Gophercon, 28 August 2025

An Operating System in GO

Patricio Whittingslow

soypat@github

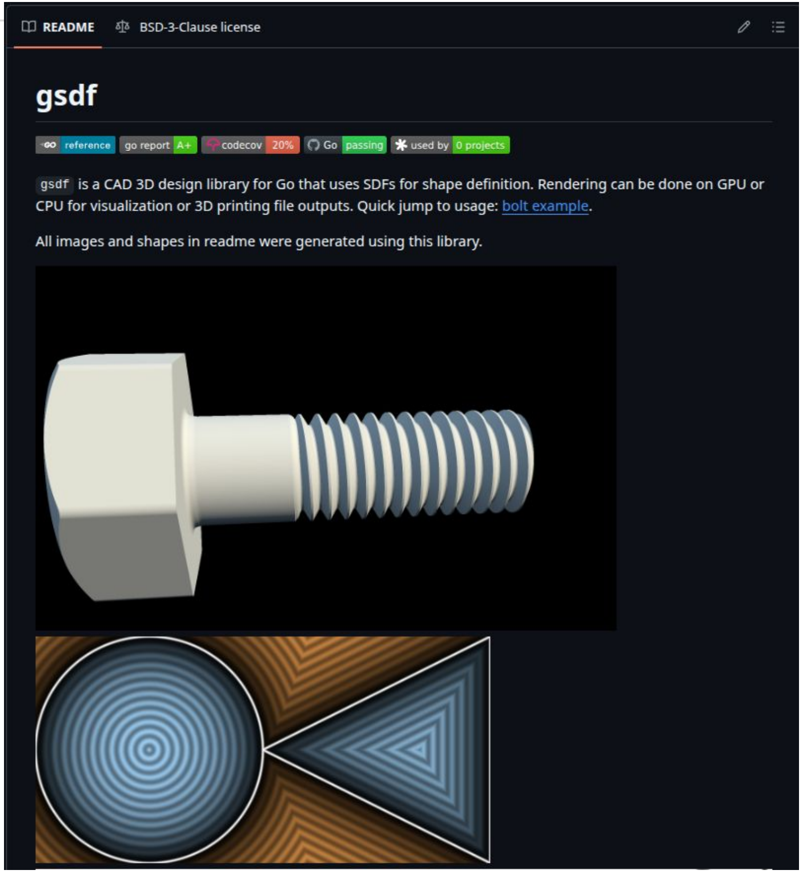
Disclaimer

All views and opinions are my own and do not reflect my employer's point of view.

à contre-courant

- GC22: Rocket engine test bench
- GC23: Great language to learn
- GC24: Embedded systems, replace C
- GCAU24: Automatic drone control
- GCAU24: 3D CAD Part design
- GC25: Operating systems

soypat / gsdf



gsdf

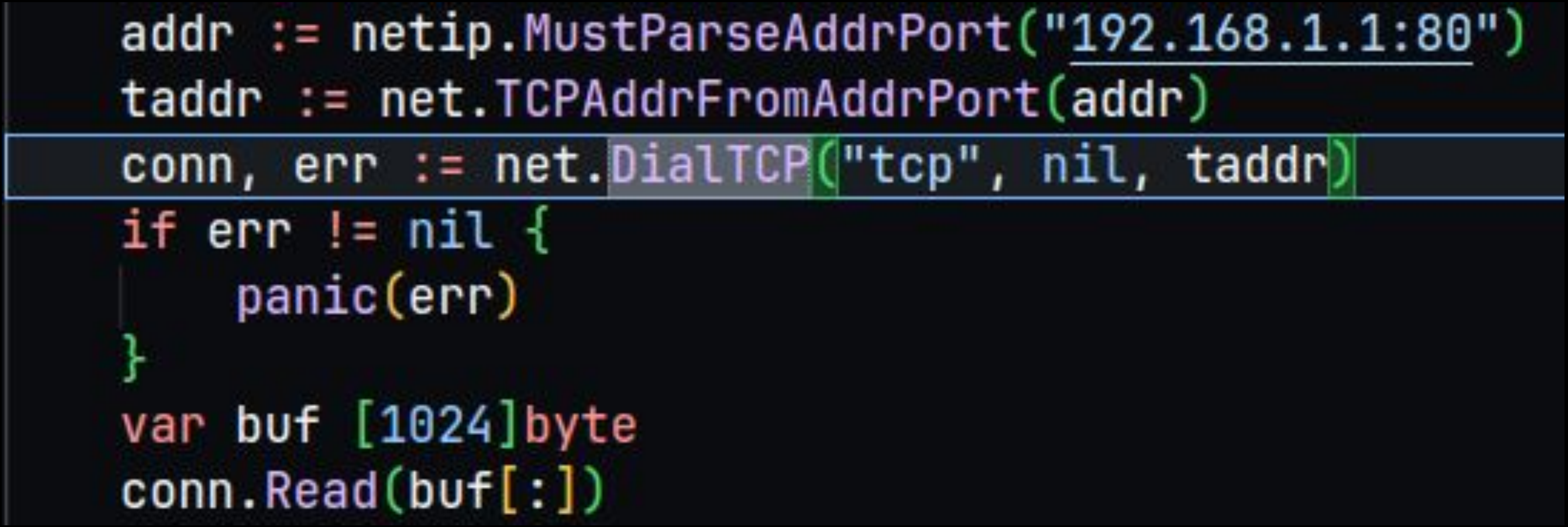
gsdf is a CAD 3D design library for Go that uses SDFs for shape definition. Rendering can be done on GPU or CPU for visualization or 3D printing file outputs. Quick jump to usage: bolt example.

All images and shapes in readme were generated using this library.

