操作系统-hw1

班级：人工智能2103

学号：202107030125

姓名：姚丁钰

第四章

4.1

用以下标志运行程序：./process-run.py -l 5:100,5:100.CPU 利用率（CPU 使用时间的百分比）应该是多少？为什么你知道这一点？利用 - c 标记查看你的答案是否正确。

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/04.第四章-抽象:进程$ ./process-run.py -l 5:100,5:100
Produce a trace of what would happen when you run these processes:
Process 0
  cpu
  cpu
  cpu
  cpu
  cpu

Process 1
  cpu
  cpu
  cpu
  cpu
  cpu

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

如本章所述，进程可以处于几种不同的状态：

运行(RUNNING) - 进程正在使用CPU
就绪(READY) - 进程现在可以使用CPU，但是其他进程正在使用
等待(WAITING) - 进程正在等待I / O完成(例如，它向磁盘发出请求)
完成(DONE) - 过程执行完毕

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/04.第四章-抽象:进程$ ./process-run.py -l 5:100,5:100 -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     RUN:cpu      READY          1
  6        DONE    RUN:cpu          1
  7        DONE    RUN:cpu          1
  8        DONE    RUN:cpu          1
  9        DONE    RUN:cpu          1
 10        DONE    RUN:cpu          1
```

答：CPU 利用率为 100%,程序没有进行 IO

4.2

现在用这些标志运行： ./process-run.py -l 4:100,1:0 这些标志指定了一个包含 4 条指令的进程(都要使用 CPU),并且只是简单地发出 IO 并等待它完成。完成这两个进程需要多长时间?利用 c 检查你的答案是否正确。

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/04.第四章-抽象:进程$ ./process-run.py -l 4:100,1:0
Produce a trace of what would happen when you run these processes:
Process 0
  cpu
  cpu
  cpu
  cpu

Process 1
  io

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)

```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/04.第四章-抽象:进程$ ./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    WAITING                    1
  7        DONE    WAITING                    1
  8        DONE    WAITING                    1
  9        DONE    WAITING                    1
 10*       DONE       DONE
```

答：完成这两个进程需要10单位的时间。首先我们可以知道命令行`./process-run.py -l 4:100,1:0`中的`4:100,1:0`表示CPU中一共有两个进程，先运行的进程4:100一共有4条指令，每条指令的CPU占用率都是100%，一共耗去4单位时间。随后的进程1:0表示仅有一条I/O指令，又因为`README.md`文档已经提示每条I/O指令的默认运行时长为5单位时间（包括了CPU发出指令即将进行I/O操作阻塞进程占去的1单位时间，实际I/O操作等待了4单位时间）。最后I/O操作完毕CPU再解除该进程的阻塞耗去1单位时间。综上一共需要10单位时间。

4.3

现在交换进程的顺序： ./process-run.py -l 1:0,4:100 现在发生了什么？交换顺序是否重要？为什么？同样，用 -c 看看你的答案是否正确。

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/04.第四章-抽象:进程$ ./process-run.py -l 1:0,4:100
Produce a trace of what would happen when you run these processes:
Process 0
  io

Process 1
  cpu
  cpu
  cpu
  cpu

Important behaviors:
```

```
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)

cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/04.第四章-抽象:进程$ ./process-run.py -l 1:0,4:100  -c
Time     PID: 0     PID: 1        CPU        IOs
  1      RUN:io      READY          1
  2     WAITING    RUN:cpu          1          1
  3     WAITING    RUN:cpu          1          1
  4     WAITING    RUN:cpu          1          1
  5     WAITING    RUN:cpu          1          1
  6*       DONE       DONE
```

答：进程 1 （PID1）执行 IO 时,进程 2 使用 CPU，运行时间由 10 变为 6 个时钟周期

4.4

现在探索另一些标志。一个重要的标志是 -S，它决定了当进程发出 IO 时系统如何反应。将标志设置为 SWITCH_ON_END，在进程进行 I/O 操作时,系统将不会切换到另一个进程,而是等待进程完成。当你运行以下两个进程时，会发生什么情况？一个执行 I/O，另一个执行 CPU 工作。 (-l 1:0,4:100 -c -S SWITCH_ON_END)

