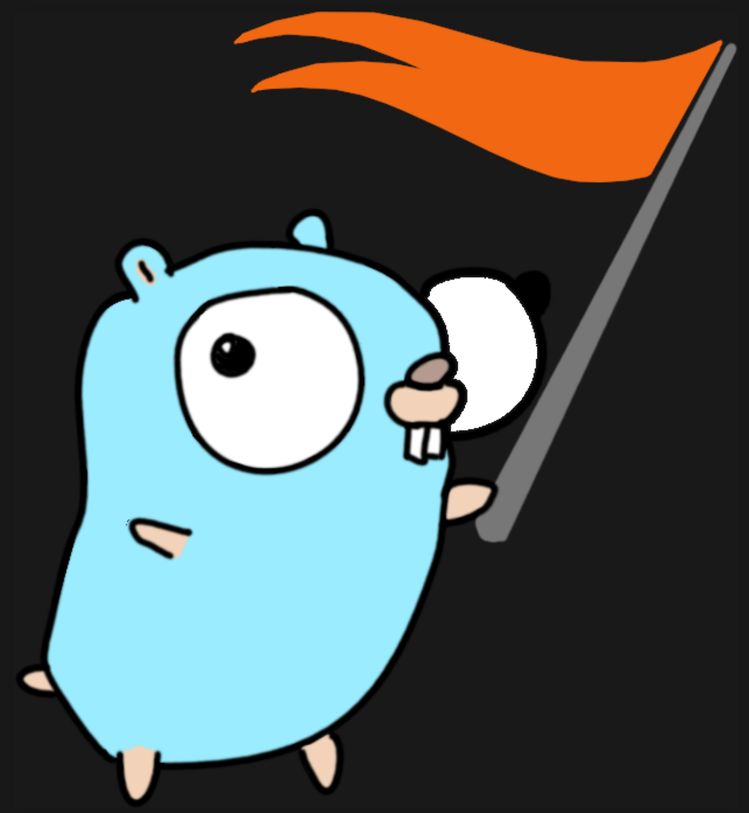
Porting Go to NetBSD/arm64

Maya Rashish <coypu@sdf.org>



Porting Go to NetBSD/arm64

- Porting: making something run on another operating system or architecture
- Go: a programming language
- NetBSD: an operating system (1993-current)
- arm64: CPU architecture (iPhone, most Android...)

Porting Go, a top-level overview

1. Adding your target to the list of supported targets
2. Several generated files
3. Operating System-specific calls

Adding your target to a list of targets

Strategy: pretend it works, look up error strings

```
~/g/src> env GOOS=netbsd GOARCH=arm64 bash ./make.bash
...
Building packages and commands for target, netbsd/arm64.
cmd/go: unsupported GOOS/GOARCH pair netbsd/arm64
```

Generated files

zsysnum, zerror...

NetBSD pretty consistent: copy the amd64 files

Operating System specific logic

open a file, create a thread...

~500 lines of assembly

WHY???

the stack

everything below the stack pointer is free to use.

Repercussions of using libc

- every thread needs its own "big enough" stack.
  constant overhead
- Need to save state Go puts in places that aren't kept by C

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.

Know your C ABI:

```
x0          x0                  x1          x2.. x7.. stack
int  open(const char *path,  int flags,    ...);
```

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
```

Debugging: ktrace

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

C ABI? syscalls aren't required to follow that.

Signal handling

```
^T
Expected:
[ 3032.0244760] load: 0.64  cmd: cat 1530 [ttyraw] 0.00u 0.01s 0% 12
Got:
Segmentation fault
```

g is nil?

g:

- Best, easiest to search name
- goroutine specific accounting

What C ABI says about thread-local storage

Memory area per-thread, each thread gets their own

- "mrs tpidr_el0, r0"

```
#ifdef TLS_linux
#define TPIDR TPIDR_EL0
#define MRS_TPIDR_R0 WORD $0xd53bd040 // MRS TPIDR_EL0, R0
#endif

#ifdef GOOS_darwin
#define TPIDR TPIDRRO_EL0
#define TLSG_IS_VARIABLE
#define MRS_TPIDR_R0 WORD $0xd53bd060 // MRS TPIDRRO_EL0, R0
#endif
```

- lwp_getprivate?

Go dual nature

cgo, using regular thread-local storage, easier to call C

Normal go, assembly, standalone, very incompatible with C.

g is x28.

SIGNAL HANDLING

Want to pass information to signal handler

NetBSD kernel signal delivery...

```
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 */
```

Tramples some registers

all the state to recover is in ucontext (ucp)

Can build hello world

Questions?