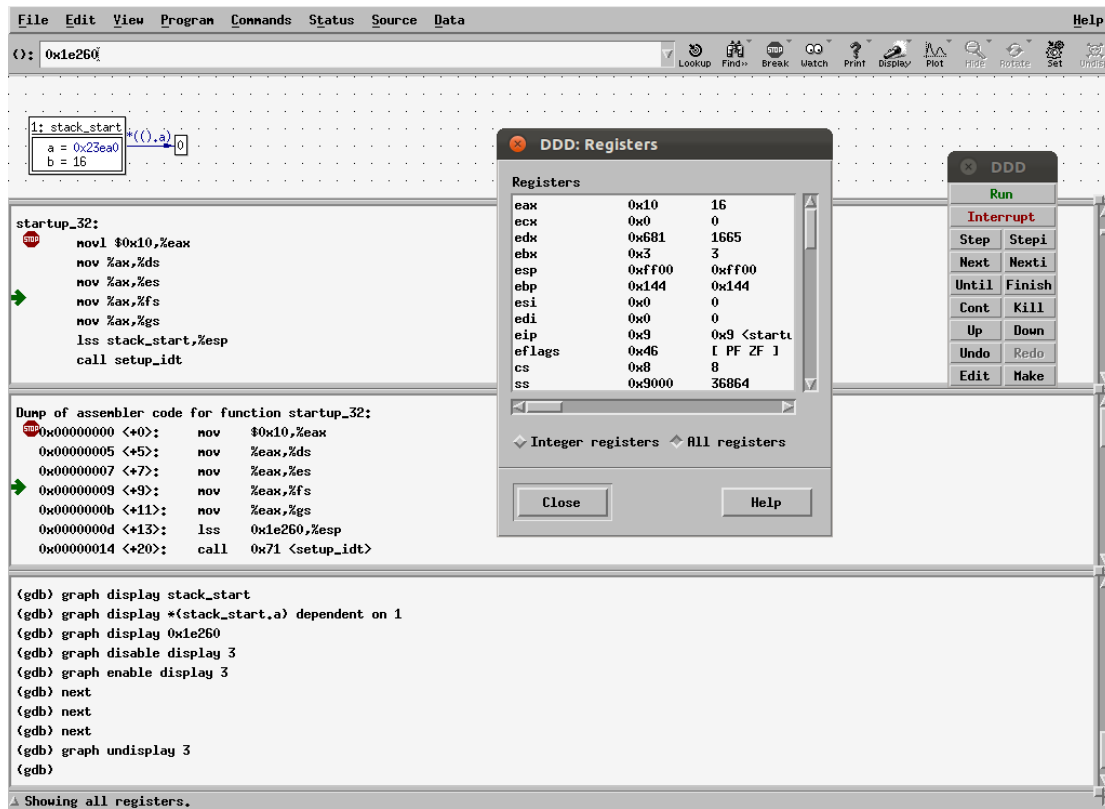


图 0.2 调试汇编 head.s

上图为调试 head.s 的情况。

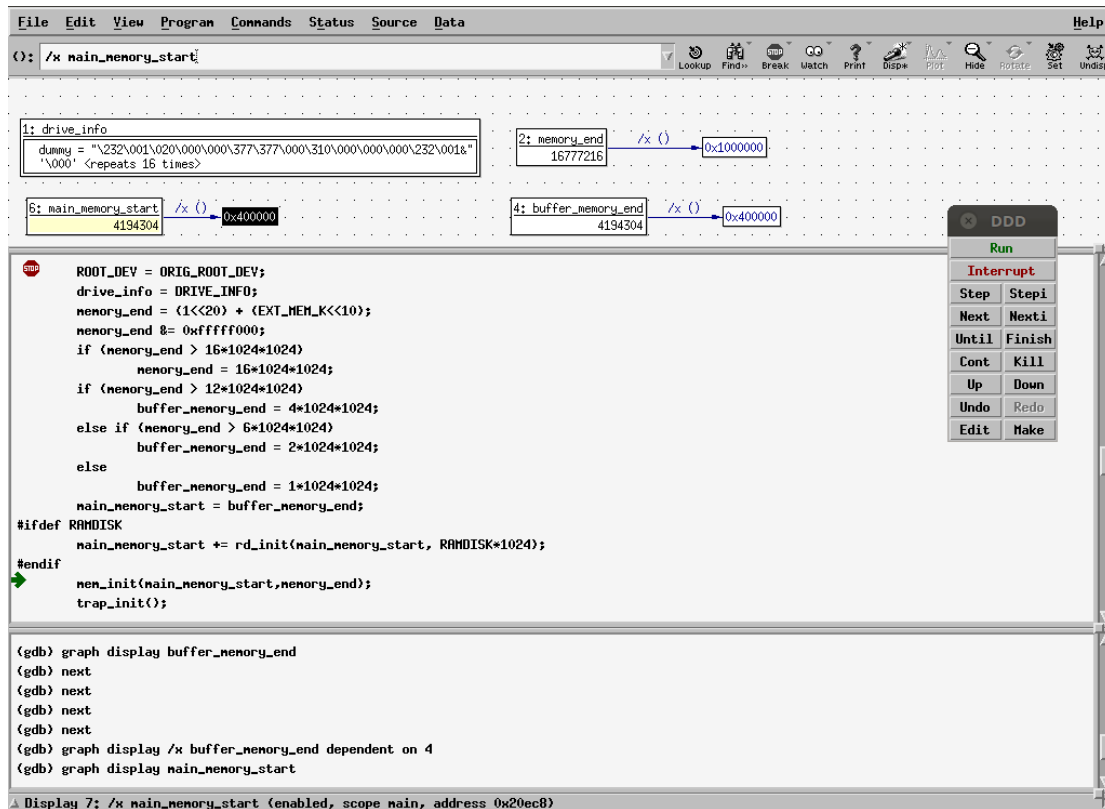


图 0.3 调试 c 代码:main 函数

环境搭建后以后，我们可以像在其他 IDE 那样，设置断点、单步运行代码、监视变量、查看寄存器等。

下面以 Ubuntu 11.10 为例展现一下整个操作过程（假定大家已经完成了 Linux 的安装并有一定的 Linux 基础）^^

## 1 下载所需文件

实验所需的文件是 linux-0.11.tar.gz、taglist\_45.zip、.vimrc，可能还需要 VMWare WorkStation 8.0 和 Ubuntu 的安装文件。

### 1.1 Linux0.11 源代码

文件名：linux-0.11.tar.gz

sha1sum: 8a1fcb6cdef98b834c72491079ee939eca337b39

源代码已经过修改，可以在 gcc 4.6.1 上编译通过，并加入了调试信息，修改自：

<http://www.oldlinux.org/oldlinux/viewthread.php?tid=13681>

### 1.2 Taglist 插件

文件名：taglist\_45.zip

sha1sum: c8056b3590a83f06ae3a16b8ae7b350dd4c87e7c

### 1.3 Vim 配置文件

文件名：.vimrc

sha1sum: a9e666d3cafe8f77248e5eaa903f535b8b0535bd

### 1.4 VMWare WorkStation 8.0

如果在虚拟机安装 Ubuntu，可以使用 VMWare WorkStation，相对比较稳定。

下载地址：

<ftp://ftp.ustb.edu.cn/pub/vmware/workstation/8.0.2/VMware-workstation-full-8.0.2-591240.exe>

sha1sum: 67af885d20a30f6074e2511f89ffff4fee321880

速度很快，使用 ipv6 下载免流量。

另外，VMTools 可以使虚拟机与物理机间更方便的进行通信。

### 1.5 Ubuntu 11.10

<ftp://ftp.ustb.edu.cn/pub/ubuntu-releases/11.10/ubuntu-11.10-desktop-i386.iso>

sha1sum: 8492d3daf0c89907c4301cb2c72094fe59037c76

速度很快，使用 ipv6 下载免流量。



## 2 编译 linux-0.11

### 2.1 解压文件

将 linux-0.11.tar.gz 解压到用户目录下:

```
$tar -xvzf linux-0.11.tar.gz -C ~/
```

内容如下:

```
boot fs hdc-0.11-new.img include init kernel lib Makefile Makefile.header mm README run.sh tools
```

图 2.1 linux-0.11.tar.gz 内容

其中 boot, fs, include, init, kernel, lib, mm, tools 为 linux-0.11 的代码; Makefile, Makefile.header 为编译所需文件; hdc-0.11-new.img 为根文件系统; run.sh 为运行及调试虚拟机的脚本。

### 2.2 安装编译所需工具

```
$sudo apt-get install build-essential  
$sudo apt-get install bin86
```

安装编译所需要的工具如 gcc、make、ld 等

### 2.3 编译 linux-0.11

```
$cd ~/linux-0.11  
$make clean  
$make
```

