Loongson Exploration

实验 2 龙芯平台初探

实验 2.0

熟悉实验环境

一．实验平台

多路处理器计算机教学实验系统是由四片四核龙芯 3A 处理器构成的 16 核 CC-NUMA 结构、内可配置外可扩展结构的实验硬件平台。实验系统特点如下：

（1）多种并行层次：多发射、多核、多路、多机

（2）多种互连方式：片上网络、HyperTransport、以太网

（3）多种存储结构：CMP/SMP、CC-NUMA

（4）多种编程模式：Pthread、MPI、OpenMP

该实验系统是由多路处理单元和前端控制单元组成，多路处理单元上有对称的 4 个计算 节点，每个计算节点包含一颗龙芯 3A 四核处理器。4 个计算节点既可通过网络互连为多处理机集群架构，也可通过 HT 总线互连为 CC-NUMA 架构。前端控制单元由龙芯 2H 构成，为计算节点提供内核和网络文件系统。实验系统主板的基本结构和软硬件结构图如下所示。

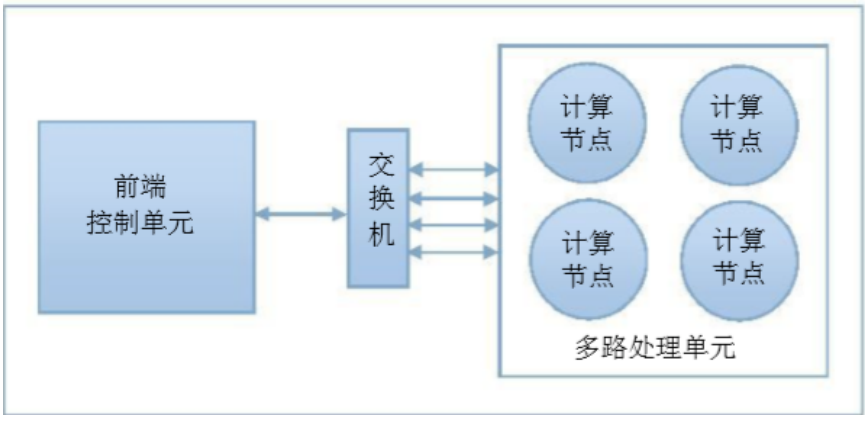


图 1 实验系统结构图

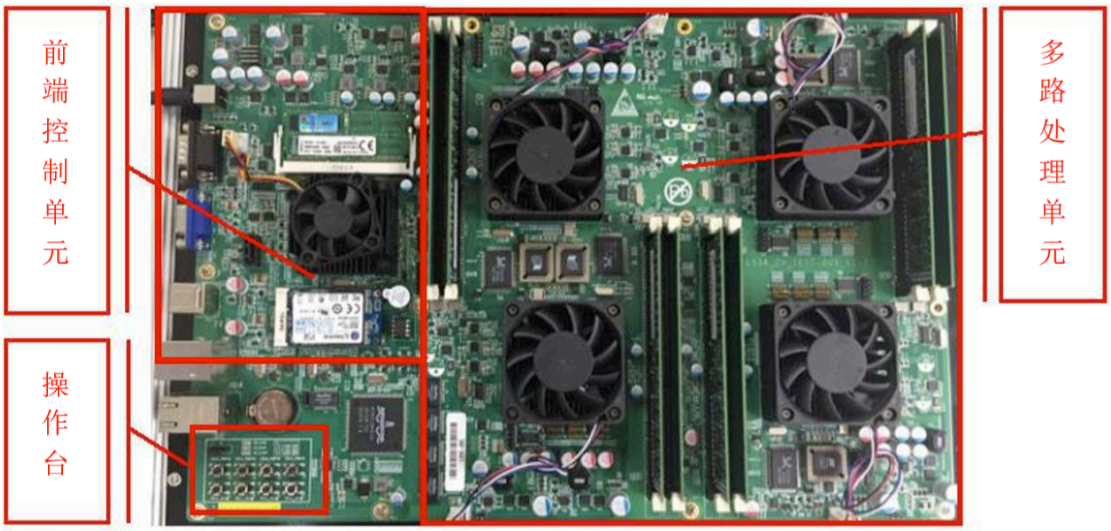


图 2 实验系统实物图

