Hello, World!

— Porting GO to ucore.

计 83 班 陈睿
计 83 班 方宇剑

Original Schedule

- Week 4~5: Get familiar with the experiment environment;
- Week 6: Implement the simplest “hello world”;
- Week 7: Research on thread scheduling;
- Week 8: Flexible;

Exact Progress

- Week 4~5: Get familiar with the experiment environment; [on time]
- Week 6: Implement the simplest “hello world”; [postponed]
- Week 7: Research on thread scheduling; [combined]
- Week 8: Flexible; [hello world done]

Maximum Requirements

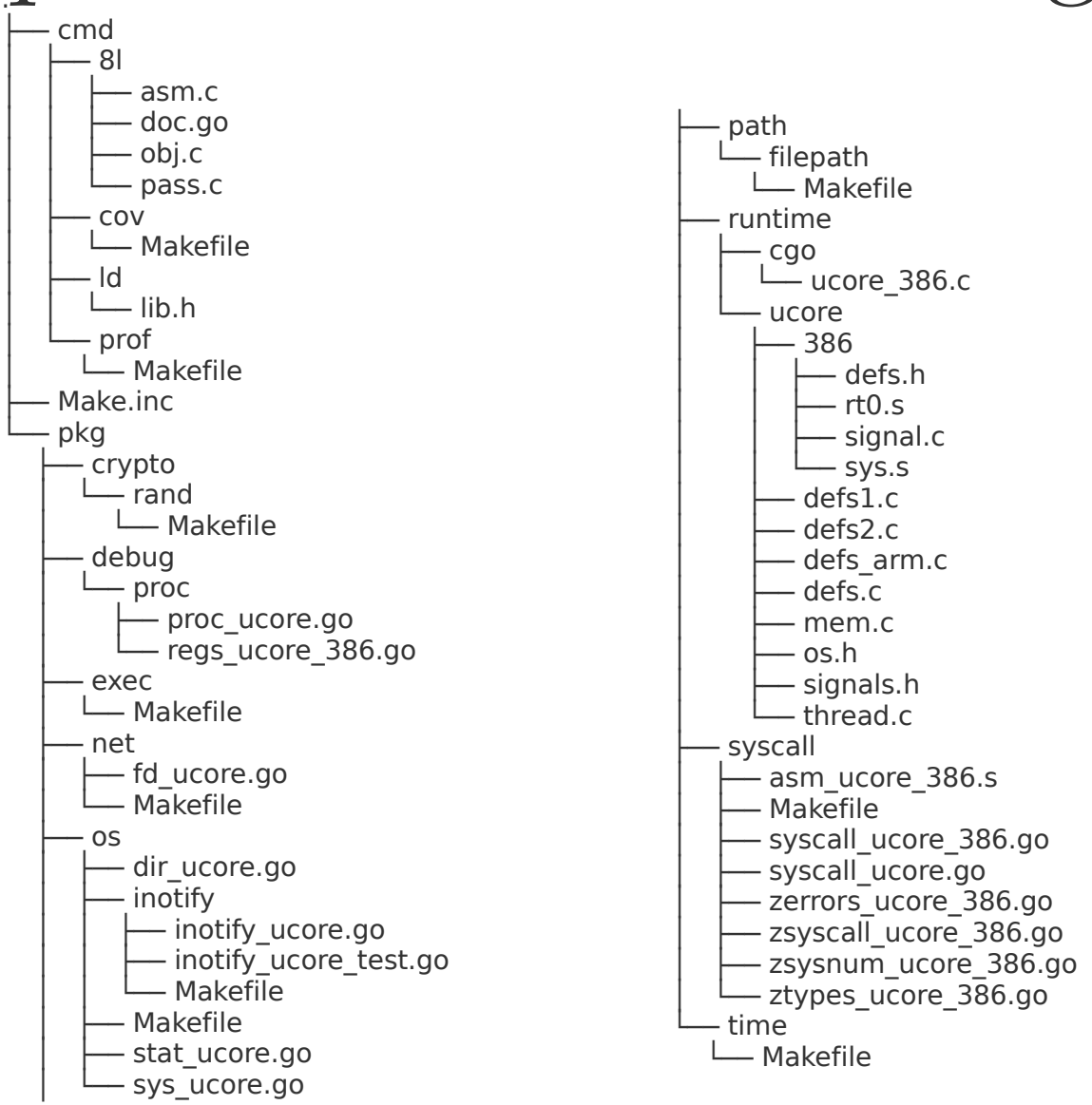
- Linker: copy the one of linux (ELF32);
- Package “runtime”:
  - Memory allocation
  - Local storage segment
  - Printing
  - Exit
  - …
- Package “syscall”, “os”...
- More OS-specific support;