编译成功后会在 当前目录下生成 Image 文件, 并在 tools 文件夹下生成含所有调试信息的 system 文件。

```
make[1]: Entering directory `/home/lyg/linux-0.11/boot'
make[1]: Leaving directory `/home/lyg/linux-0.11/boot'
make[1]: Entering directory `/home/lyg/linux-0.11/boot'
make[1]: Leaving directory `/home/lyg/linux-0.11/boot'
make[1]: Entering directory `/home/lyg/linux-0.11/boot'
make[1]: Leaving directory `/home/lyg/linux-0.11/boot'
make[1]: Entering directory `/home/lyg/linux-0.11/kernel'
make[1]: Leaving directory `/home/lyg/linux-0.11/kernel'
make[1]: Entering directory `/home/lyg/linux-0.11/mm'
make[1]: Leaving directory `/home/lyg/linux-0.11/mm'
make[1]: Entering directory `/home/lyg/linux-0.11/fs'
make[1]: Leaving directory `/home/lyg/linux-0.11/fs'
make[1]: Entering directory `/home/lyg/linux-0.11/kernel/blk_drv'
make[1]: Leaving directory `/home/lyg/linux-0.11/kernel/blk_drv'
make[1]: Entering directory `/home/lyg/linux-0.11/kernel/chr_drv'
sync
make[1]: Leaving directory `/home/lyg/linux-0.11/kernel/chr_drv'
make[1]: Entering directory `/home/lyg/linux-0.11/kernel/math'
make[1]: Leaving directory `/home/lyg/linux-0.11/kernel/math'
make[1]: Entering directory `/home/lyg/linux-0.11/lib'
make[1]: Leaving directory `/home/lyg/linux-0.11/lib'
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.000108574 s, 4.7 MB/s
0+1 records in
0+1 records out
311 bytes (311 B) copied, 0.000126935 s, 2.5 MB/s
261+1 records in
261+1 records out
133796 bytes (134 kB) copied, 0.0197884 s, 6.8 MB/s
2+0 records in
2+0 records out
2 bytes (2 B) copied, 0.000121363 s, 16.5 kB/s
lyg@lyg:~/linux-0.11$ ls
boot          cscope.po.out  Image          kernel         Makefile.header  run.sh         tools
cscope.in.out fs              include        lib            mm               System.map
cscope.out    hdc-0.11-new.img  init           Makefile       README           tags
```