二．前端控制单元

前端控制单元集成了单片高性能龙芯 2H 处理器，1 个 DDR3 SODIMM 插槽，两路 10/100/1000Mbps 自适应网络控制器，1 个千兆网口，1 个 M-SATA 插槽，1G NAND FLSAH，4 个 USB2.0 接口，1 个 VGA 接口和 1 个串口。实物图为图 3，其中标注出前端控制单元所包括的主要器件。

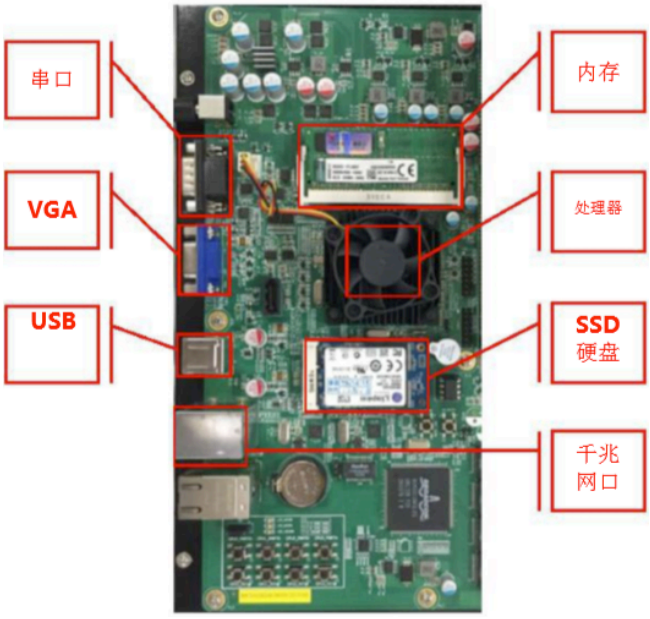


图 3 前端控制单元实物图

三．多路处理单元

多路处理单元承载 4 个对称的计算节点，编号如图 2-2 所示，每个计算节点包括一颗龙芯 3A 处理器、DDR3 内存、BIOS Flash、串口收发芯片以及电源变换电路等。四个龙芯 3A 处理器在处理板上通过 HT 总线实现互连。多路处理单元的结构示意图如图 4 所示。

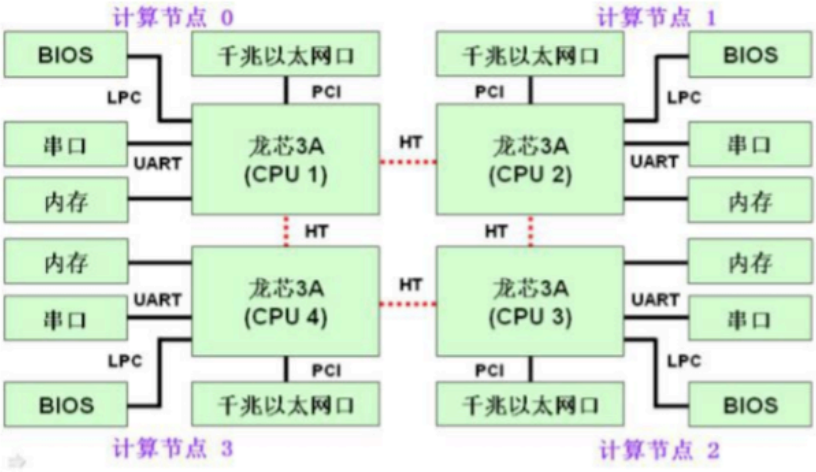


图 4 多路处理单元结构示意图

实物图为图 5，其中标注出计算节点号以及单个计算节点所包括的主要器件。本主板 CPU 频率为 800MHz，内存频率 300MHz，HT 拨码开关采用默认配置，各计算节点配置参数一致。