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/04.第四章-抽象:进程$ ./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     WAITING      READY                    1
  3     WAITING      READY                    1
  4     WAITING      READY                    1
  5     WAITING      READY                    1
  6*       DONE    RUN:cpu          1
  7        DONE    RUN:cpu          1
  8        DONE    RUN:cpu          1
```

答：进程 1 执行 IO 操作时，进程 2 等待

4.5

现在,运行相同的进程，但切换行为设置，在等待 IO 时切换到另一个进程（-l 1:0,4:100 c-S- SWITCH_ON_IO）现在会发生什么？利用 -c 来确认你的答案是否正确。

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/04.第四章-抽象:进程$ ./process-run.py -l 1:0,4:100 -S SWITCH_ON_IO
Produce a trace of what would happen when you run these processes:
Process 0
  io

Process 1
  cpu
  cpu
  cpu
  cpu

Important behaviors:
  System will switch when the current process is FINISHED or ISSUES AN IO
  After IOs, the process issuing the IO will run LATER (when it is its turn)
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/04.第四章-抽象:进程$ ./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     WAITING    RUN:cpu          1          1
  3     WAITING    RUN:cpu          1          1
  4     WAITING    RUN:cpu          1          1
  5     WAITING    RUN:cpu          1          1
  6*       DONE       DONE
```

答：进程 1 执行 IO 操作时，进程 2 执行,充分利用 CPU 资源

第五章

5.1

编写一个调用 fork()的程序。在调用之前,让主进程访问一个变量(例如 x)并将其值设置为某个值(例如 100)。子进程中的变量有什么值?当子进程和父进程都改变 x 的值时,变量会发生什么?

```c
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>

int main()
{
    int x = 100;
    printf("x = %d, (pid:%d)\n", x, (int)getpid());
    int rc = fork();
    if (rc < 0) {
        fprintf(stderr, "fork failed");
        exit(1);
    } else if (rc == 0) {
        printf("child x = %d (pid:%d)\n", x, (int)getpid());
        x = 200;
        printf("child after changed x = %d (pid:%d)\n", x,
(int)getpid());
    } else {
        printf("parent x = %d (pid:%d)\n", x, (int)getpid());
        x = 300;
        printf("parent after changed x = %d (pid:%d)\n", x,
(int)getpid());
    }
    return 0;
}
```

```
M2021-YDY@hndx-R8424-G12:~/code/Operating-Systems-Three-Easy-
Pieces-NOTES-main/05.第五章-插叙_进程API/code$ gcc -o 1 1.c
M2021-YDY@hndx-R8424-G12:~/code/Operating-Systems-Three-Easy-
Pieces-NOTES-main/05.第五章-插叙_进程API/code$ ./1
x = 100, (pid:2697709)
parent x = 100 (pid:2697709)
parent after changed x = 300 (pid:2697709)
child x = 100 (pid:2697710)
child after changed x = 200 (pid:2697710)
```

答：子进程的变量子一开始与父进程相同。但是实际上父子进程中的同名变量已经不在同一内存区域，实际上是两个变量，因此父子进程的变量值改变不会影响另外一个进程。

5.2

编写一个打开文件的程序(使用 open 系统调用),然后调用 fork 创建一个新进程。子进程和父进程都可以访问 open()返回的文件描述符吗?当它们并发(即同时)写入文件时,会发生什么

```c
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <unistd.h>

int main() {
    int fd = open("./check.txt", O_CREAT | O_RDWR | O_TRUNC, S_IRWXU);
    int pid = fork();
    if (pid < 0) {
        printf("fork error\n");
        exit(1);
    } else if (pid == 0) {
        char *buf = "child\n";
        int error = write(fd, buf, sizeof(char) * strlen(buf));
        printf("child error: %d\n", error == -1 ? 1 : 0);

    } else {
        char *buf = "parent\n";
        int error = write(fd, buf, sizeof(char) * strlen(buf));
        printf("child error: %d\n", error == -1 ? 1 : 0);

        int wc = wait(NULL);
        close(fd);
    }
    return 0;
}
```

```
M2021-YDY@hndx-R8424-G12:~/code/Operating-Systems-Three-Easy-Pieces-NOTES-main/05.第五章-插叙_进程API/code$ gcc -o 2 2.c
M2021-YDY@hndx-R8424-G12:~/code/Operating-Systems-Three-Easy-Pieces-NOTES-main/05.第五章-插叙_进程API/code$ cat check.txt
parent
child
M2021-YDY@hndx-R8424-G12:~/code/Operating-Systems-Three-Easy-Pieces-NOTES-main/05.第五章-插叙_进程API/code$ ./2
child error: 0
child error: 0
```