图 2.2 编译 linux-0.11

### 3 运行 linux-0.11

由于 bochs 的编译安装和配置较为复杂，我们选择 **qemu** 来运行 linux-0.11。

#### 3.1 安装 QEMU

```
$sudo apt-get install qemu
```

系统会自动安装 qemu 及所需工具包。

输入

```
$qemu -version
```

显示出版本信息说明安装成功。

#### 3.2 运行Linux-0.11

```
$cd ~/linux-0.11
```

```
$qemu -m 16M -boot a -fda "Image" -hda "hdc-0.11-new.img"
```

其中：

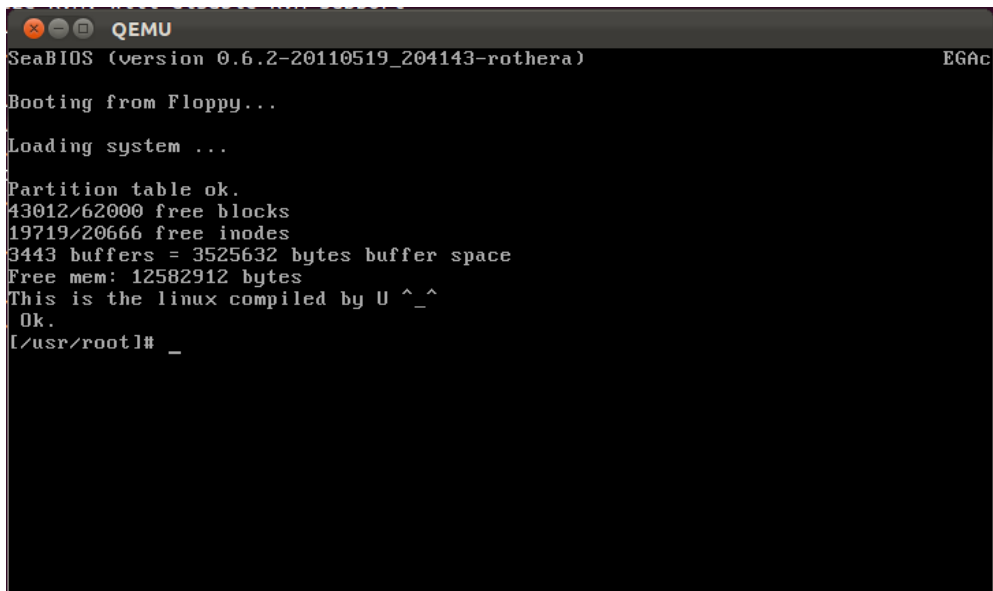
**-m**:指定虚拟机的内存大小

**-boot**:指定启动盘

**-fda**:指定软驱 **a** 的镜像

**-hda**:指定硬盘 **a** 的镜像

运行结果如下：



```
QEMU
SeaBIOS (version 0.6.2-20110519_204143-rothera) EGAc
Booting from Floppy...
Loading system ...
Partition table ok.
43012/62000 free blocks
19719/20666 free inodes
3443 buffers = 3525632 bytes buffer space
Free mem: 12582912 bytes
This is the linux compiled by U ^_^
Ok.
[usr/root]# _
```

图 3.1 运行 Linux-0.11

看到

```
This is the linux compiled by U ^_^
```

表示运行成功。大家可以加入一些自定义信息或修改一些代码，然后编译运行，一个帅气的内核就这样诞生了~

## 4 Vim 阅读 Linux-0.11 源代码

相信大家都或多或少地使用过 Vim。在 Ubuntu 下我们利用 Vim 来阅读 linux-0.11 的源代码，并通过 `ctags`、`cscope`、`taglist` 实现了变量、宏、函数等列表的显示，变量函数的定义及引用的查找等。

最终效果如下：

```

98 }
99
100 static long memory_end = 0;
101 static long buffer_memory_end = 0;
102 static long main_memory_start = 0;
103
104 struct drive_info { char dummy[32]; } drive_info;
105
106 void main(void) /* This really IS void, no error here. */
107 { /* The startup routine assumes (well, ...) this */
108 /*
109  * Interrupts are still disabled. Do necessary setups, then
110  * enable them
111  */
112
113     ROOT_DEV = ORIG_ROOT_DEV;
114     drive_info = DRIVE_INFO;
115     memory_end = (1<<20) + (EXT_MEM_K<<10);
116     memory_end &= 0xfffff000;
117     if (memory_end > 16*1024*1024)
118         memory_end = 16*1024*1024;
119     if (memory_end > 12*1024*1024)
120         buffer_memory_end = 4*1024*1024;

