进程的非局部跳转

局部跳转

局部跳转通常指的是程序在某一函数内部跳转至其它语句继续执行的行为，不受限于"程序自上而下执行"

C语言中实现局部跳转的关键字为goto — *goto LABEL;* / *LABEL: statement;*

```c
int main() {
    goto ZERO;
    printf("Never print");
    ZERO: printf("This is a ZERO jump");
    return 0;
}
```

当程序运行时将只会打印"This is a ZERO jump"

局部跳转就像一个潘多拉魔盒一样，如果无止境的滥用，则会给代码的阅读性以及可维护性造成相当大的难度。通常仅用于跳出较深层次的循环

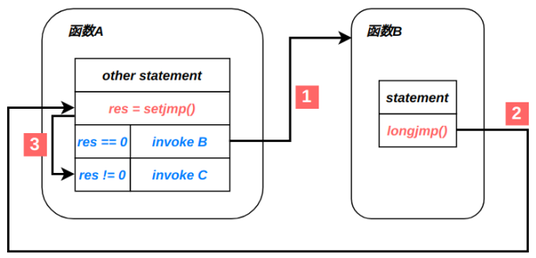
非局部跳转

非局部跳转通常指的是程序的执行语句允许跳转至当前函数之外的某个位置，偶尔使用能够带来不凡的效果 — 但切勿滥用

原型
- *int setjmp(jmp_buf env);* — 在初次调用时将返回0，后续调用则返回longjmp()调用中参数val的值
- *void longjmp(jmp_buf env, int val);* — **如果longjmp()中的val参数指定为0的话，在调用longjmp()时则会将其替换为1，用于区分setjmp()是否为初次调用**

env
- 参数env主要提供跳转的粘合剂，主要用于保存当前进程的各种信息、程序计数寄存器和栈指针寄存器等内容 — 程序计数寄存器中保存了下一条将要执行的指令，而栈指针则是函数调用的根本
- 由于setjmp()以及longjmp()通常位于不同的函数中，所以env变量也通常会设置成全局的静态变量

longjmp的关键操作
- 在调用setjmp()时，当前的进程信息(包括寄存器信息)将会被存储至env参数中，而后通常会在另一个函数中调用longjmp()
- 在调用longjmp()时，将发起longjmp()调用的函数与之前调用setjmp()的函数之间的函数栈帧从栈上剥离，就好像调用longjmp()的函数从未调用过一样
- 栈帧剥离后，重置程序计数器，使得程序能够从初始的setjmp()调用位置中继续执行



longjump()调用时将val参数传入，调用结束时将返回至原来调用setjmp()调用点，并返回val参数

```c
static jmp_buf env;

void bar() {
    longjmp(env, 25);
}

int main() {
    switch (setjmp(env)) {
        case 0:     /* 表示初次调用setjmp */
            printf("invoke setjmp first\n");
            bar();
        case 25:    /* 从longjmp调用中跳转 */
            printf("Jumped back from bar\n");
    }
    return 0;
}
```

一段简单的示例程序

setjmp()通常在在构成选择或者迭代语句中(if、switch、while)的整个表达式中使用，而诸如 s = setjmp(env) 的表达则是不符合标准的使用

信号处理函数的非局部跳转

在信号处理函数中，我们可以选择终止进程的执行(调用exit、_exit等)，或更新全局变量，亦或是进行其它的处理。除此以外，可以使用setjmp()以及longjmp()实现非局部跳转，跳转至其余函数中进行处理

但是，使用longjmp()存在一个小问题: 当进程进入到信号处理函数时， 内核会自动地将引发调用的信号以及act.sa_mask所指定的任意信号添加至进程信号掩码中，信号处理函数返回时再将其移除

longjmp()对信号掩码的处理因内核版本的不同而不同
- 在 System V 一脉中，longjmp()不会将信号掩码恢复
- 在源于 BSD 一脉的实现中，setjmp()将信号掩码保存在其 env 参数中，而信号掩码的保存值由 longjmp()恢复

宗于上述原因，要么重新约定longjmp()，要么提供一个新的系统调用，并进行统一 — POSIX选择后者，即重新定义函数

支持信号显式保存的非局部跳转

原型
- *int sigsetjmp(sigjmp_buf env, int savesigs);*
- *void siglongjmp(sigjmp_buf env, int val);*

所支持的基本功能以及调用方式未做改变，只是在原有的函数上进行拓展

savesigs
- 0值 — 表示在调用sigsetjmp()时并不会将进程信号掩码保存至env中，既然没有保存，也就不会存在恢复一说
- 非0值 — 表示在调用时将进程信号掩码保存至env中，之后siglongjmp()调用后执行恢复操作