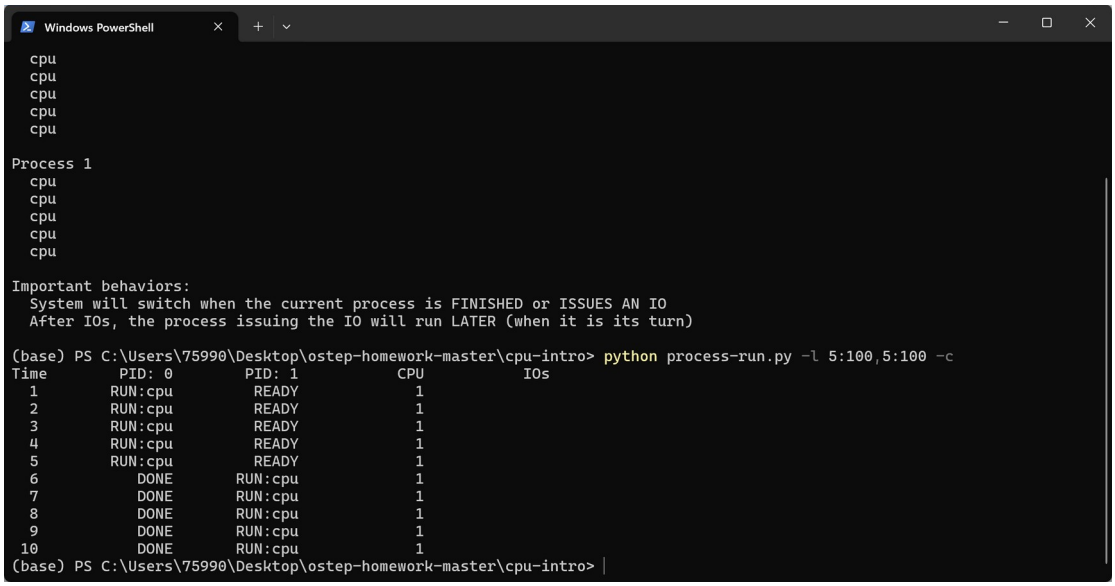
Chp 4

1. CPU 的利用率为 100%，因为没有进程执行 IO 操作。



2. 执行 4 条执行需要 4 个时间单元，一次 IO 操作的发起和结束一共需要两个时间单元，IO 操作需要等待 5 个时间单元。
   故一共需要 11 个时间单元的时间。

```
75990@Smile MINGW64 ~/Desktop/ostep-homework-master/cpu-intro
$ ./process-run.py -l 4:100,1:0 -c
Time        PID: 0        PID: 1          CPU          IOs
  1        RUN:cpu         READY            1
  2        RUN:cpu         READY            1
  3        RUN:cpu         READY            1
  4        RUN:cpu         READY            1
  5           DONE        RUN:io            1
  6           DONE       BLOCKED                           1
  7           DONE       BLOCKED                           1
  8           DONE       BLOCKED                           1
  9           DONE       BLOCKED                           1
 10           DONE       BLOCKED                           1
 11*          DONE    RUN:io_done             1
```

3. 交换顺序后，一共需要 7 个时间单元。因为进程 0 在等待 IO 时，进程 1 可以上 CPU 执行。

```
75990@Smile MINGW64 ~/Desktop/ostep-homework-master/cpu-intro
$ ./process-run.py -l 1:0,4:100 -c
Time        PID: 0        PID: 1          CPU          IOs
  1         RUN:io         READY            1
  2        BLOCKED       RUN:cpu            1            1
  3        BLOCKED       RUN:cpu            1            1
  4        BLOCKED       RUN:cpu            1            1
  5        BLOCKED       RUN:cpu            1            1
  6        BLOCKED          DONE                         1
  7*   RUN:io_done          DONE            1
```