Go and an operating system

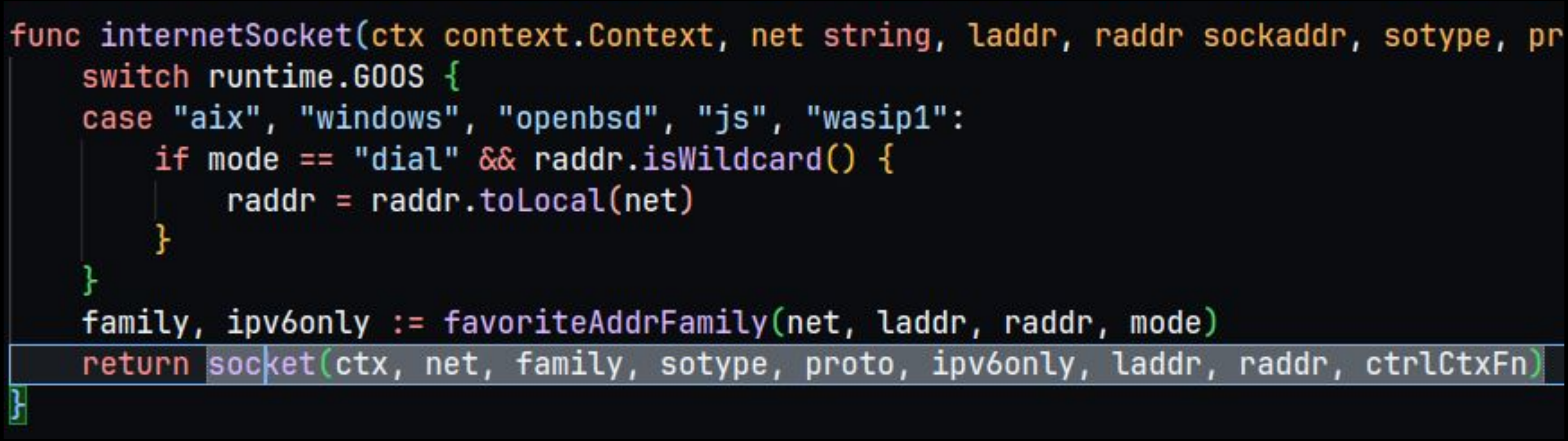
```
GOOS=linux
GOOS=windows
GOOS=darwin
```

GOOS=linux



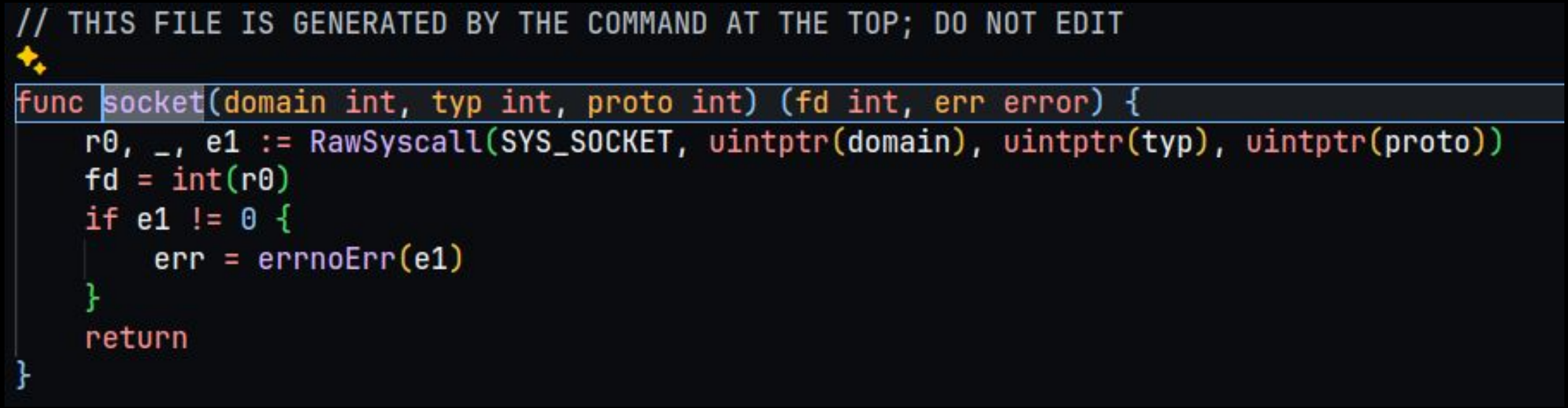
```
addr := netip.MustParseAddrPort("192.168.1.1:80")
taddr := net.TCPAddrFromAddrPort(addr)
conn, err := net.DialTCP("tcp", nil, taddr)
if err != nil {
    panic(err)
}
var buf [1024]byte
conn.Read(buf[:])
```

GOOS=linux



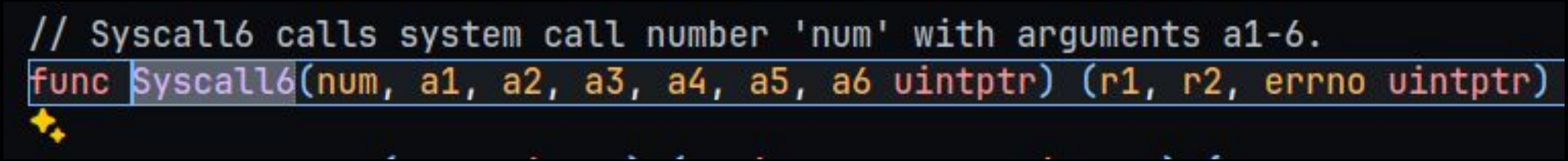
```
func internetSocket(ctx context.Context, net string, laddr, raddr sockaddr, sotype, pr
    switch runtime.GOOS {
    case "aix", "windows", "openbsd", "js", "wasip1":
        if mode == "dial" && raddr.isWildcard() {
            raddr = raddr.toLocal(net)
        }
    }
    family, ipv6only := favoriteAddrFamily(net, laddr, raddr, mode)
    return socket(ctx, net, family, sotype, proto, ipv6only, laddr, raddr, ctrlCtxFn)
}
```

