Go Signals

Go by Example https://gobyexample.com/signals

Go signal notification works by sending `os.Signal` values on a channel. We'll create a channel to receive these notifications. Note that this channel should be buffered.

```
package main

import (
    "fmt"
    "os"
    "os/signal"
    "syscall"
)

func main() {

    sigs := make(chan os.Signal, 1)
```

`signal.Notify` registers the given channel to receive notifications of the specified signals.

```
    signal.Notify(sigs, syscall.SIGINT, syscall.SIGTERM)
```

We could receive from `sigs` here in the main function, but let's see how this could also be done in a separate goroutine, to demonstrate a more realistic scenario of graceful shutdown.

```
    done := make(chan bool, 1)
```

This goroutine executes a blocking receive for signals. When it gets one it'll print it out and then notify the program that it can finish.

```
    go func() {

        sig := <-sigs
        fmt.Println()
        fmt.Println(sig)
        done <- true
    }()
```

The program will wait here until it gets the expected signal (as indicated by the goroutine above sending a value on `done`) and then exit.

```
    fmt.Println("awaiting signal")
    <-done
    fmt.Println("exiting")
}
```

When we run this program it will block waiting for a signal. By typing `ctrl-C` (which the terminal shows as `^C`) we can send a `SIGINT` signal, causing the program to print `interrupt` and then exit.

```
$ go run signals.go
awaiting signal
^C
interrupt
exiting
```

```
package main

import (
    "fmt"
    "os"
    "os/signal"
    "syscall"
)

func main() {

    sigs := make(chan os.Signal, 1)

    signal.Notify(sigs, syscall.SIGINT, syscall.SIGTERM)

    done := make(chan bool, 1)

    go func() {

        sig := <-sigs
        fmt.Println()
        fmt.Println(sig)
        done <- true
    }()

    fmt.Println("awaiting signal")
    <-done
    fmt.Println("exiting")
}
```

Another Example:

```
package main

import (
    "fmt"
    "os"
    "os/signal"
    "syscall"
    "time"
)

func main() {

    osSignalChannel := make(chan os.Signal, 1)

    signal.Notify(osSignalChannel, syscall.SIGINT, syscall.SIGTERM)

    msgChannel := make(chan string)

    go func() {

        for {
            //wait on a message from a channel
            select {
            case signal := <-osSignalChannel:
                fmt.Println(signal)

                //call shutdown
                msgChannel <- "shutdown"

            case msg := <-msgChannel:
                fmt.Println("s: Received message:", msg)
                if msg == "shutdown" {
                    fmt.Println("s: Cleaning up")

                    //sleep
                    time.Sleep(time.Second * 30)

                    fmt.Println()
                    fmt.Println("s: Shutting down")

                    //notify shutdown
                    msgChannel <- "shutdown"

                    return

                } else {
                    fmt.Println("s: Received message:", msg, " ignoring...")
                }

            }
        }

    }()

    time.Sleep(time.Second * 1)

    //sens bogus message
    fmt.Println("m: Sending bogus message")
    msgChannel <- "bogus"

    time.Sleep(time.Second * 5)

    //send shutdown
    fmt.Println("m: Sending shutdown message")
    msgChannel <- "shutdown"

    fmt.Println("m: Waiting for response message")
    response := <-msgChannel
    fmt.Println("m: Response: ", response)
    fmt.Println("m: Exiting")
```

```
}
```

Another Example

```
package main
import (
        "fmt"
        "os"
        "os/signal"
        "syscall"
        "time"
)

func main() {

        //create channel that can receive OS signals
        ch := make(chan os.Signal)

    //Submit channel to the notify signal
        signal.Notify(ch, syscall.SIGTERM, syscall.SIGINT)


        //a go function to monitor signals in the background
        go func() {
                for sig := range ch {
                        switch sig {
                        case syscall.SIGTERM:
                                fmt.Println("sigterm received')
                                os.Exit(0)
                        case syscall.SIGINT:
                                fmt.Println("sigint received')
                                oS.Exit(0)
                        }
                }
        }()

        time.Sleep(time.Minute)
}
```