GDB 调试器提供了两种不同的调试代理用于支持远程调试，即 gdbserver 方式和 stub（插桩）方式。

这两种远程调试方式是有区别的。gdbserver 本身的体积很小，能够在具有很少存储容量的目标系统上独立运行，因而非常适合于嵌入式环境；而 stub 方式则需要通过链接器把调试代理和要调试的程序链接成一个可执行的应用程序文件，如果程序运行在没有操作系统的机器上，那么 stub 需要提供异常和中断处 理序，以及串口驱动程序，如果程序运行在有操作系统的嵌入式平台上，那么 stub 需要修改串口驱动程序和操作系统异常处理。显然，对于在有嵌入式操作系统 支持下的开发而言，gdbserver 比 stub 程序更容易使用。这里使用的是 GDB+gdbserver 的方式，建立远程调试的环境。

gdbserver 是一个可以独立运行的控制程序，它可以运行在类 UNIX 操作系统上，当然，也可以运行在 Linux 的诸多变种。gdbserver 允许远程 GDB 调试器通过 target remote 命令与运行在目标板上的程序建立连接。

GDB 和 gdbserver 之间可以通过串口线或 TCP/IP 网络连接通信，采用的通信协议是标准的 GDB 远程串行协议（ Remote Serial Protocol RSP）。

使用 gdbserver 调试方式时，在目标机端需要一份要调试的程序的拷贝，这通常是通过 ftp 或 NFS 下载到目标机上的，宿主机端也需要这信一份拷贝。 由于 gdbserver 不处理程序符号表，所以如果有必要，可以用 strip 工具将要复制到目标机上的程序中的符号表去掉以节省空间。符号表是由运行在主 机端的 GDB 调试器处理的，不能将主机端的程序中的符号表去掉。

虽然大部分的 Linux 发行版都已经安装了 GDB，但是那都是基于 PC 的平台的，我们要使用的是在 ARM 平台上，所以要重新下载 gdb 的源码，并修改以 适应自己的目标平台。可以从 http://www.gnu.org/software/gdb/download,获得。这里使用的是 GDB 的最新的版本 7.1。在控制台下输入下面的解包命令解压到/opt/src 处

```
#tar -jxvf gdb-7.1.tar.bz2
```

解包之后会生成 gdb-7.1/目录，所有的源码都在这个目录下，在 gdb 目录下新建立一个目录 arm-linux-gdb，把生成的可执行文件都放在这个目录下，在 gdb-7.1 目录下，输入下面的命令，配置 GDB 源码：

```
./configure --target=arm-linux --prefix=/opt/gdb --program-prefix=arm-linux-
```

其中 --target=arm-linux 选项，表示针对的目标平台是运行 linux 内核的 ARM 体系结构的平台，后面的选项--prefix=/home /frank/gdb 则是指定编译完成后的安装目录，最后一个选项--program-prefix=arm-linux-用来指定在编译生成的二进制可执行文件名之前加上前缀，这里加上的前缀是 arm-linux-。这样就可以和宿主机上的调试文件相区别。

```
make
```

把源文件编译成相应的目标文件，并连接成可执行的二进制文件 。然后，再执行下面的命令，

```
make intall
```

执行完该命令后，就会在我们刚才建立的 arm-linux-gdb 目录下生成 bin/,lib/,share/这几个子目录，其中在 bin 下有三个可执行 文件分别为：arm-linux-gdb. arm-linux-run ,arm-linux-gdbui.arm-linux-gdb,就是我们需要的调试器。

编译完 arm-linux-gdb 之后，接下来就需要编译在目标板上运行的 gdbserver 了，在 gdb/gdb7.1/gdb 下有一个 gdbserver 的子目录，这个目录包括了编译 gdbserver 所需要的所有的东西。

首先，进行 gdbserver 目录

```
./configure --target=arm-linux --host=arm-linux
```

此时，如果要直接进行编译的话，会出现错误，

```
linux-arm-low.c:26:21:sys/reg.h: 没有那个文件 或目录
make:*****[linux-arm-low.0] Error 1
```

这里的 sys/reg.h 是指/usr/include/sys/reg.h，在该文件中定义的寄存器都是针对 x86 平台的，对于运行在 ARM 平台上的 gdbserver 显然是不对的，在 configure 后，将会生成一个 config.h 文件，在这个文件中定义了相应的宏，在 config.h 中我们可以看到这样一个 C 语言的宏定义，

```
#define HAVE_SYS_REG_H 1
在 linux-arm-low.c 文件中出错的代码行：
#ifdef HAVE_SYS_REG_H
#include <sys/reg.h>
#endif
```

这 里我们只需要把 conifg.h 文件中的#define HAVE_SYS_REG_H 1 注释掉就 OK 了。由于 gdbserver 是运行在 ARM-linux 平台上的，因此需要使用交叉编译器 arm-linux-gcc，这可以通过给 make 加上 CC 参数来实现这里我使用的是默认的，你也可以使用一个绝对的路径来指定一个 arm-linux-gcc.

