任务介绍

8. `SCHED_FIFO` 调度以及相关 syscall： （新版本starry中已实现）
   - 什么是 `SCHED_FIFO`？
     给没上过 OS 课的同学说一下，在这种调度框架下，每个进程有一个优先级，而每个优先级都对应一个队列。在同队列内，每个进程轮流运行；在不同队列之间，优先级高的必定抢占优先级低的，除非高优先级的进程退出或者主动让出。
     以 Linux 为例，优先级最高是 99 最低是 1。假设四个进程 a,b,c,d 的优先级分别是 $99,99,5,1$。那么 a 和 b 轮流运行。除非有一天 a 和 b 都调用 `sys_sleep`，那此时轮到 c 运行，一个时钟周期后内核又回到 a 和 b。假设此时 a,b 都没醒，那么还是只有 c 可以运行，a,b,d 不能运行。假设此时 a 醒了 b 没醒，那么此时只有 a 可以运行，b,c,d 不能运行。
   - MediaServer 进程执行了什么操作?

     ```
     sched_get_priority_min(SCHED_FIFO) = 1
     sched_get_priority_max(SCHED_FIFO) = 99
     sched_setscheduler(7560, SCHED_FIFO, [99]) = 0
     ```

     它首先通过 `sched_get_priority_min` 和 `sched_get_priority_max` 获取了内核允许的最大和最小优先级（一般就设置为和 Linux 一致，从 1 到 99），然后通过 `sched_setscheduler` 设置了自己的优先级。我们需要实现这三个 syscall。

代码

crates/linux_syscall_api/src/syscall_task/task_syscall_id.rs

```
SCHED_GET_PRORITY_MAX = 146,
SCHED_GET_PRORITY_MIN = 147,
```

crates/linux_syscall_api/src/syscall_task/mod.rs

```
SCHED_GET_PRORITY_MAX => syscall_sched_getscheduler_max(args),
SCHED_GET_PRORITY_MIN => syscall_sched_getscheduler_min(args),
```

crates/axtask/src/lib.rs

```
pub use taskctx::{SchedPolicy, SchedStatus, MAX_RT_PRIO};
```

crates/taskctx/src/task.rs

```
pub const MAX_RT_PRIO: usize = 99;
```

```
#[derive(PartialEq, Eq, Clone, Copy)]
#[allow(non_camel_case_types)]
/// The policy of the scheduler
pub enum SchedPolicy {
    /// The default time-sharing scheduler
    SCHED_OTHER = 0,
    /// The first-in, first-out scheduler
    SCHED_FIFO = 1,
    /// The round-robin scheduler
    SCHED_RR = 2,
    /// The batch scheduler
    SCHED_BATCH = 3,
    /// The idle task scheduler
    SCHED_IDLE = 5,
```

```
    /// The deadline scheduler
    SCHED_DEADLINE = 6,
    /// Unknown scheduler
    SCHED_UNKNOWN,
}

impl From<usize> for SchedPolicy {
    #[inline]
    fn from(policy: usize) -> Self {
        match policy {
            0 => SchedPolicy::SCHED_OTHER,
            1 => SchedPolicy::SCHED_FIFO,
            2 => SchedPolicy::SCHED_RR,
            3 => SchedPolicy::SCHED_BATCH,
            5 => SchedPolicy::SCHED_IDLE,
            6 => SchedPolicy::SCHED_DEADLINE,
            _ => SchedPolicy::SCHED_UNKNOWN,
        }
    }
}

impl From<SchedPolicy> for isize {
    #[inline]
    fn from(policy: SchedPolicy) -> Self {
        match policy {
            SchedPolicy::SCHED_OTHER => 0,
            SchedPolicy::SCHED_FIFO => 1,
            SchedPolicy::SCHED_RR => 2,
            SchedPolicy::SCHED_BATCH => 3,
            SchedPolicy::SCHED_IDLE => 5,
            SchedPolicy::SCHED_DEADLINE => 6,
            SchedPolicy::SCHED_UNKNOWN => -1,
        }
    }
}
```

