

# OS开发中阶训练营第一周课后作业

## 预备

1. Fork ArceOS的工程，clone到本地。工程链接如下

```
git@github.com:rcore-os/arceos.git
```

2. 在main分支下，创建并切换到新的分支week1，执行

```
git checkout -b week1
```

后面的实验都在该分支下进行。

## 第一节课课后作业

支持HashMap数据类型。以apps/memtest为测试应用。

本次作业的主要目的为带领大家了解 arceos 的接口层次，主要修改内容在 ulib/axstd 文件夹中。

首先修改 apps/memtest/src/main.rs，把 BTreeMap 替换为 HashMap，如下：

```
use rand::{rngs::SmallRng, RngCore, SeedableRng};
-use std::collections::BTreeMap;
+use std::collections::HashMap;
use std::vec::Vec;

fn test_vec(rng: &mut impl RngCore) {
@@ -22,9 +22,9 @@ fn test_vec(rng: &mut impl RngCore) {
    println!("test_vec() OK!");
}

-fn test_btree_map(rng: &mut impl RngCore) {
+fn test_hashmap_map(rng: &mut impl RngCore) {
    const N: usize = 50_000;
    - let mut m = BTreeMap::new();
    + let mut m = HashMap::new();
    for _ in 0..N {
        let value = rng.next_u32();
        let key = format!("key_{value}");
@@ -35,7 +35,7 @@ fn test_btree_map(rng: &mut impl RngCore) {
        assert_eq!(k.parse::<u32>().unwrap(), *v);
    }
}

- println!("test_btree_map() OK!");
+ println!("test_hashmap_map() OK!");
```

```

}

#[cfg_attr(feature = "axstd", no_mangle)]
@@ -44,7 +44,7 @@ fn main() {

    let mut rng = SmallRng::seed_from_u64(0xdead_beef);
    test_vec(&mut rng);
-   test_btree_map(&mut rng);
+   test_hashmap_map(&mut rng);

    println!("Memory tests run OK!");
}

```

然后，尝试编译运行，`make A=apps/memtest ARCH=riscv64 run`，此时会报错，因为我们目前不支持 HashMap 类型。

```

→ arceos git:(os_camp) make A=apps/memtest ARCH=riscv64 run
   Building App: memtest, Arch: riscv64, Platform: riscv64-qemu-virt, App type:
rust
cargo build --target riscv64gc-unknown-none-elf --target-dir
/home/hky/workspace/rcore-os/arceos/target --release --manifest-path
apps/memtest/Cargo.toml --features "axstd/log-level-warn"
   Compiling arceos-memtest v0.1.0 (/home/hky/workspace/rcore-
os/arceos/apps/memtest)
error[E0432]: unresolved import `std::collections::HashMap`
  --> apps/memtest/src/main.rs:9:5
   |
9  | use std::collections::HashMap;
   |     ^^^^^^^^^^^^^^^^^^^^^^^^^ no `HashMap` in `collections`

For more information about this error, try `rustc --explain E0432`.
error: could not compile `arceos-memtest` (bin "arceos-memtest") due to 1 previous
error
make: *** [scripts/make/build.mk:36: _cargo_build] Error 101

```

**要求：**在ulib/axstd中支持HashMap类型

**预期输出：**执行`make A=apps/memtest ARCH=riscv64 run`

```

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

Running memory tests...
test_vec() OK!

```

```
test_hashmap_map() OK!
Memory tests run OK!
```

提示：

1. 参考官方rust标准库中的HashMap实现，把涉及的代码拷过来，做一下修改。只需要满足memtest的测试需要即可。

( 在 axstd 中使用 rust 标准库中的代码可能需要给 axstd 增加一些 feature，可能需要增加这些： )

```
// ulib/axstd/src/lib.rs
#![cfg_attr(all(not(test), not(doc)), no_std)]
#![feature(doc_cfg)]
#![feature(doc_auto_cfg)]
#![feature(hashmap_internals)]
#![feature(extend_one)]
#![feature(hasher_prefixfree_extras)]
#![feature(error_in_core)]
#![feature(try_reserve_kind)]
#![feature(thread_local)]
#![feature(const_hash)]
```

2. 注意：官方std与ArceOS的axstd的区别。官方rust标准库主要是基于Linux/Windows这些内核，为应用提供的用户库。官方std的支持后端是libc+syscall；而ArceOS是单特权级，没有syscall一说，axstd直接通过一系列function-call调用底层的功能。
3. HashMap之所以没有像其他collections类型一样放到alloc库中实现，主要是因为它需要随机数的支持，而随机数的产生机制是平台相关的。类似的代码可以在 std/src/hash/randoms.rs 中对 RandomState 的初始化代码中找到。大家做实验可以简单点，用一个软实现的随机数函数来产生。比如

```
use spinlock::SpinNoIrq;
use crate::time;

static PARK_MILLER_LEHMER_SEED: SpinNoIrq<u32> = SpinNoIrq::new(0);
const RAND_MAX: u64 = 2_147_483_647;

pub fn random() -> u128 {
    let mut seed = PARK_MILLER_LEHMER_SEED.lock();
    if *seed == 0 {
        *seed = time::current_ticks() as u32;
    }

    let mut ret: u128 = 0;
    for _ in 0..4 {
        *seed = ((u64::from(*seed) * 48271) % RAND_MAX) as u32;
        ret = (ret << 32) | (*seed as u128);
    }
    ret
}
```

