Porting Go to NetBSD/arm64

About me:

- coypu@sdf.org
- maya@NetBSD.org

QUALIFIER

This talk will only refer to the main Go implementation
(the one in github.com/golang/go).

Several other implementations of Go exist.

Go - typical programming languages facts

- Compiled language
- Strongly typed
- Self-hosted (written in Go)
- Memory safe, garbage collected
- Build logic built into the language

Go - unusual benefits

- Lightweight, easy concurrency
  (Small auto-growing stack, goroutines)
- Static binaries by default
- Extremely easy to cross-build (another OS, another architecture)
  GOOS=plan9 GOARCH=arm

TYPICAL PROGRAMMING LANGUAGE STRUCTURE (C WITH GCC)

↓ C code ↓

Compiler (GCC)

↓ human-readable assembly ↓

Assembler (binutils)

↓ machine-readable assembly ↓

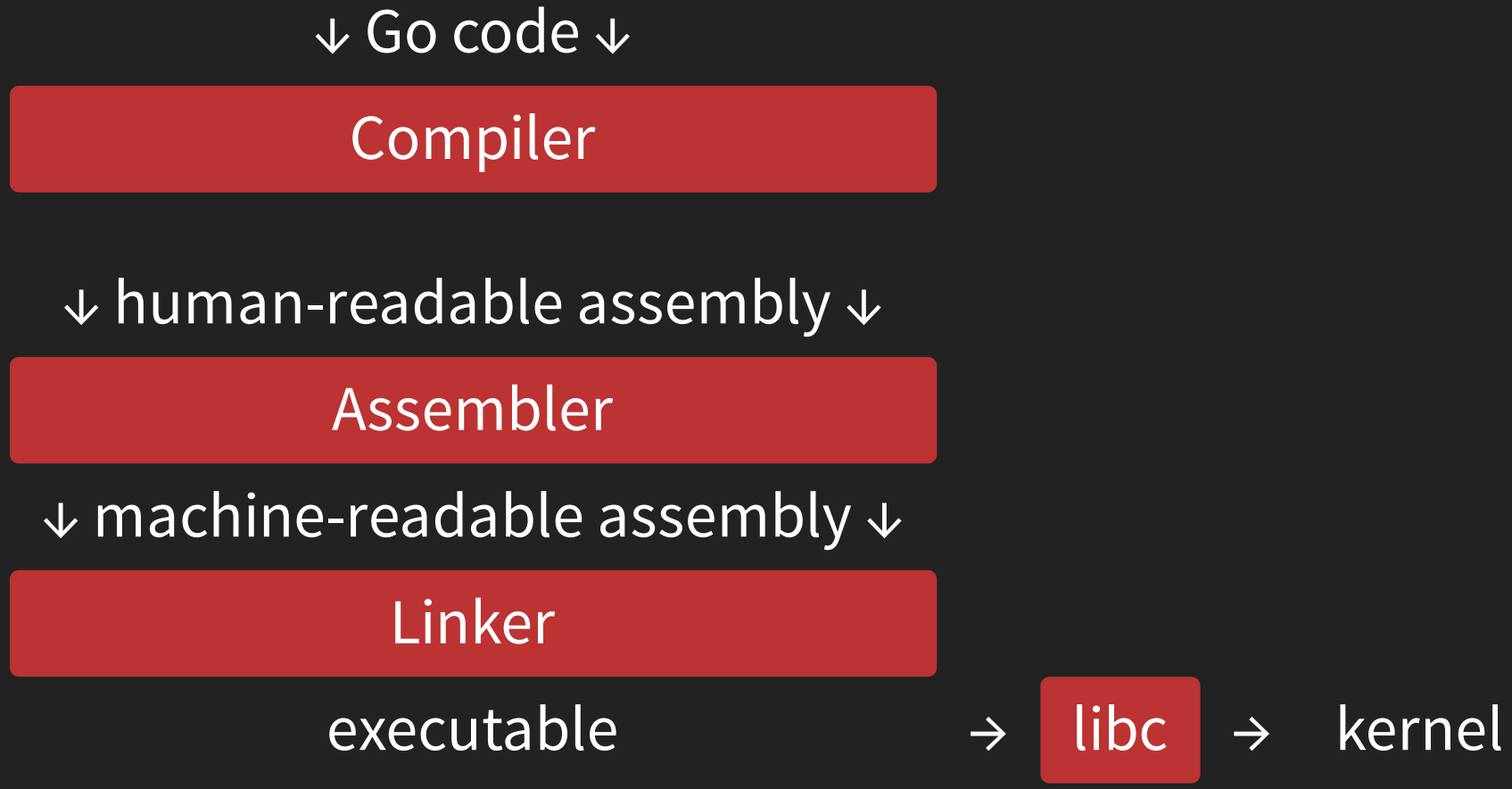
Linker (binutils)

executable → glue code (ld.so) → libc → kernel

REASONABLE PROGRAMMING LANGUAGES REUSE EXISTING TOOLS:

