

The background image is a high-angle aerial photograph of a landscape. It features a dense forest of green and yellowish trees covering a hillside. A narrow, light-colored path or road winds its way down the slope. At the bottom of the hill, there is a body of water with some darker, possibly rocky or sandy, areas visible near the shore. The overall scene is natural and scenic.

OS 大实验汇报

2024.3.24 · 致理-信计11 游宇凡

基础实验完成情况

- 通过了 lab1~5 测试，完成了实验报告、简答题
- 对框架进行了一些小的优化
 - 修复了 clippy warning
 - 重新实现 `mm::translate` 模块代替 `translated_byte_buffer` 等函数
 - 其他小的调整，例如消除重复代码，调整逻辑结构
- 实现了选做任务，多种调度算法

基础实验完成情况

mm::translate

```
impl Iterator for TranslatedByteBuffers // 使用迭代器，不用一次性读取至 `Vec`  
    type Item = &'static mut [u8];  
  
impl<T> Iterator for TranslatedArrayRead<T> // 读取任意类型的值，处理跨页读取  
    type Item = T;  
  
impl<T> TranslatedArrayWrite<T>  
    pub fn write_next(&mut self, val: &T) -> isize // 写入任意类型的值，处理跨页写入  
  
pub unsafe fn translated_str_until_nul(token: usize, ptr: *const u8) -> String // 每个 page 只进行一次地址翻译
```

基础实验完成情况

修复 UB

在 lab4 中遇到了一个 UB，经过测试找到了 workaround。

```
--- a/os/src/syscall/process.rs
+++ b/os/src/syscall/process.rs

+#[inline(never)]
+fn uninlined_new_tcb(elf_data: &[u8]) -> Arc<TaskControlBlock> {
+    Arc::new(TaskControlBlock::new(elf_data))
+}
+
pub fn sys_spawn(path: *const u8) -> isize {
    let task = current_task().unwrap();
@@ -155,7 +162,7 @@ pub fn sys_spawn(path: *const u8) -> isize {
    let path = unsafe { translated_str_until_nul(token, path) };
    if let Some(app_inode) = open_file(path.as_str(), OpenFlags::RDONLY) {
        let data = app_inode.read_all();
-        let child = Arc::new(TaskControlBlock::new(&data));
+        let child = uninlined_new_tcb(&data);
        let child_pid = child.pid.0.try_into().unwrap();
        child.inner_exclusive_access().parent = Some(Arc::downgrade(&task));
        task.inner_exclusive_access().children.push(child.clone());
    }
}
```

基础实验完成情况

多种调度算法： trait TaskManager

```
pub trait TaskManager: Default {
    type Data;

    /// data for new task
    fn new_data(&self) -> Self::Data;

    /// change data based on priority, returns priority on success, -1 on failure
    #[allow(unused_variables)]
    fn set_priority(&self, data: &mut Self::Data, priority: isize) -> isize {
        priority
    }

    /// add a ready task
    fn add(&mut self, task: Arc<TaskControlBlock>);

    /// fetch a task to schedule
    fn fetch(&mut self) -> Option<Arc<TaskControlBlock>>;
}

pub type TaskManagerData = <ChosenTaskManager as TaskManager>::Data;
```

基础实验完成情况

多种调度算法：选择算法

```
// mod rr;
// type ChosenTaskManager = rr::RoundRobin;
// mod lottery;
// type ChosenTaskManager = lottery::LotteryScheduling;
// mod stride;
// type ChosenTaskManager = stride::StrideScheduling;
mod mlfq;
type ChosenTaskManager = mlfq::MultiLevelFeedbackQueue;
```

特别地，调整对 timer interrupt 的处理即可得到 FCFS：

```
--- a/os/src/trap/mod.rs
+++ b/os/src/trap/mod.rs
@@ -95,7 +95,6 @@ pub fn trap_handler() -> ! {
    Trap::Interrupt(Interrupt::SupervisorTimer) => {
        set_next_trigger();
        check_timer();
-       suspend_current_and_run_next();
    }
-      => {
-         panic!()
```

基础实验完成情况

多种调度算法：功能测试

每种调度算法均能通过测例。

stride、MLFQ 和 lottery 支持设置优先级，stride 最符合设定的比例，lottery 最不稳定。

- stride: max ratio / min ratio = 1.031
- lottery: max ratio / min ratio = 1.576
- MLFQ: max ratio / min ratio = 1.113

基础实验完成情况

多种调度算法：性能测试

运行 50 个 sleep 和 50 个计算任务，测试指标为：

- 比设定值多 sleep 的时长（表示 response time）
- 计算任务的 turnaround time

基础实验完成情况

多种调度算法：性能测试

运行 50 个 sleep 和 50 个计算任务，测试指标为：

- 比设定值多 sleep 的时长（表示 response time）
- 计算任务的 turnaround time

	round robin	FCFS	lottery	stride	MLFQ
extra sleep / s	13.31	376.5	14.23	18.86	7.916
turnaround time / s	571.1	471.3	553.1	568.3	577.6

- MLFQ 的 response time 最低，turnaround time 稍高
- FCFS 拥有最低的 turnaround time 和极高的 response time

大实验选题计划

- 预计使用 Verus 编写 rCore tutorial lab1~5 的形式化验证（同时修复其 bug）
- 对其他工具（Prusti、Kani……）也进行了一点了解，暂时不是特别清楚各个工具有没有什么决定性的优势，只不过 Verus 的 OS 相关工作似乎稍多一些，介绍中也声称是“for low-level systems code”
- （也不排除进一步了解后换用其他工具 / 其他目标项目）

Verus : Verified Rust for low-level systems code

```
use vstd::prelude::*;

verus! {

spec fn min(x: int, y: int) -> int {
    if x <= y {
        x
    } else {
        y
    }
}

fn main() {
    assert(min(10, 20) == 10);
    assert(min(-10, -20) == -20);
    assert(forall|i: int, j: int| min(i, j) <= i && min(i, j) <= j);
    assert(forall|i: int, j: int| min(i, j) == i || min(i, j) == j);
    assert(forall|i: int, j: int| min(i, j) == min(j, i));
}

} // verus!
```

```
verification results:: 1 verified, 0 errors
```

Verus : Verified Rust for low-level systems code

```
spec fn min(x: int, y: int) -> int {  
    if x + 1 < y {  
        x  
    } else {  
        y  
    }  
}
```

```
error: assertion failed  
--> guide/getting_started.rs:16:12  
|  
16 |     assert(forall|i: int, j: int| min(i, j) <= i && min(i, j) <= j);  
|           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ assertion failed  
  
error: aborting due to previous error
```

```
verification results:: 0 verified, 1 errors
```

大实验计划

- 目前初步搭建了开发环境，可以本地运行 Verus
- 后续先学习 Verus 教程，尝试编写简单的形式化验证
- 同时思考应当怎样为 rCore tutorial 编写形式化验证，哪些功能适合验证，可能遇到什么困难