4. CPU 花了 11 个时间单元才执行完。

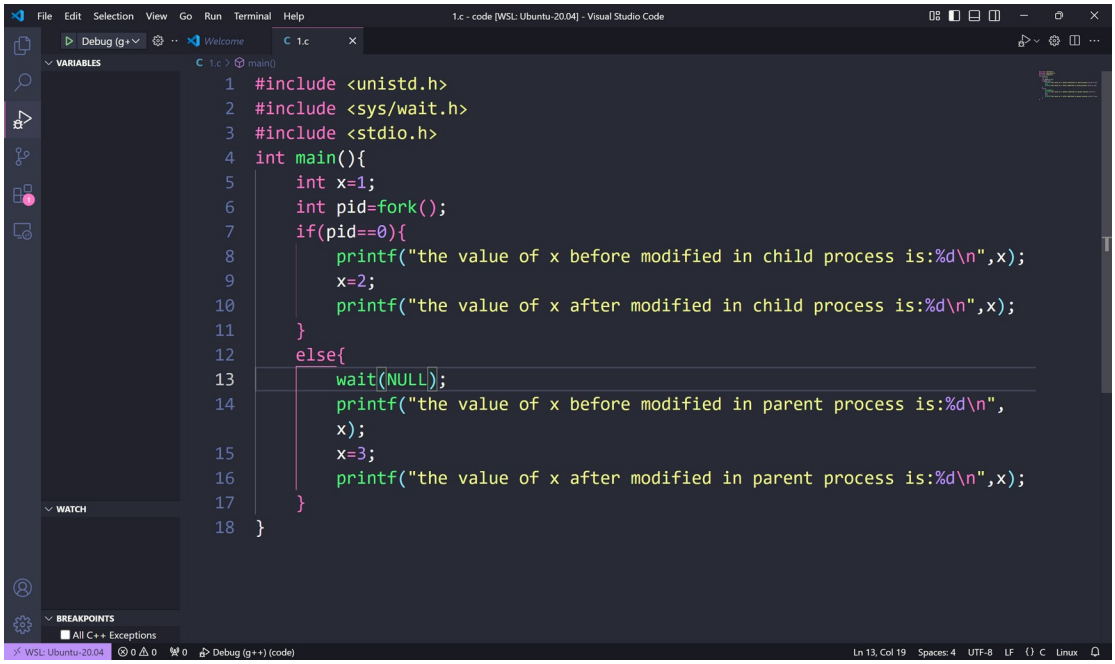
```
75990@Smile MINGW64 ~/Desktop/ostep-homework-master/cpu-intro
$ ./process-run.py -l 1:0,4:100 -c -S SWITCH_ON_END
Time        PID: 0        PID: 1           CPU           IOs
  1         RUN:io         READY             1
  2        BLOCKED         READY                           1
  3        BLOCKED         READY                           1
  4        BLOCKED         READY                           1
  5        BLOCKED         READY                           1
  6        BLOCKED         READY                           1
  7*   RUN:io_done         READY             1
  8           DONE       RUN:cpu             1
  9           DONE       RUN:cpu             1
 10           DONE       RUN:cpu             1
 11           DONE       RUN:cpu             1
```

5. CPU 花了 7 个时间单元才执行完。情况和第 3 问是一样的。

```
75990@Smile MINGW64 ~/Desktop/ostep-homework-master/cpu-intro
$ ./process-run.py -l 1:0,4:100 -c -S SWITCH_ON_IO
Time        PID: 0        PID: 1           CPU           IOs
  1         RUN:io         READY             1
  2        BLOCKED       RUN:cpu             1             1
  3        BLOCKED       RUN:cpu             1             1
  4        BLOCKED       RUN:cpu             1             1
  5        BLOCKED       RUN:cpu             1             1
  6        BLOCKED          DONE                           1
  7*   RUN:io_done          DONE             1
```

Chp5

1.