答：子进程和父进程都能访问 fd。存在竞争条件，无法同时使用 fd，但最终都会写入成功

5.4

编写一个调用 fork()的程序,然后调用某种形式的 exec()来运行程序"/bin/ls"看看是否可以尝试 exec 的所有变体,包括 execl()、 execle()、 execlp()、 execv()、 execvp()和 execve(),为什么同样的基本调用会有这么多变种?

```c
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>

int flag = 0;
const int MAX = 6;
int main()
{
    char * s = "/bin/ls";
    char * ss = "ls";
    char * s2 = "/";
    char * sv[] = { ss, s2, NULL };
    for(flag = 0;flag < MAX; ++flag) {
        int rc = fork();
        if (rc < 0) {
            fprintf(stderr, "fork failed");
            exit(1);
        } else if (rc == 0) {
            switch(flag) {
            case 0:
                execl(s, ss, s2, NULL);
                break;
            case 1:
                execle(s, ss, s2, NULL);
```

```
                break;
            case 2:
                execlp(s, s, s2, NULL);
                break;
            case 3:
                execv(s, sv);
                break;
            case 4:
                execvp(ss, sv);
                break;
            case 5:
                execvpe(ss, sv);
                break;
            default: break;
            }
        } else {
            wait(NULL);
        }
    }
    return 0;
}
```

```
M2021-YDY@hndx-R8424-G12:~/code/Operating-Systems-Three-Easy-Pieces-NOTES-main/05.第五章-插叙_进程API/code$ ./4
bin   cdrom  etc   home_old  lib32  libx32      media  opt   root  sbin  srv       sys  usr
boot  dev    home  lib       lib64  lost+found  mnt    proc  run   snap  swapfile  tmp  var
bin   cdrom  etc   home_old  lib32  libx32      media  opt   root  sbin  srv       sys  usr
boot  dev    home  lib       lib64  lost+found  mnt    proc  run   snap  swapfile  tmp  var
bin   cdrom  etc   home_old  lib32  libx32      media  opt   root  sbin  srv       sys  usr
boot  dev    home  lib       lib64  lost+found  mnt    proc  run   snap  swapfile  tmp  var
bin   cdrom  etc   home_old  lib32  libx32      media  opt   root  sbin  srv       sys  usr
boot  dev    home  lib       lib64  lost+found  mnt    proc  run   snap  swapfile  tmp  var
bin   cdrom  etc   home_old  lib32  libx32      media  opt   root  sbin  srv       sys  usr
boot  dev    home  lib       lib64  lost+found  mnt    proc  run   snap  swapfile  tmp  var
bin   cdrom  etc   home_old  lib32  libx32      media  opt   root  sbin  srv       sys  usr
boot  dev    home  lib       lib64  lost+found  mnt    proc  run   snap  swapfile  tmp  var
```

答：有多种变体是为了要适应不同的调用形式和环境要求。

在 exec 函数族中，后缀 l、v、p、e 添加到 exec 后，所指定的函数将具有某种操作能力：

- l: 希望接收以逗号分隔的参数列表,列表以 NULL 指针作为结束标志
- v: 希望接收一个以 NULL 结尾字符串数组的指针
- p: 是一个以 NULL 结尾的字符串数组指针,函数可以利用 DOS 的 PATH 变量查找自程序文件
- e 函数传递指定采纳数 envp(环境变量),允许改变子进程环境,无后缀 e 是,子进程使用当前程序环境
- c 语言没有默认参数语法,只能实现多个变体