crates/linux_syscall_api/src/syscall_task/imp/schedule.rs

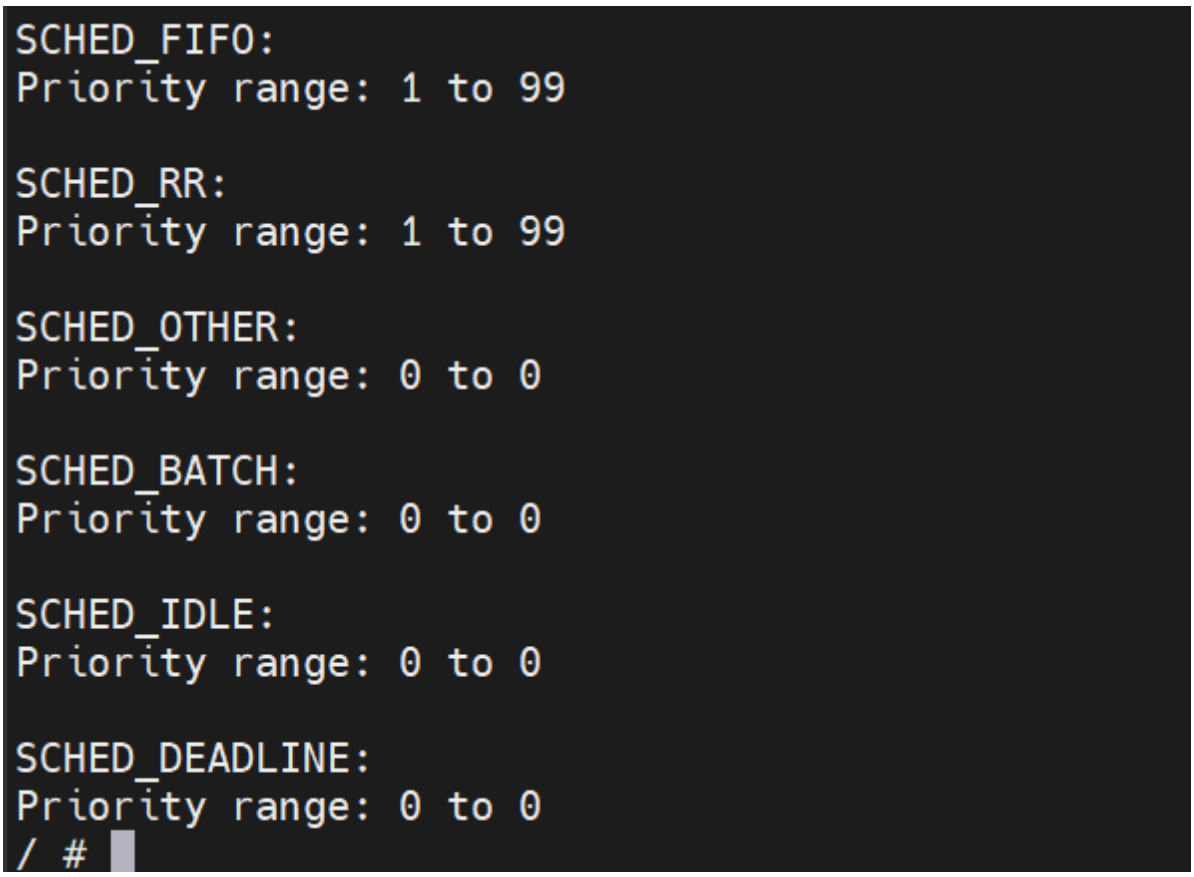
```
use axtask::{SchedPolicy, SchedStatus, MAX_RT_PRIO};

/// # Arguments
/// * `policy` - usize
pub fn syscall_sched_getscheduler_max(args: [usize; 6]) -> SyscallResult {
    let policy: usize = args[0];
    match SchedPolicy::from(policy) {
        SchedPolicy::SCHED_FIFO | SchedPolicy::SCHED_RR => Ok(MAX_RT_PRIO as
isize),
        SchedPolicy::SCHED_OTHER | SchedPolicy::SCHED_DEADLINE |
SchedPolicy::SCHED_BATCH | SchedPolicy::SCHED_IDLE => Ok(0),
        _ => Err(SyscallError::EINVAL),
    }
}
```

```
/// # Arguments
/// * `policy` - usize
pub fn syscall_sched_getscheduler_min(args: [usize; 6]) -> SyscallResult {
    let policy: usize = args[0];
    match SchedPolicy::from(policy) {
        SchedPolicy::SCHED_FIFO | SchedPolicy::SCHED_RR => Ok(1),
        SchedPolicy::SCHED_OTHER | SchedPolicy::SCHED_DEADLINE |
SchedPolicy::SCHED_BATCH | SchedPolicy::SCHED_IDLE => Ok(0),
        _ => Err(SyscallError::EINVAL),
    }
}
```

测试用例

```
#include <stdio.h>
#include <stdlib.h>
#include <linux/sched.h>
#include <errno.h>

void print_priority_range(int policy) {
    int max_priority, min_priority;
    max_priority = sched_get_priority_max(policy);
    if (max_priority == -1) {
        perror("sched_get_priority_max");
        return;
    }

    min_priority = sched_get_priority_min(policy);
    if (min_priority == -1) {
        perror("sched_get_priority_min");
        return;
    }
    printf("Priority range: %d to %d\n", min_priority, max_priority);
}

int main() {
    printf("SCHED_FIFO:\n");
    print_priority_range(SCHED_FIFO);

    printf("\nSCHED_RR:\n");
    print_priority_range(SCHED_RR);

    printf("\nSCHED_OTHER:\n");
    print_priority_range(SCHED_OTHER);

    printf("\nSCHED_BATCH:\n");
    print_priority_range(SCHED_BATCH);

    printf("\nSCHED_IDLE:\n");
    print_priority_range(SCHED_IDLE);

    printf("\nSCHED_DEADLINE:\n");
```

```
    print_priority_range(SCHED_DEADLINE);

    return 0;
}
```



说明：有的本地环境不支持部分调度策略，可以在/usr/include/linux/sched.h中手动添加，另外，linux中SCHED_OTHER为SCHED_NORMAL