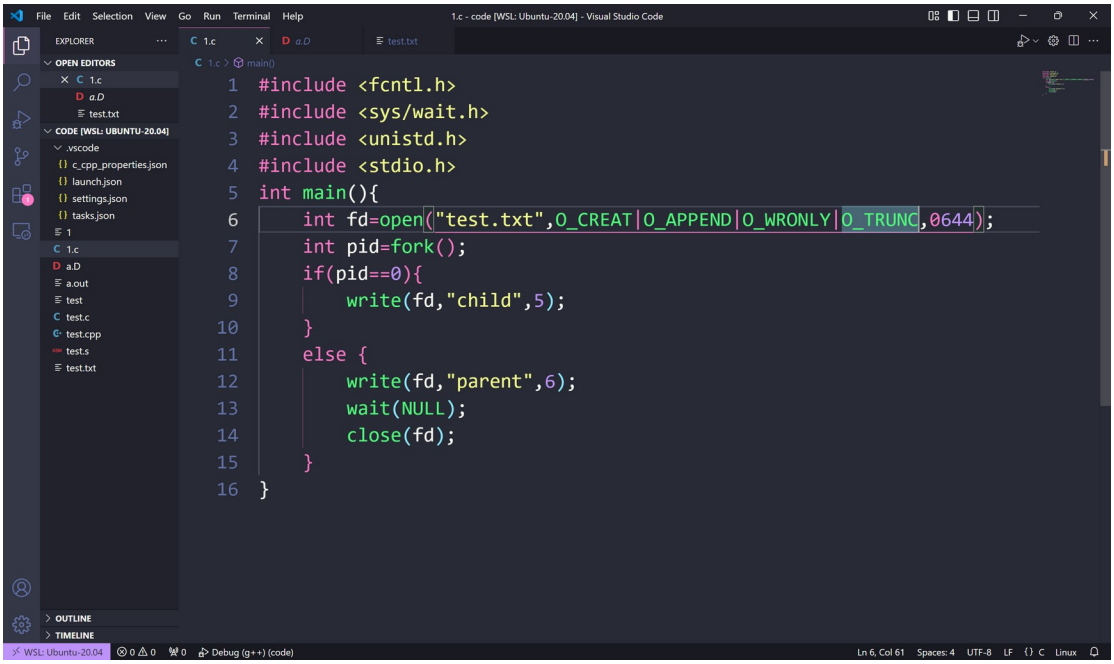

```c
#include <unistd.h>
#include <sys/wait.h>
#include <stdio.h>
int main(){
    int x=1;
    int pid=fork();
    if(pid==0){
        printf("the value of x before modified in child process is:%d\n",x);
        x=2;
        printf("the value of x after modified in child process is:%d\n",x);
    }
    else{
        wait(NULL);
        printf("the value of x before modified in parent process is:%d\n",
        x);
        x=3;
        printf("the value of x after modified in parent process is:%d\n",x);
    }
}
```

```
TERMINAL                                                                    cppdbg: 1

the value of x before modified in child process is:1
the value of x after modified in child process is:2
the value of x before modified in parent process is:1
the value of x after modified in parent process is:3
[1] + Done                    "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-1r2dsmfh.4vv" 1>"/tmp/Microsoft-MIEngine-Out-
nqj1ezl3.odc"
smile_laughter@Smile:~/code$
```

子进程中变量的值和主进程设置的值一样，子进程和父进程可以分别修改变量的值，而不会发生冲突。

2.