第七章

7.1

使用 SJF 和 FIFO 调度程序运行长度为 200 的 3 个作业时,计算响应时间和周转时间。

```
.py -p FIFO -l 200,200,200 -c
ARG policy FIFO
ARG jlist 200,200,200

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


** Solutions **

Execution trace:
  [ time   0 ] Run job 0 for 200.00 secs ( DONE at 200.00 )
  [ time 200 ] Run job 1 for 200.00 secs ( DONE at 400.00 )
  [ time 400 ] Run job 2 for 200.00 secs ( DONE at 600.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 200.00  Wait 0.00
  Job   1 -- Response: 200.00  Turnaround 400.00  Wait 200.00
  Job   2 -- Response: 400.00  Turnaround 600.00  Wait 400.00

  Average -- Response: 200.00  Turnaround 400.00  Wait 200.00
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/07.第七章-进程调度:介绍$ ./scheduler.py -p SJF -l 200,200,200 -c
ARG policy SJF
ARG jlist 200,200,200

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


** Solutions **

```

```
Execution trace:
  [ time   0 ] Run job 0 for 200.00 secs ( DONE at 200.00 )
  [ time 200 ] Run job 1 for 200.00 secs ( DONE at 400.00 )
  [ time 400 ] Run job 2 for 200.00 secs ( DONE at 600.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 200.00  Wait 0.00
  Job   1 -- Response: 200.00  Turnaround 400.00  Wait 200.00
  Job   2 -- Response: 400.00  Turnaround 600.00  Wait 400.00

  Average -- Response: 200.00  Turnaround 400.00  Wait 200.00
```

SJF:

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

FIFO:

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

调度策略	平均响应时间	平均周转时间
SJF	200	400
FIFO	200	400

7.2

现在做同样的事情,但有不同长度的作业,即 100、200 和 300

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/07.第七章-进程调度:介绍$ ./scheduler.py -p FIFO -l 100,200,300 -c
ARG policy FIFO
```

```
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
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/07.第七章-进程调度:介绍$ ./scheduler.py -p SJF -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
```

```
22
23    Average -- Response: 133.33  Turnaround 333.33  Wait 133.33
```

SJF:

作业 id	响应时间	周转时间
0	0	100
1	100	300
2	300	600

FIFO:

作业 id	响应时间	周转时间
0	0	100
1	100	300
2	300	600

调度策略	平均响应时间	平均周转时间
SJF	400/3	1000/3
FIFO	400/3	1000/3

7.3

现在做同样的事情,但采用 RR 调度程序,时间片为 1

```
.......
  [ time 590 ] Run job   2 for 1.00 secs
  [ time 591 ] Run job   2 for 1.00 secs
  [ time 592 ] Run job   2 for 1.00 secs
  [ time 593 ] Run job   2 for 1.00 secs
  [ time 594 ] Run job   2 for 1.00 secs
  [ time 595 ] Run job   2 for 1.00 secs
  [ time 596 ] Run job   2 for 1.00 secs
  [ time 597 ] Run job   2 for 1.00 secs
  [ time 598 ] Run job   2 for 1.00 secs
  [ time 599 ] Run job   2 for 1.00 secs ( DONE at 600.00 )

Final statistics:
```

```
  Job   0 -- Response: 0.00  Turnaround 298.00  Wait 198.00
  Job   1 -- Response: 1.00  Turnaround 499.00  Wait 299.00
  Job   2 -- Response: 2.00  Turnaround 600.00  Wait 300.00

  Average -- Response: 1.00  Turnaround 465.67  Wait 265.67
```

- 长度为 200:

RR:

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

平均响应时间 = 1
平均周转时间 = 599

- 长度为 100, 200, 300:

RR:

作业 id	响应时间	周转时间
0	0	298
1	1	499
2	2	600

平均响应时间 = 1
平均周转时间 = 456.67

7.4

对于什么类型的工作负载,SJF 提供与 FIFO 相同的周转时间?

答：运行时间相同的任务

7.5

对于什么类型的工作负载和量子长度(时间片长度),SJF 与 RR 提供相同的响应时间?

答：运行时间 <= 时间片

7.6

随着工作长度的增加,SJF 的响应时间会怎样?你能使用模拟程序来展示趋势吗?

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/07.第七章-进程调度:介绍$
./scheduler.py -p SJF -l 100,100,100 -c
ARG policy SJF
ARG jlist 100,100,100

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


** Solutions **

Execution trace:
  [ time   0 ] Run job 0 for 100.00 secs ( DONE at 100.00 )
  [ time 100 ] Run job 1 for 100.00 secs ( DONE at 200.00 )
  [ time 200 ] Run job 2 for 100.00 secs ( DONE at 300.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 100.00  Wait 0.00
  Job   1 -- Response: 100.00  Turnaround 200.00  Wait 100.00
  Job   2 -- Response: 200.00  Turnaround 300.00  Wait 200.00

  Average -- Response: 100.00  Turnaround 200.00  Wait 100.00
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/07.第七章-进程调度:介绍$
./scheduler.py -p SJF -l 200,200,200 -c
ARG policy SJF
ARG jlist 200,200,200

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


** Solutions **