GOOS=linux



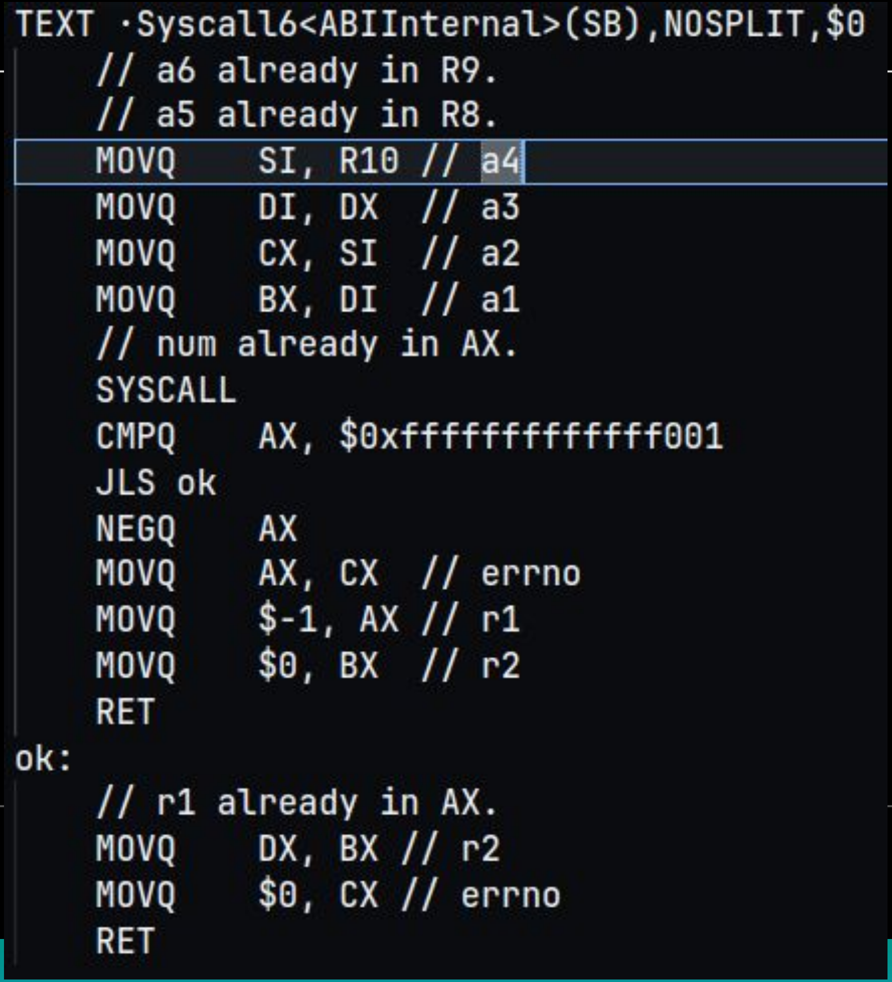
```go
// THIS FILE IS GENERATED BY THE COMMAND AT THE TOP; DO NOT EDIT

func socket(domain int, typ int, proto int) (fd int, err error) {
    r0, _, e1 := RawSyscall(SYS_SOCKET, uintptr(domain), uintptr(typ), uintptr(proto))
    fd = int(r0)
    if e1 != 0 {
        err = errnoErr(e1)
    }
    return
}
```

GOOS=linux



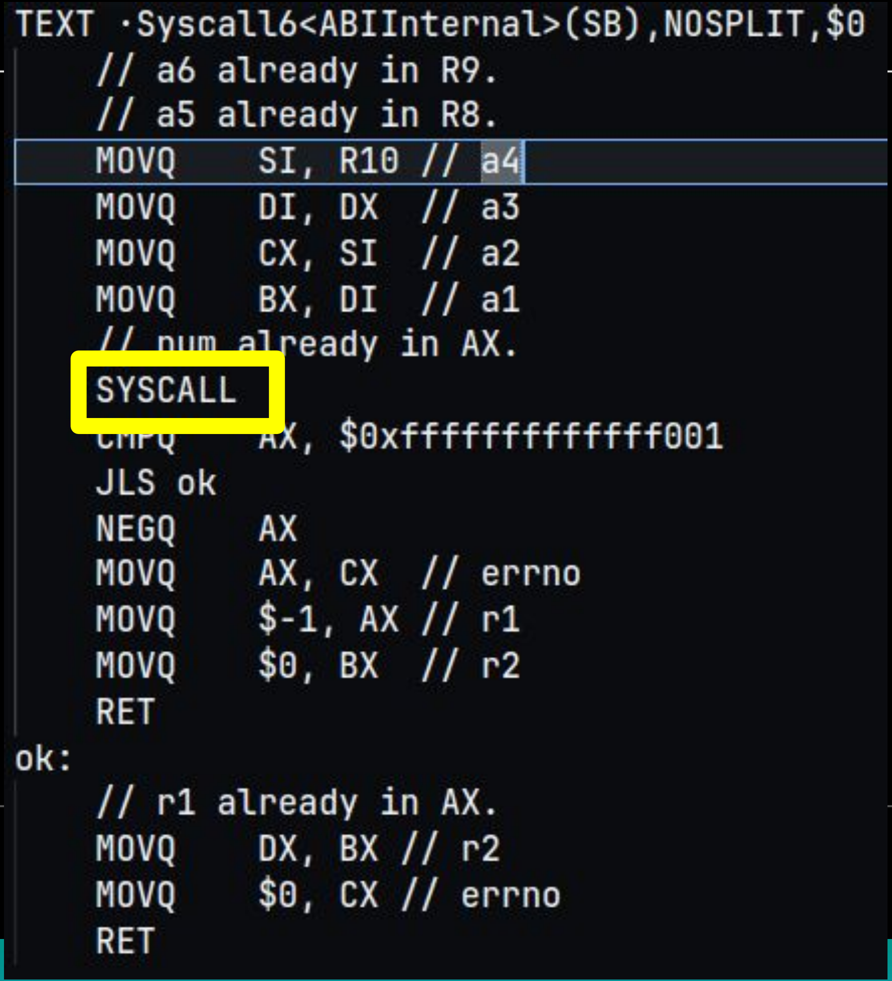
```go
// Syscall6 calls system call number 'num' with arguments a1-6.
func Syscall6(num, a1, a2, a3, a4, a5, a6 uintptr) (r1, r2, errno uintptr)
```

