1、代码修改

- 在defs.h中增加BigStride字段定义；
- 在proc.h中给进程结构体添加stride相关字段；
- 在proc.c中完成了stride算法与时钟中断；
- 在syscall.c中实现了所需的若干系统调用；

2、思考题

1）请简要描述chapter3示例代码调度策略，尽可能多的指出该策略存在的缺点。

按内存顺序不断扫描进程池，如果发现一个可运行的进程，则运行它直到完成，并扫描下一个进程。

缺点：

- 不公平，可能会先产生的进程后被调度；
- 必须等待一个进程执行完毕后才会进行下一个进程，不能抢占式调度。

2）该调度策略在公平性上存在比较大的问题，请找到一个进程产生和结束的时间序列，使得在该调度算法下发生：先创建的进程后执行的现象。你需要给出类似下面例子的信息（有更详细的分析描述更好，但尽量精简）。同时指出该序列在你实现的 stride 调度算法下顺序是怎样的?

时间	0	1	2	3	4	5	6
运行进程	-	p1[0]	p2[1]	p4[2]	p3[0]	p2[1]	-
事件	p1[0]、p2[1]	p1结束	p3[0]，p4[2]产生	p4结束	p3结束	p2结束	-

方括号代表的是进程在的进程池的位置，产生顺序：p1、p2、p3、p4。第一次执行顺序: p1、p2、p4、p3。

我实现的stride算法下的顺序是p1、p2、p3、p4，因为它会选择stride最小的进程池编号最小的进程执行。

3）stride 算法深入

1. 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？

不是，是p2，stride溢出了：p2执行后，p2.stride = 250 + 10 = 4，4<255

2. 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，**如果不考虑溢出**，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（传达思想即可，不要求严格证明）。

因为每次stride增长的步长都不超过BigStride / 2，而每次都会选择stride最小的，因此任意两个进程的stride之差都不会超过BigStride / 2，最大值减最小值也就不会超过BigStride / 2了。

3. 已知以上结论，**在考虑溢出的情况下**，假设我们通过逐个比较得到 Stride 最小的进程，请设计一个合适的比较函数，用来正确比较两个 Stride 的真正大小：

```
typedef unsigned long long Stride_t;
const Stride_t BIGSTRIDE = 0xffffffffffffffffULL;
bool Less(Stride_t s1, Stride_t s2) {
    if (s1 < s2) {
        if (s2 - s1 < BIGSTRIDE / 2) { return true; }
        else { return false; }
    }
    else if (s1 > s2){
        if (s1 - s2 < BIGSTRIDE / 2) { return false; }
        else { return true; }
    }
    else { return false; }
}
```

函数如上所示。