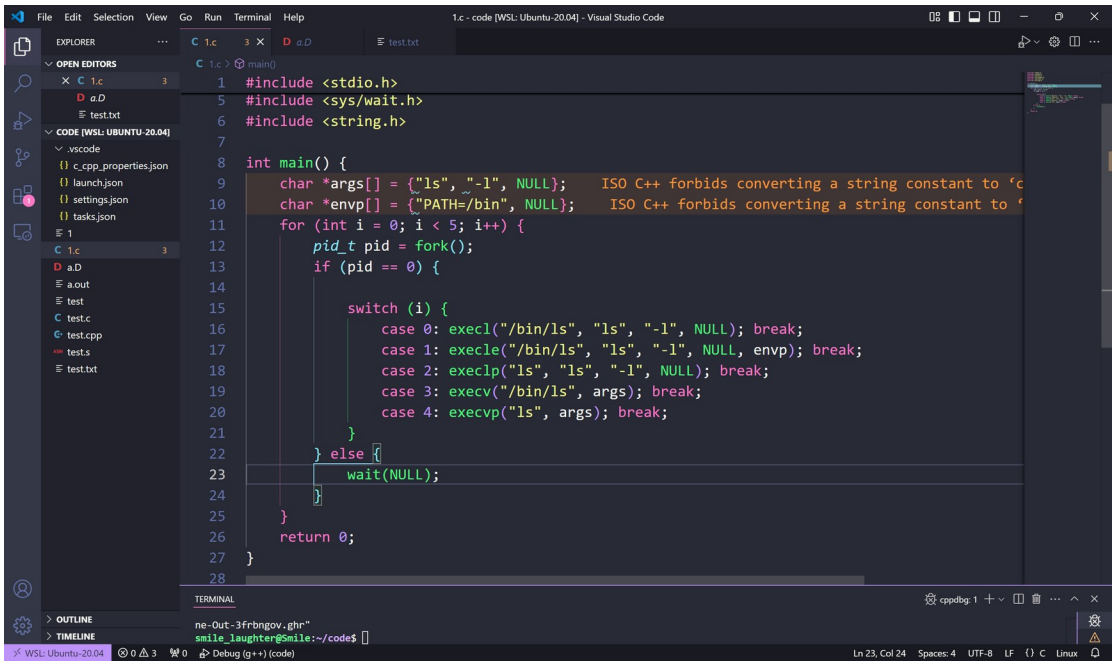


子进程和父进程都可以访问主进程创建的文件描述符，同时写入时，test.txt 中可能的结果有两种：

（1） parentchild

（2） childparent

4.



原因：

（1） 参数传递方式不同：可变参数列表与参数数组

（2） 路径解析方式不同：提供绝对路径，利用环境变量解析路径

（3） 是否可以支持自定义环境变量

Chp 7

1.

SJF	响应时间	周转时间
作业 0	0	200
作业 1	200	400
作业 2	400	600
平均	200	400

FIFO	响应时间	周转时间
作业 0	0	200

作业 1	200	400
作业 2	400	600
平均	200	400

2.

SJF	响应时间	周转时间
作业 0	0	100
作业 1	100	300
作业 2	300	600
平均	400/3	1000/3

FIFO	响应时间	周转时间
作业 0	0	100
作业 1	100	300
作业 2	300	600
平均	400/3	1000/3

3.

RR	响应时间	周转时间
作业 0	0	598
作业 1	1	599
作业 2	2	600

平均	1	599

4.作业长度按作业到达顺序非严格递增

5.

（1）所有作业长度相同，并且 RR 的时间片足够大时

（2）每个工作的时长相同且等于时间切片时

6. 工作长度最短的任务响应时间不变，其它工作的响应时间变长。

```
75990@Smile MINGW64 ~/Desktop/ostep-homework-master/cpu-sched
$ ./scheduler.py -p SJF -j 3 -l 1,2,3 -c
ARG policy SJF
ARG jlist 1,2,3

Here is the job list, with the run time of each job:
  Job 0 ( length = 1.0 )
  Job 1 ( length = 2.0 )
  Job 2 ( length = 3.0 )


** Solutions **

Execution trace:
  [ time   0 ] Run job 0 for 1.00 secs ( DONE at 1.00 )
  [ time   1 ] Run job 1 for 2.00 secs ( DONE at 3.00 )
  [ time   3 ] Run job 2 for 3.00 secs ( DONE at 6.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 1.00  Wait 0.00
  Job   1 -- Response: 1.00  Turnaround 3.00  Wait 1.00
  Job   2 -- Response: 3.00  Turnaround 6.00  Wait 3.00

  Average -- Response: 1.33  Turnaround 3.33  Wait 1.33


75990@Smile MINGW64 ~/Desktop/ostep-homework-master/cpu-sched
```

```
75990@Smile MINGW64 ~/Desktop/ostep-homework-master/cpu-sched
$ ./scheduler.py -p SJF -j 3 -l 100,200,300 -c
ARG policy SJF
ARG jlist 100,200,300

Here is the job list, with the run time of each job:
  Job 0 ( length = 100.0 )
  Job 1 ( length = 200.0 )
  Job 2 ( length = 300.0 )


** Solutions **

Execution trace:
  [ time   0 ] Run job 0 for 100.00 secs ( DONE at 100.00 )
  [ time 100 ] Run job 1 for 200.00 secs ( DONE at 300.00 )
  [ time 300 ] Run job 2 for 300.00 secs ( DONE at 600.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 100.00  Wait 0.00
  Job   1 -- Response: 100.00  Turnaround 300.00  Wait 100.00
  Job   2 -- Response: 300.00  Turnaround 600.00  Wait 300.00

  Average -- Response: 133.33  Turnaround 333.33  Wait 133.33


75990@Smile MINGW64 ~/Desktop/ostep-homework-master/cpu-sched
$
```

7. 假设工作时间远大于时间片长度：
   随着量子长度的增加，RR 的响应时间会变大
   假设量子长度为 t，则平均响应时间为

$$[0*t+1*t+2*t+.....+(N-1)*t]/n=(N-1)*t/2$$

   最坏情况的响应时间即是最后一个工作的响应时间，为(N-1)*t

Chp 8

1.