GOOS=linux



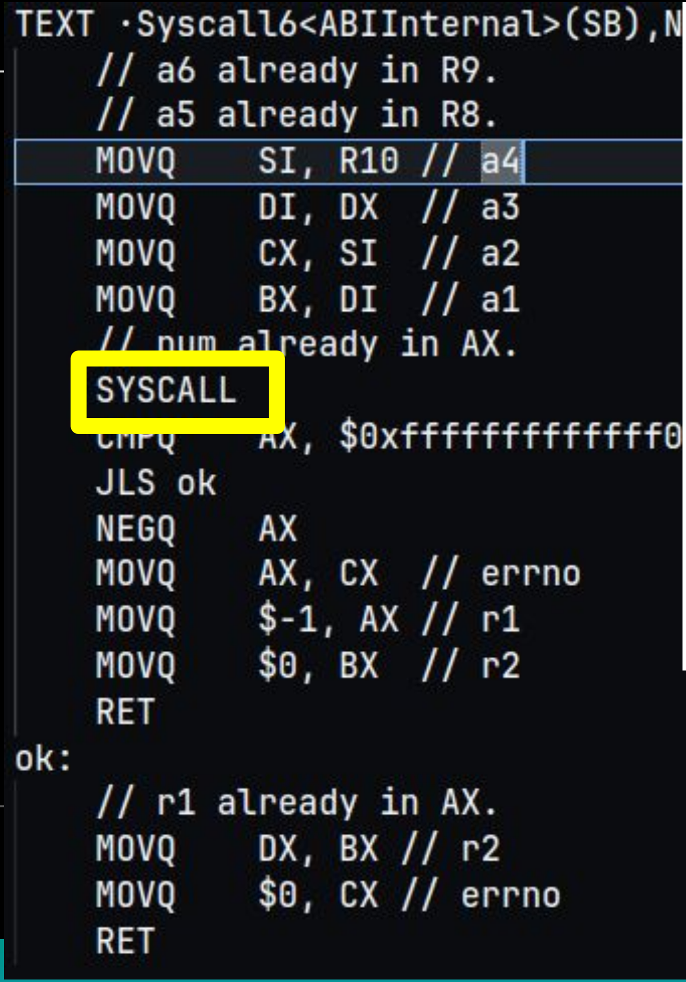
```
TEXT ·Syscall6<ABIInternal>(SB),NOSPLIT,$0
    // a6 already in R9.
    // a5 already in R8.
    MOVQ    SI, R10 // a4
    MOVQ    DI, DX  // a3
    MOVQ    CX, SI  // a2
    MOVQ    BX, DI  // a1
    // num already in AX.
    SYSCALL
    CMPQ    AX, $0xfffffffffffff001
    JLS ok
    NEGQ    AX
    MOVQ    AX, CX  // errno
    MOVQ    $-1, AX // r1
    MOVQ    $0, BX  // r2
    RET
ok:
    // r1 already in AX.
    MOVQ    DX, BX // r2
    MOVQ    $0, CX // errno
    RET
```

GOOS=linux



```
TEXT ·Syscall6<ABIInternal>(SB),NOSPLIT,$0
	// a6 already in R9.
	// a5 already in R8.
	MOVQ	SI, R10 // a4
	MOVQ	DI, DX  // a3
	MOVQ	CX, SI  // a2
	MOVQ	BX, DI  // a1
	// num already in AX.
	SYSCALL
	CMPQ	AX, $0xfffffffffffff001
	JLS	ok
	NEGQ	AX
	MOVQ	AX, CX  // errno
	MOVQ	$-1, AX // r1
	MOVQ	$0, BX  // r2
	RET
ok:
	// r1 already in AX.
	MOVQ	DX, BX // r2
	MOVQ	$0, CX // errno
	RET
```

GOOS=linux



```
TEXT ·Syscall6<ABIInternal>(SB),N
    // a6 already in R9.
    // a5 already in R8.
    MOVQ    SI, R10 // a4
    MOVQ    DI, DX  // a3
    MOVQ    CX, SI  // a2
    MOVQ    BX, DI  // a1
    // num already in AX.
    SYSCALL
    CMPQ    AX, $0xfffffffffffff0
    JLS ok
    NEGQ    AX
    MOVQ    AX, CX  // errno
    MOVQ    $-1, AX // r1
    MOVQ    $0, BX  // r2
    RET
ok:
    // r1 already in AX.
    MOVQ    DX, BX // r2
    MOVQ    $0, CX // errno
    RET
```

GOOS=linux

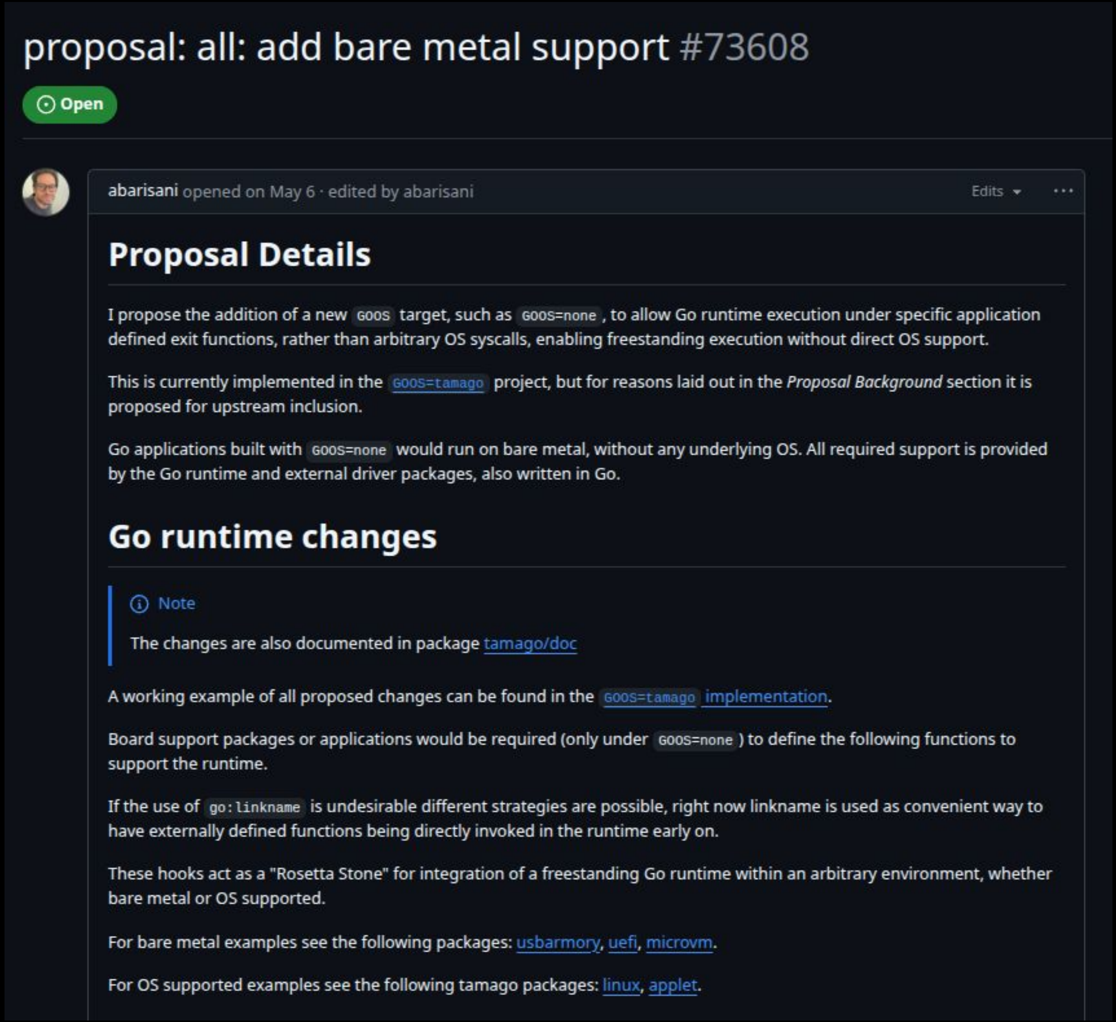
GOOS=aix
GOOS=android
GOOS=darwin
GOOS=dragonfly
GOOS=freebsd
GOOS=illumis
GOOS=ios

