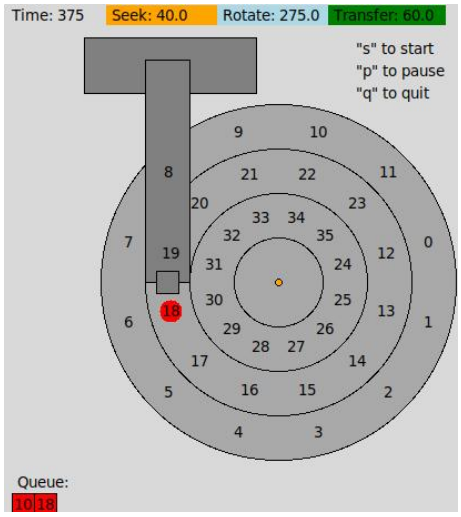
第三十七章

作业

本作业使用 disk.py 来帮助读者熟悉现代磁盘的工作原理。它有很多不同的选项，与大多数其他模拟不同，它有图形动画，可以准确显示磁盘运行时发生的情况。详情请参阅 README 文件。

为了计算传输效率，需要了解磁盘的一些细节。首先，转速默认设置为每时间单位 1 度。完成一圈公转需要 360 个时间单位。第二，转移开始和结束于扇区之间的中点。因此，要读取扇区 10，传输在 9 和 10 之间开始，在 10 和 11 之间结束。最后，在默认磁盘中，每个磁道有 12 个扇区，这意味着每个扇区占用 30 度的旋转空间。因此，读取一个扇区需要 30 个时间单位(考虑到我们默认的旋转速度)。最后，转移扇区需要 30 个时间单位。寻道时间为 40 个时间单位，默认的策略为 FIFO。-G 为图形模式，我们可以通过该参数查看具体的图形。

如果我们计算./dick.py -a 10,18 -G



逆时针旋转，最初的位置为扇区 6 的中间位置。访问扇区 10 所需的时间是 135 个时间单位(105 个旋转（3.5*30），30 个传输)。一旦这个请求完成，磁盘开始寻找扇区 18 所在的中间磁道，花费 40 个时间单位。然后磁盘旋转到扇区 18，传输 30 个时间单位，完成模拟。

要计算 18 的旋转延迟，首先计算磁盘从扇区 10 的访问结束到扇区 18 的访问开始需要多长时间。假设是零代价寻道，从模拟器中可以看到，外层轨道上的第 10 扇区与中间轨道上的第 22 扇区排列在一起，并且有 7 个扇区将 22 与 18 分开(23、12、13、14、15、16 和 17，因为磁盘是逆时针旋转的)。旋转 7 个扇区需要 210 个时间单位(每个扇区 30 个)。

然而，这个旋转的第一部分实际上是在寻找中间轨道，40 个时间单位。因此，访问扇区 18 的实际旋转延迟为 210 减去 40，即 170 个时间单位。

最终有：

```
Block:  10  Seek:  0  Rotate:105  Transfer: 30  Total: 135
Block:  18  Seek: 40  Rotate:170  Transfer: 30  Total: 240

TOTALS      Seek: 40  Rotate:275  Transfer: 60  Total: 375
```

```
Options:
  -h, --help            show this help message and exit
  -s SEED, --seed=SEED  Random seed
  -a ADDR, --addr=ADDR  Request list (comma-separated) [-1 -> use addrDesc]
  -A ADDRDESC, --addrDesc=ADDRDESC
                        Num requests, max request (-1->all), min request
  -S SEEKSPEED, --seekSpeed=SEEKSPEED
                        Speed of seek
  -R ROTATESPEED, --rotSpeed=ROTATESPEED
                        Speed of rotation
  -p POLICY, --policy=POLICY
                        Scheduling policy (FIFO, SSTF, SATF, BSATF)
  -w WINDOW, --schedWindow=WINDOW
                        Size of scheduling window (-1 -> all)
  -o SKEW, --skewOffset=SKEW
                        Amount of skew (in blocks)
  -z ZONING, --zoning=ZONING
                        Angles between blocks on outer,middle,inner tracks
  -G, --graphics        Turn on graphics
```

-a 提供待访问的数组
-S 修改寻道速率（第 2 题），默认寻道速度为 1，寻道路程为 40，寻道时间为 40.
-R 修改旋转速率（第 3 题），默认旋转速度为 1。
-p 提供调度算法，默认 FIFO，可以替换为 SATF,SSTF 等（第 4 题）
-o 引入磁道偏移（第 6 题）
-G 查看可视化内容

1. 计算以下几组请求的寻道、旋转和传输时间：-a 0，-a 6，-a 30，-a 7，30，8，最后

-a 10，11，12，13。

-a 0：

初始位于 6 的中间（最外圈），寻道时间为 0；旋转时间为 5.5*30；传输时间为 30。

-a 6：

初始位于 6 的中间（最外圈），寻道时间为 0；由于初始位置是 6 的中间，我们需要重新回到 6 的初始，即 56 之间，旋转时间为 11.5*30；传输时间为 30。

-a 30：

初始位于 6 的中间（最外圈），我们需要进入最内圈，寻道时间为 40*2；初始位置是 6 的中间，30 在紧挨着 6 的位置，由于有寻道时间，当我们到达最内圈的位置时，30 已经被旋转走过了，我们需要等待它下次到达，旋转时间为 345-40*2；传输时间为 30。

-a 7,30,8：

模拟了 FIFO 策略下的随机读取情况。

7:无需寻道；旋转 0.5*30 后，直接读取；传输时间为 30.

30：寻道进入最内圈，寻道时间 40*2；旋转的位置为 7-8 之间到 29-30 之间，时间为 300-40*2；传输时间为 30.

8：回到最外圈，寻道时间 40*2；旋转的位置为 30-31 之间到 7-8 之间，同一圈之间的最小距离为 30，由于寻道时间为 80，我们需要再等一圈，旋转时间为 390-40*2；传输时间为 30.

-a 10,11,12,13：

模拟了 FIFO 策略下的顺序读取情况。

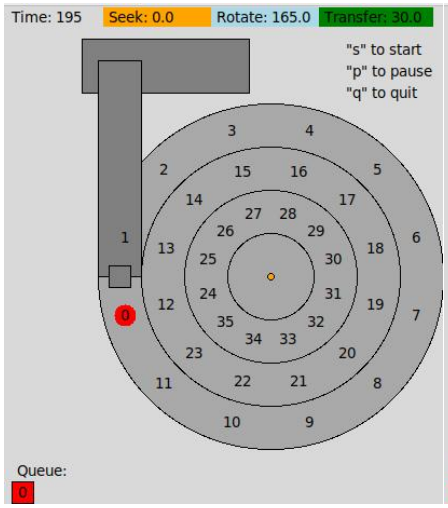
10：无需寻道；旋转 6 到 9-10 之间，旋转时间 30*3.5；传输时间为 30.

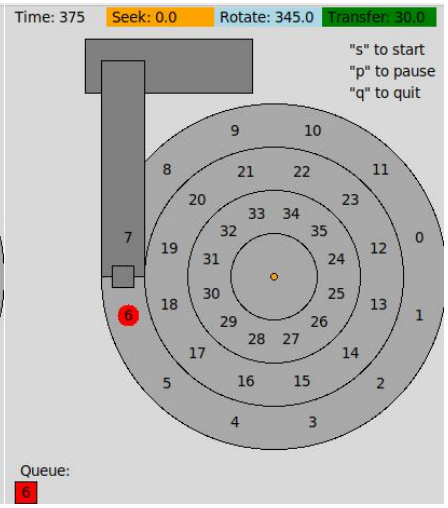
11：无需寻道；顺序读取无需旋转；传输时间为 30.

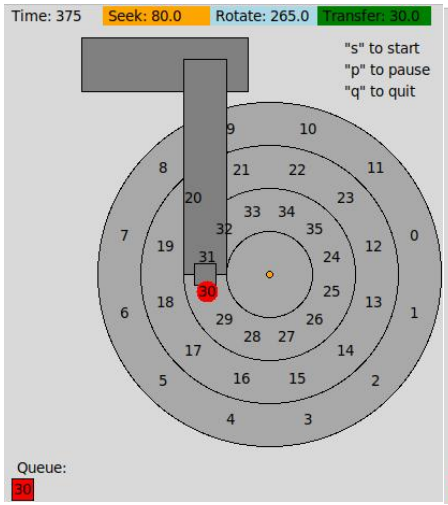
12：从最外圈到中间圈，寻道时间 40；由于从同圈位置 11-0 和 23-12 之间的时间为 0，我们需要再等一圈，旋转时间为 360-40；传输时间为 30.

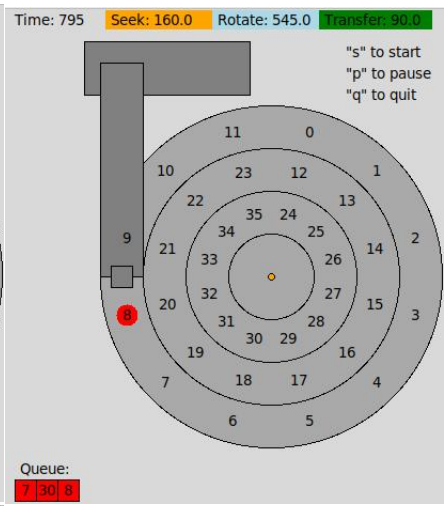
13：无需寻道；顺序读取无需旋转；传输时间为 30.

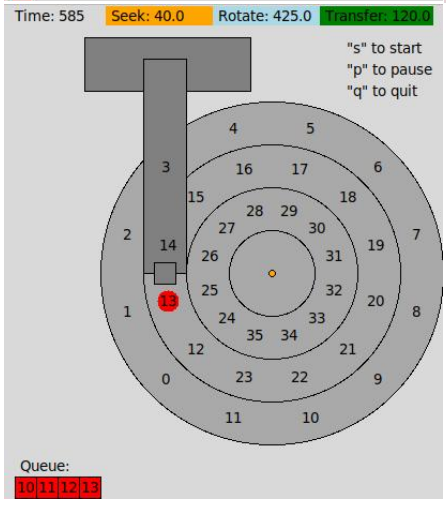
由于未设置磁道偏斜，哪怕是顺序读取，在跨道的时候仍然会有等待下一圈的问题，可将磁道偏斜设置为长度 40，或者为 2 块，可以优化旋转时间。











总结：

	寻道时间	旋转时间	传输时间	总时间
-a 0	0	165	30	195
-a 6	0	345	30	375
-a 30	80	265	30	375
-a 7,30,8 /7	0	15	30	45
-a 7,30,8 /30	80	220	30	330
-a 7,30,8 /8	80	310	30	360
总时间	/160	/545	/90	/795
-a 10,11,12,13/10	0	105	30	135
-a 10,11,12,13/11	0	0	30	30
-a 10,11,12,13/12	40	320	30	390
-a 10,11,12,13/13	0	0	30	30
总时间	/40	/425	/120	/585

2. 执行上述相同请求，但将寻道速率更改为不同值：-S 2，-S 4，-S 8，-S 10，-S 40，-S 0.1。时代如何变化？

对-a 0 和-a 6，寻道时间为 0，不会变化。对其余而言，可能发生寻道时间减小导致旋转时间的减小（无需多转一圈），可能发生寻道时间增加导致旋转时间增加，也可能不变。

-a 30：

-S 2：寻道时间变为 40；等待旋转时间 345-40；传输时间 30；总时间 375.