```

Tag List: 28,5 All init/main.c 110,9 47%

cscope tag: printf

#	line	filename / context / line
1	179	init/main.c <<init>>
2	181	init/main.c <<init>>
3	182	init/main.c <<init>>
4	183	init/main.c <<init>>
5	196	init/main.c <<init>>
6	210	init/main.c <<init>>

Type number and <Enter> (empty cancels):

图 4.1vim 阅读 linux-0.11 源代码

双击 `main` 即可跳转到 `main` 函数~

## 4.1 安装 Vim 及所需工具

### 4.1.1 安装 vim, cscope 及 ctags。

```
$ sudo apt-get install vim cscope exuberant-ctags
```

其中 **Ctags** 用于从程序源代码树中产生索引文件，以便于文本编辑器来实现快速定位；**Cscope** 用于查找变量、函数定义及引用关系等。

### 4.1.2 配置 .vimrc

.vimrc 为 vim 的配置文件，其中包含了 vim、ctags、taglist、cscope 的配置，可使 vim 变得更加方便，强大。

将下载得到的.vimrc 文件复制到~/目录下

```
$cp .vimrc ~/
```

也可以根据自己的习惯进行自定义。

### 4.1.3 安装 Taglist

将 taglist\_45.zip 解压到~/vim 中

```
$unzip taglist_45.zip -d ~/vim/
```

打开 vim，并使用下面的命令生成帮助标签（此步可忽略）

```
$vim  
:helptags ~/vim/doc
```

这样我们就可以使用

```
:help taglist.txt
```

来查看帮助文件

## 4.2 生成代码阅读所需文件

### 4.2.1 生成 tags 文件

```
$ cd ~/linux-0.11  
$ ctags -R .
```

用于从程序源代码中产生索引文件 tags。

### 4.2.2 生成 cscope 所需文件

```
$ cscope -Rbkq
```

执行后，产生 cscope.out, cscope.in.out, cscope.po.out 三个文件。

其中：

-R: 在生成索引文件时，搜索子目录树中的代码

- b: 只生成索引文件，不进入 cscope 的界面
- k: 在生成索引文件时，不搜索/usr/include 目录
- q: 生成 cscope.in.out 和 cscope.po.out 文件，用于加快 cscope 的索引速度

#### 4.3 浏览代码

插件已经配置完成，现在 vim 已经具有了较好的代码阅读功能。

```
$cd ~/linux-0.11
$vim init/main.c
```

我们用 vim 打开 main.c

```
:cscope add cscope.out
```

用于加载 cscope 生成的文件，否则 cscope 无法工作。

```
:TlistOpen
```

打开 Taglist 列表。由于我们预先在 .vimrc 中设置了

```
cmap t! TlistToggle
```

即将 “:t!” 映射为 “:TlistToggle”，所以我们也可使用 :t! 打开 Taglist 列表。结果如下：

```
" Press <F1> to display help
1 /*
2  * linux/init/main.c
3  *
4  * (C) 1991  Linus Torvalds
5  */
6
7 #define __LIBRARY__
8 #include <unistd.h>
9 #include <time.h>
10
11 /*
12  * we need this inline - forking from kernel space will result
13  * in NO COPY ON WRITE (!!!), until an execve is executed. This
14  * is no problem, but for the stack. This is handled by not letting
15  * main() use the stack at all after fork(). Thus, no function
16  * calls - which means inline code for fork too, as otherwise we
17  * would use the stack upon exit from 'fork()'.
18  */
19
20 * Actually only pause and fork are needed inline, so that there
21 * won't be any messing with the stack from main(), but we define
22 * some others too.
23
24 static inline fork(void) __attribute__((always_inline));
25 static inline pause(void) __attribute__((always_inline));
26 static inline _syscall0(int,fork)
27 static inline _syscall1(int,pause)
28 static inline _syscall1(int,setup,void *,BIOS)
29 static inline _syscall0(int,sysync)
30
31 #include <linux/tty.h>
32 #include <linux/sched.h>
33 #include <linux/head.h>
34 #include <asm/system.h>
35 #include <asm/io.h>
36 #include <stddef.h>
37 #include <stdarg.h>
```

图 4.2 代码浏览

当光标位于 Tag 列表是，可使用以下快捷键。

<CR> 跳到光标下 tag 所定义的位置，用鼠标双击此 tag 功能也一样  
o 在一个新打开的窗口中显示光标下 tag  
<Space>显示光标下 tag 的原型定义  
u 更新 taglist 窗口中的 tag  
s 更改排序方式，在按名字排序和按出现顺序排序间切换  
x taglist 窗口放大和缩小，方便查看较长的 tag  
+ 打开一个折叠，同 zo  
- 将 tag 折叠起来，同 zc  
\* 打开所有的折叠，同 zR  
= 将所有 tag 折叠起来，同 zM  
[[ 跳到前一个文件  
]] 跳到后一个文件  
q 关闭 taglist 窗口  
<F1> 显示帮助

