lab2 实验报告

——181220010-人工智能-丁豪

函数行为说明

add

```
// 有符号整型加法
int64_t asm_add(int64_t a, int64_t b) {
    return a + b;
}
```

popcnt

```
// 统计二进制表示中1的个数
int asm_popcnt(uint64_t x) {
    int s = 0;
    for (int i = 0; i < 64; i++) {
        if ((x >> i) & 1) s++;
    }
    return s;
}
```

mencpy

```
// 内存拷贝
void *asm_memcpy(void *dest, const void *src, size_t n) {
    return memcpy(dest, src, n);
}
```

set_jmp

```
// 对程序状态保存轻量级快照
int asm_setjmp(asm_jmp_buf env);
```

经过STFW以及对glibc source code的直接阅读，发现某一个版本的实现中保存的内容为——rip,rsp,rbp,rbx,r12,r13,r14,r15总共8个寄存器的值。因而我在实现其功能的时候就参考这一版本set_jmp，保存此8个寄存器的值。

long_jmp

```
// 根据set_jmp保存的快照回调，同时给set_jmp变量赋val的值
void asm_longjmp
```

阅读和上文set_jmp同一套的资料发现，set_jmp除了重新载入对应的8个寄存器的值，还把val的值存入eax（也就是之后的伪返回值），然后跳转到了set_jmp处继续往下执行。

内联汇编语法

基本格式为

```
asm ( // 1. 汇编代码 (字符串)
: // 2. 汇编代码的输出操作数
: // 3. 汇编代码的输入操作数
: // 4. 汇编代码可能改写的寄存器 (clobber)
);
```

汇编代码段多行代码之间用;隔开。当使用原生寄存器时要%%

x86-64 的C语言中，输出、输入操作数常用的修饰有

tag	类型	实例
a,b,c,d	指定通用寄存器	rax,rbx,rcx,rdx
q	a,b,c,d寄存器中的任意一个	ecx
D,S	指定通用寄存器	rdi,rsi
r	q与D,S中的任意一个	ecx
m	内存	
rm或g	r或m	

其中在作为输出操作数时，需要指定以下前缀

tag	含义
=	只写
+	读写

可能改写寄存器那一段，如果发现在汇编代码中存在push或者pop，则还要填入“memory”表示可能改变内存。但实际上这一段仅对编译器起提示作用，并不强制保留。因此在我的实践中，很多时候写与不写并没有任何差别。

以上即为本人在学习使用内联汇编时的主要知识。

实验中遇到的bug与解决

1. 自己写的memcpy测试样例，总会触发segmentation fault，一开始还以为是mencpy的实现有问题。后来发现，我在测试中直接写了两个字面字符串进行复制，并把复制结果与应该得到的结果进行memcpy比较，这样做的时候，字面字符串已经变成了野指针，于是才出发了segmentation fault。最后的解决方法是定义3个字符串string1 string2 string_result，并对他们3者进行操作。这样就不会自动收回局部变量的空间，指针仍有意义，自然不会仿存出错咯。

```
// 原测试-segmentation fault
assert(mencmp((char*)asm_memcpy("string1","string2,length),string3)==0);
// 新测试-能过
char string1[]="...";
char string2[]="...";
char string3[]="...";
char* st = (char*)asm_memcpy("string1","string2,length);
assert(memcmp(st,string3)==0);
```

- 自己写了一个set_jmp,long_jmp的测试程序（其实是复制了lab2里面用来解释其行为而举的例子），但是会触发segmentation fault。在舍友的提示下，告诉我不妨先试试本身的测试程序Main，结果一顿调试后发现能过，但仍然过不了自己的测试。
  于是我开始怀疑是编译环境不同导致的差异，于是复制了make中的编译环境，将main.c替换成我自己的测试，果然就能正常运行了，经过对冗余参数的去除已知参数的修改，发现问题被锁定在了-O优化等级上面。
  现在可以发现的现象是-O1 -O2都能正常过，-O0过不了。
  考虑到在编程途中，long_jmp曾有报错bp can not be used here，因而感觉大概率是因为更高的优化等级把使用bp的行为给优化掉了，因此能过。
  出于不希望编译器对我们的汇编代码过度优化的目的，我们在每一个asm加上一个volatile关键字。由于我们学院并未学过汇编语言，因此在此不做深究，待将来对汇编有更深入了结之后将回头解决此问题。

```
    gcc -m64 -O1 -std=gnu11 -ggdb -Wall -Werror -Wno-unused-result ./asm-impl.c
./main.c -o asm-64
```