./mlfq.py -j 2 -n 2 -M 0 -m 100

```
75990@Smile MINGW64 ~/Desktop/ostep-homework-master/cpu-sched-mlfq
$ ./mlfq.py -j 2 -n 2 -M 0 -m 100
Here is the list of inputs:
OPTIONS jobs 2
OPTIONS queues 2
OPTIONS allotments for queue  1 is   1
OPTIONS quantum length for queue  1 is  10
OPTIONS allotments for queue  0 is   1
OPTIONS quantum length for queue  0 is  10
OPTIONS boost 0
OPTIONS ioTime 5
OPTIONS stayAfterIO False
OPTIONS iobump False


For each job, three defining characteristics are given:
  startTime : at what time does the job enter the system
  runTime   : the total CPU time needed by the job to finish
  ioFreq    : every ioFreq time units, the job issues an I/O
              (the I/O takes ioTime units to complete)

Job List:
  Job  0: startTime   0 - runTime  84 - ioFreq   0
  Job  1: startTime   0 - runTime  42 - ioFreq   0

Compute the execution trace for the given workloads.
If you would like, also compute the response and turnaround
times for each of the jobs.

Use the -c flag to get the exact results when you are finished.


75990@Smile MINGW64 ~/Desktop/ostep-homework-master/cpu-sched-mlfq
$
```

time 0-9	Run JOB 0 at PRIORITY 1
time 10-19	Run JOB 1 at PRIORITY 1
time 20-29	Run JOB 0 at PRIORITY 0
time 30-39	Run JOB 1 at PRIORITY 0
time 40-49	Run JOB 0 at PRIORITY 0
time 50-59	Run JOB 1 at PRIORITY 0
time 60-69	Run JOB 0 at PRIORITY 0
time 70-79	Run JOB 1 at PRIORITY 0
time 80-89	Run JOB 0 at PRIORITY 0
time 90-91	Run JOB 1 at PRIORITY 0
time 92-101	Run JOB 0 at PRIORITY 0
time 102-111	Run JOB 0 at PRIORITY 0

time 112-121	Run JOB 0 at PRIORITY 0
time 122-125	Run JOB 0 at PRIORITY 0

```
[ time 103 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 22 (of 84) ]
[ time 104 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 21 (of 84) ]
[ time 105 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 20 (of 84) ]
[ time 106 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 19 (of 84) ]
[ time 107 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 18 (of 84) ]
[ time 108 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 17 (of 84) ]
[ time 109 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 16 (of 84) ]
[ time 110 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 15 (of 84) ]
[ time 111 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 14 (of 84) ]
[ time 112 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 13 (of 84) ]
[ time 113 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 12 (of 84) ]
[ time 114 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 11 (of 84) ]
[ time 115 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 10 (of 84) ]
[ time 116 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 9 (of 84) ]
[ time 117 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 8 (of 84) ]
[ time 118 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 7 (of 84) ]
[ time 119 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 6 (of 84) ]
[ time 120 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 5 (of 84) ]
[ time 121 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 4 (of 84) ]
[ time 122 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 3 (of 84) ]
[ time 123 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 2 (of 84) ]
[ time 124 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 1 (of 84) ]
[ time 125 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 0 (of 84) ]
[ time 126 ] FINISHED JOB 0

Final statistics:
  Job  0: startTime   0 - response   0 - turnaround 126
  Job  1: startTime   0 - response  10 - turnaround  92

  Avg  1: startTime n/a - response 5.00 - turnaround 109.00


7599O@Smile MINGW64 ~/Desktop/ostep-homework-master/cpu-sched-mlfq
$
```

3. 当工作在同一队列时，进行轮转调度工作，因此只需要将 mlfq 调度的队列数设置为 1，所以答案为：./mlfq.py -n 1

5. 假设每隔 T 时间，系统使用-B 标志将所有进程（包括这个被饥饿的）推回最高优先级队列。那么，这个进程每 T 时间可以运行 10ms。那么它获得的 CPU 时间比例是 10ms / T。根据题目要求，这个比例至少要是 5%，也就是：
10ms / T ≥ 5% → T ≤ 10ms / 0.05 → T ≤ 200ms。
所以答案是每 200ms 将工作推回到最高优先级级别。

