Unikernel基础与框架

1.组件化的意义

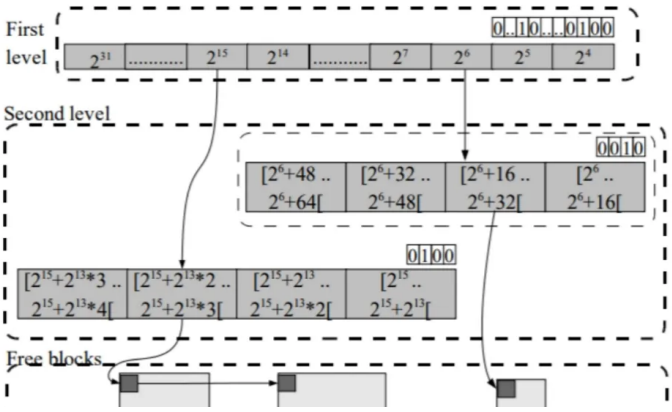
- 设计思路面向场景和需求
- 降低内核开发和维护的难度

2.Unikernel内核

- 应用和内核处于同一特权级
- 共享地址空间
- 既是应用也是内核

3.组件化内存分配方式——框架与算法分离

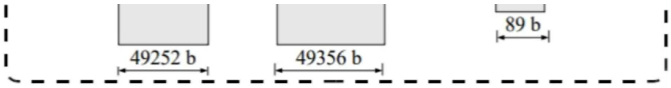
- TLSF
  - 先范围，再等分，最后列表



两级bitmap+List管理空闲块

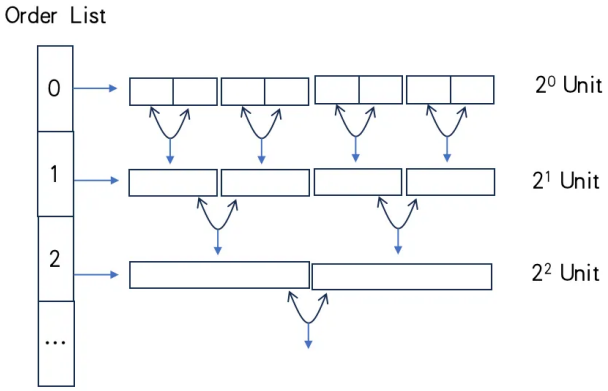
bitmap第一级First Level：
每一位对应一个范围的内存块，示例中分别对应2^4 ~ 2^{31}。1表示空闲。图中两个1。

bitmap第二级Second Level：
有几位就表示几等分。例如， 2^6表示64~127，然后进行4等分就是64~79，80~95，96~107，108~127，每一位对应一个范围，同样1表示空闲。

- 

然后就能找到包含对应范围大小的空闲块链表List。链表耗尽或者新建时，对应维护两级bitmap。

- Buddy
  - 分配时，二分，直至相等
  - 释放时，相同大小的邻居合并

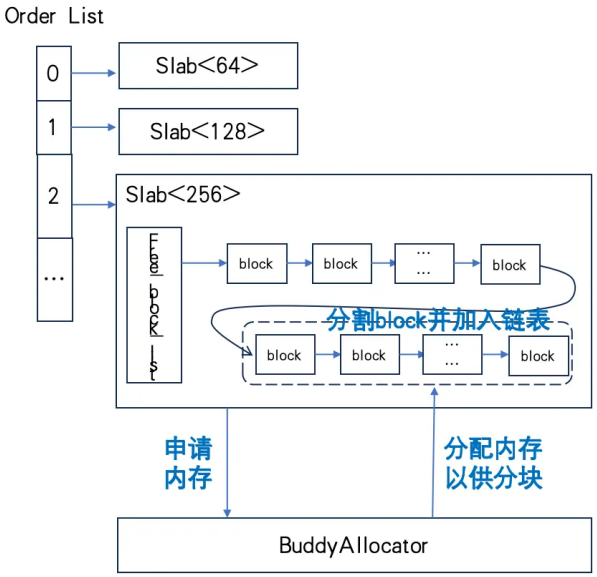


分配单元Unit：
一般不会采用1字节，通常8，16，32…字节

分配：
寻找匹配alloc需要(order)的最小块
如果order大于目标，则二分切割，直至相等，每级剩余的部分挂到对应的Order List

释放：
查看是否有邻居空闲块，有则尽可能向高Oder合并，直至无法合并，挂到OrderList。

- Slab
  - 用链表来维护空闲块



结构：
1）通过OrderList维护一系列Slab
2）Slab维持一个空闲的block链表

分配：从block空闲链表中弹出一个block。
依靠BuddyAllocator提供内存分配支持，初始时以及block不足时，从BuddyAllocator申请，分割block后加入block空闲链表。