Step1: "hello world" analysis

-	morestack	-	gosave	-	gosched	-	MHeap_Lookup Maybe	-	MSpanList_Remove	-	semacquire
*	lock	-	memclr	-	SizeToClass	-	unmarkspan	-	MSpan_Init	-	MCentral_Grow
*	unlock	*	exit	-	class_to_size	-	MHeap_Free	-	MHeap_FreeLocked	-	semrelease
-	malloc	-	xadd	-	MCache_!Alloc	-	MCache_Free	-	MHeap_SysAlloc	-	nanotime
-	free	*	write	-	MHeap_!Alloc	-	MProf_Free	-	FixAlloc_Free	-	stoptheworld
-	printstring	*	gotraceback	-	markspan	-	gogo	-	MSpanList_Insert	-	newproc1
-	throwinit	*	traceback	-	markallocated	-	MCentral_AllocList	?	casp	-	starttheworld
-	throw	-	getcallerpc	-	MemProfileRate	-	MCentral_FreeList	*	SysAlloc	-	MCache_ReleaseAll
-	malg	-	getcallersp	-	setblockspecial	-	class_to_transfercount	-	mcmp	-	MGetSizeClassInfo
-	runcgo	*	tracebackothers	-	MProf_!Malloc	-	MCentral_Init	-	rnd	*	SysFree
*	newosproc	-	FixAlloc_Init	-	gc	-	MSpanList_Init	-	memmove	-	findnull
-	startpanic	-	FixAlloc_Alloc	-	blockspecial	-	MHeap_AllocLocked	?	caller	*	gettime
-	printf	-	mallocgc	-	mlookup	-	MSpanList_IsEmpty	-	strcmp	*	noteclear
-	dopanic	?	cas	-	markfreed	-	MHeap_AllocLarge	-	getenv	*	notesleep
-	stackalloc	-	gcwaiting	-	checkfreed	-	MHeap_Grow	-	atoi	-	mcpy

* To do - Don't care ? Not clear

Step2: maximum working set



Step3: cross-compile

- Analysis of the GO makefiles' structure;

- Strip the OS-dependent sources and package the ucore resources as an independent patch outside the original GO source;

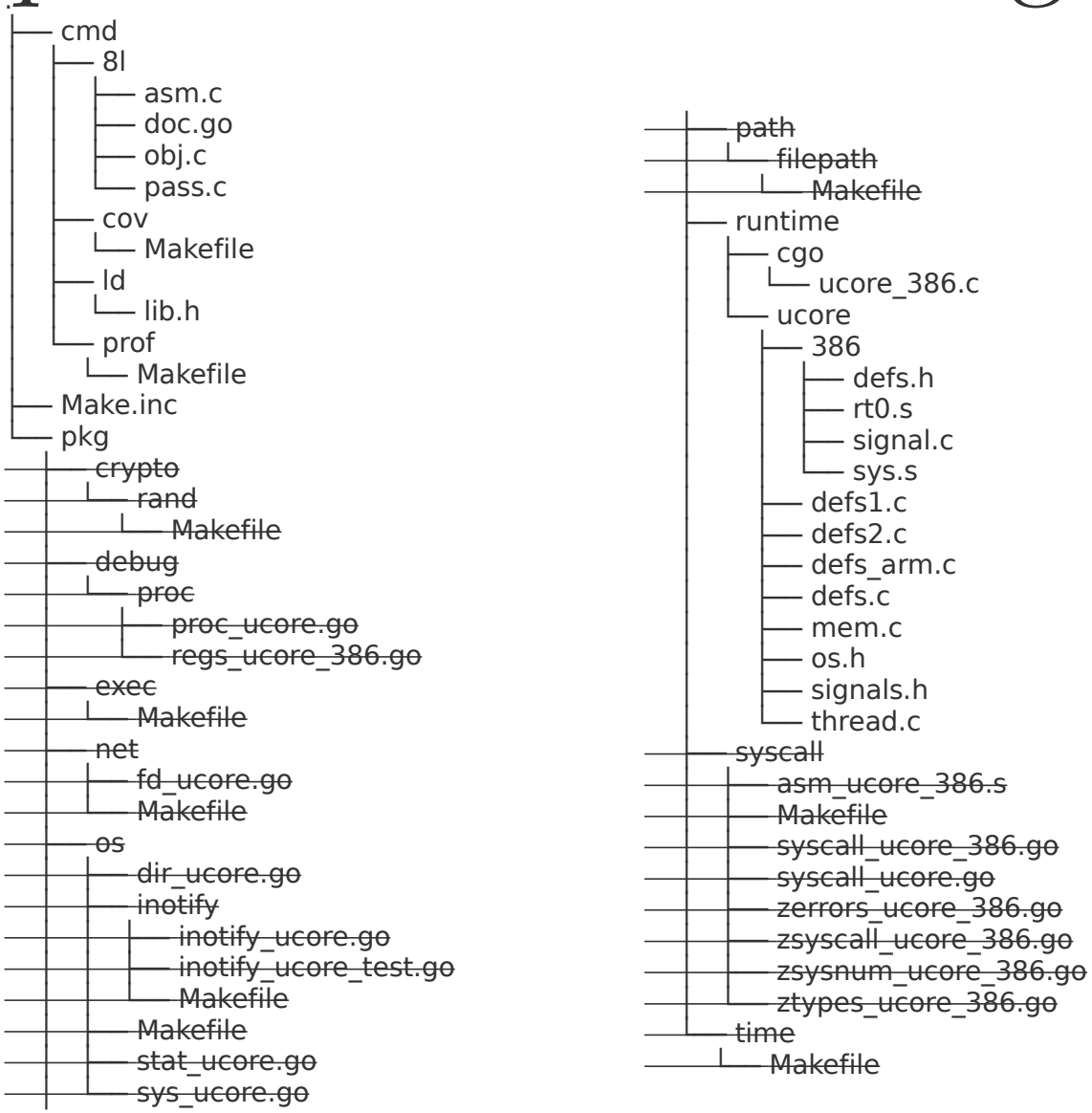
- Here ucore resources are simply a carbon copy of the linux implementation;