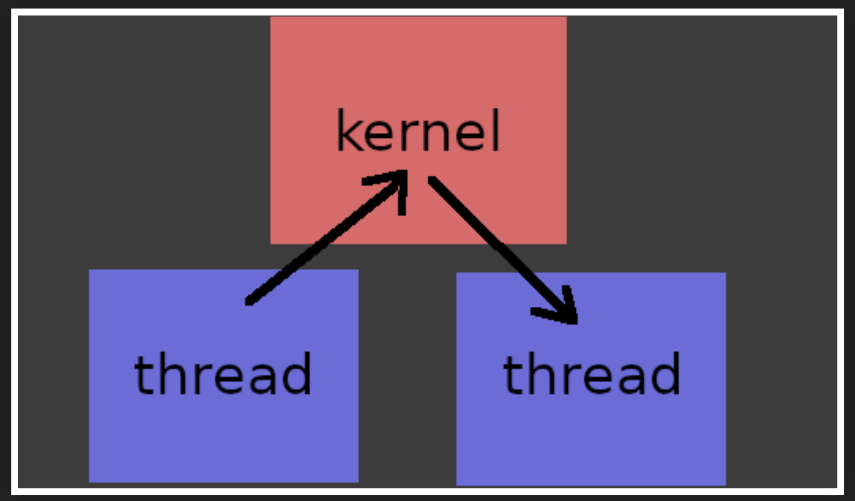
- Mangle into valid C names:
  _ZNKSbIwSt11char_traitsIwESaIwEE4dataEv
  (c++filt) std::basic_string<wchar_t, std::char_traits<wchar_t>, std::allocator<wchar_t> >::data() const
- No reason to write a new assembler/linker!