GOOS=linux
GOOS=netbsd
GOOS=openbsd
GOOS=plan9
GOOS=solaris
GOOS=windows
GOOS=js

The Operating System

- System startup
- Task/thread scheduling
- Interrupt handling
- Timing services
- System tick, delays, timers
- Inter Process Communication: So queues, semaphores, mutexes
- Memory management: static regions, heap allocation
- Hardware I/O: GPIO, UART/SPI/I^{2}C, PWM, ADC/DAC, CAN, USB, etc
- DMA coordination
- Power management: sleep and deep sleep modes, wakeup sources
- File systems: LittleFS/FAT
- Networking stacks: Ethernet/Wifi/BLE, TCP/IP, DHCP, NTP, DNS
- USB device/host classes: CDC, HID
- Shell/CLI and logging
- Watchdogs: kick/monitor watchdogs

GOOS=noos



proposal: all: add bare metal support #73608

abarisani opened on May 6 · edited by abarisani

Proposal Details

I propose the addition of a new `GOOS` target, such as `GOOS=none`, to allow Go runtime execution under specific application defined exit functions, rather than arbitrary OS syscalls, enabling freestanding execution without direct OS support.

This is currently implemented in the `GOOS=tamago` project, but for reasons laid out in the *Proposal Background* section it is proposed for upstream inclusion.

Go applications built with `GOOS=none` would run on bare metal, without any underlying OS. All required support is provided by the Go runtime and external driver packages, also written in Go.

Go runtime changes

> ⓘ Note
>
> The changes are also documented in package tamago/doc

A working example of all proposed changes can be found in the `GOOS=tamago` implementation.

Board support packages or applications would be required (only under `GOOS=none`) to define the following functions to support the runtime.

If the use of `go:linkname` is undesirable different strategies are possible, right now linkname is used as convenient way to have externally defined functions being directly invoked in the runtime early on.

These hooks act as a "Rosetta Stone" for integration of a freestanding Go runtime within an arbitrary environment, whether bare metal or OS supported.

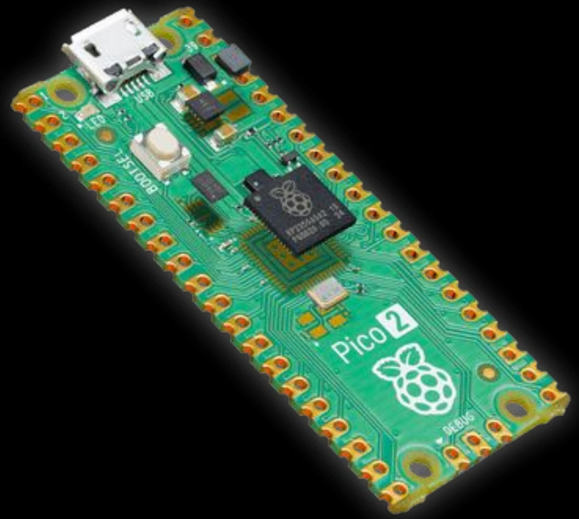
For bare metal examples see the following packages: usbarmory, uefi, microvm.

For OS supported examples see the following tamago packages: linux, applet.

proposal: all: add bare metal support #73608

GOOS=noos

```
var (
   Bloc           uintptr          // Heap start addr.
   Exit           func(int32)      // Runtime termination.
   Idle           func(until int64) // Runtime idle CPU until timestamp.
   ProcID         func() int64     // Processor hardware identifier.
   RamSize        uint             // Size available to runtime for allocation.
   RamStackOffset uint             // Negative offset from end of available memory for stack.
   RamStart       uint             // Start addr of allocation memory.
   SocketFunc     func(ctx context.Context, net string, family, sotype int, laddr, raddr netip.Addr)
(interface{}, error)
   Task           func(sp, mp, gp, fn unsafe.Pointer)
)

func GetRandomData(b []byte)
func InitRNG()
func HwInit0()
func HwInit1()
func Nanotime() int64
func Printk(char byte)
```



Who

Embedded Gophers

Who? - TamaGo

TamaGo enables compilation and execution of Go applications running on baremetal AMD64/ARM/RISC-V processors. Objective: Reduce attack surface of embedded systems by **removing** C dependencies and OS.

https://github.com/usbarmory/tamago

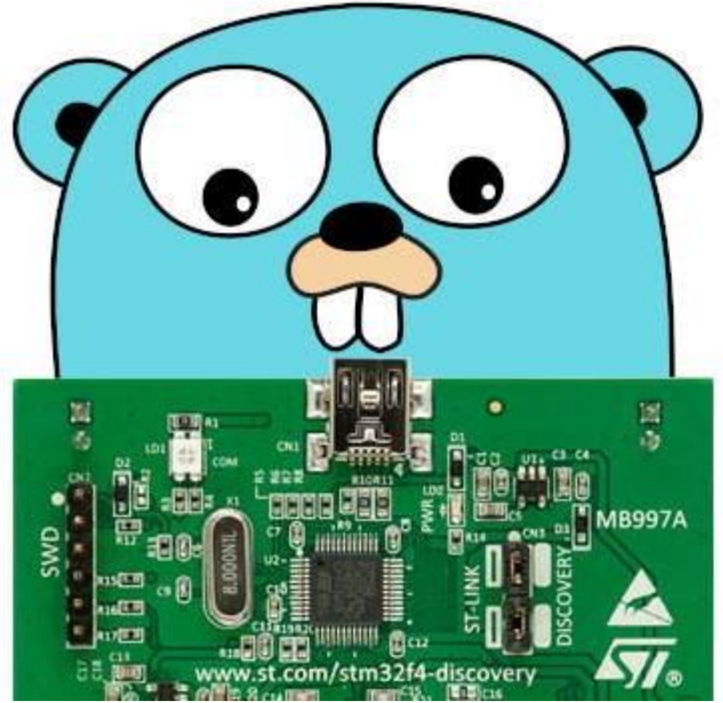
Who? - TinyGo

TinyGo is a Go compiler with modified internals to reduce the footprint of Go programs so that they fit on MCUs.

