Proj 3：进程调度与进程同步

> 现场验收：2025.5.25晚上18:00-21:00，实验楼103
> 报告提交截止日期：验收通过后、2025.5.28晚上22:00前

本实验用于理解和掌握进程调度方法和进程同步方法。

1. 轮转调度（30%）

xv6目前的轮转调度机制不完善，每个时钟tick都会导致进程切换，它在时钟中断处理过程中（trap函数）如果发现当前是某个进程正在运行（而不是内核本身），则调用yield切换回scheduler，从而触发下一次调度。虽然逻辑上类似轮转调度，但时间片过小（长度为1个tick），实际操作系统中不可能如此频繁调度。

本实验中，你需要扩展轮转调度的能力，实现以下系统调用，设置当前进程的时间片大小。

```
int set_quantum(int quantum)
```

含义是，将当前进程的时间片大小设置为quantum个ticks。进程在被创建时，默认的quantum值为1，即，如果不专门设置，则xv6相当于没有启用该扩充版的轮转调度机制。

验收要求

测试用例

```
$ rrtest
================================
Parent (pid=3)
Child (pid=4) created!
Child (pid=5) created!
Child (pid=6) created!
Child (pid=7) created!
================================
-5--6--6--6--6--6--6--6--7--7--7--7--7--7--7--4--5---6--6--6--6--6--6--6--7--7--7--7--7--7--7--7--7--4--4--5--5--6--6--6--6--6--6--6--6--6--7--7--7--7--7--7--7--4--5--5--5--6--6--6--6--6--7--7--7--7--7--7--7--5--6--6--6--4--5--5--4--5--4--4--5--5--4--5--4--4--5--5--4--4--5---4--5--5--4--4--5--5--5--4--4--5--5--4--5--4--4--5--5--4--5--4--4--5--5---
$

```

说明：

- 观察点：前2个进程打印的数量差不多、后2个进程数量差不多、后两个比前2个打印的数量多（具体多多少不要求，因为具有随机性）
- 4个子进程中，前2个时间片长度为默认长度，后2个时间片长度被设置为5倍的原始长度，所以可以看到，4、5进程连续打印的数量少于6、7进程打印的数量
- 例外情况，如果你的电脑性能非常好，可能需要在rrtest中修改busy_computing函数中fib函数的参数值，否则可能一轮就打印完了，看不到轮转情况，一个时间片都执行结束了，如果你做了这个改动，需要在验收时、实验报告中提及

实验提示

- 可以在pcb中定义quantum变量，并在合适的时机维护它，比如，创建时、用户执行set_quantum系统调用时
- 你还需要定义一个计时器记录进程运行了多少时间，建议命名为time_elapsed，然后和进程的quantum对比，发现超时后，触发切换操作，并把计时器清零
- 子进程是否继承父进程的quantum？没有要求，可以继承也可以不继承
- 子进程不应该继承父进程的time_elapsed。

实验报告要求

- 描述修改过程和遇到的问题、解决方法。
- xv6实现的并不是严格意义上的轮转调度，因为轮转调度严格意义上需要考虑进程进入就绪队列的时间，假如要求完成严格意义上的轮转调度，你觉得应该怎么做？

2. 优先级调度（30%）

在xv6的时间片轮转调度算法基础上，实现一个优先级调度算法，具有三个优先级：

- 优先级3是最低优先级，2次之，1最高
- 每个进程在创建时默认的优先级是2
- 每次需要调度时，总是优先级最高的进程被调度
- 具有同一个优先级的多个进程采用轮转调度
- 设计的时候注意考虑晚到的高优先级进程，例如它从阻塞进入就绪状态，也就是说，每次触发调度时，都需要检查是否有更高优先级的进程

为了控制实验的复杂度，不需要将进程划分多个队列，新的进程即使具有高优先级也不必考虑立即为它分配CPU，即不必在时间片未到时剥夺其他进程的CPU。（注：你需要在struct proc结构体中添加一个标识优先级的域）

你需要实现设置进程优先级的系统调用。

```
1    int set_priority(int pid, int prior)
```

其中pid指进程编号，prior是需要设置的优先级。

验收要求

测试程序的预期运行结果如下。

```
$ schedtest
================================
Parent (pid=4, prior=1)
Child (pid=5, prior=1) created!
Child (pid=6, prior=2) created!
Child (pid=7, prior=3) created!
Child (pid=8, prior=1) created!
Child (pid=9, prior=2) created!
Child (pid=10, prior=3) created!
Child (pid=11, prior=2) created!
================================
4 -5 -8 -5 -8 -5 -8 -5 -8 -4 -5 -4 -6 -9 -11 -6 -9 -11 -6 -9 -11 -6 -9 -11 -6
-4 -9 -4 -11 -11 -4 -7 -10 -7 -10 -7 -10 -7 -4 -10 -10 -
```