#### 4.3.1 代码跳转

我们双击 main 即可跳转至 main 函数处。

```

" Press <F1> to display help
main.c (/home/lyg/linux-0.11)
macro
  _LIBRARY_
  EXT_MEM_K
  DRIVE_INFO
  ORIG_ROOT_DEV
  CMOS_READ
  BCD_TO_BIN
struct
  drive_info
variable
  printbuf
  memory_end
  buffer_memory_end
  main_memory_start
  drive_info
  argv_rc
  envp_rc
  argv
  envp
function
  time_init
  main
  printf
  init
88      time.tm_year = CMOS_READ(0);
89  } while (time.tm_sec != CMOS_READ(0));
90  BCD_TO_BIN(time.tm_sec);
91  BCD_TO_BIN(time.tm_min);
92  BCD_TO_BIN(time.tm_hour);
93  BCD_TO_BIN(time.tm_mday);
94  BCD_TO_BIN(time.tm_mon);
95  BCD_TO_BIN(time.tm_year);
96  time.tm_mon--;
97  startup_time = kernel_mktime(&time);
98 }
99
100 static long memory_end = 0;
101 static long buffer_memory_end = 0;
102 static long main_memory_start = 0;
103
104 struct drive_info { char dummy[32]; } drive_info;
105
106 void main(void) /* This really IS void, no error here. */
107 { /* The startup routine assumes (well, ...) this */
108 /*
109  * Interrupts are still disabled. Do necessary setups, then
110  * enable them
111  */
112
113  ROOT_DEV = ORIG_ROOT_DEV;
114  drive_info = DRIVE_INFO;
115  memory_end = (1<<20) + (EXT_MEM_K<<10);
116  memory_end &= 0xffff000;
117  if (memory_end > 16*1024*1024)
118      memory_end = 16*1024*1024;
119  if (memory_end > 12*1024*1024)
120      buffer_memory_end = 4*1024*1024;
121  else if (memory_end > 6*1024*1024)
122      buffer_memory_end = 2*1024*1024;
123  else
124      buffer_memory_end = 1*1024*1024;

```

图 4.3 跳转至 main

我们还可以跳转至某个函数，使用“Ctrl+J”快捷键可跳转至光标下的变量或函数的定义处。使用“Ctrl+T”或“Ctrl+O”快捷键可返回上一次跳转处。

### 4.3.2 查找变量或函数

利用 Cscope 我们可以查找函数、变量、文件的定义、引用已经引用的情况。命令如下：

```
:cs find c|d|e|f|g|i|s|t keyword
```

其中

s:查找 C 语言符号，即查找函数名、宏、结构体等出现的位置；

g:查找函数、宏、结构体等定义的位置，类似 ctags；

d:查找本函数所调用的函数；

c:查找调用本函数的函数；

t:查找指定的字符串；

e:查找 egrep 模式，相当于 egrep 功能，但查找速度很快；





## 5 调试 linux-0.11

我们利用 QEMU+gdb+ddd 实现像 Eclipse 或 Visual Studio 等 IDE 那样的调试 linux-0.11~

### 5.1 安装所需工具

```
$sudo apt-get install ddd
```

GNU DDD 全称为 Data Display Debugger，它可以把数据结构以图形的方式显示出来。

### 5.2 调试 linux-0.11

```
$cd ~/linux-0.11
$ qemu -m 16M -boot a -fda "Image" -hda "hdc-0.11-new.img" -S -gdb
tcp::1234&
```

