ch5实验报告

设计分析与实现

spawn

sys_spawn部分我是先执行fork中复制进程的操作,再调用exec执行复制后的进程,最后加入调度队列里等待执行

Stride调度

TCB添加stride和priority字段

修改 `TaskManager.fetch` ,使得run_tasks循环每次都能拿到stride最小的task,拿到最小的task之后给这个task的stride加上 `GLOBAL_BIG_STRIDE / priority` ,再 `__switch` 进这个task

问答作业

stride 算法深入

stride 算法原理非常简单，但是有一个比较大的问题。例如两个 pass = 10 的进程，使用 8bit 无符号整形储存 stride， p1.stride = 255, p2.stride = 250，在 p2 执行一个时间片后，理论上下一次应该 p1 执行。

- 实际情况是轮到 p1 执行吗？为什么?

  实际情况不会轮到p1执行,因为p2溢出变成5,下一次选择到的进程还是p2

我们之前要求进程优先级 >= 2 其实就是为了解决这个问题。可以证明， ***在不考虑溢出的情况下*** *，在进程优先级全部 >= 2 的情况下，如果严格按照算法执行，那么 STRIDE_MAX – STRIDE_MIN <= BigStride / 2。*

- 为什么？尝试简单说明（不要求严格证明）。

  因为优先级>=2,而且算法是每次选择stride最小的进程

- 已知以上结论，**考虑溢出的情况下**，可以为 Stride 设计特别的比较器，让 BinaryHeap 的 pop 方法能返回真正最小的 Stride。补全下列代码中的 `partial_cmp` 函数，假设两个 Stride 永远不会相等。

```
use core::cmp::Ordering;
use std::cmp::Ordering::Less;
use std::ops::Sub;
```

```rust
const BIG_STRIDE :u64 = 1000000;

struct Stride(u64);

impl PartialOrd for Stride {
    fn partial_cmp(&self, other: &Self) -> Option<Ordering> {
        // 因为 STRIDE_MAX – STRIDE_MIN <= BigStride / 2
        // 所以两值相减大于等于 BIG_STRIDE / 2 则判定为溢出
        if self.0.wrapping_sub(other.0) >= BIG_STRIDE / 2
        {
            // 值的比较结果反过来
            if self.0 > other.0
            {
                return Some(Ordering::Less);
            }
            else
            {
                return Some(Ordering::Greater);
            }
        }

        // 正常比较
        if self.0 > other.0
        {
            return Some(Ordering::Greater);
        }
        else if self.0 < other.0
        {
            return Some(Ordering::Less);
        }
        return Some(Ordering::Equal);
    }
}

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

fn main() {
    let s1 = Stride(900000);
    let s2 = Stride(400000);

    println!("r = {}",s2 < s1);

    println!("Hello, world!");
}
```

荣誉准则

感谢群内大佬提示,这个test需要先完成spawn函数才能跑起来...,看到有人说我的pid是-1就反应过来了