第 6 讲：The Programming Languages of OS

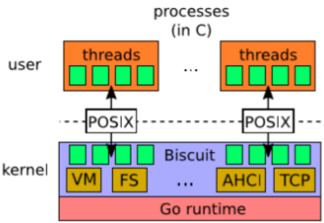
第四节：The benefits and costs of writing kernel in a high-level language

陈渝

清华大学计算机系

yuchen@tsinghua.edu.cn

2020 年 3 月 22 日



58 syscalls, LOC: *28k Go,*
1.5k assembly (boot, entry/exit)

go-lang based Biscuit OS, MIT, OSDI'2018

Should we use high-level languages to build OS kernels?

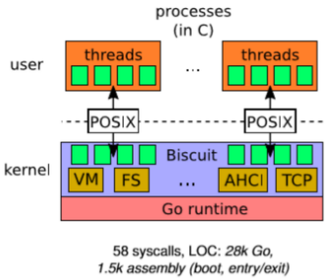
Benefits

- Easier to program
- Simpler concurrency with GC
- Prevents classes of kernel bugs

Downside

- ~~Bounds,~~ cast, ~~nil-pointer checks~~
- ~~Reflection~~
- ~~Garbage collection~~

Introduction – Biscuit



go-lang based Biscuit OS, MIT, OSDI'2018

Goal: measure HLL impact

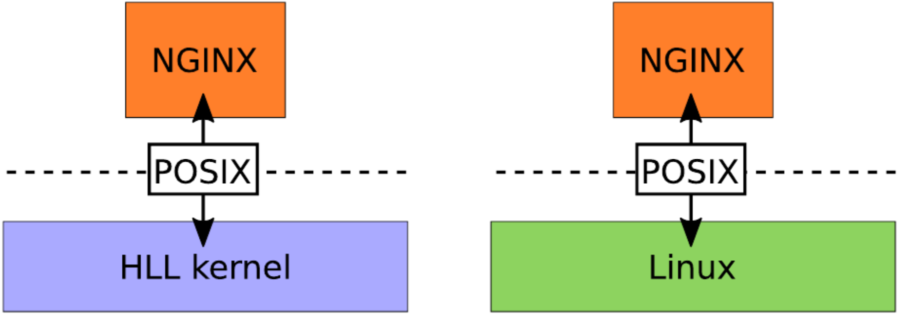
Pros:

- Reduction of bugs
- Simpler code

Cons:

- HLL safety tax
- GC CPU and memory overhead
- GC pause times

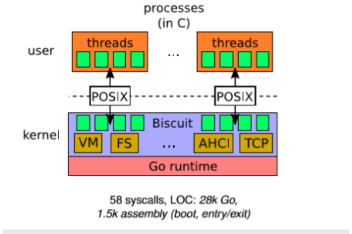
Introduction – Biscuit



Methodology

- None measure HLL impact in a monolithic POSIX kernel
- Build new HLL kernel, compare with Linux
- Isolate HLL impact:
    - Same apps, POSIX interface, and monolithic organization

Introduction – Biscuit



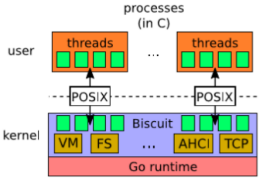
go-lang based Biscuit OS, MIT, OSDI'2018

58 syscalls, LOC: 28k Go, 1.5k assembly (boot, entry/exit)

Go-lang

- Easy to call asm
- Compiled to machine code w/good compiler
- Easy concurrency & static analysis
- GC
  - Concurrent mark and sweep
  - Stop-the-world pauses of 10s of μs

Introduction – Biscuit



58 syscalls, LOC: *28k Go*, *1.5k assembly (boot, entry/exit)*

go-lang based Biscuit OS, MIT, OSDI'2018

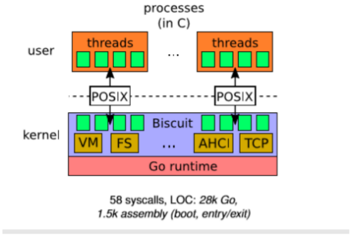
GOLANG

58 syscalls, LOC: 28k Go, 1.5k assembly (boot, entry/exit)

Biscuit

- Multicore
- Threads
- Journaled FS (7k LOC)
- Virtual memory (2k LOC)
- TCP/IP stack (5k LOC)
- Drivers: AHCI and Intel 10G NIC (3k LOC)

Introduction – Biscuit



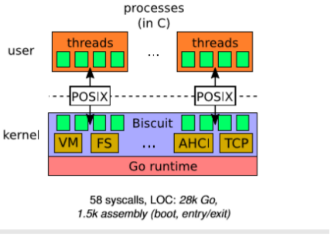
go-lang based Biscuit OS, MIT, OSDI'2018

58 syscalls, LOC: 28k Go, 1.5k assembly (boot, entry/exit)

Many implementation puzzles in Biscuit

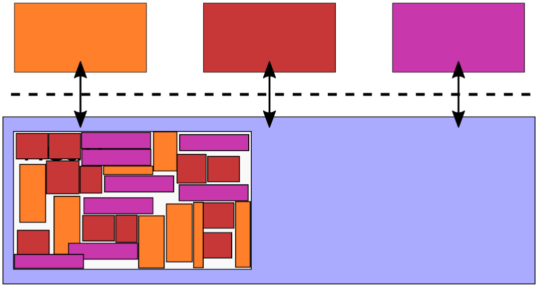
- Interrupts
- Threads
- Kernel threads are lightweight
- Runtime on bare-metal
- Heap exhaustion (Surprising)
- etc.