所有的工作都已经完成了，只要把生成的 gdbserver 下载到目标板上，或者是通过 NFS 挂载到目标板上就可能进行远程的调试了，如果就是足够幸运的 话，编译中没有出现什么错误的话，这样完全可以了，不过在我的系统上却不是那么幸运。我一直使用的是友善之臂的 arm-linux-gcc-4.3.2 交 叉编译器，这个版本的编译器中，已经自带了 arm-linux-gdb.

在终端中输入 arm-linux-gdb -v 可以看到下面的信息

```
#make CC = arm-linux-gcc
```

```
GNU gdb (Sourcery G++ Lite 2008q3-72) 6.8.50.20080821-cvs

Copyright (C) 2008 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-gnueabi".
For bug reporting instructions, please see:
<https:
```

这里可以看出 arm-linux-gcc-4.3.2 自带的 arm-linux-gdb 的版本是 6.8 版的，而我们一直编译的是 7.1 版本的。一开始我并 没有注意 arm-linux-gdb 的版本信息，不过在使用过程中，我把用生成的 gdbserver 的 7.1 版本用 NFS 加载到目标板上，而在宿主机上用 的 arm-linux-gcc 6.8 版本，一直出现下面的错误：

先在目标机上运行下面的 gdbserver，注意这里的 IP 地址是宿主机上的 IP 地址。

```
[root@Frankzfz]$gdbserver 10.27.10.48:9000 ./test_seg_fault
Process ./test_seg_fault created; pid = 760
Listening on port 9000
Remote debugging from host 10.27.10.48
```

然后在宿主机上运行

```
arm-linux-gdb/bin$ arm-linux-gdb
GNU gdb (Sourcery G++ Lite 2008q3-72) 6.8.50.20080821-cvs
Copyright (C) 2008 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-gnueabi".
For bug reporting instructions, please see:
<https://support.codesourcery.com/GNUToolchain/>.

(gdb) target remote 10.27.10.23:9000
Remote debugging using 10.27.10.23:9000
Malformed packet(b) (missing colon): ore:0;
Packet: 'T050b:00000000;0d:804ebdbe;0f:b0070040;thread:2f8;core:0;'

(gdb) symbol-file test_seg_fault
Reading symbols from /home/zfz/kernel/fs/arm-linux-gdb/bin/test_seg_fault...done.
(gdb) break main
Breakpoint 1 at 0x84a8: file test_seg_fault.c, line 7.
(gdb) info main
Undefined info command: "main". Try "help info".
(gdb) info b
Num Type Disp Enb Address What
```

```
1 breakpoint keep y 0x000084a8 in main at test_seg_fault.c:7
(gdb) c
The program is not being run.
(gdb)
```

输入 c 命令时说是程序没有运行，可是程序已经在目标板上运行了，不用输入 run 命令，当输入 target remote 10.27.10.23:9000 结束后，在 minicom 中可以看到目标板上的输出信息有了变化.

```
[root@Frankzfz]$gdbserver 10.27.10.48:9000 ./test_seg_fault
Process ./test_seg_fault created; pid = 760
Listening on port 9000
Remote debugging from host 10.27.10.48
readchar: Got EOF
Remote side has terminated connection. GDBserver will reopen the connection.
Listening on port 9000
```

在这里捣鼓了一整天没有弄明白错误在那里，看到了 arm-linux-gdb 和 gdbserver 的版本不匹配，所以就变换 arm-linux-gdb，这里要把 arm-linux-gdb 加入到.profile 文件中，或者 environment 文件中，

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/lib/jvm/j
dk1.6.0_17/bin:/home/zfz/kernel/fs/arm-linux-gdb/bin:/home/zfz/linux-
tool/usr/local/arm/4.3.2/bin"
```

当时一直在，arm-linux-gdb/bin 目录下执行，其实执行的也都是 arm-linux-gcc-4.3.2 中的 arm-linux-gdb， 如果这样你看到的版本信息还是 6.8 版本的话，也可以直接在 arm-linux-gcc-4.3.2 中把 arm-linux-gdb 文件删除掉。

这样再一次查看 arm-linux-gdb 的信息。

```
GNU gdb (GDB) 7.1
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=x86_64-unknown-linux-gnu --target=arm-linux".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
```

使用 GDB 进行调试，首先，我的宿主机的 IP 10.27.10.48，开发板上的 IP 10.27.10.23 在开发板上运行 gdbserver 10.27.10.48:9000 test,这里是通过 NFS，把要调试的程序加载到目标版上面的。这样就可从在宿主机上运行。