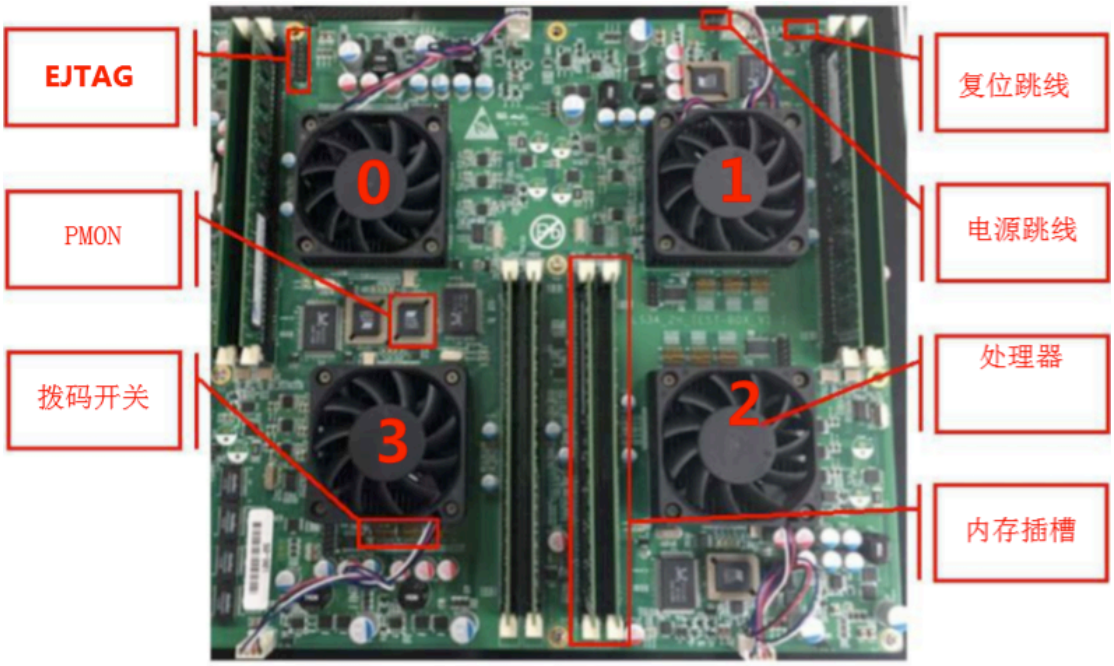


图 5 多路处理单元实物图

实验 2.1

C 语言编程—字符操作

一、 实验要求

(1) 显示键入的字符的 16 进制 ASCII 码；

(2) 将键入的字符进行大小写转换并显示。

二、 实验目的

(1) 了解数据在计算机内的编码表示，熟悉常见字符的 16 进制 ASCII 码；

(2) 初步掌握 C 语言，能够用 C 语言进行简单编程实验，掌握键盘输入及显示输出的方法；

(3) 熟悉 Linux 环境，学会如何在 Linux 环境下完成 C 程序的编写和运行。

附录：知识点总结

1. Linux 环境下文本编辑器的使用

Linux 环境下可以使用 vi/vim 文本编辑器编写程序。vi 的基本使用方式总结如下：

进入 vi 编辑界面一般分为三种模式，一般模式、编辑模式、命令行模式。

- 一般模式

以 vi 打开一个文件就直接进入一般模式（这是默认模式）。在这个模式中，你可以上下左右的移动光标，也可以粘贴复制或是删除文件数据，但不可以编辑文件内容。

- 编辑模式

在一般模式中按下“i，I，o，O，a，A，r，R”任一字母之后才可进入编辑模式。通常在 Linux 中按下这些按键后，在界面左下方会出现“INSERT”或“REPLACE”的字样，此时就可以进行编辑。如果要回到一般模式，则需按下【Esc】。