https://tinygo.org/

Who? - Embedded Go

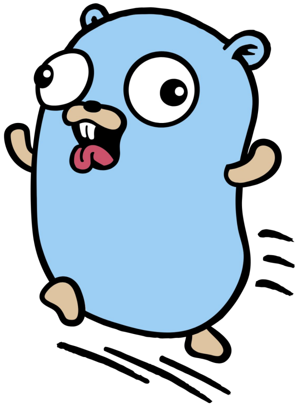
Embedded Go targets the ARMv7M/Thumb2 architecture. Similar in spirit to TamaGo. Eliminate C and OS.



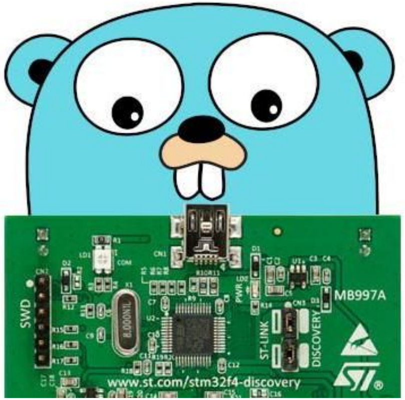
https://embeddedgo.github.io/

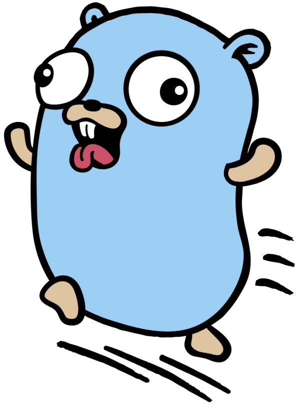
Who? - Honourable mentions

- eggos - github.com/icexin/eggos
- gokrazy - gokrazy.org/



Who? - Honourable mentions

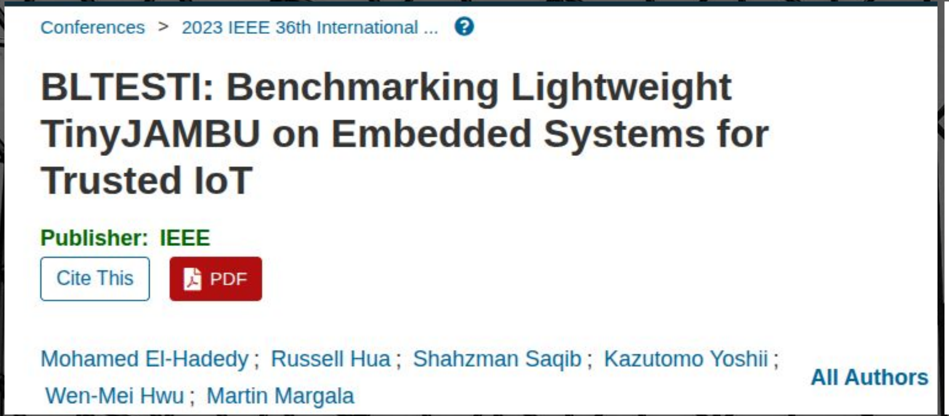




Operating Systems - Disadvantages

- Security: Reduced threat surface
- Size: Can fit more program on small devices.
- Resource control: You do you.
- Deterministic timing: No jitter (TinyGo)
- C: No Cgo required.
- No FFI: Performance benefits
- No C compiler dependency
- No C: Safety
- Portability: develop with interfaces, not syscalls.
- Power efficiency: Less CPU work, deep sleep, timings
- Reproducible deployment: No environment, no kernel version

02 Real Life Academic Papers on GOOS=noos



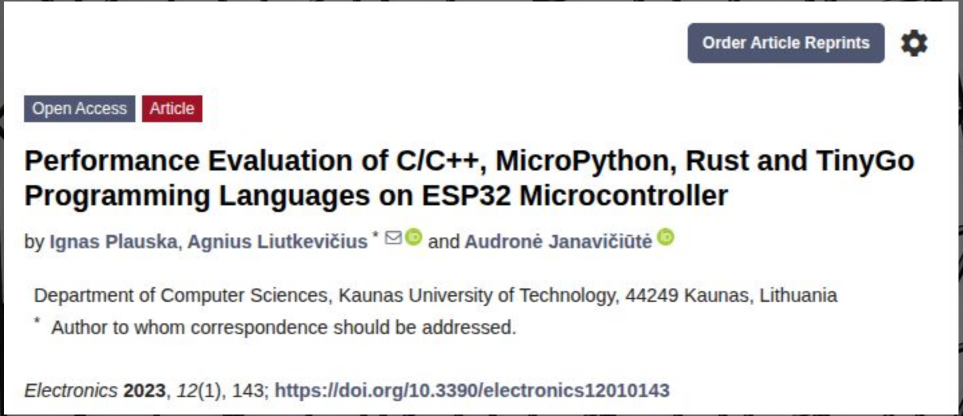
Conferences > 2023 IEEE 36th International ... ?

BLTESTI: Benchmarking Lightweight TinyJAMBU on Embedded Systems for Trusted IoT

Publisher: IEEE

Mohamed El-Hadedy ; Russell Hua ; Shahzman Saqib ; Kazutomo Yoshii ; Wen-Mei Hwu ; Martin Margala

All Authors



Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller

by **Ignas Plauska**, **Agnius Liutkevičius** * and **Audronė Janavičiūtė**

Department of Computer Sciences, Kaunas University of Technology, 44249 Kaunas, Lithuania

* Author to whom correspondence should be addressed.

