**Lab 2: 数据表示**

姓名：刘瑞洋 学号：2200400618 班级：2204104 实验教师：华栋 日期：2024.11.19

|  |  |
| --- | --- |
| **实验内容** | 1. 环境搭建 2. 配置 GDB 3. 快速排序程序调试（附源码） 4. CMU Data Lab系列实验 |
| 操作记录：   1. 环境搭建   gdb安装    Data Lab 实验包解压缩，编译生成 btest 验证工具  进入到实验目录，利用make命令编译生成btest工具     1. 配置GDB   利用vim ~/.gdbinit创建并打开配置文件后，加入一些参数来改变gdb的提示符，设置gdb提示符为绿色并且设置语法高亮，下图为配置文件.gdbinit的内容      我们可以发现gdb已经配置好，是我们想要的结果。  在gdb下调试hello.c文件      break main设置断点，显示出了我们的信息，Breakpoint 1 at line 4,我们的hello.c文件中确实是在第4行，是正确的。后run一下，而后采用n单步执行（不进入函数内部），但是出现了一个问题，没有那个文件目录，这是说明我们的为调试文件debug的位置位于其他位置，我们可以执行这样一条语句set debug-file-directory就相当于把debug文件放到了我们现在的这个路径当中，就可以执行了。一直到最后，我们发现输出了正确信息  我们再尝试使用 s 命令执行 printf，s命令是单步执行（进入函数内部）    我们可以发现，我们进入了printf函数的内部，但是也出现了没有文件或目录，同上的解决方法，我们可以采用set debug-file-directory的方式，将debug文件放置我们现在目录   1. 快速排序程序调试     设置Breakpoint在main处，利用p/x a命令可以得到数组的各个元素的值，分析可知，数组最大的元素值为0x7f（127），最小值为128（-128）。    设置两个断点，一个是main()，一个是quickQsort()，r运行到main()处    运行到main()处，利用continue命令跳过当前程序一直到quickSort断点处，利用s命令进入了quickSort()函数    因为要看执行完首个递归后的数组变化，所以设置下一个断点quickSort()，先利用p/x a[0]@9输出当前数组每个元素的值，再利用c命令执行首个递归，发现已经进行了一次排序，也就是找到了中间元素，在这个元素左边的大，在这个元素右边的小。我们可以发现，中心元素是a[6]。    这是首轮递归完后的元素排列情况。    运行结束，我们可以发现最终排序的结果。   1. CMU Data Lab系列实验 2. lsbZero   int lsbZero(int x)  {  return x & ~1;  }   1. byteNot   int byteNot(int x, int n)  {  int mask = 0xFF << (n << 3);  return x ^ mask;  }   1. byteXor   int byteXor(int x, int y, int n)  {  int byte\_x = (x >> (n << 3)) & 0xFF;  int byte\_y = (y >> (n << 3)) & 0xFF;  return !!(byte\_x ^ byte\_y);  }   1. logicalAnd   int logicalAnd(int x, int y)  {  return !!x & !!y;  }   1. logicalOr   int logicalOr(int x, int y)  {  return !!x | !!y;  }   1. rotateLeft   int rotateLeft(int x, int n)  {  int mask = (~0) + (1<<n);  int r = (x>>(32+(~n)+1))&mask;  return ((x<<n)&(~mask))|r;  }   1. parityCheck   int parityCheck(int x)  {  x ^= x >> 16;  x ^= x >> 8;  x ^= x >> 4;  x ^= x >> 2;  x ^= x >> 1;  return x & 1;  }   1. mul2Ok   int mul2OK(int x)  {  int sign\_bit = (1 << 16) << 15; // 1 << 31  int overflow\_bit = (1 << 16) << 14; // 1 << 30  return ~((x & sign\_bit) | (x & overflow\_bit)) & 1;  }   1. mult3div2   int mult3div2(int x)  {  int mult3 = (x << 1) + x;  int round\_up = (mult3 >> 31) & 1;  return (mult3 + round\_up) >> 1;  }   1. subOk   int subOK(int x, int y)  {  int x\_sign = x >> 31;  int y\_sign = y >> 31;  int diff = x + ~y + 1;  int diff\_sign = diff >> 31;  return ~((x\_sign ^ y\_sign) & (x\_sign ^ diff\_sign)) & 1;  }   1. absVal   int absVal(int x)  {  int mask = x >> 31;  return (x + mask) ^ mask;  }  对x处理可以分为两种情况，取反+1，不变+0。  众所周知，一个数取反可以异或1，不变可以异或0。  当x<0时，x>>31为0xFFFFFFFF，x^(x>>31)即取反，(x>>31)&1为0x1。   1. float\_abs   unsigned float\_abs(unsigned uf)  {  if ((uf & 0x7F800000) == 0x7F800000 && (uf & 0x007FFFFF) != 0) {  return uf; // NaN  }  return uf & 0x7FFFFFFF;  }   1. float\_f2i   int float\_f2i(unsigned uf)  {  unsigned sign = uf >> 31 & 0x1;  unsigned exp = (uf & 0x7F800000) >> 23;  unsigned frac = uf & 0x7FFFFF;  int E = exp-127;  if (E < 0)  {  return 0;  }  else if (E >= 31)  {  return 0x80000000u;  }  else  {  frac = frac | (1 << 23); // add the omitted 1  if (E < 23) // exist decimal, omit decimal  {  frac >>= (23 - E);  }  else // not exist decimal  {  frac <<= (E - 23);  }  }  if (sign) // neg  {  frac = -frac;  }  return frac;  }  按照DataLab实验的要求，将bits.c文件中的各个题目用上面的代码进行填充，后经过DLC编译器进行编译，用make命令生成btest测试程序，后用btest对上面的题目答案进行测试，测试结果如下： | |

|  |
| --- |
| 总结：通过本次实验，我成功搭建了成功搭建了实验环境，并安装了GDB调试工具。通过配置GDB的初始化文件（.gdbinit），我学会了如何自定义GDB的提示符和语法高亮，提升了调试效率和体验。在GDB中调试快排程序时，掌握了设置断点、单步执行（n命令）、进入函数内部（s命令）等基本操作，掌握了如何在调试过程中观察数组元素的变化，理解了快速排序算法的执行过程。成功分析了数组在排序过程中的变化，加深了对快速排序算法的理解。我还完成了CMU Data Lab系列实验中的多个题目，涵盖了位操作、逻辑运算、浮点数转换等多个方面。通过这些实验，我深入理解了计算机底层数据表示和操作的原理。在实验过程中，我学会了如何利用位操作和逻辑运算来实现特定的功能，并通过DLC编译器和btest测试程序验证了代码的正确性。本次实验不仅提升了我的编程能力和调试技巧，还加深了我对计算机底层数据表示和操作的理解。通过实际操作和调试，我能够更好地将理论知识应用于实际问题中。 |