特征：先是优先级为1的两个child进程交替运行；然后是优先级为2的child；优先级为3的child；另外，父进程周期性出现。

实验提示

- 一种简单的做法是，每次切回scheduler时，先搜索全部进程，找到当前最高的优先级数字，下一个被调度的进程可以是当前进程在proc数组之后第一个具有最高优先级的进程（可以验证下这样做对不对，比如，如果当前进程已经是最高优先级，那么下一个调度的就是其他相同优先级的进程；如果当前进程不是最高级，那么下一个就是调度最高优先级的进程）
- 在排查错误时，关键地方插入输出语句，比如，优先级是否成功设置？最后调试通过后再注释掉。

实验报告要求

请在实验报告中描述以下内容：

- 修改过程和遇到的问题（走的弯路？）
- 父进程的优先级为1，为何有时优先级低的子进程会先于它执行？父进程似乎周期性出现在打印列表中，为什么？（你需要阅读schedtest.c，再次提醒，进程调度的对象处于什么状态？）
- set_priority系统调用会否和scheduler函数发生竞争条件？如何解决？

3. 信号量 （20%）

借助spinlock，为xv6添加信号量的支持。

定义一个信号量结构体`struct semaphore`，成员自定。

在内核里开辟一个包含100个信号量的空间，可以模仿ptable对struct proc的组织方式。以下说明假设信号量数组为s[100]。提供以下**系统调用**给用户程序：

```
int alloc_sem (int v);
```

创建一个初始值为v的信号量，返回信号量的下标。若返回值为-1，表明分配失败。分配失败的原因可能是初始值为负数或者系统中信号量资源不足（会有相应测试用例，建议第一次写的时候就考虑）。

```
int wait_sem(int i);
```

对信号量s[i]执行wait操作；成功返回1，出错返回-1。出错的原因可能是该信号量不存在，如i不在0~99之间，或者信号量尚未分配。由于信号量是独立的同步机制，**不要使用xv6中的sleep和wakeup**，否则会出奇怪的错误。由wait_sem导致阻塞的进程，只能由signal_sem唤醒。注意wait的实现有两个关键点，

- 哪部分是临界区，
- “放锁和阻塞”需要实现为原子操作，即不能被打断。

> 阻塞时，修改进程状态需要拿到ptable.lock锁、需要释放信号量的锁，否则其他进程无法访问信号量。那么，如何协调这两个锁？可以参考sleep的处理方式。此外，进程阻塞时，需要对其进行标记，以便将来强制删除信号量时能够终止阻塞在信号量上的进程，可以复用proc的chan变量。

```
int signal_sem(int i);
```

对信号量s[i]执行signal操作；成功返回1，出错返回-1。出错的原因同上。

```
int dealloc_sem(int i);
```

删除信号量s[i]，将s[i]标记为未分配。同时将所有等待s[i]的进程终止（终止进程时参考kill的实现）。成功返回1，失败返回-1。失败的情况包括下标不在合法范围内以及信号量未被分配。

实验提示

- 每个信号量用独立的spinlock保护，同时，用额外的spinlock保护信号量资源的分配，类似于ptable.lock保护所有struct proc结构体的state域。
- 每个信号量上睡眠的进程数量小于NPROC（严格小于NPROC，因为init进程不会等待信号量），可以定义包含NPROC个元素的地址数组，然后维护队头head和队尾tail。
- 可以模仿pinit函数，写seminit，初始化信号量表，但在main中调用时，建议在userinit之后调用（否则可能内存不足）。
- 对单个信号量的初始化应当在alloc_sem中完成。
- 不需要考虑进程死亡时未释放信号量的情况（实际操作系统中，未释放的信号量以虚拟文件的形式依然存在系统中）。

验收要求

测试用例: 运行semtest命令。

- 第1部分你应当看到两个success。
- 第2部分应当有以下规律：当信号量值为1时，只有一个进程能输出，所以输出不会杂乱；当信号量值为2时，有两个进程可以同时输出，所以输出会交叉；以此类推。
- 第3部分，你应当能够看到一行字“parent leaving”【前面没有“should not get here"提示】。

多次运行semtest。如果资源释放不正确，第二次会卡住。

实验报告要求

- 给出你的wait_sem和signal_sem的代码。
- 描述遇到的问题和解决方案。
- 如果dealloc_sem和sem_wait或者sem_signal同时执行，你的设计是如何避免并发错误的。

4. 实验报告和代码（20%）

在助教现场验收结束后，需要准备两份文件：

- 源代码：zip压缩包
  - 进入`proj3-revise`目录，执行`make clean`，然后在proj3-revise的父目录下执行`zip -r proj3-revise.zip proj3-revise`，此命令会创建一个源代码压缩文件`proj3-revise.zip`。提交此文件。
  - 请确保提交的代码压缩包内无其他文件（比如.o文件，或者vscode目录）。
- 实验报告：pdf，
  - 在实验报告中回答上面提到的问题；
  - 给出参考资料（网址）和工具（哪个大模型）；
  - 报告中需要添加姓名学号。不限定模板，尽量整洁、美观。