-S 4：寻道时间变为 20；等待旋转时间 345-20；传输时间 30；总时间 375.

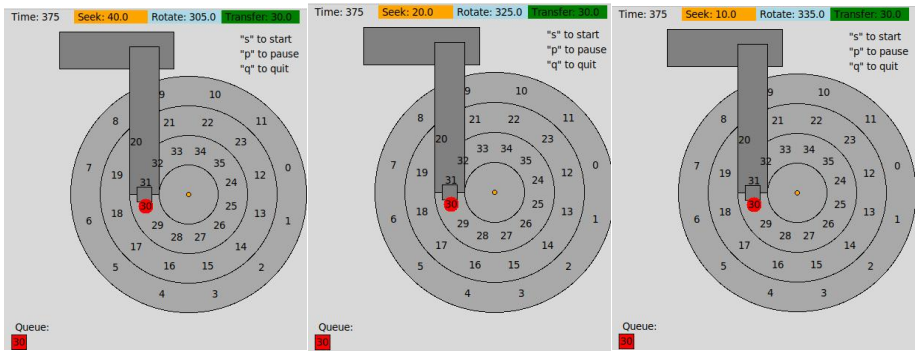
-S 8：寻道时间变为 10；等待旋转时间 345-10；传输时间 30；总时间 375.

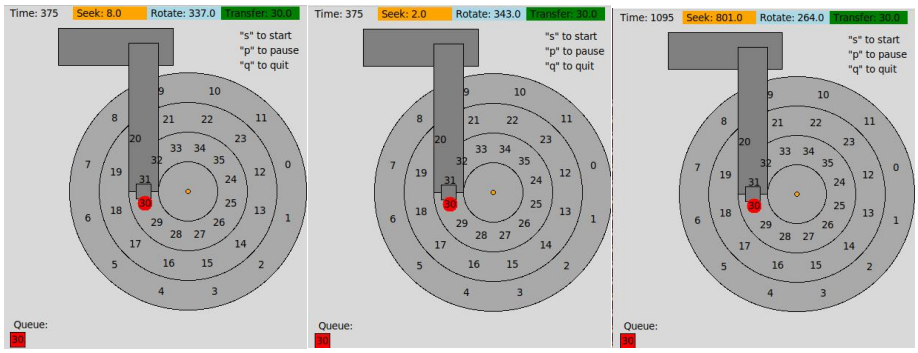
-S 10：寻道时间变为 8；等待旋转时间 345-8；传输时间 30；总时间 375.

-S 40：寻道时间变为 2；等待旋转时间 345-2；传输时间 30；总时间 375.

-S 0.1：寻道时间变为 800；等待旋转时间 720+345-800；传输时间 30；总时间 1095.

总时间不变是由于寻道时间减少的不显著而并未跳出旋转时间的“吞并”效应；寻道速率过小时会多转多圈。





-a 7,30,8：

-S 2：转换时的寻道时间变为 40+40；第一个旋转时间不变 15，第二个为 300-40，第三个为 390-40；传输时间 30+30+30；总时间 795.

-S 4：转换时的寻道时间变为 20+20；第一个旋转时间不变 15，第二个为 300-20，第三个为 30-20；传输时间 30+30+30；总时间 435.

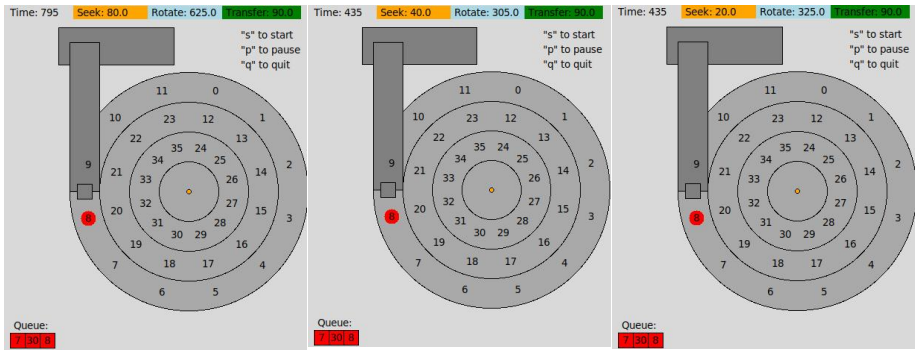
-S 8：转换时的寻道时间变为 10+10；第一个旋转时间不变 15，第二个为 300-10，第三个为 30-10；传输时间 30+30+30；总时间 435.

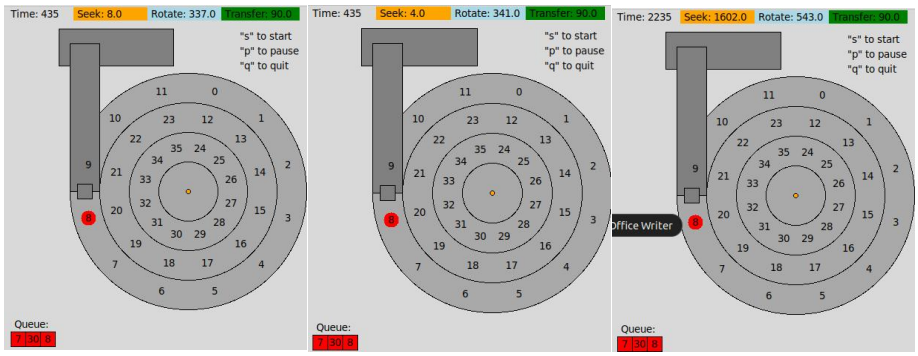
-S 10:转换时的寻道时间变为 8+8;第一个旋转时间不变 15,第二个为 300-8，第三个为 30-8；传输时间 30+30+30；总时间 435.

-S 40:转换时的寻道时间变为 2+2;第一个旋转时间不变 15,第二个为 300-2，第三个为 30-2；传输时间 30+30+30；总时间 435.

-S 0.1：转换时的寻道时间变为 800+800；第一个旋转时间不变 15，第二个为 720+300-800，第三个为 720+390-800；传输时间 30+30+30；总时间 2235.

总时间不变是由于寻道时间减少的不显著而并未跳出旋转时间的“吞并”效应；寻道速率过小时会多转多圈。





-a 10,11,12,13：

-S 2：转换时的寻道时间变为 20；第一个旋转时间不变 105，第二个不变为

0，第三个为 360-20，第四个不变为 0；传输时间 30+30+30+30；总时间 585.

-S 4：转换时的寻道时间变为 10；第一个旋转时间不变 105，第二个不变为 0，第三个为 360-10，第四个不变为 0；传输时间 30+30+30+30；总时间 585.

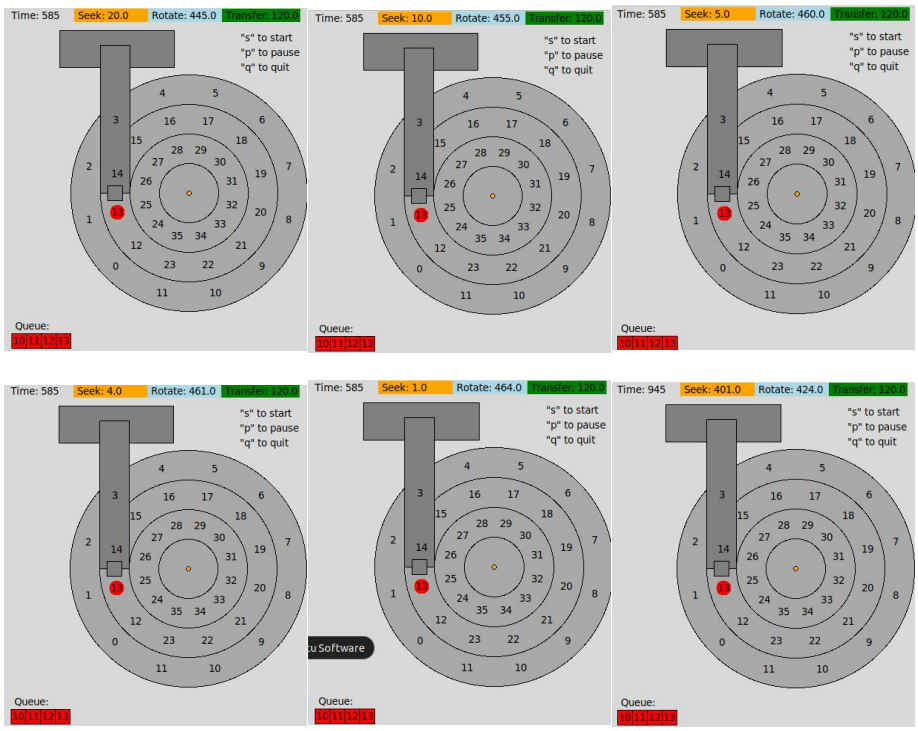
-S 8：转换时的寻道时间变为 5；第一个旋转时间不变 105，第二个不变为 0，第三个为 360-5，第四个不变为 0；传输时间 30+30+30+30；总时间 585.

-S 10：转换时的寻道时间变为 4；第一个旋转时间不变 105，第二个不变为 0，第三个为 360-4，第四个不变为 0；传输时间 30+30+30+30；总时间 585.

-S 40：转换时的寻道时间变为 1；第一个旋转时间不变 105，第二个不变为 0，第三个为 360-1，第四个不变为 0；传输时间 30+30+30+30；总时间 585.

-S 0.1：转换时的寻道时间变为 400；第一个旋转时间不变 105，第二个不变为 0，第三个为 360+360-400，第四个不变为 0；传输时间 30+30+30+30；总时间 945.