Introduction – Biscuit



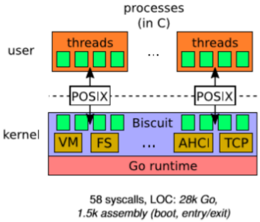
go-lang based Biscuit OS, MIT, OSDI'2018

58 syscalls, LOC: 28k Go, 1.5k assembly (boot, entry/exit)



- Can' t allocate heap memory ==> nothing works
- All kernels face this problem

Introduction – Biscuit



go-lang based Biscuit OS, MIT, OSDI'2018

GOLANG

58 syscalls, LOC: 28k Go, 1.5k assembly (boot, entry/exit)

Strawman 1: Wait for memory in allocator?

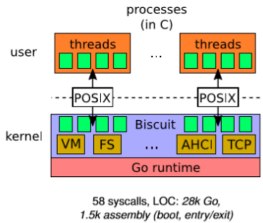
- May deadlock!

Strawman 2: Check/handle allocation failure, like C kernels?

- Difficult to get right
- Can' t! Go doesn' t expose failed allocations
- and implicitly allocates

Both cause problems for Linux; see "too small to fail" rule

Introduction – Biscuit



go-lang based Biscuit OS, MIT, OSDI'2018

GOLANG

58 syscalls, LOC: 28k Go, 1.5k assembly (boot, entry/exit)

To execute syscall...

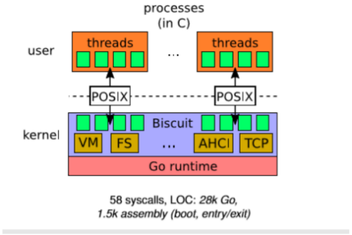
```
reserve()
    (no locks held)
    evict, kill
    wait...
sys_read()
    ...
unreserve()
```

No checks, no error handling code, no deadlock

Reservations

- HLL easy to analyze
- Tool computes reservation via escape analysis
- ≈ three days of expert effort to apply tool

Introduction – Biscuit



go-lang based Biscuit OS, MIT, OSDI'2018

GOLANG

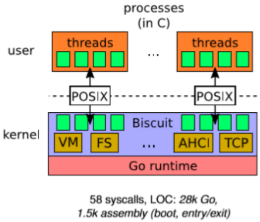
58 syscalls, LOC: 28k Go, 1.5k assembly (boot, entry/exit)

BISCUIT adopted many Linux optimizations:

- large pages for kernel text
- per-CPU NIC transmit queues
- RCU-like directory cache
- concurrent FS transactions
- pad structs to remove false sharing

Good OS performance more about optimizations, less about HLL

Introduction – Biscuit



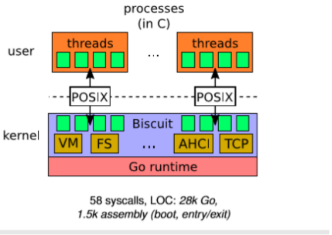
go-lang based Biscuit OS, MIT, OSDI'2018

58 syscalls, LOC: 28k Go, 1.5k assembly (boot, entry/exit)

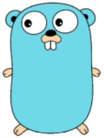
Eval: Should we use high-level languages to build OS kernels?

- Did BISCUIT benefit from HLL features?
- Is BISCUIT performance in the same league as Linux?
- What is the breakdown of HLL tax?
- What is the performance cost of Go compared to C?

Introduction – Biscuit



go-lang based Biscuit OS, MIT, OSDI'2018



58 syscalls, LOC: 28k Go, 1.5k assembly (boot, entry/exit)

Eval: Qualitative benefits of HLL features

- GC' ed allocation
- defer
- multi-valued return
- closures
- maps

Inspected fixes for all publicly-available execute code CVEs in Linux kernel for 2017

Category	#	Outcome in Go
—	11	unknown
logic	14	same
use-after-free/double-free	8	disappear due to GC
out-of-bounds	32	`panic` or disappear due to GC

panic likely better than malicious code execution

Introduction – Biscuit

Biscuit and Linux in the same league

	BISCUIT ops/s	Linux ops/s	Ratio
CMailbench (mem)	15,862	17,034	1.07
NGINX	88,592	94,492	1.07
Redis	711,792	775,317	1.09

the breakdown of HLL tax in Biscuit

	GC cycles	GCs	Prologue cycles	Write barrier cycles	Safety cycles
CMailbench	3%	42	6%	$< 1\%$	3%
NGINX	2%	32	6%	$< 1\%$	2%
Redis	1%	30	4%	$< 1\%$	2%

Introduction – Biscuit

C is 15% faster

Prologue/safety-checks $\Longrightarrow$ 16% more instructions

C (ops/s)	Go (ops/s)	Ratio
536,193	465,811	1.15

- The HLL worked well for kernel development
- Performance is paramount $\Longrightarrow$ use C (up to 15%)
- Minimize memory use $\Longrightarrow$ use C ($\downarrow$ mem. budget, $\uparrow$ GC cost)
- Safety is paramount $\Longrightarrow$ use HLL (40 CVEs stopped)
- Performance merely important $\Longrightarrow$ use HLL (pay 15%, memory)

References

- Multiprogramming a 64 kB Computer Safely and Efficiently,SOSP 2017
- The benefits and costs of writing a POSIX kernel in a high-level language ,OSDI 2018