```

```
Execution trace:
  [ time   0 ] Run job 0 for 200.00 secs ( DONE at 200.00 )
  [ time 200 ] Run job 1 for 200.00 secs ( DONE at 400.00 )
  [ time 400 ] Run job 2 for 200.00 secs ( DONE at 600.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 200.00  Wait 0.00
  Job   1 -- Response: 200.00  Turnaround 400.00  Wait 200.00
  Job   2 -- Response: 400.00  Turnaround 600.00  Wait 400.00

  Average -- Response: 200.00  Turnaround 400.00  Wait 200.00
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/07.第七章-进程调度:介绍$
./scheduler.py -p SJF -l 300,300,300 -c
ARG policy SJF
ARG jlist 300,300,300

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


** Solutions **

Execution trace:
  [ time   0 ] Run job 0 for 300.00 secs ( DONE at 300.00 )
  [ time 300 ] Run job 1 for 300.00 secs ( DONE at 600.00 )
  [ time 600 ] Run job 2 for 300.00 secs ( DONE at 900.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 300.00  Wait 0.00
  Job   1 -- Response: 300.00  Turnaround 600.00  Wait 300.00
  Job   2 -- Response: 600.00  Turnaround 900.00  Wait 600.00

  Average -- Response: 300.00  Turnaround 600.00  Wait 300.00
```

答：因为SJF是非抢占式的策略，所以每个作业开始响应必须等待之前的作业执行完成，当作业工作长度增加了，后面的作业等待响应时间也会相应增加。

7.7

随着量子长度(时间片长度)的增加,RR 的响应时间会怎样?你能写出一个方程,计算给定 N 个工作时,最坏情况的响应时间吗?

答：平均响应时间增加

时间片 >= 工作最长运行时间时,有最坏情况
即 t_i 为作业运行时间
$res_{avg} = (0 + t_1 + (t_1 + t_2) + (t_1 + t_2 + t_3) + \ldots (t_1 + t_2 + t_3 + \ldots t_{N-1}))/N$

第八章

8.1

只用两个工作和两个队列运行几个随机生成的问题。针对每个工作计算 MLFQ 的执行记录。限制每项作业的长度并关闭 I/O,让你的生活更轻松。

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/08.第八章-调度:多级反馈队列$ ./mlfq.py -j 2 -m 10 -M 0 -n 2 -q 3 -c
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   3
OPTIONS allotments for queue  0 is   1
OPTIONS quantum length for queue  0 is   3
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   8 - ioFreq   0
  Job  1: startTime   0 - runTime   4 - ioFreq   0


```

```
Execution Trace:

[ time 0 ] JOB BEGINS by JOB 0
[ time 0 ] JOB BEGINS by JOB 1
[ time 0 ] Run JOB 0 at PRIORITY 1 [ TICKS 2 ALLOT 1 TIME 7 (of 8) ]
[ time 1 ] Run JOB 0 at PRIORITY 1 [ TICKS 1 ALLOT 1 TIME 6 (of 8) ]
[ time 2 ] Run JOB 0 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 5 (of 8) ]
[ time 3 ] Run JOB 1 at PRIORITY 1 [ TICKS 2 ALLOT 1 TIME 3 (of 4) ]
[ time 4 ] Run JOB 1 at PRIORITY 1 [ TICKS 1 ALLOT 1 TIME 2 (of 4) ]
[ time 5 ] Run JOB 1 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 1 (of 4) ]
[ time 6 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 4 (of 8) ]
[ time 7 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 3 (of 8) ]
[ time 8 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 2 (of 8) ]
[ time 9 ] Run JOB 1 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 0 (of 4) ]
[ time 10 ] FINISHED JOB 1
[ time 10 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 1 (of 8) ]
[ time 11 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 0 (of 8) ]
[ time 12 ] FINISHED JOB 0