- 命令行模式

在一般模式中输入“:，/，？”任一个按键，就可以将光标移动到最下面那一行。在这个模式中可以提供你查找数据的操作，而读取、保存、大量替换字符、离开 vi、显示行号等则是在此模式中完成。如果要回到一般模式，则需按下【Esc】。

进入 vi 编辑界面后按下“i”按键进入编辑模式，开始编辑 C 语言代码。代码写完后按【Esc】退回到一般模式，按下“:wq”保存退出。如果发现程序有错误，可以“vi 文件名”重新编辑。

2. GCC 编译器的使用

GCC 常用命令总结如下：

- 使用-o 可以指定输出的可执行文件名称

```
lin@lin-desktop:~$ gcc test.c -o test
```

- 指定-E 编译选项，使得只输出预编译结果

```
lin@lin-desktop:~$ gcc -E test.c -o test.i
lin@lin-desktop:~$ cat test.i
```

文件 test.i 中存放着 test.c 经预处理之后的代码。“cat test.i”可以打开 test.i 文件。

- 通过编译选项-S 输出汇编代码

```
lin@lin-desktop:~$ gcc -S test.c -o test.s
lin@lin-desktop:~$ cat test.s
```

文件 test.s 会包含 test.c 的汇编输出代码。

- 指定-C 输出编译后的代码

```
lin@lin-desktop:~$ gcc -C test.c -o test.o
```

文件 test.o 包含机器指令或编译后的代码。

- 通过编译选项-save-temps 输出所有中间代码

```
lin@lin-desktop:~$ gcc -save-temps test.c
lin@lin-desktop:~$ ls
a.out  test.c  test.o  公共的  配置文件  图
study  test.i  test.s  模板    视频      文
```

大家可以自行搜取更多资料学习 linux 环境下的文本编辑和程序运行知识。

实验 2.2

文件读写及加解密

一.实验要求

(1) 读取已定义内容的文件，将其内容加密后，保存到另一个文件。

(2) 读取加密后的文件内容，实现解密算法进行解密，并与源文件对照验证正确性。

二.实验目的

以读写文件为例，掌握计算机系统中程序的执行流程。

Defusing Binary-Bomb

实验 3 二进制炸弹拆除

一. 实验要求

要求根据反汇编指令分析程序运行需要的参数,即需要正确的输入,以拆除炸弹。

二. 实验目的

(1) 熟悉 MIPS 指令集；
(2) 根据反汇编程序可以分析程序的功能和执行流程；
(3) 熟悉 GDB 调试工具，帮助程序理解。

三. 实验配置与步骤

1.准备工作

本实验设计为一个黑客拆解二进制炸弹的游戏。我们仅给黑客（同学）提供一个 MIPS 二进制可执行文件 bomb ,不提供源代码(提供汇编代码帮助理解)。程序运行中有 6 个关卡（6 个 phase），每个关卡需要用户输入正确的字符串或数字才能通关，否则会引爆炸弹（打印出一条错误信息）!

要求同学运用 GDB 调试工具,通过分析汇编代码,找到在每个 phase 程序段中,引导程序跳转到“explode_bomb”程序段的地方,并分析其成功跳转的条件，以此为突破口寻找应该在命令行输入何种字符串来通关。