总时间不变是由于寻道时间减少的不显著而并未跳出旋转时间的“吞并”效应；寻道速率过小时会多转多圈。



3. 同样的请求，但改变旋转速率：-R 0.1，-R 0.5，-R 0.01。时间如何变化？

同条道路上，旋转时间线性增加；可能发生旋转时间的增加导致寻道时间小于旋转时间，无需再转一圈，减少总时间。

-a 0：

寻道时间为 0.

-R 0.1:旋转时间 1650，传输时间 300

-R 0.5:旋转时间 330，传输时间 60

-R 0.01:旋转时间 16500，传输时间 3000

-a 6：

寻道时间为 0.

-R 0.1:旋转时间 3450，传输时间 300

-R 0.5:旋转时间 690，传输时间 60

-R 0.01:旋转时间 34500，传输时间 3000

-a 30：

寻道时间为 80.

-R 0.1:旋转时间 3370，传输时间 300

-R 0.5:旋转时间 610，传输时间 60

-R 0.01:旋转时间 34420，传输时间 3000

-a 7,30,8：

先等待旋转 15 度访问 7 号扇区，然后移动到最内圈访问 30 号扇区，最终移动汇最外圈 8 号，寻道时间不变均为 160.

-R 0.1:旋转时间 3290，传输时间 900

-R 0.5:旋转时间 1250，传输时间 180

-R 0.01:旋转时间 34340，传输时间 9000

-a 10,11,12,13:

访问 10,11 号扇区后，移动至中间磁道，访问 12,13 号扇区，寻道时间均为 40。

-R 0.1:旋转时间 4610，传输时间 1200

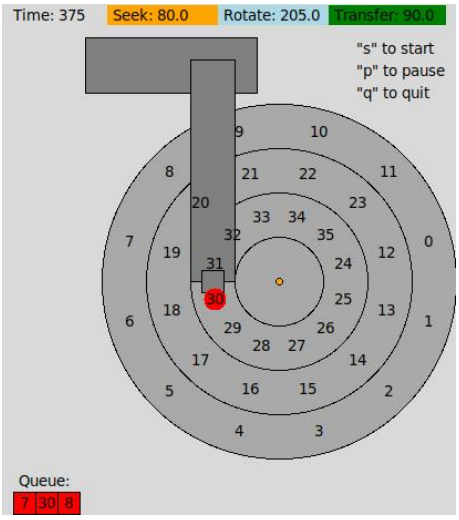
-R 0.5:旋转时间 890，传输时间 240

-R 0.01:旋转时间 46460，传输时间 12

	-R 0.1	-R 0.5	-R 0.01
-a 0	旋转时间增加	旋转时间增加	旋转时间增加
-a 6	旋转时间增加	旋转时间增加	旋转时间增加
-a 30	旋转时间增加	旋转时间增加	旋转时间增加
-a 7,30,8	少转一圈，但旋转时间增加	旋转时间增加	少转一圈，但旋转时间增加
-a 10,11,12,13	旋转时间增加	旋转时间增加	旋转时间增加

4. 你可能已经注意到，对于一些请求流，一些策略比 FIFO 更好。例如，对于请求流 -a 7，30，8，处理请求的顺序是什么？现在在相同的工作负载上运行最短寻道时间优先（SSTF）调度程序（-p SSTF）。每个请求服务需要多长时间（寻道、旋转、传输）？

对于-a 7,30,8，FIFO 的处理顺序为 7,30,8，会增加一次寻道和一圈旋转；SSTF、SCAN,FSACN,SPTF 的顺序均为 7,8,30.



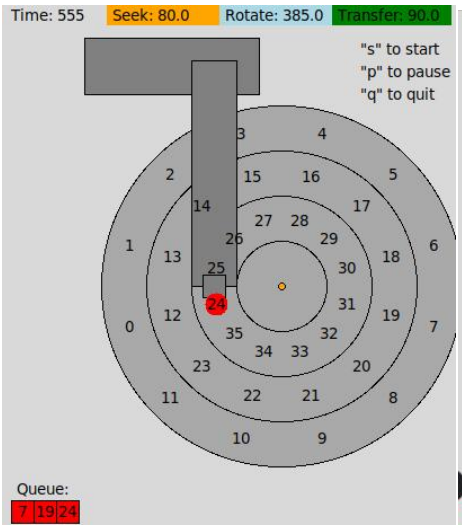
寻道时间为 80，旋转时间 205，传输时间 90.

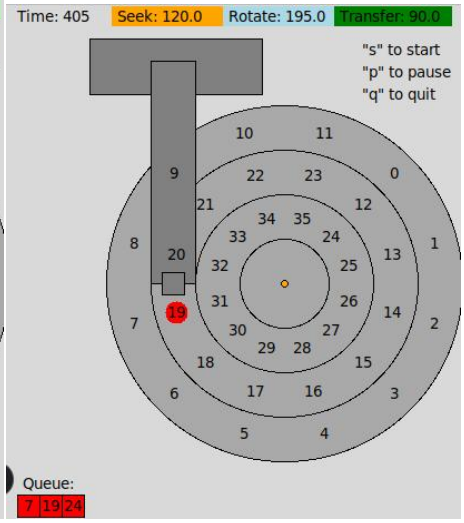
5. 现在做同样的事情，但使用最短的访问时间优先（SATF）调度程序（-p SATF）。它是否对-a 7，30.8 指定的一组请求有所不同？找到 SATF 明显优于 SSTF 的一组请求。出现显著差异的条件是什么？

对于-a 7,30,8，SATF 的顺序没有改变。

我们可以构造一个序列-a 7,19,24，对于 SSTF，访问 7 过后，它会先去访问中间圈的 19，但由于 7 和 19 在同一行，需要再等一圈，再去访问 24。总时间为 555.

对于 SATF，它会先去找 24，无需再等一圈，访问过后回来找 19，最终总时间为 405.





6. 你可能已经注意到，该磁盘没有特别好地处理请求流-a 10，11，12，13。这是为什么？你可以引入一个磁道偏斜来解决这个问题（-o skew，其中 skew 是一个非负整数）？考虑到默认寻道速率，偏斜应该是多少，才能尽量减少这一组请求的总时间？对于不同的寻道速率（例如，-S 2，-S 4）呢？一般来说，考虑到寻道速率和扇区布局信息，你能否写出一个公式来计算偏斜？

12,13 号扇区在与 10,11 号扇区不同的磁道上，但他们在圆上是连续的，由于寻道时间的存在，每次访问都需要等待一圈。对于默认寻道速率，取大于等于 40 的角度偏移就可以解决这个问题，如果按扇区，可以设置为 2 个扇区。

对于不同的寻道速率 v，磁道间距离 s，旋转速率 p，则寻道时转过的角度为 p*s/v。若磁道有 n 个扇区，一个扇区角度为 360/n，设偏移为 x，有 p*s/v<=(360/n)*x。

故 x>pns/360v。

第三十八章

作业

本节引入 raid.py，这是一个简单的 RAID 模拟器，你可以使用它来增强你对 RAID 系统工作方式的了解。详情请参阅 README 文件。

```
Options:
  -h, --help            show this help message and exit
  -s SEED, --seed=SEED  the random seed
  -D NUMDISKS, --numDisks=NUMDISKS
                        number of disks in RAID
  -C CHUNKSIZE, --chunkSize=CHUNKSIZE
                        chunk size of the RAID
  -n NUMREQUESTS, --numRequests=NUMREQUESTS
                        number of requests to simulate
  -S SIZE, --reqSize=SIZE
                        size of requests
  -W WORKLOAD, --workload=WORKLOAD
                        either "rand" or "seq" workloads
  -w WRITEFRAC, --writeFrac=WRITEFRAC
                        write fraction (100->all writes, 0->all reads)
  -R RANGE, --randRange=RANGE
                        range of requests (when using "rand" workload)
  -L LEVEL, --level=LEVEL
                        RAID level (0, 1, 4, 5)
  -5 RAID5TYPE, --raid5=RAID5TYPE
                        RAID-5 left-symmetric "LS" or left-asym "LA"
  -r, --reverse         instead of showing logical ops, show physical
  -t, --timing          use timing mode, instead of mapping mode
  -c, --compute         compute answers for me
```

1. 使用模拟器执行一些基本的 RAID 映射测试。运行不同的级别（0、1、4、5），看看你是否可以找出一组请求的映射。对于 RAID-5，看看你是否可以找出左对称（left-symmetric）和左不对称（left-asymmetric）布局之间的区别。使用一些不同的随机种子，产生不同于上面的问题。

给出与书上例子条件相同的请求映射。

RAID0：

表 38.1　　RAID-0：简单条带化

磁盘 0	磁盘 1	磁盘 2	磁盘 3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

```
guorulling@guorulling-virtual-machine:~/os/hw4$ python3 ./raid.py -D 4 -n 5 -L 0 -R 16
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 5
ARG reqSize 4k
ARG workload rand
ARG writeFrac 0
ARG randRange 16
ARG level 0
ARG raid5 LS
ARG reverse False
ARG timing False

13 1
LOGICAL READ from addr:13 size:4096
  Physical reads/writes?

6 1
LOGICAL READ from addr:6 size:4096
  Physical reads/writes?

8 1
LOGICAL READ from addr:8 size:4096
  Physical reads/writes?

12 1
LOGICAL READ from addr:12 size:4096
  Physical reads/writes?

7 1
LOGICAL READ from addr:7 size:4096
  Physical reads/writes?
```

答案：
磁盘号=地址%磁盘数，偏移=地址/磁盘数

```
13 1
LOGICAL READ from addr:13 size:4096
  read  [disk 1, offset 3]

6 1
LOGICAL READ from addr:6 size:4096
  read  [disk 2, offset 1]

8 1
LOGICAL READ from addr:8 size:4096
  read  [disk 0, offset 2]

12 1
LOGICAL READ from addr:12 size:4096
  read  [disk 0, offset 3]

7 1
LOGICAL READ from addr:7 size:4096
  read  [disk 3, offset 1]
```

RAID1:

表 38.3　　简单 RAID-1：镜像

磁盘 0	磁盘 1	磁盘 2	磁盘 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -D 4 -n 5 -L 1 -R 16
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 5
ARG reqSize 4k
ARG workload rand
ARG writeFrac 0
ARG randRange 16
ARG level 1
ARG raid5 LS
ARG reverse False
ARG timing False

13 1
LOGICAL READ from addr:13 size:4096
  Physical reads/writes?

6 1
LOGICAL READ from addr:6 size:4096
  Physical reads/writes?