Step4: minimize working set

- Minimize the current working set according to the requirements of the simplest “hello world”;

- Focus mainly on package “runtime”, which is the bottom package that could rely on nothing;

Step4: minimize working set



Step5: Setting fake LDT

- runtime·setldt: spare a new GDTe for GO;

- In tiny mode, a GDTe is used as a LDTe, and we decided to do the same thing for the moment;

Step6: Something relatively simple

- `runtime·exit`

- `runtime·write (two revisions), setting stderr to stdout: now we can throw runtime error info and exit;`

- `runtime·rt_sigaction`

Step7: Memory map

- Three kinds of mapping:
  - runtime·SysAlloc: anywhere;
  - runtime·SysReserve: right after text & data segment, 768MB reserved;
  - runtime·SysMap: map area that has been reserved in runtime·SysReserve;
- Zeroize;
- COW: fixed during zeroization;

Problem1: TLS offset

- Thread local storage requires that 0(GS) and 4(GS) actually referring to -8(GS) and -4(GS);

- In GO, tls_offset should be set correctly for a specific OS;

Problem2: set_ldt

- Solved for the moment using a fake LDTe;

Problem3: cross-compiling

- GO uses the environment constant GOOS to decide the compiling target and the compiler compiling target.

- First try: full OS-dependent source;
  - ✗ Compiler compiling fail;
  - ✗ Impossible to make modifications;

Problem3: cross-compiling

- Solved using part-compiling;

- Part-compiling: compile with GOOS unset (default to the current OS), and part-recompile the OS-dependent package;

Problem4: the so-called "hint"

- sys_mmap's requested address is usually regarded as a "hint";

- runtime·SysReserve: the 768MB reserved area is a LITTLE coincident with the data segment;
  - ✗ Killed by the original ucore;

Problem4: the so-called "hint"

- second chance: find it with the hint;

- get_unmapped_area_with_hint(): looks for the suitable place forwards instead of backwards, starting from the requested address;

- Even linux would not always care about the hint... Time for us to cry.

Problem5: reserving mapping

- Difference between runtime·SysReserve and runtime·SysAlloc;
  - ✗ Fault in original ucore;

- Temporary solution: direct return when runtime·SysAlloc is called;

Problem6: argc, argv...

- Process arguments;
  - Unix-like: followed by all the envirment constants ended with a null,

- Manually disable probing environment constants for GO (ucore);

What's next?

- Current implementation is not that elegant, for example,
  - Direct return of runtime·SysMap;
  - Multiple LDTe...
  - Frequent call to putc;
  - mmap flag ignored;

- An advanced “hello world”
  - Package “fmt”;
    - Package “syscall”;
    - Package “os”;
    - ...

What's next?

- More is to be implemented based on thread scheduling, such as futex (mutex in ucore), lock & unlock mechanism, clone (for creating a new thread) etc.;

What can be tough?

- Go: Some signal handlers replaced;
  - ✔ Request for more stack at the stack overflow time;
  - …
  - ✘ We may need to implement signal;

- We have not found TLS in ucore, but it is likely to be a must for GO;

What can be tough?

- We have not compared ucore mutex with the futex GO will be using; the same situation goes for lock & unlock;
- Clone;
- Dynamic or not?
  - runtime·iscgo

Thanks!

```
$ ./hw2.out
Hello, world!
```