需要准备的软件：

- bomb 可执行文件（MIPS）
- bomb 汇编文件

  链接：https://pan.baidu.com/s/18QUuHK_60kW_FXUK13jMcQ

  提取码：ppl5

需要准备的硬件：

- 龙芯实验平台（MIPS）

2. 拆除 phase_1 演示

2.1 参考命令

```
Dump of assembler code for function phase_1:
   0x00400d6c <+0>: addiu   sp,sp,-32
   0x00400d70 <+4>: sw  ra,28(sp)
   0x00400d74 <+8>: sw  s8,24(sp)
   0x00400d78 <+12>:    move    s8,sp
   0x00400d7c <+16>:    sw  a0,32(s8)
   0x00400d80 <+20>:    lw  a0,32(s8)
   0x00400d84 <+24>:    lui v0,0x40
   0x00400d88 <+28>:    addiu   a1,v0,10092
   0x00400d8c <+32>:    jal 0x401cf8 <strings_not_equal>
   0x00400d90 <+36>:    nop
   0x00400d94 <+40>:    beqz    v0,0x400da4 <phase_1+56>
   0x00400d98 <+44>:    nop
   0x00400d9c <+48>:    jal 0x4021f0 <explode_bomb>
   0x00400da0 <+52>:    nop
   0x00400da4 <+56>:    move    sp,s8
   0x00400da8 <+60>:    lw  ra,28(sp)
   0x00400dac <+64>:    lw  s8,24(sp)
   0x00400db0 <+68>:    addiu   sp,sp,32
   0x00400db4 <+72>:    jr  ra
   0x00400db8 <+76>:    nop
End of assembler dump.
```

图一　phase_1 的二进制代码

(1) **addiu　sp,sp,-32**　:调用子函数为当前函数开辟栈空间；

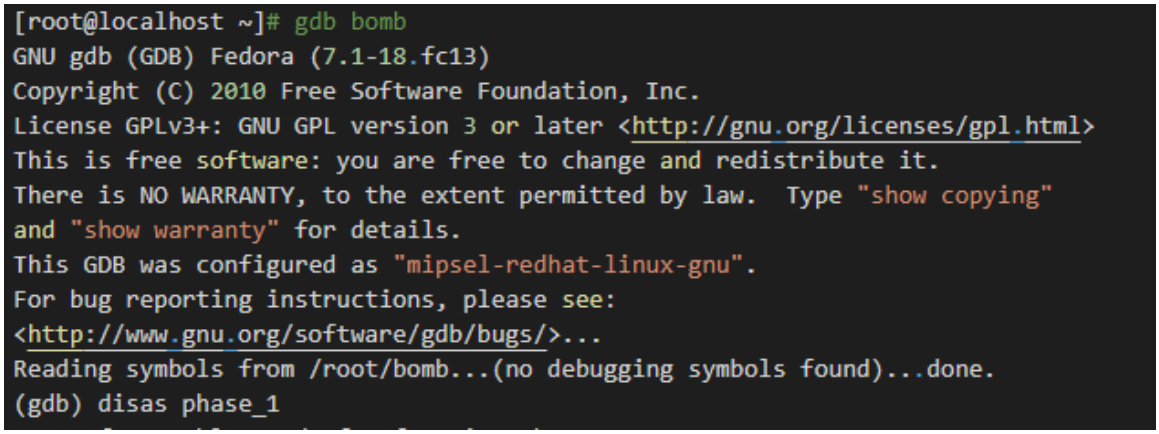
(2) **sw　　a0,32(s8)**　:$a0(参数寄存器)的内容被存入栈；
(思考：为什么需要对$a0 执行 sw 和 lw 操作？)

(3) **addiu a1,v0,10092**　：加载另外一个参数到$a1;（结合前一步的 lui 命令理解）

参数寄存器$a0 $a1 将参数传入 **strings_not_equal** 函数后进行比较，返回值放在$v0 内，看是否为 0（beqz），不为 0 则炸弹爆炸，因此可以推断出两个参数的值需要相同。接下来在实验平台中使用 GDB 调试一下，看看寄存器内是不是我们期望的内容。