Chp 9

1.
Seed=1:

随机种子：1　份额：84,25,44　工作长度：1,7,4

时间	job0	job1	job2	随机数	总份额	模	运行
1	0	0	1	651593	153	119	job2
2	1(完成)	0	1	788724	153	9	job0
3		1	1	93859	69	19	job1
4		1	2	28347	69	57	job2
5		1	3	835765	69	37	job2
6		1	4(完成)	432767	69	68	job2
7		2		762280	25	5	job1
8		3		2106	25	6	job1
9		4		445387	25	12	job1
10		5		721540	25	15	job1
11		6		228762	25	12	job1
12		7(完成)		945271	25	21	job1

Seed=2:

随机种子：2　份额：94,73,30　工作长度：9,8,6

时间	job0	job1	job2	随机数	总份额	模	运行
1	0	0	1	605944	197	169	job2
2	1	0	1	606802	197	42	job0
3	2	0	1	581204	197	54	job0
4	2	0	2	158383	197	192	job2
5	3	0	2	430670	197	28	job0
6	3	1	2	393532	197	123	job1
7	4	1	2	723012	197	22	job0
8	4	1	3	994820	197	167	job2
9	5	1	3	949396	197	53	job0
10	6	1	3	544177	197	63	job0
11	7	1	3	444854	197	28	job0
12	7	2	3	268241	197	124	job1
13	8	2	3	35924	197	70	job0
14	9(完成)	2	3	27444	197	61	job0
15		3	3	464894	103	55	job1
16		3	4	318465	103	92	job2
17		4	4	380015	103	48	job1
18		5	4	891790	103	16	job1
19		6	4	525753	103	41	job1
20		6	5	560510	103	87	job2
21		7	5	236123	103	47	job1
22		8(完成)	5	23858	103	65	job1

Seed=3:

随机种子：3　份额：54,60,6　工作长度：2,3,6

时间	job0	job1	job2	随机数	总份额	模	运行
1	0	1	0	13168	120	88	job1
2	0	2	0	837469	120	109	job1
3	1	2	0	259354	120	34	job0
4	1	3(完成)	0	234331	120	91	job1
5	2(完成)		0	995645	60	5	job0
6			1	470263	6	1	job1
7			2	836462	6	2	job1
8			3	476353	6	1	job2
9			4	639068	6	2	job2
10			5	150616	6	4	job2
11			6(完成)	634861	6	1	job2

2.

份额：1,100　工作长度：10

时间	job0	job1	随机数	总份额	模	运行
1	0	1	844422	101	62	job1
2	0	2	757955	101	51	job1
3	0	3	420572	101	8	job1
4	0	4	258917	101	54	job1
5	0	5	511275	101	13	job1
6	0	6	404934	101	25	job1
7	0	7	783799	101	39	job1
8	0	8	303313	101	10	job1
9	0	9	476597	101	79	job1
10	0	10(完成)	583382	101	6	job1
11	1		908113	1	0	job0
12	2		504687	1	0	job0
13	3		281838	1	0	job0
14	4		755804	1	0	job0
15	5		618369	1	0	job0
16	6		250506	1	0	job0
17	7		909747	1	0	job0
18	8		982786	1	0	job0
19	9		810218	1	0	job0

这里工作 0 与工作 1 的彩票数量差距过大了，导致在工作 1 完成之前工作 0 占用 CPU 几乎是不可能的。从模拟结果上看，在工作 1 完成之前，工作 0 没有运行。
在这种情况下，持有份额小的工作响应时间与周转时间非常长，且基本上占用不了 CPU。极有可能会“饿死”。

3.

种子编号	工作0完成时间	工作1完成时间	提前完成时间
0	192	200	8
1	200	196	4
2	200	190	10
3	196	200	4
4	200	199	1
5	200	181	19
6	200	193	7
7	200	185	15
8	200	191	9
9	200	192	8
10	197	200	3
11	196	200	4
12	200	189	11
Average	198.5	193.5	7.9

提前完成的工作所用时间平均比后完成的快了 8 左右，对比工作长度只能保证基本接近公

平，假设以提前完成时间比工作长度为指标，不公平度约为 7.9%。