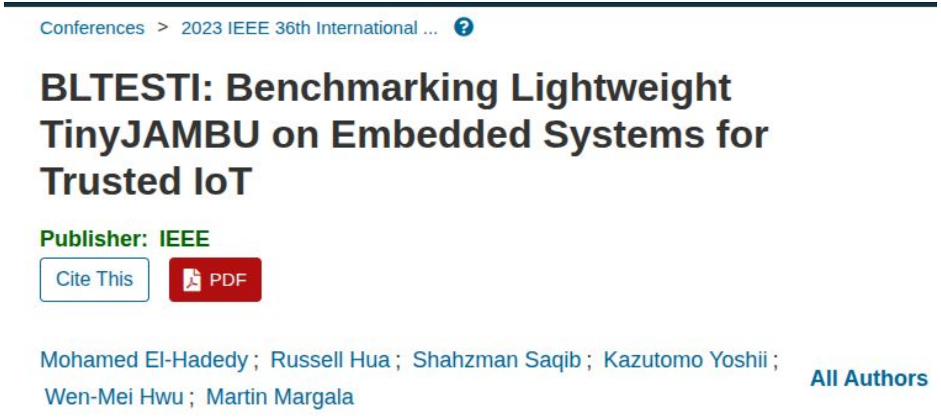
Electronics **2023**, *12*(1), 143; **https://doi.org/10.3390/electronics12010143**

https://ieeexplore.ieee.org/abstract/document/10256731

https://www.mdpi.com/2079-9292/12/1/143

TinyJambu Paper

- Benchmarks performance of Rust/C/TinyGo on RP2040
  - MCU is RP2040 (Raspberry Pi Pico)
  - Algorithm is TinyJambu, a NIST finalist
- Result: Binary Sizes: TinyGo 19kB, Rust 66kB, C 70kB,
- Result: TinyGo binary size does not scale with message size
- Result: TinyGo was 1.5 to 2 times slower than C
- Note: Code not provided



Conferences > 2023 IEEE 36th International ...

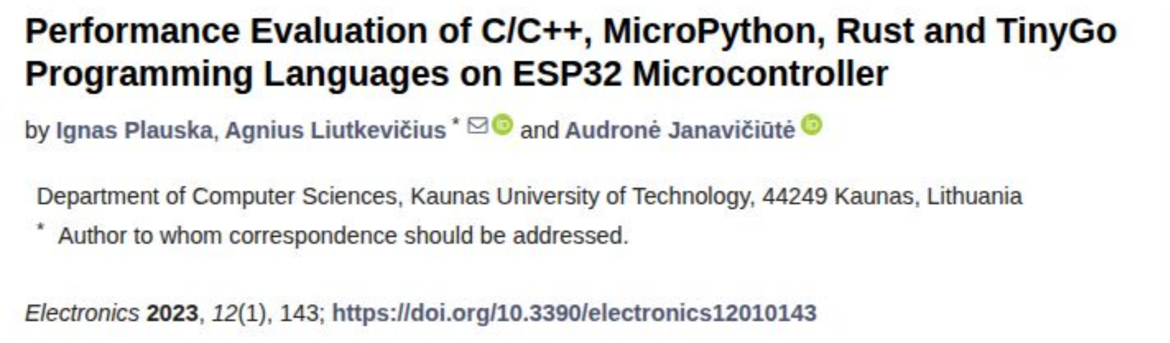
BLTESTI: Benchmarking Lightweight TinyJAMBU on Embedded Systems for Trusted IoT

Publisher: IEEE

Mohamed El-Hadedy ; Russell Hua ; Shahzman Saqib ; Kazutomo Yoshii ; Wen-Mei Hwu ; Martin Margala

Performance Evaluation Paper

- Algorithms benchmarked: FFT, CRC, SHA, IIR and FIR Filters
- Languages: C/MicroPython/Rust/TinyGo
  - All except TinyGo running on FreeRTOS
- Result: TinyGo programs run as fast as C programs
- Result: TinyGo programs show no jitter and run in constant time
- Conclusion: TinyGo shows no jitter due to not running an OS



Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller

by **Ignas Plauska**, **Agnius Liutkevičius** * and **Audronė Janavičiūtė**

Department of Computer Sciences, Kaunas University of Technology, 44249 Kaunas, Lithuania

* Author to whom correspondence should be addressed.

Electronics **2023**, *12*(1), 143; **https://doi.org/10.3390/electronics12010143**

Operating System tasks

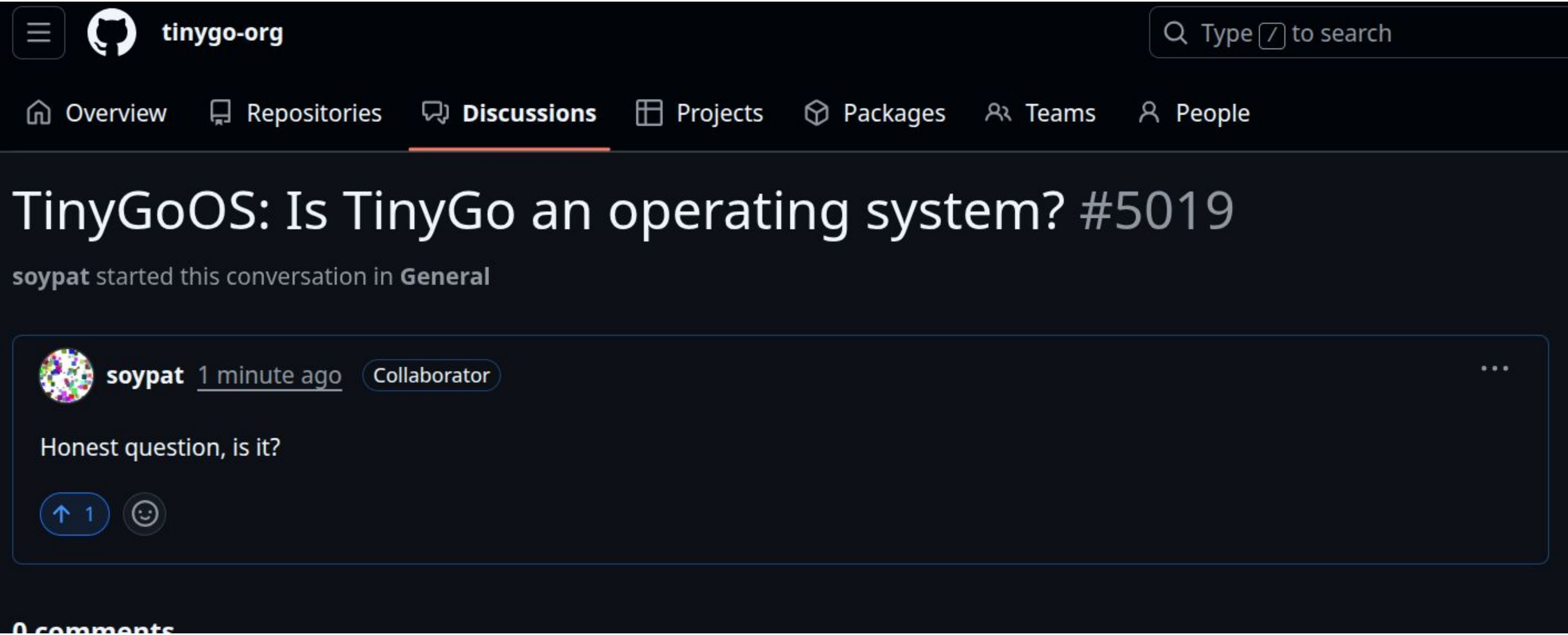
- System startup
- Task/thread scheduling
- Interrupt handling
- Timing services
- System tick, delays, timers
- Inter Process Communication: So queues, semaphores, mutexes
- Memory management: static regions, heap allocation
- Hardware I/O: GPIO, UART/SPI/I^{2}C, PWM, ADC/DAC, CAN, USB, etc
- DMA coordination
- Power management: sleep and deep sleep modes, wakeup sources
- File systems: LittleFS/FAT
- Networking stacks: Ethernet/Wifi/BLE, TCP/IP, DHCP, NTP, DNS
- USB device/host classes: CDC, HID
- Shell/CLI and logging
- Watchdogs: kick/monitor watchdogs