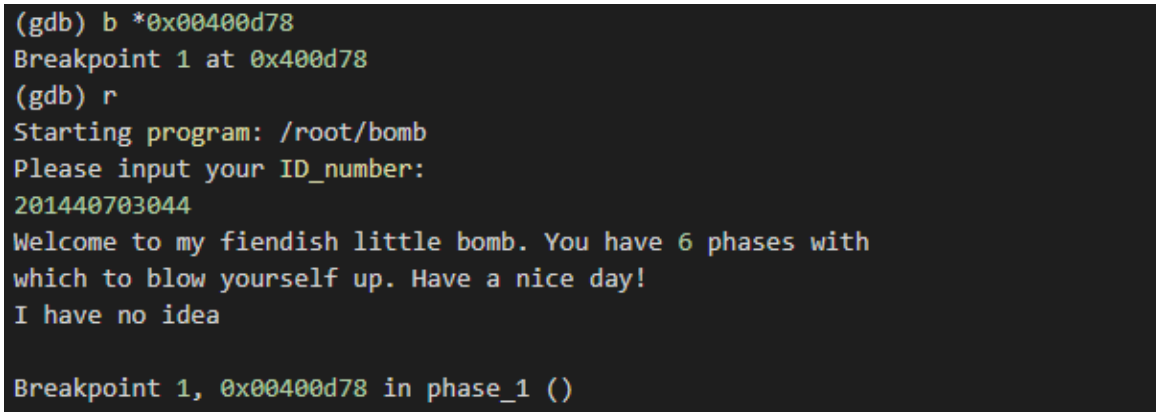
2.2 调试步骤

(1)打开终端，进入 gdb 调试窗口。



图二 使用 GDB 进行反汇编

(2)设置断点，然后用 r 命令运行程序。输入一串字符后会得到命中断点的提示。



图三 设置断点并运行

(3)使用 ni 命令进行单步调试，之后使用 **i r $a0** 和 **i r $a1** 可以查看当前 a0 和 a1 的寄存器状态（也可以直接使用 **i r** 命令查看所有寄存器的状态）。同时可以配合 x 命令查看内存中存储的数据，可以看到 a0 里面确实保存着我们输入的字串。

```
(gdb) i r $a0
a0: 0x413294
(gdb) x $a0
0x413294 <input_strings>:    0x61682049
(gdb) i r $a1
a1: 0x0
```

图四 查看寄存器内容

(4)继续使用 ni 命令，可以看到 a1 里面保存的字符串是“Let’s begin now!”。

```
(gdb) x /16c $a1
0x40276c:   76 'L'  101 'e' 116 't' 39 '\'' 115 's' 32 ' '  98 'b'  101 'e'
0x402774:   103 'g' 105 'i' 110 'n' 32 ' '  110 'n' 111 'o' 119 'w' 33 '!'
```

图五 参数寄存器内容

(5)退出当前调试，正确输入第一个炸弹内容并继续调试。

```
(gdb) r
Starting program: /root/bomb
Please input your ID_number:
201440703044
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Let's begin now!
Phase 1 defused. How about the next one?
```

图六 拆除第一个炸弹

3.接下来的提示

上面给大家演示了拆除炸弹 1 的过程，这也是一个最简单的炸弹。接下来的 5 个炸弹难度逐级递增，大家不要拘泥于代码，死读代码；要结合 GDB 调试器，查看内存以及各个寄存器的值，这样对于绕过一些棘手的函数很有帮助，可以让大家切中炸弹爆炸条件的要害进行分析。GDB 的具体使用也不止上面所说的几个命令，希望大家可以自行上网搜索，掌握 GDB 的其他命令。GDB 调试器可以在拆除炸弹的过程中给予你很大的帮助，但是仍希望你将它当作一个辅助的工具来使用，帮助你更好的了解与汇编代码有关的内容，对汇编代码的理解是该实验的重点。

祝大家拆弹顺利！

Buffer Overflow

实验 4 缓冲区溢出

一．实验题目及要求

缓冲区溢出实验：要求了解缓冲区溢出原理，程序栈结构及参数传递规则，完成后面的实验。

二．实验目的

- 了解缓冲区溢出的原理；
- 了解程序的栈结构、函数调用以及参数传递的规则；
- 了解 MIPS 指令是如何编码的；
- 加深对 debug 工具（如 GDB）的了解。