8 1
LOGICAL READ from addr:8 size:4096
  Physical reads/writes?

12 1
LOGICAL READ from addr:12 size:4096
  Physical reads/writes?

7 1
LOGICAL READ from addr:7 size:4096
  Physical reads/writes?
```

答案：
磁盘号=2*地址%磁盘数，偏移=2*地址/磁盘数

```
13 1
LOGICAL READ from addr:13 size:4096
  read  [disk 2, offset 6]

6 1
LOGICAL READ from addr:6 size:4096
  read  [disk 1, offset 3]

8 1
LOGICAL READ from addr:8 size:4096
  read  [disk 0, offset 4]

12 1
LOGICAL READ from addr:12 size:4096
  read  [disk 0, offset 6]

7 1
LOGICAL READ from addr:7 size:4096
  read  [disk 3, offset 3]
```

RAID4:

表 38.4　　具有奇偶校验的 RAID-4

磁盘 0	磁盘 1	磁盘 2	磁盘 3	磁盘 4
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -D 5 -n 5 -L 4 -R 16
ARG blockSize 4096
ARG seed 0
ARG numDisks 5
ARG chunkSize 4k
ARG numRequests 5
ARG reqSize 4k
ARG workload rand
ARG writeFrac 0
ARG randRange 16
ARG level 4
ARG raid5 LS
ARG reverse False
ARG timing False

13 1
LOGICAL READ from addr:13 size:4096
  Physical reads/writes?

6 1
LOGICAL READ from addr:6 size:4096
  Physical reads/writes?

8 1
LOGICAL READ from addr:8 size:4096
  Physical reads/writes?

12 1
LOGICAL READ from addr:12 size:4096
  Physical reads/writes?

7 1
LOGICAL READ from addr:7 size:4096
  Physical reads/writes?
```

答案：

磁盘号=地址%(磁盘数-1)，偏移=地址/(磁盘数-1)

```
13 1
LOGICAL READ from addr:13 size:4096
  read  [disk 1, offset 3]
6 1
LOGICAL READ from addr:6 size:4096
  read  [disk 2, offset 1]
8 1
LOGICAL READ from addr:8 size:4096
  read  [disk 0, offset 2]
12 1
LOGICAL READ from addr:12 size:4096
  read  [disk 0, offset 3]
7 1
LOGICAL READ from addr:7 size:4096
  read  [disk 3, offset 1]
```

RAID5：

表 38.9　具有旋转奇偶校验的 RAID-5

磁盘 0	磁盘 1	磁盘 2	磁盘 3	磁盘 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -D 5 -n 20 -L 5 -R 20 -5 LS -W seq -c
```

所有情况全部标出，可以得出如下结果。

左对称：

DISK0	DISK1	DISK2	DISK3	DISK4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

左不对称：

DISK0	DISK1	DISK2	DISK3	DISK4
0	1	2	3	P0
4	5	6	P1	7
8	9	P2	10	11
12	P3	13	14	15
P4	16	17	18	19

左对称中，块按照顺序分布，非左对称中，块会跳过校验块。

2. 与第一个问题一样，但这次使用-C 来改变大块的大小。大块的大小如何改变映射？

RAID0:

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -D 4 -n 16 -L 0 -R 16  -W seq -c -C 8192
```

同样，所有情况全部列出，可以得到：

DISK0	DISK1	DISK2	DISK3
0	2	4	6
1	3	5	7
8	10	12	14
9	11	13	15

RAID1:

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -D 4 -n 8 -L 1 -R 8  -W seq -c -C 8192
```

所有情况全部列出，可以得到：

DISK0	DISK1	DISK2	DISK3
0	0	2	2
1	1	3	3
4	4	6	6
5	5	7	7

RAID4:

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -D 4 -n 20 -L 4 -R 20  -W seq -c -C 8192
```

所有情况全部列出，可以得到：

DISK0	DISK1	DISK2	DISK3
0	2	4	P
1	3	5	P
6	8	10	P
7	9	11	P

RAID4:

```
guorulling@guorulling-virtual-machine:~/os/hw4$ python3 ./raid.py -D 4 -n 20 -L 5 -R 20  -W seq -c -C 8192 -5 LA
```

所有情况全部列出，可以得到：

DISK0	DISK1	DISK2	DISK3
0	2	4	P
1	3	5	P
6	8	P	10
7	9	P	11
12	P	14	16
13	P	15	17

只是改变了块的大小，映射规则没有改变。

3. 执行上述测试，但使用-r 标志来反转每个问题的性质。

给出与书上例子条件相同的请求映射。加-r 翻转后，问题是给出磁盘号和磁盘偏移，计算地址。

RAID0：地址=磁盘数*偏移+磁盘号。

RAID1：地址=(磁盘数*偏移+磁盘号)/2

RAID4：地址=(磁盘数-1)*偏移+磁盘号(可能有 1 的偏差，因为不确定条带前面是否已有校验块)

RAID5：可以根据布局直接找到地址。

RAID0：

表 38.1　　RAID-0：简单条带化

磁盘 0	磁盘 1	磁盘 2	磁盘 3
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

```
guorulling@guorulling-virtual-machine:~/os/hw4$ python3 ./raid.py -D 4 -n 5 -L 0 -R 16 -r -c
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 5
ARG reqSize 4k
ARG workload rand
ARG writeFrac 0
ARG randRange 16
ARG level 0
ARG raid5 LS
ARG reverse True
ARG timing False

13 1
LOGICAL READ from addr:13 size:4096
  read  [disk 1, offset 3]

6 1
LOGICAL READ from addr:6 size:4096
  read  [disk 2, offset 1]

8 1
LOGICAL READ from addr:8 size:4096
  read  [disk 0, offset 2]

12 1
LOGICAL READ from addr:12 size:4096
  read  [disk 0, offset 3]

7 1
LOGICAL READ from addr:7 size:4096
  read  [disk 3, offset 1]
```

RAID1:

表 38.3 简单 RAID-1：镜像

磁盘 0	磁盘 1	磁盘 2	磁盘 3
0	0	1	1
2	2	3	3
4	4	5	5
6	6	7	7

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -D 4 -n 5 -L 1 -R 16 -r -c
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 5
ARG reqSize 4k
ARG workload rand
ARG writeFrac 0
ARG randRange 16
ARG level 1
ARG raid5 LS
ARG reverse True
ARG timing False

13 1
LOGICAL READ from addr:13 size:4096
  read  [disk 2, offset 6]

6 1
LOGICAL READ from addr:6 size:4096
  read  [disk 1, offset 3]

8 1
LOGICAL READ from addr:8 size:4096
  read  [disk 0, offset 4]

12 1
LOGICAL READ from addr:12 size:4096
  read  [disk 0, offset 6]

7 1
LOGICAL READ from addr:7 size:4096
  read  [disk 3, offset 3]
```

RAID4:

表 38.4 具有奇偶校验的 RAID-4

磁盘 0	磁盘 1	磁盘 2	磁盘 3	磁盘 4
0	1	2	3	P0
4	5	6	7	P1
8	9	10	11	P2
12	13	14	15	P3

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -D 4 -n 5 -L 4 -R 16 -r -c
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 5
ARG reqSize 4k
ARG workload rand
ARG writeFrac 0
ARG randRange 16
ARG level 4
ARG raid5 LS
ARG reverse True
ARG timing False

13 1
LOGICAL READ from addr:13 size:4096
  read  [disk 1, offset 4]
6 1
LOGICAL READ from addr:6 size:4096
  read  [disk 0, offset 2]
8 1
LOGICAL READ from addr:8 size:4096
  read  [disk 2, offset 2]
12 1
LOGICAL READ from addr:12 size:4096
  read  [disk 0, offset 4]
7 1
LOGICAL READ from addr:7 size:4096
  read  [disk 1, offset 2]
```

RAID5：

表 38.9　具有旋转奇偶校验的 RAID-5

磁盘 0	磁盘 1	磁盘 2	磁盘 3	磁盘 4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -D 5 -n 20 -L 5 -R 20 -5 LS -W seq -
c
```

所有情况全部标出，可以得出如下结果。

左对称：

DISK0	DISK1	DISK2	DISK3	DISK4
0	1	2	3	P0
5	6	7	P1	4
10	11	P2	8	9
15	P3	12	13	14
P4	16	17	18	19

左不对称：

DISK0	DISK1	DISK2	DISK3	DISK4
0	1	2	3	P0
4	5	6	P1	7
8	9	P2	10	11
12	P3	13	14	15
P4	16	17	18	19

左对称中，块按照顺序分布，非左对称中，块会跳过校验块。

4. 现在使用反转标志，但用-S 标志增加每个请求的大小。尝试指定 8KB、12KB 和 16KB 的大小，同时改变 RAID 级别。当请求的大小增加时，底层 I/O 模式会发生什么？请务必在顺序工作负载上尝试此操作（-W sequential）。对于什么请求大小，RAID-4 和 RAID-5 的 I/O 效率更高？

-S 8k:

RAID 0:

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -n 5 -L 0 -R 20 -r -S 8K -W seq
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 5
ARG reqSize 8K
ARG workload seq
ARG writeFrac 0
ARG randRange 20
ARG level 0
ARG raid5 LS
ARG reverse True
ARG timing False

0 2
LOGICAL OPERATION is ?
  read  [disk 0, offset 0]
  read  [disk 1, offset 0]

2 2
LOGICAL OPERATION is ?
  read  [disk 2, offset 0]
  read  [disk 3, offset 0]

4 2
LOGICAL OPERATION is ?
  read  [disk 0, offset 1]
  read  [disk 1, offset 1]

6 2
LOGICAL OPERATION is ?
  read  [disk 2, offset 1]
  read  [disk 3, offset 1]

8 2
LOGICAL OPERATION is ?
  read  [disk 0, offset 2]
  read  [disk 1, offset 2]
```

读操作需要两次完成。

RAID1：

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -n 5 -L 1 -R 20 -r -S 8K -W seq
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 5
ARG reqSize 8K
ARG workload seq
ARG writeFrac 0
ARG randRange 20
ARG level 1
ARG raid5 LS
ARG reverse True
ARG timing False

0 2
LOGICAL OPERATION is ?
  read  [disk 0, offset 0]
  read  [disk 2, offset 0]

2 2
LOGICAL OPERATION is ?
  read  [disk 1, offset 1]
  read  [disk 3, offset 1]

4 2
LOGICAL OPERATION is ?
  read  [disk 0, offset 2]
  read  [disk 2, offset 2]

6 2
LOGICAL OPERATION is ?
  read  [disk 1, offset 3]
  read  [disk 3, offset 3]