## 第二节课课后作业

在第一节课的基础上，针对字节内存分配，新增一个简单的内存分配算法。

本次作业的主要目的为带领大家了解arceos的内存分配接口，主要修改内容在 `crates/allocator` 文件夹中。

**准备：**阅读 `crates/allocator/src/lib.rs`，了解 `arceos` 目前支持的内存分配算法，思考它们的区别与适用场景。

**要求：**在 `crates/allocator` 中支持一个新的内存分配算法，传递 `feature` 为 `"new"`。

```
// crates/allocator/src/lib.rs
#[cfg(feature = "bitmap")]
mod bitmap;
#[cfg(feature = "bitmap")]
pub use bitmap::BitmapPageAllocator;

#[cfg(feature = "buddy")]
mod buddy;
#[cfg(feature = "buddy")]
pub use buddy::BuddyByteAllocator;

#[cfg(feature = "slab")]
mod slab;
#[cfg(feature = "slab")]
pub use slab::SlabByteAllocator;

#[cfg(feature = "new")]
mod new;
#[cfg(feature = "new")]
pub use new::YourNewAllocator;
```

注意修改 `allocator` 的 `Cargo.toml`

```
[features]
default = []
full = ["bitmap", "tlsf", "slab", "buddy", "allocator_api"]

bitmap = ["dep:bitmap-allocator"]

tlsf = ["dep:rlsf"]
slab = ["dep:slab_allocator"]
buddy = ["dep:buddy_system_allocator"]
new = ["dep:your_dep_if_needed"]
```

**提示：**

可以使用 <https://github.com/rust-osdev/linked-list-allocator>

也可以去 [crates.io](https://crates.io) 里面寻找更多的合适的 allocator

大部分现成的 allocator 都没有现成的 `add_memory` 功能实现，针对 `BaseAllocator` 中的该接口可以之间返回 `Ok`

```
impl BaseAllocator for YourNewByteAllocator {
    fn init(&mut self, start: usize, size: usize) {
        /// Your implementation.
    }

    fn add_memory(&mut self, _start: usize, _size: usize) -> AllocResult {
        // unsafe { self.inner.add_to_heap(start, start + size) };
        Ok(())
    }
}
```

注意:

参考 `ulib/axstd/Cargo.toml` 中关于内存管理算法的 `features` 增加一个新的 `feature` 选项。

```
# ulib/axstd/Cargo.toml
# Memory
alloc = ["arceos_api/alloc", "axfeat/alloc", "axio/alloc"]
alloc-tlsf = ["axfeat/alloc-tlsf"]
alloc-slab = ["axfeat/alloc-slab"]
alloc-buddy = ["axfeat/alloc-buddy"]
alloc-new = ["axfeat/alloc-new"]
```

参考 `api/axfeat/Cargo.toml` 中关于内存管理算法的 `features` 增加一个新的 `feature` 选项。

```
# api/axfeat/Cargo.toml
# Memory
alloc = ["axalloc", "axruntime/alloc"]
alloc-tlsf = ["axalloc/tlsf"]
alloc-slab = ["axalloc/slab"]
alloc-buddy = ["axalloc/buddy"]
alloc-new = ["axalloc/new"]
```

最后修改 `modules/axalloc/Cargo.toml` 中关于内存管理算法的 `features` 增加一个新的 `feature` 选项。

```
# modules/axalloc/Cargo.toml
[features]
default = ["tlsf"]
tlsf = ["allocator/tlsf"]
slab = ["allocator/slab"]
```

```
buddy = ["allocator/buddy"]
new = ["allocator/new"]
```

**预期输出：**执行`make A=apps/memtest ARCH=riscv64 run FEATURES=alloc-new`，结果应该与第一次作业一致。

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

Running memory tests...
test_vec() OK!
test_hashmap_map() OK!
Memory tests run OK!
```

## 作业提交

1、题目做完以后, 使用 `git diff` 将本次作业修改的内容提取成补丁文件,命名为 用户名-题号.patch

例如:

```
git diff main > pengzechen-lesson1.patch
```

2、提交之前**删除根目录的target目录**, 将剩余文件(包括上一步生成的patch文件)打包成**zip压缩包**，使用固定邮箱发送到指定邮箱: 173701980@qq.com，**邮箱标题**命名为 **学员名称-小组名称-lesson{题号}**，没有加入小组可命名为 **单人**。

(例如：第一题课后作业邮箱标题为 "唐翰文-测试小组-lesson1" 或 "唐翰文-单人-lesson1")

**注意：**

第一周的作业主要是为了让大家都熟悉 `arceos` 的代码结构，作业本身难度并不高，只是需要有限的工作量。

作业的预期输出比较简单，老师与助教会手动查看代码实现，**希望大家不要通过修改输出结果来欺骗自动评测脚本。**