其中：

**-S:**在启动时冻结 CPU

**-gdb tcp::1234:**利用 tcp 的 1234 端口与 gdb 通信

**&:**表示后台运行

这时 qemu 启动，但为黑屏，因为 CPU 被冻结~

```
$ddd tools/system
```

或直接运行：

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

用 ddd 打开带有调试信息的 linux-0.11 的内核镜像

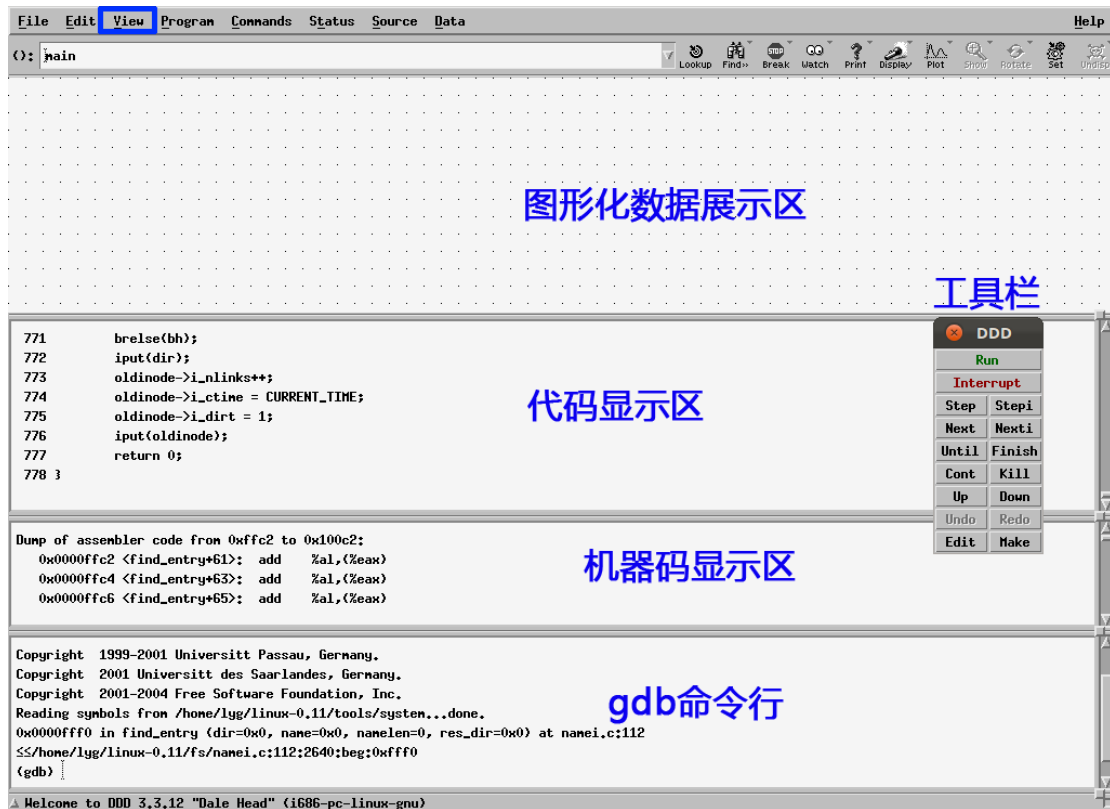


图 5.1 运行 ddd

上图 ddd 主界面中主要由“图形化数据显示区”、“代码显示区”、“机器码显示区”、“gdb 命令区”和工具栏组成，可通过“View”菜单选项进行管理。

### 5.2.1 简单调试命令

在 gdb 命令行中输入：

```
break main
```

在 main 函数入口处设置断点

```
continue
```

使虚拟机运行至 main 函数，如果提示错误可输入：

```
target remote :1234
```

指明调试对象然后再次输入 continue，结果如下图：

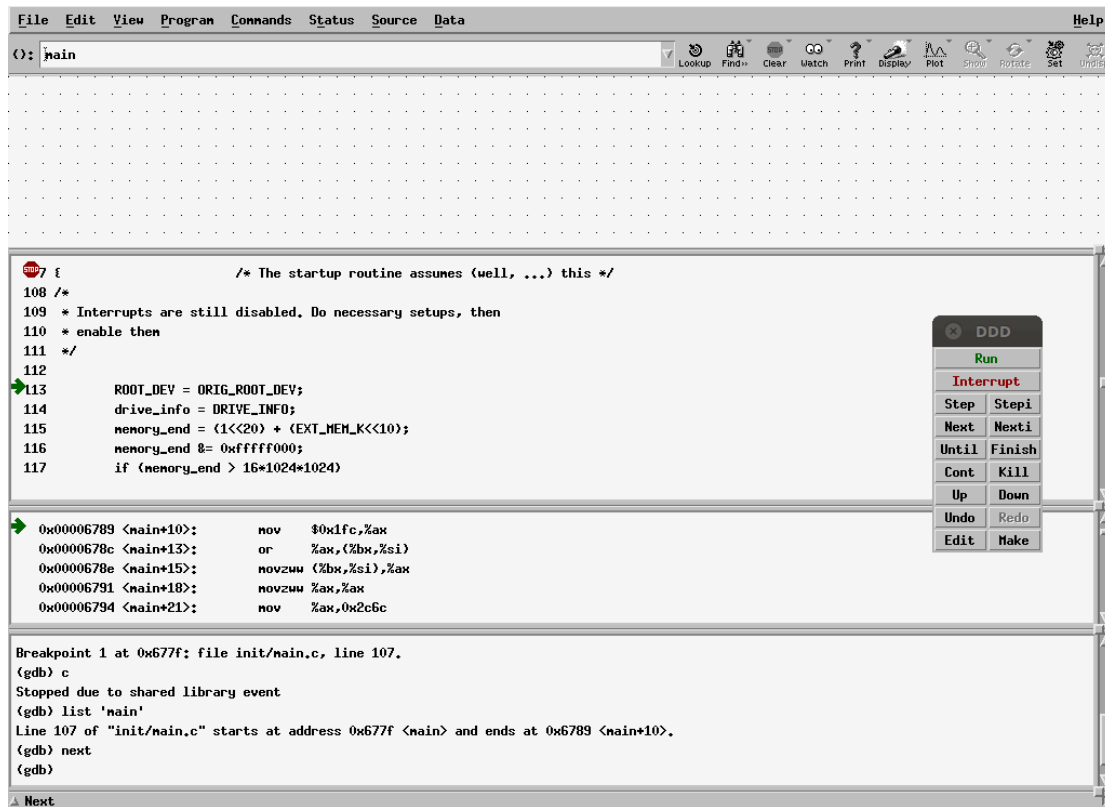


图 5.2 设置断点

我们可以通过“点击工具栏的按钮”或“在 gdb 命令中输入命令”或使用快捷键来控制内核的运行。部分命令如下：

<F5>	step	运行一行代码，可进入子程序
<Shift_F5>	stepi	运行一句指令，可进入子程序
<F6>	next	运行一行代码，跳过子程序
<Shift_F6>	nexti	运行一句指令，跳过子程序
<F9>	continue	运行至断点