8 2
LOGICAL OPERATION is ?
  read  [disk 0, offset 4]
  read  [disk 2, offset 4]
```

需要两次读。

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -n 5 -L 1 -R 20 -r -S 8K -W seq -w 100
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 5
ARG reqSize 8K
ARG workload seq
ARG writeFrac 100
ARG randRange 20
ARG level 1
ARG raid5 LS
ARG reverse True
ARG timing False

0 2
LOGICAL OPERATION is ?
  write [disk 0, offset 0]    write [disk 1, offset 0]
  write [disk 2, offset 0]    write [disk 3, offset 0]

2 2
LOGICAL OPERATION is ?
  write [disk 0, offset 1]    write [disk 1, offset 1]
  write [disk 2, offset 1]    write [disk 3, offset 1]

4 2
LOGICAL OPERATION is ?
  write [disk 0, offset 2]    write [disk 1, offset 2]
  write [disk 2, offset 2]    write [disk 3, offset 2]

6 2
LOGICAL OPERATION is ?
  write [disk 0, offset 3]    write [disk 1, offset 3]
  write [disk 2, offset 3]    write [disk 3, offset 3]

8 2
LOGICAL OPERATION is ?
  write [disk 0, offset 4]    write [disk 1, offset 4]
  write [disk 2, offset 4]    write [disk 3, offset 4]
```

每次写入两个块，需要四次写。

RAID 4：

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -n 5 -L 4 -R 20 -r -S 8K -W seq -w 100
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 5
ARG reqSize 8K
ARG workload seq
ARG writeFrac 100
ARG randRange 20
ARG level 4
ARG raid5 LS
ARG reverse True
ARG timing False

0 2
LOGICAL OPERATION is ?
  read  [disk 2, offset 0]
  write [disk 0, offset 0]    write [disk 1, offset 0]    write [disk 3, offset 0]

2 2
LOGICAL OPERATION is ?
  read  [disk 2, offset 0]    read  [disk 3, offset 0]
  write [disk 2, offset 0]    write [disk 3, offset 0]
  read  [disk 0, offset 1]    read  [disk 3, offset 1]
  write [disk 0, offset 1]    write [disk 3, offset 1]

4 2
LOGICAL OPERATION is ?
  read  [disk 0, offset 1]
  write [disk 1, offset 1]    write [disk 2, offset 1]    write [disk 3, offset 1]

6 2
LOGICAL OPERATION is ?
  read  [disk 2, offset 2]
  write [disk 0, offset 2]    write [disk 1, offset 2]    write [disk 3, offset 2]

8 2
LOGICAL OPERATION is ?
  read  [disk 2, offset 2]    read  [disk 3, offset 2]
  write [disk 2, offset 2]    write [disk 3, offset 2]
  read  [disk 0, offset 3]    read  [disk 3, offset 3]
  write [disk 0, offset 3]    write [disk 3, offset 3]
```

写入时，需先读地址对应块和校验块，判断是否需要改变，再写入块和校验块。如果块的偏移量不同，重复两次上述操作（减法奇偶校验）；如果块的偏移量相同，由于共用同一个校验位，只需要读另一个块，写入这两个块和校验位即可（加法奇偶校验）。

RAID 5：

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -n 5 -L 5 -R 20 -r -S 8K -W seq -w 100
ARG blockSize 4096
ARG seed 0
ARG numDisks 4
ARG chunkSize 4k
ARG numRequests 5
ARG reqSize 8K
ARG workload seq
ARG writeFrac 100
ARG randRange 20
ARG level 5
ARG raid5 LS
ARG reverse True
ARG timing False

0 2
LOGICAL OPERATION is ?
  read  [disk 2, offset 0]
  write [disk 0, offset 0]    write [disk 1, offset 0]    write [disk 3, offset 0]

2 2
LOGICAL OPERATION is ?
  read  [disk 2, offset 0]    read  [disk 3, offset 0]
  write [disk 2, offset 0]    write [disk 3, offset 0]
  read  [disk 3, offset 1]    read  [disk 2, offset 1]
  write [disk 3, offset 1]    write [disk 2, offset 1]

4 2
LOGICAL OPERATION is ?
  read  [disk 3, offset 1]
  write [disk 0, offset 1]    write [disk 1, offset 1]    write [disk 2, offset 1]

6 2
LOGICAL OPERATION is ?
  read  [disk 0, offset 2]
  write [disk 2, offset 2]    write [disk 3, offset 2]    write [disk 1, offset 2]

8 2
LOGICAL OPERATION is ?
  read  [disk 0, offset 2]    read  [disk 1, offset 2]
  write [disk 0, offset 2]    write [disk 1, offset 2]
  read  [disk 1, offset 3]    read  [disk 0, offset 3]
  write [disk 1, offset 3]    write [disk 0, offset 3]
```

与 RAID 4 基本相同。

-S 12K：

对于 RAID0 与 RAID1，只是需要多对一个块进行处理。因此与 8K 类似，RAID0 需要 3 次 I/O 完成请求，RAID1 需要 3 次读操作完成读请求，6 次写操作完成写请求。

RAID4 的随机读和顺序读也与 8K 类似，需要 3 次读完成。随机写有所不同，如果在同一个条带上进行写，那只需要三个块进行异或，然后一次将包括校验块在内的四个块全部写入，故需要 4 次写。如果有 2 个块在同一条带上，那么这两个块的写入可以采取加法奇偶校验(4 次写操作)，另一个单独在其他条带的块不论采用哪种方式，都需要 4 次写操作，故一共 8 次写操作。

RAID4 顺序写时，写操作数明显减少了，因为每次请求都是对一个条带上的三个块进行写请求，可以采用全条带写入，即直接将三个块异或，然后全部和奇偶校验块一起写入。

RAID5 的情况与 RAID4 基本相同。

-S 16K：

RAID0 与 RAID1 的情况没有发生变化，只是需要多处理一个块。RAID0 完成请求需要 4 次 I/O。RAID1 完成读需要 4 次读操作，完成写需要 8 次写操作。

RAID4 的随机读和顺序读 4 次读操作完成。随机写时，有以下 2 种情况：

情况 1：一个请求分布在两个条带上，两个条带上的块数分别为 3,1，

情况 2：一个请求分布在两个条带上，两个条带上的块数分别为 2,2。

考虑 3,1 的情况，3 个块在同一个条带上可以使用全条带写入(4 次写)，剩下一个块 4 次写单独处理，共 8 次写。另一种 2,2 的情况，每一个条带上采用加法奇偶校验，各需要 4 次写，故也需要 8 次写。

对于顺序写，情况是与随机写相同的，因为请求大小比 1 个条带的数据块要多。因此顺序写也是以上的两种模式。

RAID5 与 RAID4 基本相同。

对于 4 个磁盘的情况下，请求块数越接近(小于等于)一个条带的块数，RAID4 和 RAID5 的写性能更好。即 RAID4/5 更适合接近一个条带块数的顺序写入。因为在这种情况下，加法奇偶校验可以比减法奇偶校验使用更少的写操作完成请求，最好的情况下，可以使用全条带写入直接完成写入，而不需要读取数据块。

5. 使用模拟器的定时模式（-t）来估计 100 次随机读取到 RAID 的性能，同时改变 RAID 级别，使用 4 个磁盘。

表 38.10 RAID 容量、可靠性和性能

	RAID-0	RAID-1	RAID-4	RAID-5
容量	N	$N/2$	$N-1$	$N-1$
可靠性	0	1（肯定）		
		$N/2$（如果走运）		
吞吐量				
顺序读	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
顺序写	$N \cdot S$	$(N/2) \cdot S$	$(N-1) \cdot S$	$(N-1) \cdot S$
随机读	$N \cdot R$	$N \cdot R$	$(N-1) \cdot R$	$N \cdot R$
随机写	$N \cdot R$	$(N/2) \cdot R$	$1/2 \cdot R$	$N/4 \cdot R$

RAID 0:

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -L 0 -t -n 100 -c
```

```
disk:0  busy: 100.00  I/Os:    28 (sequential:0 nearly:1 random:27)
disk:1  busy:  93.91  I/Os:    29 (sequential:0 nearly:6 random:23)
disk:2  busy:  87.92  I/Os:    24 (sequential:0 nearly:0 random:24)
disk:3  busy:  65.94  I/Os:    19 (sequential:0 nearly:1 random:18)

STAT totalTime 275.69999999999993
```

RAID 1:

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -L 1 -t -n 100 -c

 disk:0  busy: 100.00  I/Os:    28 (sequential:0 nearly:1 random:27)
 disk:1  busy:  86.98  I/Os:    24 (sequential:0 nearly:0 random:24)
 disk:2  busy:  97.52  I/Os:    29 (sequential:0 nearly:3 random:26)
 disk:3  busy:  65.23  I/Os:    19 (sequential:0 nearly:1 random:18)

 STAT totalTime 278.7
```

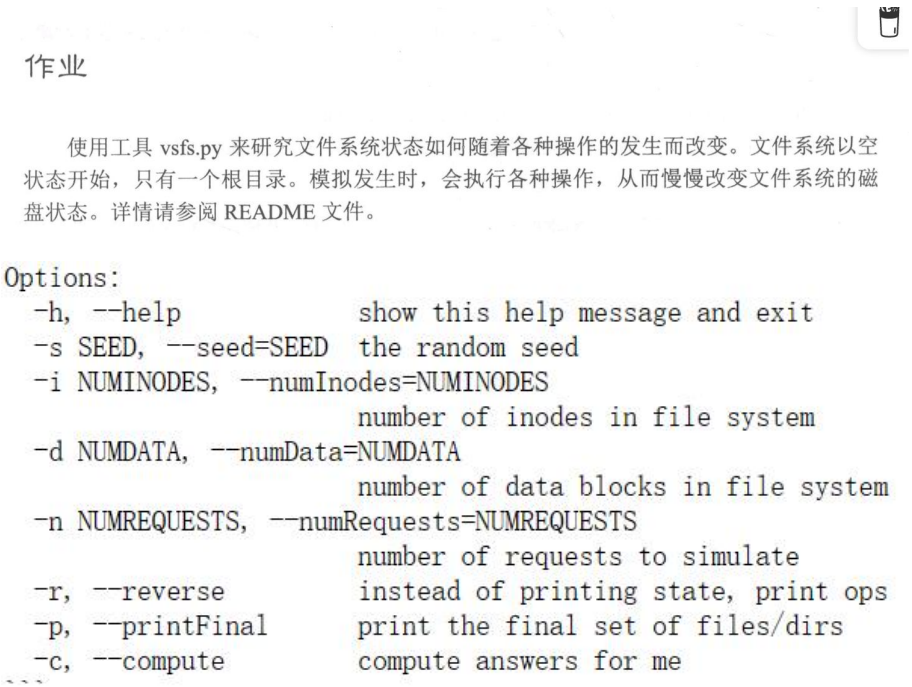
RAID 4:

