# 標準入力が元に戻らない

# Grok 回答

## Issue Analysis

The provided code implements an interactive proxy for an external process's stdin/stdout, which works during the external process's lifetime. However, the FIXME highlights a problem: after `EchoProxy` returns (i.e., after the external process ends and control returns to the calling code), subsequent reads from `os.Stdin` in the parent program may fail or behave unexpectedly. This is because:

- The `receiveStdin` goroutine runs indefinitely in a loop, continuously consuming input from `os.Stdin` via `bufio.Scanner`.
- When the external process exits, `externalProcess.Wait()` unblocks, but the goroutine doesn't stop—it keeps scanning and attempting to write to the now-closed `exStdin` pipe.
- Writes to a closed pipe return an error (e.g., "broken pipe"), but the code doesn't check for this error, so the goroutine silently continues looping and "steals" input from `os.Stdin`.
- When the parent program later tries to read from `os.Stdin` (e.g., via `fmt.Scanln` or another scanner), the input is consumed by the lingering goroutine, causing hangs or empty reads.

Other minor issues:

- `print(...)` should be `fmt.Print(...)` for standard output (Go's builtin `print` is for simple debugging and doesn't flush reliably).
- Reading 1 byte at a time in `receiveStdout` is inefficient and can fragment UTF-8 characters, but it's functional for ASCII/echo-like use cases. A larger buffer (e.g., 1024 bytes) would be better.
- The `parameters` slice is empty, which matches the comment (no params), but `strings.Split("", " ")` is a no-op—could simplify to `[]string{}`.
- Deferring `Close()` on pipes is fine, but explicit closes after `Wait()` ensure cleanup.
- No error handling on `scanner.Err()` in `receiveStdin`, which could lead to panics on I/O errors.

## Fixed Code

Here's the corrected version. Key changes:

- In `receiveStdin`: Check `Write` errors and exit the goroutine if the pipe is broken/closed (this stops input consumption post-process).
- In `receiveStdout`: Use a larger buffer (1024 bytes) for efficiency and handle UTF-8 better.
- Replace `print` with `fmt.Print` and add `fmt.Fprint(os.Stderr, ...)` for status messages to avoid interfering with proxied output.
- Add `scanner.Err()` check in `receiveStdin`.
- Explicitly close pipes after `Wait()` (defers still handle cleanup).
- Minor cleanups for clarity.

```
package exercise
```

```
import (
    "bufio"
    "fmt"
    "io"
    "os"
    "os/exec"
    "strings"
)

// EchoProxy - 外部プロセスの標準入出力をプロキシする練習
func EchoProxy(externalProcessPath string) {
    parameters := []string{} // No parameters, as per comment

    externalProcess := exec.Command(externalProcessPath, parameters...)
    // ワーキング・ディレクトリーは特に指定なし

    exStdin, err := externalProcess.StdinPipe()
    if err != nil {
        panic(err)
    }
    defer exStdin.Close()

    exStdout, err := externalProcess.StdoutPipe()
    if err != nil {
        panic(err)
    }
    defer exStdout.Close()

    err = externalProcess.Start()
    if err != nil {
        panic(fmt.Errorf("cmd.Start() --> [%s]", err))
    }

    go receiveStdout(exStdout)

    go receiveStdin(exStdin)

    fmt.Fprint(os.Stderr, "外部プロセスと接続しました。文字を入力してください。\n")
    externalProcess.Wait()

    // Explicitly close pipes to trigger errors in goroutines
    exStdin.Close()
    exStdout.Close()

    fmt.Fprint(os.Stderr, "外部プロセスが終了しました。\n")

    // Now, goroutines will exit cleanly, and os.Stdin is free for the parent
program
}

// receiveStdin - 標準入力受信
// `epStdin` - External process stdin
func receiveStdin(epStdin io.WriteCloser) {
    defer epStdin.Close() // Ensure close if goroutine exits early
```

```
    scanner := bufio.NewScanner(os.Stdin)
    for scanner.Scan() {
        command := scanner.Text()
        _, err := epStdin.Write([]byte(command + "\n"))
        if err != nil {
            // Pipe closed/broken: external process ended, stop consuming input
            return
        }
    }
    if err := scanner.Err(); err != nil {
        // Handle scanner errors (e.g., I/O issues)
        fmt.Fprintf(os.Stderr, "Scanner error: %v\n", err)
    }
}

// receiveStdout - 標準出力受信
// `epStdout` - External process stdout
func receiveStdout(epStdout io.ReadCloser) {
    defer epStdout.Close()

    const bufferSize = 1024
    buffer := make([]byte, bufferSize)

    for {
        n, err := epStdout.Read(buffer)
        if err != nil {
            if err == io.EOF {
                return
            }
            panic(err)
        }
        if n > 0 {
            fmt.Print(string(buffer[:n]))
        }
    }
}
```

## How to Test

- **Main function example** (add this to test):

```
func main() {
    // Test with a simple external process like "cmd" (Windows) or "bash"
(Unix)
    EchoProxy("cmd") // Or "bash", etc.

    // After proxy ends, test input here—it should now work!
    fmt.Print("Back in main. Enter something: ")
    var input string
    fmt.Scanln(&input)
```

```
    fmt.Printf("You entered: %s\n", input)
}
```

- Run: `go run main.go`. Interact with the external process (e.g., type commands), let it exit (Ctrl+C or natural end), then confirm the main prompt accepts input.
- **UTF-8 note**: As commented, use a UTF-8 terminal (e.g., PowerShell with `chcp 65001`, or non-VSCode). The larger buffer in `receiveStdout` helps with multi-byte chars.

This fix ensures the goroutines terminate promptly after the process ends, restoring `os.Stdin` for the caller without side effects. If this is part of a larger exercise, let me know for further tweaks!