Final statistics:
  Job  0: startTime   0 - response   0 - turnaround  12
  Job  1: startTime   0 - response   3 - turnaround  10

  Avg  1: startTime n/a - response 1.50 - turnaround 11.00
```

使用 -h 参数查看帮助：

```
Usage: mlfq.py [options]
Options:
  -h, --help            显示此帮助信息并退出
  -s SEED, --seed=SEED  指定随机种子
  -n NUMQUEUES, --numQueues=NUMQUEUES
                        MLFQ中的队列数（如果没有使用-Q）
  -q QUANTUM, --quantum=QUANTUM
                        时间片长度(如果没有使用-Q参数)
  -Q QUANTUMLIST, --quantumList=QUANTUMLIST
                        指定为x，y，z，...为每个队列级别的时间片长
度，
                        其中x是优先级最高的队列的时间片长度，y是第二
高的队列的时间片长度，依此类推
  -j NUMJOBS, --numJobs=NUMJOBS
                        系统中的作业数
  -m MAXLEN, --maxlen=MAXLEN
                        作业的最大运行时间（如果是随机的）
  -M MAXIO, --maxio=MAXIO
                        作业的最大I/O频率（如果是随机的）
  -B BOOST, --boost=BOOST
                        将所有作业的优先级提高到高优先级的频率（0表示
从不）
  -i IOTIME, --iotime=IOTIME
                        I/O 持续时间（固定常数)
  -S, --stay            发出I/O时重置并保持相同的优先级
  -l JLIST, --jlist=JLIST
                        以逗号分隔的要运行的作业列表，格式为x1，y1，
z1：x2，y2，z2：...。
                        其中x是开始时间，y是运行时间，z是作业I/O的
频率
  -c                    计算答案
]
```

对于每个作业，给出了三个特征值：

- tartTime：该作业在什么时候进入系统
- runTime：任务完成所需的总 CPU 时间
- ioFreq：I/O 需要 ioTime 时间来完成