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -L 4 -t -n 100 -c

disk:0  busy:  78.48  I/Os:    30 (sequential:0 nearly:0 random:30)
disk:1  busy: 100.00  I/Os:    40 (sequential:0 nearly:3 random:37)
disk:2  busy:  76.46  I/Os:    30 (sequential:0 nearly:2 random:28)
disk:3  busy:   0.00  I/Os:     0 (sequential:0 nearly:0 random:0)

STAT totalTime 386.1000000000002
```

RAID 5:

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./raid.py -L 5 -t -n 100 -c

disk:0  busy: 100.00  I/Os:    28 (sequential:0 nearly:1 random:27)
disk:1  busy:  95.84  I/Os:    29 (sequential:0 nearly:5 random:24)
disk:2  busy:  87.60  I/Os:    24 (sequential:0 nearly:0 random:24)
disk:3  busy:  65.70  I/Os:    19 (sequential:0 nearly:1 random:18)

STAT totalTime 276.7
```

第四十章



作业

使用工具 vsfs.py 来研究文件系统状态如何随着各种操作的发生而改变。文件系统以空状态开始，只有一个根目录。模拟发生时，会执行各种操作，从而慢慢改变文件系统的磁盘状态。详情请参阅 README 文件。

```
Options:
  -h, --help            show this help message and exit
  -s SEED, --seed=SEED  the random seed
  -i NUMINODES, --numInodes=NUMINODES
                        number of inodes in file system
  -d NUMDATA, --numData=NUMDATA
                        number of data blocks in file system
  -n NUMREQUESTS, --numRequests=NUMREQUESTS
                        number of requests to simulate
  -r, --reverse         instead of printing state, print ops
  -p, --printFinal      print the final set of files/dirs
  -c, --compute         compute answers for me
```

mkdir() - 创建文件夹：修改 inode 位图，增加一个 inode 用来存放新目录元数据，向存放新目录的目录块中增加一个条目，修改 data 位图，增加一个数据块用于存放新目录的内容，更新相应 inode 中的引用计数

creat() - 创建新的空文件：修改 inode 位图，增加一个 inode 用来存放新文件元数据，向存放新文件的目录块中增加一个条目，更新相应 inode 中的引用计数
open(), write() ,close() - 将一个块添加到文件：修改 data 位图，增加一个数据块用于存放文件的新内容，修改 inode 中的数据块地址字段
link() - 创建一个文件的硬链接：修改 inode，增加其中的引用计数，在保存链接的目录块中增加一个条目
unlink() - 删除一个硬链接 (如果 linkcnt==0,删除文件)：修改 inode，减小其中的引用计数，在保存链接的目录块中删除一个条目，当引用计数减为 0 时，删除文件，释放 inode、数据块，修改 inode 位图、data 位图

创建文件夹：修改 inode 位图和 data 位图。文件夹被创建时自带一个 data 块来记录子目录信息。

创建文件：只修改 inode 位图。文件只需要被记录。

每个 inode 都有三个字段：

第一个字段指示文件的类型（例如，f 表示常规文件，d 表示目录）；

第二个表示数据块属于一个文件（在这里，文件只能为空，这将数据块的地址设置为-1，或者大小为一个块，这将具有非负数地址）；

第三个显示文件的引用计数或目录。

例如，下面的 inode 是一个常规文件，它是空（地址字段设置为-1），并且在文件系统中只有一个链接：

[f a:-1 r:1]

如果同一个文件分配了一个块（比如块 10），则会显示如下所示：

[f a:10 r:1]

如果有人创建了一个指向该索引节点的硬链接，它就会变成：

[f a:10 r:2]

最后，数据块可以保留用户数据或目录数据。如果已填充对于目录数据块内的每个条目的形式(name，inumber)，其中“name”是文件或目录的名称“inumber”是文件的索引节点编号。因此一个空的根目录看起来是这样的，假设根索引节点是 0:

[（.，0）（..，0）]

如果我们在根目录中添加一个文件“f”，它已经被分配 inode 编号 1，则根目录内容将变为：

[（.，0）（..0）（f，1）]

如果数据块包含用户数据，则仅显示为单个字符在块内，例如“h”。如果它是空的并且未分配，那么只需要一对显示了空括号[]。

因此，整个文件系统如下所示：

inode bitmap 11110000
inodes [d a:0 r:6] [f a:1 r:1] [f a:-1 r:1] [d a:2 r:2] [] ...
data bitmap 11100000
data [(.,0) (..,0) (y,1) (z,2) (f,3)] [u] [(.,3) (..,0)] [] ...

1. 用一些不同的随机种子（比如 17、18、19、20）运行模拟器，看看你是否能确定每次状态变化之间一定发生了哪些操作。

seed 17:

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./vsfs.py -n 6 -s 17 -c
ARG seed 17
ARG numInodes 8
ARG numData 8
ARG numRequests 6
ARG reverse False
ARG printFinal False

Initial state

inode bitmap  10000000
inodes        [d a:0 r:2][][][][][][][]
data bitmap   10000000
data          [(.,0) (..,0)][][][][][][][]

mkdir("/u");

inode bitmap  11000000
inodes        [d a:0 r:3][d a:1 r:2][][][][][][]
data bitmap   11000000
data          [(.,0) (..,0) (u,1)][(.,1) (..,0)][][][][][][]

creat("/a");

inode bitmap  11100000
inodes        [d a:0 r:3][d a:1 r:2][f a:-1 r:1][][][][][]
data bitmap   11000000
data          [(.,0) (..,0) (u,1) (a,2)][(.,1) (..,0)][][][][][][]

unlink("/a");

inode bitmap  11000000
inodes        [d a:0 r:3][d a:1 r:2][][][][][][]
data bitmap   11000000
data          [(.,0) (..,0) (u,1)][(.,1) (..,0)][][][][][][]

mkdir("/z");
```

```
inode bitmap  11100000
inodes        [d a:0 r:4][d a:1 r:2][d a:2 r:2][][][][][]
data bitmap   11100000
data          [(.,0) (..,0) (u,1) (z,2)][(.,1) (..,0)][(.,2) (..,0)][][][][][]

mkdir("/s");

inode bitmap  11110000
inodes        [d a:0 r:5][d a:1 r:2][d a:2 r:2][d a:3 r:2][][][][]
data bitmap   11110000
data          [(.,0) (..,0) (u,1) (z,2) (s,3)][(.,1) (..,0)][(.,2) (..,0)][(.,3) (..,0)][][][][]

creat("/z/x");

inode bitmap  11111000
inodes        [d a:0 r:5][d a:1 r:2][d a:2 r:2][d a:3 r:2][f a:-1 r:1][][][]
data bitmap   11110000
data          [(.,0) (..,0) (u,1) (z,2) (s,3)][(.,1) (..,0)][(.,2) (..,0) (x,4)][(.,3) (..,0)][][][][]
```

操作 1 同时修改了 inode 位图和 data 位图。查看 1 号 inode，发现新建了一个目录，其数据存放在 1 号数据块，在 0 号数据块中查看新增加的条目，指示新建的目录名为“u”，所以操作 1 是 mkdir(“/u”)

操作 2 只修改了 inode 位图，所以是 creat()。查看 2 号 inode,发现新建了一个文件，在 0 号数据块中查看新增加的条目，指示新建的文件名为“a”，所以操作 2 是 creat(“/a”)

操作 3 修改了 inode 位图，删除了 inode 和目录块中的条目，所以是 unlink()。发现删除的是 2 号 inode，所以操作 3 是 unlink(“/a”)

操作 4 同时修改了 inode 位图和 data 位图。查看 2 号 inode,发现新建了一个目录，其数据存放在 2 号数据块，在 0 号数据块中查看新增加的条目，指示新建的目录名为“z”，所以操作 4 是 mkdir(“/z”)

操作 5 同时修改了 inode 位图和 data 位图。查看 3 号 inode,发现新建了一个目录，其数据存放在 3 号数据块，在 0 号数据块中查看新增加的条目，指示新建的目录名为“s”，所以操作 5 是 mkdir(“/s”)

操作 6 只修改了 inode 位图，所以是 creat()。查看 4 号 inode,发现新建了一个文

件，在 3 号数据块（目录 z 的目录块）中查看新增加的条目，指示新建的文件名为“x”，所以操作 6 是 creat(“/z/x”)

seed 18：

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./vsfs.py -n 6 -s 18 -c
ARG seed 18
ARG numInodes 8
ARG numData 8
ARG numRequests 6
ARG reverse False
ARG printFinal False

Initial state

inode bitmap  10000000
inodes        [d a:0 r:2][][][][][][][][]
data bitmap   10000000
data          [(.,0) (..,0)][][][][][][][]

mkdir("/f");

inode bitmap  11000000
inodes        [d a:0 r:3][d a:1 r:2][][][][][][]
data bitmap   11000000
data          [(.,0) (..,0) (f,1)][(.,1) (..,0)][][][][][][]

creat("/s");

inode bitmap  11100000
inodes        [d a:0 r:3][d a:1 r:2][f a:-1 r:1][][][][][]
data bitmap   11000000
data          [(.,0) (..,0) (f,1) (s,2)][(.,1) (..,0)][][][][][][]

mkdir("/h");

inode bitmap  11110000
inodes        [d a:0 r:4][d a:1 r:2][f a:-1 r:1][d a:2 r:2][][][][]
data bitmap   11100000
data          [(.,0) (..,0) (f,1) (s,2) (h,3)][(.,1) (..,0)][(.,3) (..,0)][][][][][]

fd=open("/s", O_WRONLY|O_APPEND); write(fd, buf, BLOCKSIZE); close(fd);
```

```
inode bitmap  11110000
inodes        [d a:0 r:4][d a:1 r:2][f a:3 r:1][d a:2 r:2][][][][]
data bitmap   11110000
data          [(.,0) (..,0) (f,1) (s,2) (h,3)][(.,1) (..,0)][(.,3) (..,0)][f][][][][]

creat("/f/o");

inode bitmap  11111000
inodes        [d a:0 r:4][d a:1 r:2][f a:3 r:1][d a:2 r:2][f a:-1 r:1][][][]
data bitmap   11110000
data          [(.,0) (..,0) (f,1) (s,2) (h,3)][(.,1) (..,0) (o,4)][(.,3) (..,0)][f][][][][]

creat("/c");

inode bitmap  11111100
inodes        [d a:0 r:4][d a:1 r:2][f a:3 r:1][d a:2 r:2][f a:-1 r:1][f a:-1 r:1][][]
data bitmap   11110000
data          [(.,0) (..,0) (f,1) (s,2) (h,3) (c,5)][(.,1) (..,0) (o,4)][(.,3) (..,0)][f][][][][]
```

