Chapter 5

王拓为 2018011917

一、实现

实现 spawn 系统调用方面：

实现了思路类似于将 fork 和 exec 整合在一起，但是区别在于 spawn 不必像 fork 一样复制父进程的地址空间。

实现 stride 调度方面：

实现了 sys_set_priority 的系统调用。

在 TaskControlBlock 中加入了 priority, stride 等新字段。

在 TaskManager 中实现了 stride 算法，遍历 ready_queue 找到 stride 最小的进程作为 fetch_task 的返回值。

二、问答

1. 实际情况不是轮到 p1 执行，因为 (p2.stride + 10) % 256 = 260 % 256 = 4 < p1.stride，出现了溢出从而导致 p2 的 stride 依然小于 p1。
2. 上述结论可以扩展为 stride_max - stride_min <= pass_max，在进程优先级 >= 2 的情况下，即有 pass_max = big_stride / 2。

   可以使用反证法证明上述结论。假设该命题不成立，由于一定需要两次以上的 pass 才能使 stride_max - stride_min >= pass_max，则一定存在 stride_max 对应进程的 stride 并非最小的情况下依然被选择了的情况，与 stride 调度算法不符。
3. 一种可能的代码实现如下：

```rust
use core::cmp::Ordering;

struct Stride(u64);

impl PartialOrd for Stride {
    fn paritial_cmp(&self, other: &Self) -> Option<Ordering> {
        Some(((self.0 - other.0) as isize).cmp(&0))
    }
}

impl PartialEq for Stride {
    fn eq(&self, other: &Self) -> bool {
        false
    }
}
```