TinyGo is not an OS

- TinyGo
- FreeRTOS
- Azure RTOS
- Zephyr RTOS
- SEGGER embOS
- ChibiOS
- RIOT OS

TinyGo is not an OS

- TinyGo__
- FreeRTOS
- Azure RTOS
- Zephyr RTOS
- SEGGER embOS
- ChibiOS
- RIOT OS



TinyGoOS: Is TinyGo an operating system? #5019

soypat started this conversation in General

soypat 1 minute ago Collaborator

Honest question, is it?

0 comments

Where

Future

TinyGo is an OS (?)

- Multicore Processing
- Userspace networking stack
  - https://github.com/soypat/lneto

TinyGo is an OS (?)

- Multicore Processing
- Userspace networking stack
  - https://github.com/soypat/lneto

"[Go] is about language design in the service of software engineering" -Rob Pike

Passing the torch

- 5 years of TinyGo
- Raspberry Pi Pico
- 30 years of Pato

Reuse Slice Idiom

```
type Object struct {
    buf []float32
}

func (obj *Object) Reset(size int) {
    if cap(obj.buf) < size {
        obj.buf = make([]float32, size)
    }
    obj.buf = obj.buf[:size]
}
```

Addendum: Beware when reusing slices of pointers (or slices). Remember to reset the underlying memory.

Generational Indices

```go
type Object struct {
    gen int // And more fields...
}

func (obj *Object) Reset(cfg Configuration)   {
    obj.gen++
}

func (obj *Object) GetHandle(i idx) Handle { /* */ }

type Handle struct {
    obj *Object
    gen int
}

func (h Handle) IsInvalid() bool {
    return h.obj == nil || h.gen != h.obj.gen
}

func (h Handle) Do() error {
    if h.IsInvalid() {
        return errInvalidGen
    }
    // ...
}
```

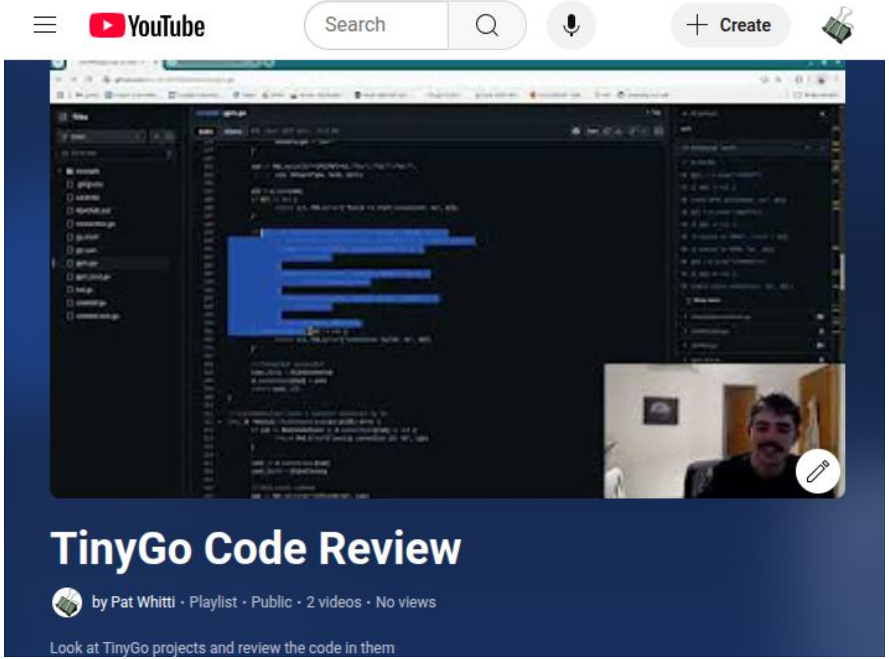
Heap allocation detection

```
tinygo build -print-allocs=. ./cmd/program
```

```
tinygo build -print-allocs=. -target=pico examples/blinky1
/home/pato/src/tg/tinygo/src/runtime/baremetal.go:43:14: object allocated on the heap:
size is not constant

/home/pato/src/tg/tinygo/src/internal/task/task_stack.go:43:24: object allocated on the
heap: object size 8192 exceeds maximum stack allocation size 256

/home/pato/src/tg/tinygo/src/internal/task/task_stack.go:75:12: object allocated on the
heap: escapes at line 77
```

Going further

- Youtube series
- Gophers slack
  - #tinygo
  - #tinygo-dev



TinyGo Code Review

by Pat Whitti · Playlist · Public · 2 videos · No views

Look at TinyGo projects and review the code in them

Closing thoughts

- GOOS=linux/windows/darwin
- GOOS=noos
- GOOS=tinygoos
- GOOS=goos

Thank you!

Patricio Whittingslow
Github: **soypat**
Email: **graded.sp@gmail.com**
Mastodon**: hachyderm.io/@whittileaks**
Twitter: **@whittileaks**

```
//go:build pico
const partAndManufBits = (1 << 28) - 1
info := rp.SYSINFO
print("gc25{", info.CHIP_ID.Get()&partAndManufBits, "}")
```