GO OVERVIEW. PARTS IMPLEMENTED BY GO ARE IN RED

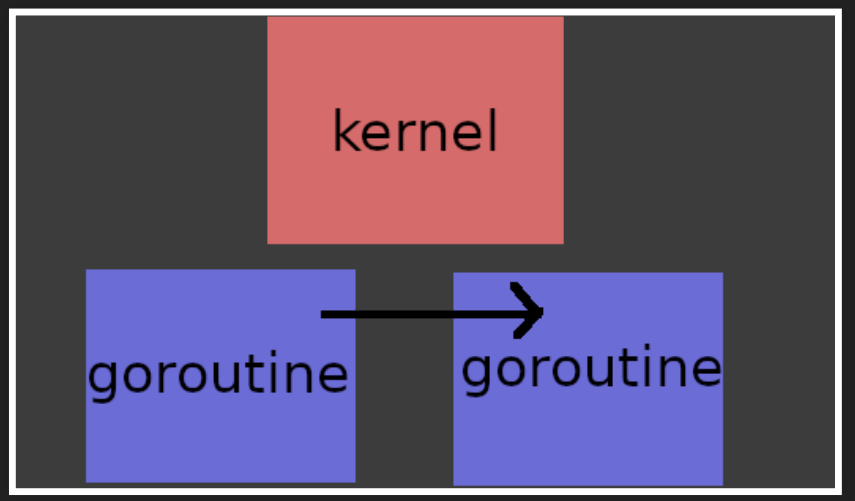


GOROUTINES VS OS THREADS

Regular thread overhead



Goroutine overhead

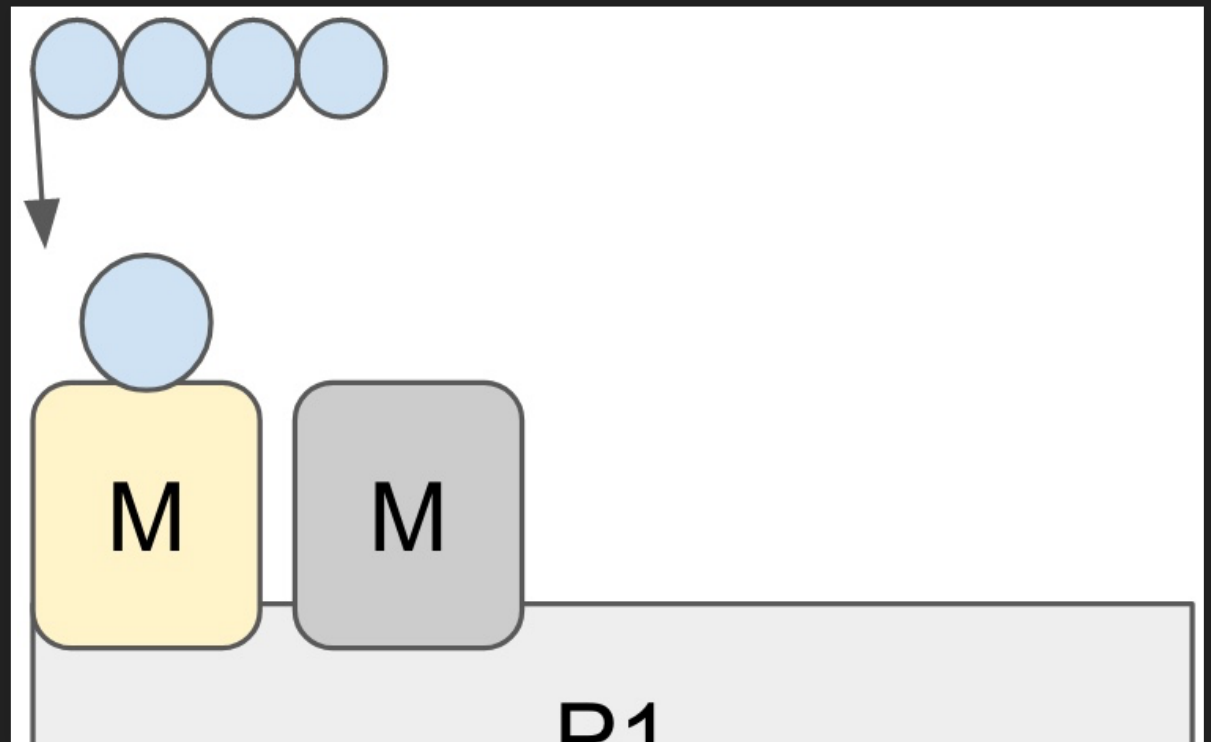


GO PARALLELISM

P - processor

m - OS thread

g - goroutine



GO CALLING CONVENTION

All registers are caller-saved

(after calling a function, your registers might be garbage)

Arguments passed on stack

Limitation, not feature

r28 is struct g

PROCEDURE CALL STANDARD FOR ARM64

Register	Role
SP	Stack pointer
r30	Link register
r19..r28	Callee-saved registers
r9..r15	Temporary registers
r0..r7	Parameter/result registers

CAN'T DELIVER SIGNALS.. ^T.

```
Thread 3 received signal SIGSEGV, Segmentation fault.
0x0000000000004e2b4 in runtime.sigtrampgo (sig=<optimized out>, info=0x4000449bd0, ctx=0x4000449c50) a
307             if sp < g.m.gsignal.stack.lo || sp >= g.m.gsignal.stack.hi {
(gdb) bt
#0  0x0000000000004e2b4 in runtime.sigtrampgo (sig=<optimized out>, info=0x4000449bd0, ctx=0x4000449c5
#1  0x0000000000640ec in runtime.sigtramp () at /home/fly/go/src/runtime/sys_netbsd_arm64.s:297
#2  0x0000000000064060 in runtime.sigprocmask () at /home/fly/go/src/runtime/sys_netbsd_arm64.s:253
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
```

THREAD-LOCAL STORAGE (TLS)

two ways to adjust:

- mrs tpidr_el0 r0
- lwp_setprivate / lwp_getprivate

tpidr - thread pointer

someone sets it up...

FOREIGN FUNCTION INTERFACE (CGO)

Slightly different code when supporting CGo

...thread-local storage interoperability?

```
sendsig_siginfo(const ksiginfo_t *ksi, const sigset_t *mask)
...
        tf->tf_reg[0] = ksi->ksi_signo;
        tf->tf_reg[1] = sip;
        tf->tf_reg[2] = ucp;
        tf->tf_reg[28] = ucp;   /* put in a callee saved register */
```

"GO LIBC"

Why not libc?

