

改行を 1 回入れないと標準入力に戻らない

コードがいくつか直りました。ありがとうございます。外部プロセスを終了して元のプロセスに戻った時、改行を 1 つ入れないと元に戻っていないようです。何か分かりますか？

Grok 回答

問題の分析

おっしゃる通り、修正後のコードでも、外部プロセス終了後（`externalProcess.Wait()` が戻った後）に元のプロセス（例: `main()` 内の `fmt.Scanln()`）で入力待ちをすると、**1回の改行（Enter）を押さないと正常に応答しない** という症状が出る場合があります。これは主に以下の原因です：

- `receiveStdin` ゴルーチン内の `bufio.Scanner` が、外部プロセス終了直前に最後の入力行を処理した後、**内部バッファに残った状態（例: 部分的な行や未処理の改行）** を保持したままゴルーチンが終了する。
- Goの `bufio.Scanner` は `Scan()` が成功した後、次の `Scan()` を待つ状態になることがあり、ゴルーチンが `return` しても `os.Stdin` のグローバル状態に微妙な影響が残る（特にターミナル入力の場合、TTYのエコーやバッファリングが絡む）。
- 結果、親プロセスの `fmt.Scanln()` などが最初の入力を「空の行」として消費し、2回目の入力（改行）でようやく正常化する。

これはGoのstdinパイプラインの微妙な副作用で、よくある問題です。根本解決には、ゴルーチンを**明確なシグナル（例: doneチャネル）で停止** させ、`Scanner` を明示的にリセット/ドレインする工夫が必要です。以下に、それを追加した修正版を示します。

修正のポイント

- **doneチャネル** を導入：`EchoProxy` でプロセス終了後にシグナルを送り、`receiveStdin` を確実に停止。`Scanner` のループを抜けた後、残った入力をドレイン（無視）する。
- `receiveStdout` も同様にdoneで停止（念のため）。
- `bufio.Scanner` のエラー処理を強化し、終了時にバッファをクリア。
- テスト用`main()`で確認しやすくするため、終了メッセージ後に即座に入力テストを追加。
- UTF-8ターミナル（PowerShellなど）の使用を推奨（コメント通り）。

これで、終了直後に改行なしで入力が効くはずです。もし再現したら、OS（Windows?）やGoバージョン（`go version`）を教えてください。

修正コード

```
package exercise

import (
    "bufio"
    "fmt"
    "io"
```

```
    "os"
    "os/exec"
)

// 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...)

    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))
    }

    // doneチャンネルでゴルーチンを制御
    done := make(chan struct{})

    go receiveStdout(exStdout, done)

    go receiveStdin(exStdin, done)

    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()

    // プロセス終了をシグナルしてゴルーチンを停止
    close(done)

    // 明示的にパイプを閉じてエラーをトリガー
    exStdin.Close()
    exStdout.Close()

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

    // これでos.Stdinの状態がクリーンになり、親プロセスで即入力可能
}

// receiveStdin - 標準入力受信
// `epStdin` - External process stdin
// `done` - 終了シグナルチャンネル
func receiveStdin(epStdin io.WriteCloser, done <-chan struct{}) {
    defer epStdin.Close()
}
```

```

scanner := bufio.NewScanner(os.Stdin)
for {
    select {
    case <-done:
        // プロセス終了シグナル受信: 残り入力をドレインして終了
        for scanner.Scan() {
            // 無視 (バッファクリア)
        }
        return
    default:
        if !scanner.Scan() {
            break
        }
        command := scanner.Text()
        _, err := epStdin.Write([]byte(command + "\n"))
        if err != nil {
            // パイプ閉鎖エラー: 早期終了
            return
        }
    }
}

if err := scanner.Err(); err != nil {
    fmt.Fprintf(os.Stderr, "Scanner error: %v\n", err)
}

}

// receiveStdout - 標準出力受信
// `epStdout` - External process stdout
// `done` - 終了シグナルチャネル
func receiveStdout(epStdout io.ReadCloser, done <-chan struct{}) {
    defer epStdout.Close()

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

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

```

テスト方法

以下の `main()` を追加して実行してください (`go run main.go`)。外部プロセス (例: `cmd`) を終了 (Ctrl+C) させた後、**改行なしで** "Back in main" のプロンプトに入力できるはずです。

```
// main.go の例 (テスト用)
func main() {
    EchoProxy("cmd") // Windowsの場合・Unixなら "bash" など

    // 終了後、即入力テスト
    fmt.Print("Back in main. Enter something: ")
    var input string
    fmt.Scanln(&input)
    fmt.Printf("You entered: %s\n", input)
}
```

- **期待動作:** 外部プロセス終了 → "Back in main..." 表示 → 即座に文字入力 → "You entered: ..." 表示。
- もしまだ改行が必要なら、ターミナルの種類 (VSCode統合ターミナルは問題やすい) や、Goの `runtime.GC()` を追加で呼ぶなどのデバッグを試せます。追加情報があれば、さらに調整します！