### 5.2.2 图形化数据显示

DDD 可以用图形化的方式来显示数据结构，下面我们以 schedule 函数中的 struct task\_struct \*p 为例：

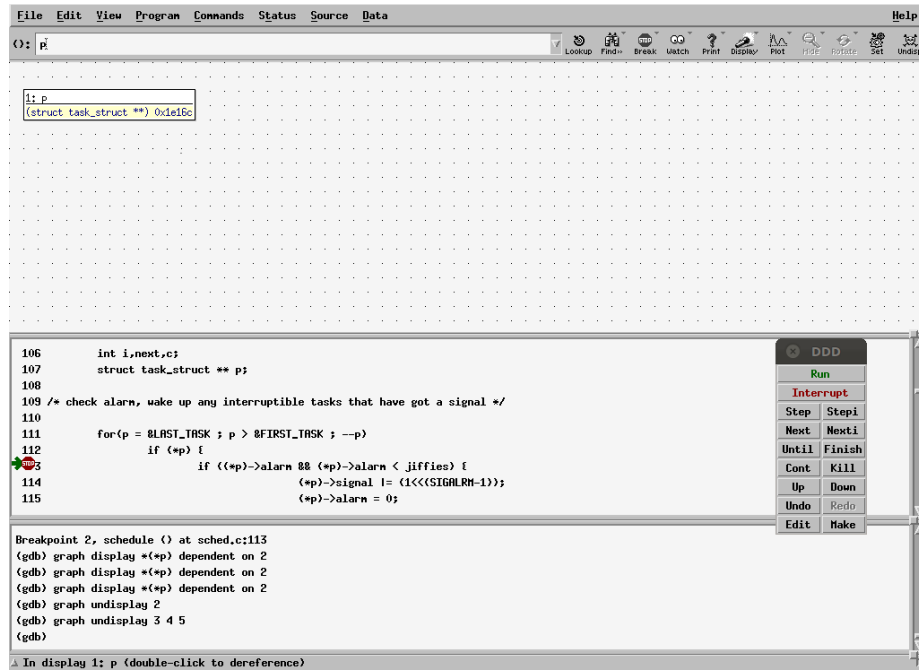


图 5.3 task\_struct \*p

双击 `p` 即可将 `p` 添加至图形显示区，我们可以通过双击指针来查看其所指的内容。

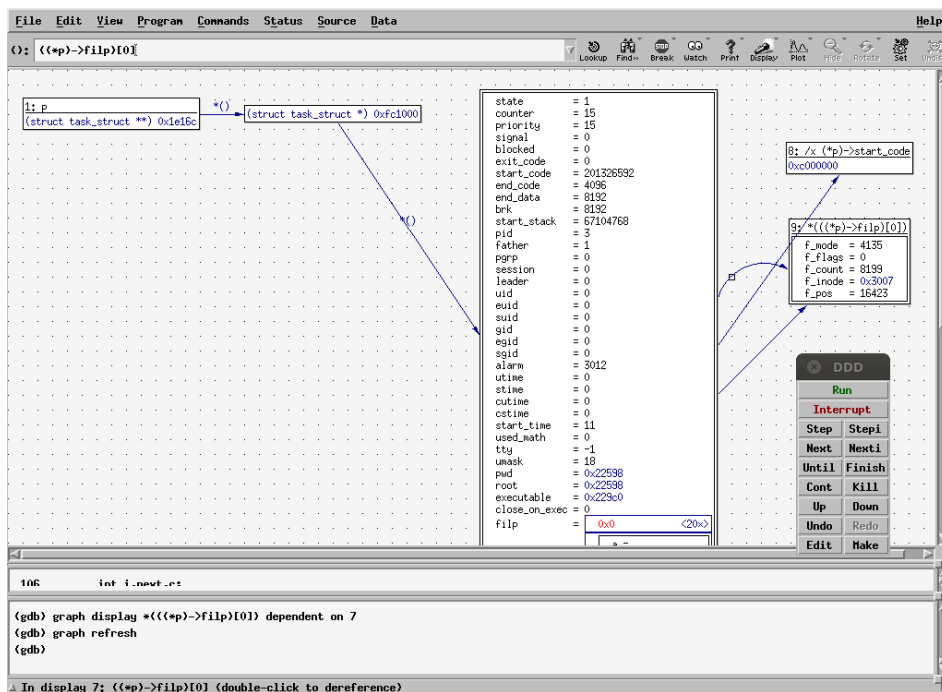


图 5.4 图形化数据显示 1

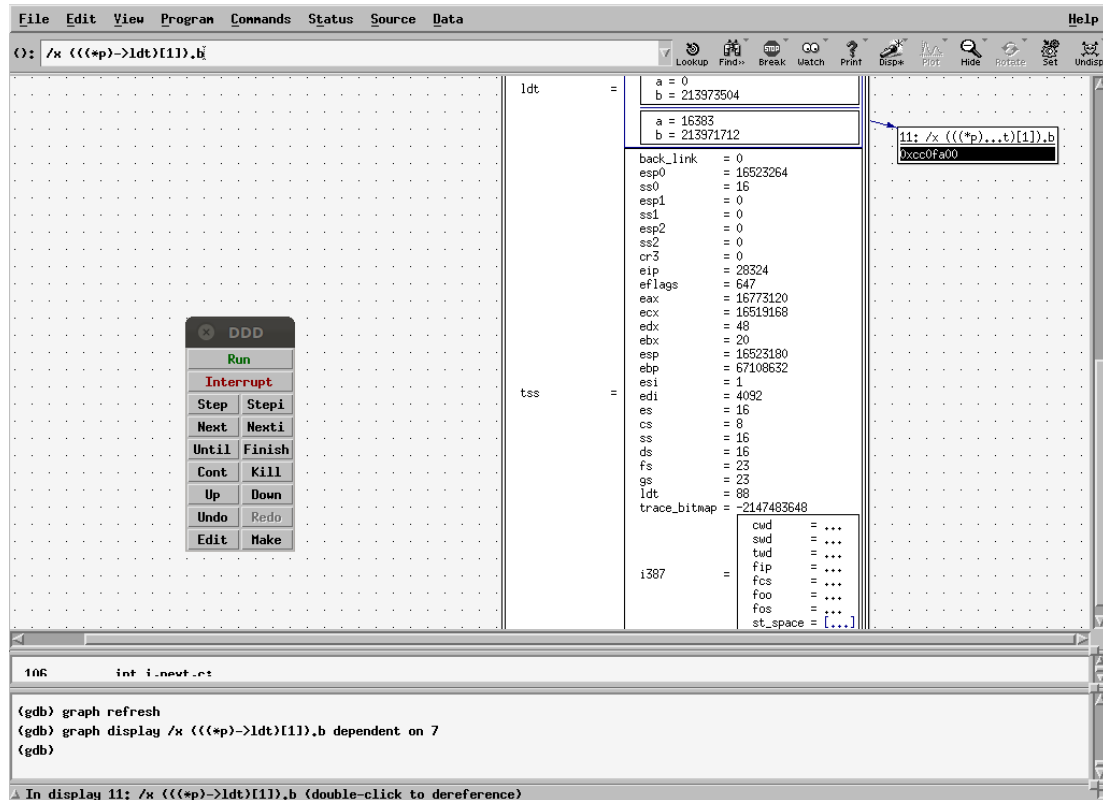


图 5.5 图形化数据显示 2

ddd 的数据展示能力远不止如此~

### 5.3 调试汇编程序

我们以 head.s 为例来展示汇编语言的调试。关闭 ddd 及 qemu 并输入

```
$cd ~/linux-0.11
$./run.sh
```

利用 run.sh 运行 ddd 及 qemu。

在 gdb 命令行内输入:

```
break startup_32
```

即在 head.s 的起点处设置断点。

```
1 /*
2  * linux/boot/head.s
3  *
4  * (C) 1991 Linus Torvalds
5  */
6
7 /*