释放：放回block空闲链表。

3.实验1.支持带颜色打印输出

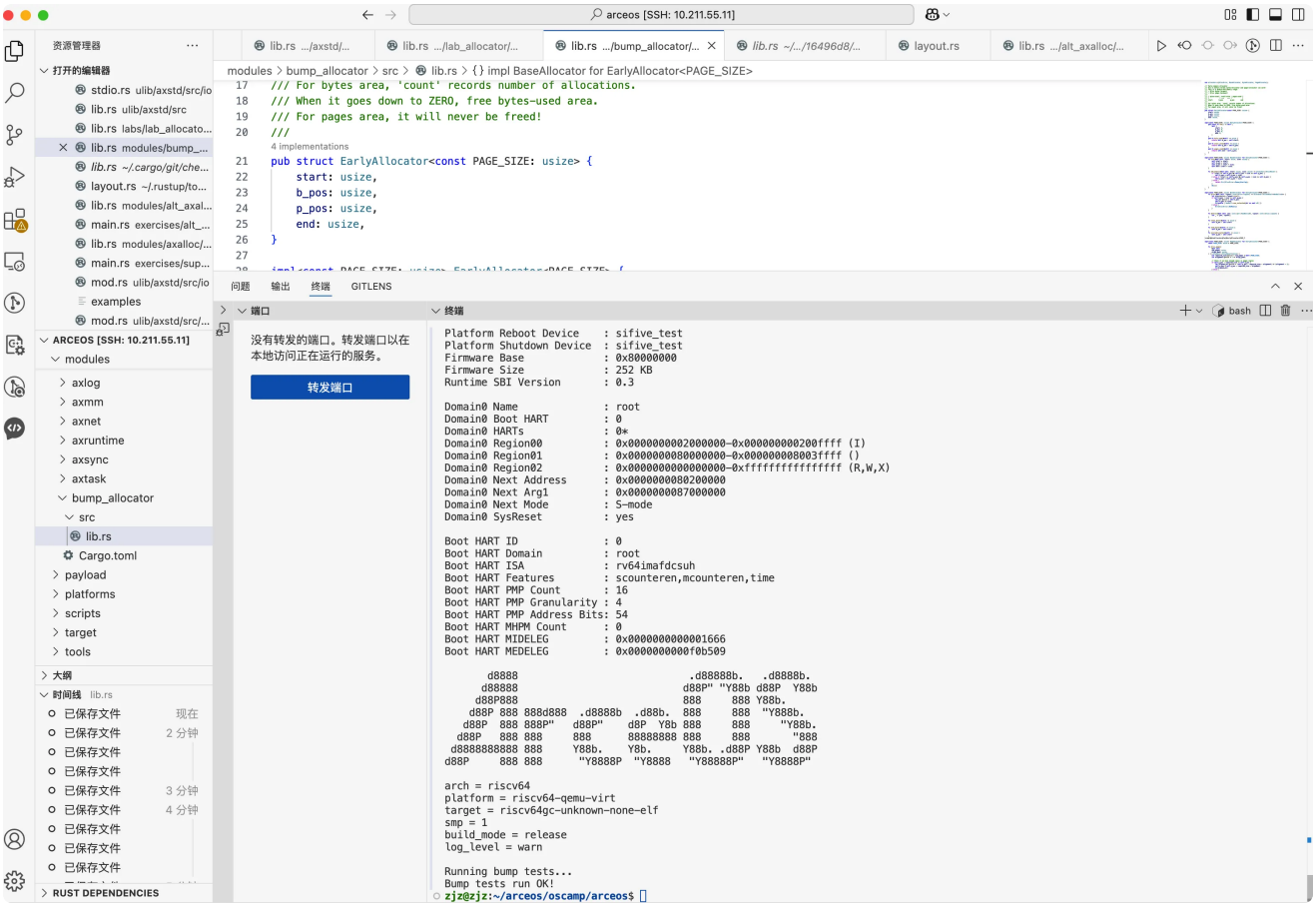
- os采用串口进行输出，因此“\x1B[<style>;<fg>m”的形式在字符串前面输出即可

```
26  impl Write for StdoutRaw {
27      fn write(&mut self, buf: &[u8]) -> io::Result<usize> {
28          arceos_api::stdio::ax_console_write_bytes(buf: b"\x1B[1;31m");      You, 1秒钟前 • Uncommitted
29          arceos_api::stdio::ax_console_write_bytes(buf)
30      }
31      fn flush(&mut self) -> io::Result<()> {
32          Ok(())
33      }
34  }
```

```
问题  输出  终端  GITLENS
终端                                                          bash
Domain0 SysReset          : yes

Boot HART ID              : 0
Boot HART Domain          : root
Boot HART ISA             : rv64imafdcsuh
Boot HART Features        : scounteren,mcounteren,time
Boot HART PMP Count       : 16
Boot HART PMP Granularity : 4
Boot HART PMP Address Bits: 54
Boot HART MHPM Count      : 0
Boot HART MIDELEG         : 0x0000000000001666
Boot HART MEDELEG         : 0x0000000000f0b509

       d8888                            .d88888b.   .d8888b.
      d88888                           d88P" "Y88b d88P  Y88b
     d88P888                           888     888 Y88b.
    d88P 888 888d888  .d8888b  .d88b.  888     888  "Y888b.
   d88P  888 888P"   d88P"    d8P  Y8b 888     888     "Y88b.
  d88P   888 888     888      88888888 888     888       "888
 d8888888888 888     Y88b.    Y8b.     Y88b. .d88P Y88b  d88P
d88P     888 888      "Y8888P  "Y8888   "Y88888P"   "Y8888P"

arch = riscv64
platform = riscv64-qemu-virt
target = riscv64gc-unknown-none-elf
smp = 1
build_mode = release
log_level = warn

Hello, Arceos!
```

