95 lines (79 sloc) | 1.78 KB

```
1  package main
2
3  import (
4          "fmt"
5          "io/ioutil"
6          "log"
7          "net"
8          "net/http"
9          "os"
10         "strconv"
11         "strings"
12         "syscall"
13 )
14
15 func ListenFds() []*os.File {
16         // Minimal implementation of systemd's socket activation protocol
17         pid, err := strconv.Atoi(os.Getenv("LISTEN_PID"))
18         if err != nil || pid != os.Getpid() {
19                 return nil
20         }
21         nfds, err := strconv.Atoi(os.Getenv("LISTEN_FDS"))
22         if err != nil || nfds == 0 {
23                 return nil
24         }
25         files := []*os.File(nil)
26         for fd := 3; fd < 3+nfds; fd++ {
27                 syscall.CloseOnExec(fd)
28                 files = append(files, os.NewFile(uintptr(fd), ""))
29         }
```

```
        return files
}

func handler(w http.ResponseWriter, r *http.Request) {
        fn := os.Getenv("STATE_DIRECTORY") + "/counter"

        file, err := os.OpenFile(fn, os.O_RDWR|os.O_CREATE, 0644)
        if err != nil {
                log.Fatal(err)
        }
        defer file.Close()

        // Let's take a BSD lock to synchronize access to the counter file
        err = syscall.Flock(int(file.Fd()), syscall.LOCK_EX)
        if err != nil {
                log.Fatal(err)
        }

        contents, err := ioutil.ReadAll(file)
        if err != nil {
                log.Fatal(err)
        }

        counter := 0

        trimmed := strings.TrimSpace(string(contents))
        if trimmed != "" {
                counter, err = strconv.Atoi(trimmed)
                if err != nil {
                        log.Fatal(err)
                }
        }

        counter++

        fmt.Fprintf(w, "Hello! You are visitor #%d!\n", counter)

        file.Truncate(0)
        file.Seek(0, 0)
        fmt.Fprintf(file, "%d\n", counter)
}

func run_http(f *os.File) {
        s, err := net.FileListener(f)
        if err != nil {
                log.Fatal(err)
        }

        log.Fatal(http.Serve(s, nil))
```

```
}

func main() {
        http.HandleFunc("/", handler)

        listeners := ListenFds()
        if len(listeners) == 0 {
                log.Fatal(http.ListenAndServe(":8080", nil))
        } else {
                // If multiple sockets are passed, spawn all but the last one as go-routin
                for _, l := range listeners[:len(listeners)-1] {
                        go run_http(l)
                }

                run_http(listeners[len(listeners)-1])
        }
}
```