8  * head.s contains the 32-bit startup code.
9  *
10 * NOTE!!! Startup happens at absolute address 0x00000000, which is also where
11 * the page directory will exist. The startup code will be overwritten by
12 * the page directory.
13 */
14 .text
15 .globl idt,gdt,pg_dir,tmp_floppy_area
16 pg_dir:
17 .globl startup_32
18 startup_32:
19     movl $0x10,%eax
20     mov %ax,%ds
21     mov %ax,%es
22     mov %ax,%fs
```

图 5.6 head.s 断点

输入:

continue

运行至 startup\_32

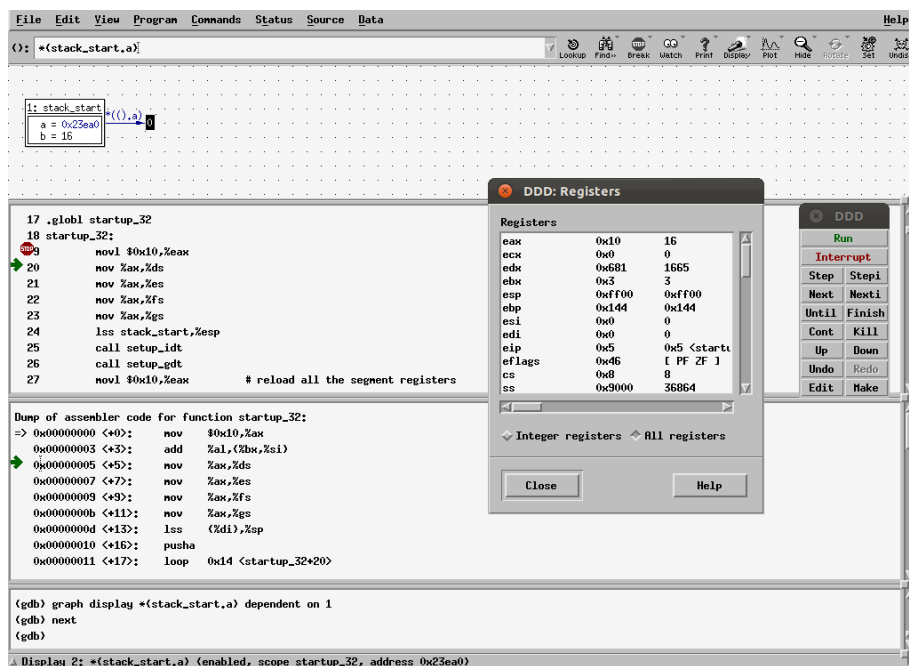


图 5.7 调试 head.s

通过“Status”->“Registers...”可查看寄存器的值，同样我们也可以像调试 c 语言那样单步执行，跟踪变量的值，查看寄存器的值等。