```
zfz@zfz:~/kernel/fs/arm-linux-gdb/bin$ ./arm-linux-gdb test
./arm-linux-gdb: Symbol `acs_map' has different size in shared object, consider re-linking
GNU gdb (GDB) 7.1
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=x86_64-unknown-linux-gnu --target=arm-linux".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/zfz/kernel/fs/arm-linux-gdb/bin/test...done.
(gdb) target remote 10.27.10.23:9000
Remote debugging using 10.27.10.23:9000
warning: Unable to find dynamic linker breakpoint function.
GDB will be unable to debug shared library initializers
and track explicitly loaded dynamic code.
0x400007b0 in ?? ()
(gdb) 1
Cannot access memory at address 0x0
1    #include <stdio.h>
2    int main( void )
3    {
4     int i=2;
5     int x, y;
6
7     x=(++i);
8     printf(" %d %d\n", i,x);
9     x+=(++i);
10     printf(" %d %d\n", i,x);
(gdb) 1
11     x+=(++i);
12     printf(" %d %d\n", i,x);
13     i=2;
14     y=(i++)+(i++)+(i++);
15     printf(" %d %d\n", i,y);
16
17     return 0;
18    }
19
(gdb) break 9
Breakpoint 1 at 0x83b8: file test.c, line 9.
(gdb) c
Continuing.
Error while mapping shared library sections:
`/lib/libc.so.6': not in executable format: File format not recognized
Error while mapping shared library sections:
/lib/ld-linux.so.3: No such file or directory.

Breakpoint 1, main () at test.c:9
9     x+=(++i);
(gdb) step
10     printf(" %d %d\n", i,x);
(gdb) next
11     x+=(++i);
(gdb) next
12     printf(" %d %d\n", i,x);
```

```
(gdb) next
13      i=2;
(gdb) next
14      y=(i++)+(i++)+(i++);
(gdb) next
15      printf(" %d %d\n", i,y);
(gdb) next
17      return 0;
(gdb) next
18    }
(gdb) next
```

在目标板上的输出：

```
[root@Frankzfz]$gdbserver 10.27.10.48:9000 ./test
Process ./test created; pid = 746
Listening on port 9000
Remote debugging from host 10.27.10.48
 3 3
 4 7
 5 12
 5 6
```

补充：前面还有库文件错误的问题，这里可以通过下面这样的方法的来解决。需要说明的是，远程调试，可能程序动态库，我们可以这样设置：

```
set solib-absolute-prefix /nfsroot/rootfs
set solib-search-path /nfsroot/rootfs/lib
```

调试找不到源代码，如下设置：

```
directory xfer-1.0.0/src/
```

下面是调试的情况：

```
(gdb) set solib-absolute-prefix /home/kernel/fs/root_nfs
(gdb) set solib-search-path /home/kernel/fs/root_nfs/lib
(gdb) target remote 10.27.10.23:9000
Remote debugging using 10.27.10.23:9000
Reading symbols from /home/kernel/fs/root_nfs/lib/ld-linux.so.3...(no debugging symbols

found)...done.
Loaded symbols for /home/kernel/fs/root_nfs/lib/ld-linux.so.3
0x400007b0 in ?? () from /home/kernel/fs/root_nfs/lib/ld-linux.so.3
(gdb) 1
3     {
4       int i=2;
5       int x, y;
6
7       x=(++i);
8       printf(" %d %d\n", i,x);
9       x+=(++i);
10       printf(" %d %d\n", i,x);
11       x+=(++i);
12       printf(" %d %d\n", i,x);
(gdb) 1
13      i=2;
```

```
14      y=(i++)+(i++)+(i++);
15      printf(" %d %d\n", i,y);
16
17      return 0;
18     }
19
(gdb) b 8
Note: breakpoint 1 also set at pc 0x83a8.
Breakpoint 2 at 0x83a8: file test.c, line 8.
(gdb) c
Continuing.

Breakpoint 1, main () at test.c:8
8      printf(" %d %d\n", i,x);
(gdb) n
9      x+=(++i);
(gdb) b 12
Breakpoint 3 at 0x8400: file test.c, line 12.
(gdb) n
10      printf(" %d %d\n", i,x);
(gdb) n
11      x+=(++i);
(gdb) n

Breakpoint 3, main () at test.c:12
12      printf(" %d %d\n", i,x);
(gdb) n
13      i=2;
(gdb) n
14      y=(i++)+(i++)+(i++);
(gdb) n
15      printf(" %d %d\n", i,y);
(gdb) n
17      return 0;
(gdb) n
18     }
(gdb) n
0x4003b004 in __libc_start_main ()
   from /home/kernel/fs/root_nfs/lib/libc.so.6
(gdb)
```