4.实验2.bump_allocator

- 即是字节分配器，也是页分配器
- 用途：在不知道具体的物理空间大小时，只知道一块区域的物理内存大小时，可以在一开始初始化时使用


arceos [SSH: 10.211.55.11]
资源管理器
打开的编辑器
stdio.rs ulib/axstd/src/io
lib.rs ulib/axstd/src
lib.rs labs/lab_allocato...
lib.rs modules/bump_...
lib.rs ~/.cargo/git/che...
layout.rs ~/.rustup/to...
lib.rs modules/alt_axal...
main.rs exercises/alt_...
lib.rs modules/axalloc/...
main.rs exercises/sup...
mod.rs ulib/axstd/src/io
examples
mod.rs ulib/axstd/src/...
ARCEOS [SSH: 10.211.55.11]
modules
axlog
axmm
axnet
axruntime
axsync
axtask
bump_allocator
src
lib.rs
Cargo.toml
payload
platforms
scripts
target
tools
大纲
时间线 lib.rs
已保存文件 现在
已保存文件 2 分钟
已保存文件
已保存文件
已保存文件 3 分钟
已保存文件 4 分钟
已保存文件
已保存文件
已保存文件
RUST DEPENDENCIES
lib.rs .../axstd/...
lib.rs .../lab_allocator/...
lib.rs .../bump_allocator/...
lib.rs ~/.../16496d8/...
layout.rs
lib.rs .../alt_axalloc/...
modules > bump_allocator > src > lib.rs > { } impl BaseAllocator for EarlyAllocator<PAGE_SIZE>
/// For bytes area, 'count' records number of allocations.
/// When it goes down to ZERO, free bytes-used area.
/// For pages area, it will never be freed!
///
4 implementations
pub struct EarlyAllocator<const PAGE_SIZE: usize> {
start: usize,
b_pos: usize,
p_pos: usize,
end: usize,
}
问题
输出
终端
GITLENS
端口
没有转发的端口。转发端口以在本地访问正在运行的服务。
转发端口
终端
bash
Platform Reboot Device : sifive_test
Platform Shutdown Device : sifive_test
Firmware Base : 0x80000000
Firmware Size : 252 KB
Runtime SBI Version : 0.3
Domain0 Name : root
Domain0 Boot HART : 0
Domain0 HARTs : 0*
Domain0 Region00 : 0x0000000002000000-0x000000000200ffff (I)
Domain0 Region01 : 0x0000000080000000-0x000000008003ffff ()
Domain0 Region02 : 0x0000000000000000-0xffffffffffffffff (R,W,X)
Domain0 Next Address : 0x0000000080200000
Domain0 Next Arg1 : 0x0000000087000000
Domain0 Next Mode : S-mode
Domain0 SysReset : yes
Boot HART ID : 0
Boot HART Domain : root
Boot HART ISA : rv64imafdcsuh
Boot HART Features : scounteren,mcounteren,time
Boot HART PMP Count : 16
Boot HART PMP Granularity : 4
Boot HART PMP Address Bits: 54
Boot HART MHPM Count : 0
Boot HART MIDELEG : 0x0000000000001666
Boot HART MEDELEG : 0x0000000000f0b509
d8888 .d88888b. .d8888b.
d88888 d88P" "Y88b d88P Y88b
d88P888 888 888 Y88b.
d88P 888 888d888 .d8888b .d88b. 888 888 "Y888b.
d88P 888 888P" d88P" d8P Y8b 888 888 "Y88b.
d88P 888 888 888 88888888 888 888 "888
d88888888888 888 Y88b. Y8b. Y88b. .d88P Y88b d88P
d88P 888 888 "Y8888P "Y8888 "Y88888P" "Y8888P"
arch = riscv64
platform = riscv64-qemu-virt
target = riscv64gc-unknown-none-elf
smp = 1
build_mode = release
log_level = warn
Running bump tests...
Bump tests run OK!
zjz@zjz:~/arceos/oscamp/arceos$