操作 1 同时修改了 inode 位图和 data 位图。查看 1 号 inode,发现新建了一个目录，其数据存放在 1 号数据块，在 0 号数据块中查看新增加的条目，指示新建的目录名为“f”，所以操作 1 是 mkdir(“/f”)

操作 2 只修改了 inode 位图，所以是 creat()。查看 2 号 inode,发现新建了一个文件，在 0 号数据块中查看新增加的条目，指示新建的文件名为“s”，所以操作 2 是 creat(“/s”)

操作 3 同时修改了 inode 位图和 data 位图，只有 mkdir()可以做到。查看 3 号 inode,发现新建了一个目录，其数据存放在 2 号数据块，在 0 号数据块中查看新增加的条目，指示新建的目录名为“h”，所以操作 3 是 mkdir(“/h”)

操作 4 修改了 data 位图，修改了 2 号 inode（文件 s）中的地址字段，增加了 3 号数据块，所以操作 4 是 fd=open(“/s”,O_WRONLY|O_APPEND); write(fd, buf, BLOCKSIZE); close(fd)

操作 5 只修改了 inode 位图，所以是 creat()。查看 4 号 inode,发现新建了一个文件，在 1 号数据块（目录 f 的目录块）中查看新增加的条目，指示新建的文件名为“o”，所以操作 5 是 creat(“/f/o”)
操作 6 只修改了 inode 位图，所以是 creat()。查看 5 号 inode,发现新建了一个文件，在 0 号数据块中查看新增加的条目，指示新建的文件名为“c”，所以操作 6 是 creat(“/c”)
seed 19：

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./vsfs.py -n 6 -s 19 -c
ARG seed 19
ARG numInodes 8
ARG numData 8
ARG numRequests 6
ARG reverse False
ARG printFinal False

Initial state

inode bitmap  10000000
inodes       [d a:0 r:2][][][][][][][]
data bitmap   10000000
data         [(.,0) (..,0)][][][][][][][]

creat("/k");

inode bitmap  11000000
inodes       [d a:0 r:2][f a:-1 r:1][][][][][][]
data bitmap   10000000
data         [(.,0) (..,0) (k,1)][][][][][][][]

creat("/g");

inode bitmap  11100000
inodes       [d a:0 r:2][f a:-1 r:1][f a:-1 r:1][][][][][]
data bitmap   10000000
data         [(.,0) (..,0) (k,1) (g,2)][][][][][][][]

fd=open("/k", O_WRONLY|O_APPEND); write(fd, buf, BLOCKSIZE); close(fd);

inode bitmap  11100000
inodes       [d a:0 r:2][f a:1 r:1][f a:-1 r:1][][][][][]
data bitmap   11000000
data         [(.,0) (..,0) (k,1) (g,2)][g][][][][][][]

link("/k", "/b");

inode bitmap  11100000
inodes       [d a:0 r:2][f a:1 r:2][f a:-1 r:1][][][][][]
data bitmap   11000000
data         [(.,0) (..,0) (k,1) (g,2) (b,1)][g][][][][][][]
```

```
link("/b", "/t");

inode bitmap  11100000
inodes       [d a:0 r:2][f a:1 r:3][f a:-1 r:1][][][][][]
data bitmap   11000000
data         [(.,0) (..,0) (k,1) (g,2) (b,1) (t,1)][g][][][][][][]

unlink("/k");

inode bitmap  11100000
inodes       [d a:0 r:2][f a:1 r:2][f a:-1 r:1][][][][][]
data bitmap   11000000
data         [(.,0) (..,0) (g,2) (b,1) (t,1)][g][][][][][][]
```

操作 1 只修改了 inode 位图，所以是 creat()。查看 1 号 inode,发现新建了一个文件，在 0 号数据块中查看新增加的条目，指示新建的文件名为“k”，所以操作 1 是 creat(“/k”)
操作 2 只修改了 inode 位图，所以是 creat()。查看 2 号 inode,发现新建了一个文件，在 0 号数据块中查看新增加的条目，指示新建的文件名为“g”，所以操作 2

是 creat(“/g”)
操作 3 修改了 data 位图，修改了 1 号 inode（文件 k）中的地址字段，增加了 1 号数据块，所以操作 3 是 fd=open(“/k”,O_WRONLY|O_APPEND); write(fd, buf, BLOCKSIZE); close(fd)
操作 4 增加了 1 号 inode（文件 k）中的引用计数字段，所以是 link()。在 0 号数据块中查看新增加的条目，指示新建的链接名为“b”，所以操作 4 是 link(“/k”,“/b”)
操作 5 增加了 1 号 inode（文件 k）中的引用计数字段，所以是 link()。在 0 号数据块中查看新增加的条目，指示新建的链接名为“t”，所以操作 5 是 link(“/k”,“/t”)
操作 6 减小了 1 号 inode（文件 k）中的引用计数字段，所以是 unlink()。在 0 号数据块中查看删除的条目，指示删除的链接名为“k”，所以操作 6 是 unlink(“/k”)
seed 20：

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./vsfs.py -n 6 -s 20 -c
ARG seed 20
ARG numInodes 8
ARG numData 8
ARG numRequests 6
ARG reverse False
ARG printFinal False

Initial state

inode bitmap  10000000
inodes        [d a:0 r:2][][][][][][][]
data bitmap   10000000
data          [(.,0) (..,0)][][][][][][][]

creat("/x");

inode bitmap  11000000
inodes        [d a:0 r:2][f a:-1 r:1][][][][][][]
data bitmap   10000000
data          [(.,0) (..,0) (x,1)][][][][][][][]

fd=open("/x", O_WRONLY|O_APPEND); write(fd, buf, BLOCKSIZE); close(fd);

inode bitmap  11000000
inodes        [d a:0 r:2][f a:1 r:1][][][][][][]
data bitmap   11000000
data          [(.,0) (..,0) (x,1)][x][][][][][][]

creat("/k");

inode bitmap  11100000
inodes        [d a:0 r:2][f a:1 r:1][f a:-1 r:1][][][][][]
data bitmap   11000000
data          [(.,0) (..,0) (x,1) (k,2)][x][][][][][][]
```

```
creat("/y");

inode bitmap  11110000
inodes        [d a:0 r:2][f a:1 r:1][f a:-1 r:1][f a:-1 r:1][][][][]
data bitmap   11000000
data          [(.,0) (..,0) (x,1) (k,2) (y,3)][x][][][][][][]

unlink("/x");

inode bitmap  10110000
inodes        [d a:0 r:2][][f a:-1 r:1][f a:-1 r:1][][][][]
data bitmap   10000000
data          [(.,0) (..,0) (k,2) (y,3)][][][][][][][]

unlink("/y");

inode bitmap  10100000
inodes        [d a:0 r:2][][f a:-1 r:1][][][][][]
data bitmap   10000000
data          [(.,0) (..,0) (k,2)][][][][][][][]
```

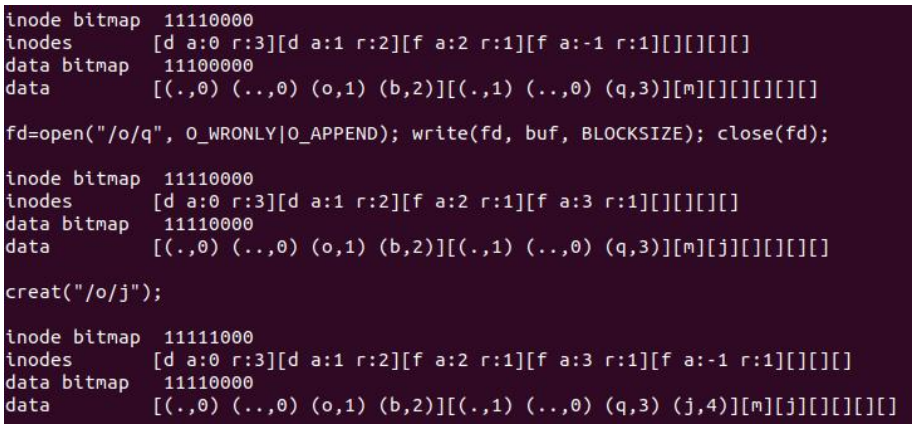
操作 1 只修改了 inode 位图，所以是 creat()。查看 1 号 inode,发现新建了一个文

件，在 0 号数据块中查看新增加的条目，指示新建的文件名为“x”，所以操作 1 是 creat(“/x”)
操作 2 修改了 data 位图，修改了 1 号 inode（文件 x）中的地址字段，增加了 1 号数据块，所以操作 2 是 fd=open(“/x”,O_WRONLY|O_APPEND); write(fd, buf, BLOCKSIZE); close(fd)
操作 3 只修改了 inode 位图，所以是 creat()。查看 2 号 inode,发现新建了一个文件，在 0 号数据块中查看新增加的条目，指示新建的文件名为“k”，所以操作 3 是 creat(“/k”)
操作 4 只修改了 inode 位图，所以是 creat()。查看 3 号 inode,发现新建了一个文件，在 0 号数据块中查看新增加的条目，指示新建的文件名为“y”，所以操作 4 是 creat(“/y”)
操作 5 修改了 inode 位图和 data 位图，删除了 inode、数据块和目录块中的条目，所以是 unlink()。发现删除的是 1 号 inode（文件 x），所以操作 5 是 unlink(“/x”)
操作 6 修改了 inode 位图，删除了 inode 和目录块中的条目，所以是 unlink()。发现删除的是 3 号 inode（文件 y），所以操作 6 是 unlink(“/y”)

2. 现在使用不同的随机种子（比如 21、22、23、24），但使用-r 标志运行，这样做可以让你在显示操作时猜测状态的变化。关于 inode 和数据块分配算法，根据它们喜欢分配的块，你可以得出什么结论？

seed 21：

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./vsfs.py -n 6 -s 21 -r -c
ARG seed 21
ARG numInodes 8
ARG numData 8
ARG numRequests 6
ARG reverse True
ARG printFinal False

