命令	效果
开始和停止	
quit	退出 GDB
run	运行程序(在此给出命令行参数)
kill	停止程序
断点	
break multstore	在函数 multstore 入口处设置断点
break * 0x400540	在地址 0x400540 处设置断点
delete 1	删除断点 1
delete	删除所有断点
执行	
stepi	执行 1 条指令
stepi 4	执行 4 条指令
nexti	类似于 stepi，但以函数调用为单位
continue	继续执行
finish	运行到当前函数返回
检查代码	
disas	反汇编当前函数
disas multstore	反汇编函数 multstore
disas 0x400544	反汇编位于地址 0x400544 附近的函数
disas 0x400540,0x40054d	反汇编指定地址范围内的代码
print /x $rip	以十六进制输出程序计数器的值
检查数据	
print $rax	以十进制输出%rax 的内容
print /x $rax	以十六进制输出%rax 的内容
print /t $rax	以二进制输出%rax 的内容
print 0x100	输出 0x100 的十进制表示
print /x 555	输出 555 的十六进制表示
print /x ($rsp+ 8)	以十六进制输出%rsp 的内容加上 8
print *(long *) 0x7fffffffe818	输出位于地址 0x7fffffffe818 的长整数
print *(long *)($rsp+ 8)	输出位于地址%rsp+8 处的长整数
x/2g 0x7fffffffe818	检查从地址 0x7fffffffe818 开始的双(8 字节)字
x/20bmultstore	检查函数 multstore 的前 20 个字节
有用的信息	
info frame	有关当前栈帧的信息
info registers	所有寄存器的值
help	获取有关 GDB 的信息

图 3-39　GDB 命令示例。说明了一些 GDB 支持机器级程序调试的方式

正如我们的示例表明的那样，GDB 的命令语法有点晦涩，但是在线帮助信息(用 GDB 的 help 命令调用)能克服这些毛病。相对于使用命令行接口来访问 GDB，许多程序员更愿意使用 DDD，它是 GDB 的一个扩展，提供了图形用户界面。

3.10.3　内存越界引用和缓冲区溢出

我们已经看到，C 对于数组引用不进行任何边界检查，而且局部变量和状态信息(例如保存的寄存器值和返回地址)都存放在栈中。这两种情况结合到一起就能导致严重的程序错误，对越界的数组元素的写操作会破坏存储在栈中的状态信息。当程序使用这个被破