Overhead:

- save all registers used in Go (not needed for direct syscall)
- garbage collection accounting
- change to a C-appropriate stack

HOW TO BEGIN?

Naively.

```
~/g/g/src> env GOOS=netbsd GOARCH=arm64 bash ./bootstrap.bash
...
Bootstrap toolchain for netbsd/arm64 installed in /home/fly/go/go-netbsd-arm64-bootstrap.
```

Error strings come from somewhere.

LIST OF THINGS TO IMPLEMENT IN "GO LIBC"

```
lwp_create, lwp_tramp, osyield, lwp_park, lwp_unpark, lwp_self, exit, exitThread,
open, closefd, read, write, usleep, raise, raiseproc, setitimer, walltime, nanotime,
getcontext, sigprocmask, sigreturn_tramp, sigaction, sigfwd, sigtramp, mmap, munmap,
madvise, sigaltstack, settls, sysctl, kqueue, kevent, closeonexec.
```

SIMPLE IMPLEMENTATION: EXIT

x86_64:

```
// Exit the entire program (like C exit)
TEXT runtime·exit(SB),NOSPLIT,$-8
        MOVL    code+0(FP), DI          // arg 1 - exit status
        MOVL    $1, AX                  // sys_exit
        SYSCALL

        MOVL    $0xf1, 0xf1             // crash
        RET
```

arm64:

```
#define SYS_exit                        1

// Exit the entire program (like C exit)
TEXT runtime·exit(SB),NOSPLIT,$-8
        MOVD    code+0(FP), R0          // arg 1 - exit status
        SVC     $SYS_exit
        MOVD    $0, R0                  // If we're still running,
        MOVD    R0, (R0)                // crash
```

/usr/include/sys/syscall.h

```
/* syscall: "exit" ret: "void" args: "int" */
#define SYS_exit        1
```

arm64 exit:

```
#define SYS_exit                        1

// Exit the entire program (like C exit)
TEXT runtime·exit(SB),NOSPLIT,$-8
        MOVD    code+0(FP), R0          // arg 1 - exit status
        SVC     $SYS_exit
        MOVD    $0, R0                  // If we're still running,
        MOVD    R0, (R0)                // crash
```

objdump -d /lib/libc.so

```
_exit:
  191810:       21 00 00 d4     svc     #0x1
  191814:       c0 03 5f d6     ret
```

arm64 exit:

```
#define SYS_exit                        1

// Exit the entire program (like C exit)
TEXT runtime·exit(SB),NOSPLIT,$-8
        MOVD    code+0(FP), R0          // arg 1 - exit status
        SVC     $SYS_exit
        MOVD    $0, R0                  // If we're still running,
        MOVD    R0, (R0)                // crash
```

EVENTUALLY WANTED TO HAVE THEM ALL AVAILABLE...

auto-generate, easier with SYS_syscall

```
func Sync() (err error) {
        _, _, e1 := Syscall(SYS_SYNC, 0, 0, 0)
        if e1 != 0 {
                err = errnoErr(e1)
        }
        return
}
```

IMPLEMENT SYSCALL3, SYSCALL6, SYSCALL9

```
// func RawSyscall(trap uintptr, a1, a2, a3 uintptr) (r1, r2, err uintptr)
TEXT ·RawSyscall(SB),NOSPLIT,$0-56
        MOVD    trap+0(FP), R17 // syscall entry
        MOVD    a1+8(FP), R0
        MOVD    a2+16(FP), R1
        MOVD    a3+24(FP), R2

        SVC     $SYS_syscall
        BCC     ok
        MOVD    $-1, R1
        MOVD    R1, r1+32(FP)   // r1
        MOVD    ZR, r2+40(FP)   // r2
        MOVD    R0, err+48(FP)  // err
        RET
ok:
        MOVD    R0, r1+32(FP)   // r1
        MOVD    R1, r2+40(FP)   // r2
        MOVD    ZR, err+48(FP)  // err
        RET
```

CALLING CONVENTION? NOT FOR SYSCALLS

```
> ktruss -i ./hello
...
    34      1 hello    __sigprocmask14(0x3, 0, 0x1840c0) = 0
    34      1 hello    __clock_gettime50(0x3, 0xffffffffe8b8) = 0
```

Demo

STATUS UPDATE

- upstreamed!
  (Had to fix 'go vet'...)
- Other people contributed, and used it for other BSDs too!
- Thanks to Joel Sing and Thomas Klauser
- Small programs!

Questions?