Initial state

inode bitmap  10000000
inodes        [d a:0 r:2][][][][][][][]
data bitmap   10000000
data          [(.,0) (..,0)][][][][][][][]

mkdir("/o");

inode bitmap  11000000
inodes        [d a:0 r:3][d a:1 r:2][][][][][][]
data bitmap   11000000
data          [(.,0) (..,0) (o,1)][(.,1) (..,0)][][][][][][]

creat("/b");

inode bitmap  11100000
inodes        [d a:0 r:3][d a:1 r:2][f a:-1 r:1][][][][][]
data bitmap   11000000
data          [(.,0) (..,0) (o,1) (b,2)][(.,1) (..,0)][][][][][][]

creat("/o/q");

inode bitmap  11110000
inodes        [d a:0 r:3][d a:1 r:2][f a:-1 r:1][f a:-1 r:1][][][][]
data bitmap   11000000
data          [(.,0) (..,0) (o,1) (b,2)][(.,1) (..,0) (q,3)][][][][][][][]

fd=open("/b", O_WRONLY|O_APPEND); write(fd, buf, BLOCKSIZE); close(fd);
```



操作 1 是 mkdir("/o")，修改 inode 位图，增加一个 inode 用来存放新目录元数据，向存放新目录的目录块中增加一个条目，修改 data 位图，增加一个数据块用于存放新目录的内容，更新相应 inode 中的引用计数

操作 2 是 creat("/b")，修改 inode 位图，增加一个 inode 用来存放新文件元数据，向存放新文件的目录块中增加一个条目，更新相应 inode 中的引用计数

操作 3 是 creat("/o/q")，修改 inode 位图，增加一个 inode 用来存放新文件元数据，向存放新文件的目录块中增加一个条目，更新相应 inode 中的引用计数

操作 4 是 fd=open("/b", O_WRONLY|O_APPEND); write(fd, buf, BLOCKSIZE); close(fd)，修改 data 位图，增加一个数据块用于存放文件的新内容，修改 inode 中的数据块地址字段

操作 5 是 fd=open("/o/q", O_WRONLY|O_APPEND); write(fd, buf, BLOCKSIZE); close(fd)，修改 data 位图，增加一个数据块用于存放文件的新内容，修改 inode 中的数据块地址字段

操作 6 是 creat("/o/j")，修改 inode 位图，增加一个 inode 用来存放新文件元数据，向存放新文件的目录块中增加一个条目，更新相应 inode 中的引用计数

seed 22：

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./vsfs.py -n 6 -s 22 -r -c
ARG seed 22
ARG numInodes 8
ARG numData 8
ARG numRequests 6
ARG reverse True
ARG printFinal False

Initial state

inode bitmap  10000000
inodes       [d a:0 r:2][][][][][][][]
data bitmap   10000000
data         [(.,0) (..,0)][][][][][][][]

creat("/z");

inode bitmap  11000000
inodes       [d a:0 r:2][f a:-1 r:1][][][][][][]
data bitmap   10000000
data         [(.,0) (..,0) (z,1)][][][][][][][]

fd=open("/z", O_WRONLY|O_APPEND); write(fd, buf, BLOCKSIZE); close(fd);

inode bitmap  11000000
inodes       [d a:0 r:2][f a:1 r:1][][][][][][]
data bitmap   11000000
data         [(.,0) (..,0) (z,1)][q][][][][][][]

unlink("/z");

inode bitmap  10000000
inodes       [d a:0 r:2][][][][][][][]
data bitmap   10000000
data         [(.,0) (..,0)][][][][][][][]

creat("/y");

inode bitmap  11000000
inodes       [d a:0 r:2][f a:-1 r:1][][][][][][]
data bitmap   10000000
data         [(.,0) (..,0) (y,1)][][][][][][][]
```

```
link("/y", "/s");

inode bitmap  11000000
inodes       [d a:0 r:2][f a:-1 r:2][][][][][][]
data bitmap   10000000
data         [(.,0) (..,0) (y,1) (s,1)][][][][][][][]

creat("/e");

inode bitmap  11100000
inodes       [d a:0 r:2][f a:-1 r:2][f a:-1 r:1][][][][][]
data bitmap   10000000
data         [(.,0) (..,0) (y,1) (s,1) (e,2)][][][][][][][]
```

seed 23：

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./vsfs.py -n 6 -s 23 -r -c
ARG seed 23
ARG numInodes 8
ARG numData 8
ARG numRequests 6
ARG reverse True
ARG printFinal False

Initial state

inode bitmap  10000000
inodes       [d a:0 r:2][][][][][][][]
data bitmap   10000000
data         [(.,0) (..,0)][][][][][][][]

mkdir("/c");

inode bitmap  11000000
inodes       [d a:0 r:3][d a:1 r:2][][][][][][]
data bitmap   11000000
data         [(.,0) (..,0) (c,1)][(.,1) (..,0)][][][][][][]

creat("/c/t");

inode bitmap  11100000
inodes       [d a:0 r:3][d a:1 r:2][f a:-1 r:1][][][][][]
data bitmap   11000000
data         [(.,0) (..,0) (c,1)][(.,1) (..,0) (t,2)][][][][][][]

unlink("/c/t");

inode bitmap  11000000
inodes       [d a:0 r:3][d a:1 r:2][][][][][][]
data bitmap   11000000
data         [(.,0) (..,0) (c,1)][(.,1) (..,0)][][][][][][]

creat("/c/q");
```

```
inode bitmap  11100000
inodes       [d a:0 r:3][d a:1 r:2][f a:-1 r:1][][][][][]
data bitmap   11000000
data         [(.,0) (..,0) (c,1)][(.,1) (..,0) (q,2)][][][][][][]

creat("/c/j");

inode bitmap  11110000
inodes       [d a:0 r:3][d a:1 r:2][f a:-1 r:1][f a:-1 r:1][][][][]
data bitmap   11000000
data         [(.,0) (..,0) (c,1)][(.,1) (..,0) (q,2) (j,3)][][][][][][]

link("/c/q", "/c/h");

inode bitmap  11110000
inodes       [d a:0 r:3][d a:1 r:2][f a:-1 r:2][f a:-1 r:1][][][][]
data bitmap   11000000
data         [(.,0) (..,0) (c,1)][(.,1) (..,0) (q,2) (j,3) (h,2)][][][][][][]
```

seed 24：

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./vsfs.py -n 6 -s 24 -r -c
ARG seed 24
ARG numInodes 8
ARG numData 8
ARG numRequests 6
ARG reverse True
ARG printFinal False

Initial state

inode bitmap  10000000
inodes        [d a:0 r:2][][][][][][][]
data bitmap   10000000
data          [(.,0) (..,0)][][][][][][][]

mkdir("/z");

inode bitmap  11000000
inodes        [d a:0 r:3][d a:1 r:2][][][][][][]
data bitmap   11000000
data          [(.,0) (..,0) (z,1)][(.,1) (..,0)][][][][][][]

creat("/z/t");

inode bitmap  11100000
inodes        [d a:0 r:3][d a:1 r:2][f a:-1 r:1][][][][][]
data bitmap   11000000
data          [(.,0) (..,0) (z,1)][(.,1) (..,0) (t,2)][][][][][][]

creat("/z/z");
```

```
inode bitmap  11110000
inodes        [d a:0 r:3][d a:1 r:2][f a:-1 r:1][f a:-1 r:1][][][][]
data bitmap   11000000
data          [(.,0) (..,0) (z,1)][(.,1) (..,0) (t,2) (z,3)][][][][][][]

fd=open("/z/z", O_WRONLY|O_APPEND); write(fd, buf, BLOCKSIZE); close(fd);

inode bitmap  11110000
inodes        [d a:0 r:3][d a:1 r:2][f a:-1 r:1][f a:2 r:1][][][][]
data bitmap   11100000
data          [(.,0) (..,0) (z,1)][(.,1) (..,0) (t,2) (z,3)][y][][][][][]

creat("/y");

inode bitmap  11111000
inodes        [d a:0 r:3][d a:1 r:2][f a:-1 r:1][f a:2 r:1][f a:-1 r:1][][][]
data bitmap   11100000
data          [(.,0) (..,0) (z,1) (y,4)][(.,1) (..,0) (t,2) (z,3)][y][][][][][]

fd=open("/y", O_WRONLY|O_APPEND); write(fd, buf, BLOCKSIZE); close(fd);

inode bitmap  11111000
inodes        [d a:0 r:3][d a:1 r:2][f a:-1 r:1][f a:2 r:1][f a:3 r:1][][][]
data bitmap   11110000
data          [(.,0) (..,0) (z,1) (y,4)][(.,1) (..,0) (t,2) (z,3)][y][v][][][][]
```

分配算法会使用最近可分配的 inode 与数据块。

3. 现在将文件系统中的数据块数量减少到非常少（比如两个），并用 100 个左右的请求来运行模拟器。在这种高度约束的布局中，哪些类型的文件最终会出现在文件系统中？什么类型的操作会失败？

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./vsfs.py -d 2 -c -n 100
ARG seed 0
ARG numInodes 8
ARG numData 2
ARG numRequests 100
ARG reverse False
ARG printFinal False

Initial state

inode bitmap  10000000
inodes        [d a:0 r:2][][][][][][][]
data bitmap   10
data          [(.,0) (..,0)][]

mkdir("/g");
File system out of data blocks; rerun with more via command-line flag?
```

数据块太少，第一个文件都无法创建。
因为 mkdir()和 open(), write(), close()需要数据块，而 creat()、link()、unlink()不需要数据块，所以 mkdir()和 open(), write(), close()操作会失败，creat()、link()、unlink()操作不会失败。

4. 现在做同样的事情，但针对 inodes。只有非常少的 inode，什么类型的操作才能成功？哪些通常会失败？文件系统的最终状态可能是什么？

```
guoruiling@guoruiling-virtual-machine:~/os/hw4$ python3 ./vsfs.py -i 2 -c -n 100
ARG seed 0
ARG numInodes 2
ARG numData 8
ARG numRequests 100
ARG reverse False
ARG printFinal False

Initial state

inode bitmap  10
inodes       [d a:0 r:2][]
data bitmap   10000000
data         [(.,0) (..,0)][][][][][][][]

mkdir("/g");
File system out of inodes; rerun with more via command-line flag?
```

同样，第一个文件也无法创建。
因为 mkdir()和 creat()需要 inode，而 open(), write(), close() 、link()、unlink()不需要 inode，所以 mkdir()和 creat()操作会失败，open(), write(), close() 、link()、unlink()操作不会失败。