三.实验配置与步骤

0 准备工作：

本实验利用了缓冲区溢出的原理，需要你来**构造特定的字符串**，这些字符串将被加载到缓冲区中，来完成溢出的操作。在实验中，你将要完成的任务是：对**栈中变量进行修改**，以此达成**跳转到指定函数**的目的。该实验将帮助你了解 MIPS 架构下栈中的数据是怎么组织的，程序跳转的时候栈和寄存器是如何配合的，以及如何来编写一段 MIPS 汇编程序。

我们提供给你的有两个可执行文件：**attack** 和 **hex2str**。我们的实验设计在 attack 中，hex2str 是提供给你的一个将 **16 进制编码的数据转化为字符串**的小工具。实验中你需要使用 **GDB** 工具来帮你查看程序执行中寄存器和相应内存单元的值。

有一些需要注意的事项请查看下面的 **说明** 部分。

需要准备的软件：

- attack 可执行文件（MIPS）
- hex2str 可执行文件（MIPS）
- GDB

链接：https://pan.baidu.com/s/1jLO6kU-4ImWdVob4cU0h7Q

提取码：vrqn

1 说明：

- 下载文件之后你可能无法运行(**如果可以正常运行请忽略**),因为没有足够的权限，你需要为两个文件添加可执行的权限：**sudo chmod +x target_file**，输入用户密码之后就可以为 target_file 文件添加可执行的权限。
- attack 是我们实验主要用到的文件。attack 文件命令行调用方式为: **./attack input_file**，其中 input_file 中存放了我们想要注入缓冲区的内容。
- attack 程序接受字符串的输入，我们在**构造要填充的缓冲区内容的时候更容易理解的是 16 进制形式，**这里我们为了方便你将 16 进制转换为字符串形式，提供了这样的一个工具 :hex2str。其调用方法为： **./hex2str input_file > output_file。** input_file 是你构造的 16 进制的文件， output_file 中保存的就是转化后的字符串。input_file 需要注意的是，每两个 16 进制字符为一组，不同组之间可以用空格或者回车隔开，如下：

```
30 30 30
30 30 30 30 30
```

另外，要保证你输入的数据中**不能包含 0x0a**，因为它代表了换行符，在 attack 程序将字符串载入缓冲区的时候碰到换行符就会中断对后面数据的读取。

- 在你开始做实验之前，需要首先关闭系统的地址随机化机制。使用 **sudo –i (root 用户请忽略)** 获取管理员权限，接下来输入命令：

  echo 0 > /proc/sys/kernel/randomize_va_space

- MIPS 默认的数据存储方式是**小端模式**，在填充数据的时候应**注意顺序**以保证向缓冲区中写入的数据符合你的预期。

2 实验要求

- 实验目的：

  通过缓冲区溢出的方式跳转到 touch 函数。

- 实验内容：

实验二用到的缓冲区位于 getbuf 函数中。getbuf 函数源代码如下：

```
void getbuf(){
    char buf[BUFFER_SIZE];
    Gets(buf);
}
```

图 1 getbuf 源码

在 getbuf 中我们申请了一个长度为 BUFFER_SIZE(40)的字符型数组 ,然后调用 Gets 函数对其进行填充。touch 函数的源代码为：

```
void touch(){
    printf("Touch!: You called touch()\n");
    exit(0);
}
```

图 2 touch 源码

如果成功跳转，系统会打印出 Touch1!: You called touch1()。

提示：跳转到 touch1 的关键是**覆盖 getbuf 函数的返回地址**，这样**当 getbuf 函数执行结束之后就会转到你指定的位置去执行。**首先你应该找到 **getbuf 函数的返回地址在哪里，然后确定你要跳转到的位置。**