```
Job List:
  Job  0: startTime   0 - runTime   8 - ioFreq   0
  Job  1: startTime   0 - runTime   4 - ioFreq   0
```

执行情况：

0 作业0、1到达，运行作业0的1时间单位
1 运行作业0的1时间单位
2 运行作业0的1时间单位
3 运行作业1的1时间单位
4 运行作业1的1时间单位
5 运行作业1的1时间单位
6 运行作业0的1时间单位
7 运行作业0的1时间单位
8 运行作业0的1时间单位
9 运行作业1的1时间单位
10 作业1完成 运行作业0的1时间单位
11 运行作业0的1时间单位
12 作业0完成

8.3

将如何配置调度程序参数，像轮转调度程序那样工作?

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/08.第八章-调度:多级反馈队列$ ./mlfq.py -l 0,50,0:0,50,0 -n 1 -q 10 -c
Here is the list of inputs:
OPTIONS jobs 2
OPTIONS queues 1
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  50 - ioFreq   0
  Job  1: startTime   0 - runTime  50 - ioFreq   0


Execution Trace:

[ time 0 ] JOB BEGINS by JOB 0
[ time 0 ] JOB BEGINS by JOB 1
[ time 0 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 49
(of 50) ]
[ time 1 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 48
(of 50) ]
[ time 2 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 47
(of 50) ]
[ time 3 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 46
(of 50) ]
[ time 4 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 45
(of 50) ]
[ time 5 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 44
(of 50) ]
[ time 6 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 43
(of 50) ]
[ time 7 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 42
(of 50) ]
[ time 8 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 41
(of 50) ]
[ time 9 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 40
(of 50) ]
[ time 10 ] Run JOB 1 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 49
(of 50) ]
[ time 11 ] Run JOB 1 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 48
(of 50) ]
[ time 12 ] Run JOB 1 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 47
(of 50) ]
[ time 13 ] Run JOB 1 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 46
(of 50) ]
[ time 14 ] Run JOB 1 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 45
(of 50) ]
[ time 15 ] Run JOB 1 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 44
(of 50) ]
[ time 16 ] Run JOB 1 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 43
(of 50) ]
```

```
[ time 17 ] Run JOB 1 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 42 (of 50) ]
[ time 18 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 41 (of 50) ]
[ time 19 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 40 (of 50) ]
[ time 20 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 39 (of 50) ]
[ time 21 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 38 (of 50) ]
[ time 22 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 37 (of 50) ]
[ time 23 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 36 (of 50) ]
[ time 24 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 35 (of 50) ]
[ time 25 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 34 (of 50) ]
[ time 26 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 33 (of 50) ]
[ time 27 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 32 (of 50) ]
[ time 28 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 31 (of 50) ]
[ time 29 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 30 (of 50) ]
[ time 30 ] Run JOB 1 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 39 (of 50) ]
[ time 31 ] Run JOB 1 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 38 (of 50) ]
[ time 32 ] Run JOB 1 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 37 (of 50) ]
[ time 33 ] Run JOB 1 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 36 (of 50) ]
[ time 34 ] Run JOB 1 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 35 (of 50) ]
[ time 35 ] Run JOB 1 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 34 (of 50) ]
[ time 36 ] Run JOB 1 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 33 (of 50) ]
[ time 37 ] Run JOB 1 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 32 (of 50) ]
[ time 38 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 31 (of 50) ]
```

```
[ time 39 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 30 (of 50) ]
[ time 40 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 29 (of 50) ]
[ time 41 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 28 (of 50) ]
[ time 42 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 27 (of 50) ]
[ time 43 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 26 (of 50) ]
[ time 44 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 25 (of 50) ]
[ time 45 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 24 (of 50) ]
[ time 46 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 23 (of 50) ]
[ time 47 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 22 (of 50) ]
[ time 48 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 21 (of 50) ]
[ time 49 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 20 (of 50) ]
[ time 50 ] Run JOB 1 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 29 (of 50) ]
[ time 51 ] Run JOB 1 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 28 (of 50) ]
[ time 52 ] Run JOB 1 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 27 (of 50) ]
[ time 53 ] Run JOB 1 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 26 (of 50) ]
[ time 54 ] Run JOB 1 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 25 (of 50) ]
[ time 55 ] Run JOB 1 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 24 (of 50) ]
[ time 56 ] Run JOB 1 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 23 (of 50) ]
[ time 57 ] Run JOB 1 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 22 (of 50) ]
[ time 58 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 21 (of 50) ]
[ time 59 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 20 (of 50) ]
[ time 60 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 19 (of 50) ]
```

```
 89  [ time 61 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 18
     (of 50) ]
 90  [ time 62 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 17
     (of 50) ]
 91  [ time 63 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 16
     (of 50) ]
 92  [ time 64 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 15
     (of 50) ]
 93  [ time 65 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 14
     (of 50) ]
 94  [ time 66 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 13
     (of 50) ]
 95  [ time 67 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 12
     (of 50) ]
 96  [ time 68 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 11
     (of 50) ]
 97  [ time 69 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 10
     (of 50) ]
 98  [ time 70 ] Run JOB 1 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 19
     (of 50) ]
 99  [ time 71 ] Run JOB 1 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 18
     (of 50) ]
100  [ time 72 ] Run JOB 1 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 17
     (of 50) ]
101  [ time 73 ] Run JOB 1 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 16
     (of 50) ]
102  [ time 74 ] Run JOB 1 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 15
     (of 50) ]
103  [ time 75 ] Run JOB 1 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 14
     (of 50) ]
104  [ time 76 ] Run JOB 1 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 13
     (of 50) ]
105  [ time 77 ] Run JOB 1 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 12
     (of 50) ]
106  [ time 78 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 11
     (of 50) ]
107  [ time 79 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 10
     (of 50) ]
108  [ time 80 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 9
     (of 50) ]
109  [ time 81 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 8
     (of 50) ]
110  [ time 82 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 7
     (of 50) ]
```

```
[ time 83 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 6 (of 50) ]
[ time 84 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 5 (of 50) ]
[ time 85 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 4 (of 50) ]
[ time 86 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 3 (of 50) ]
[ time 87 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 2 (of 50) ]
[ time 88 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 1 (of 50) ]
[ time 89 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 0 (of 50) ]
[ time 90 ] FINISHED JOB 0
[ time 90 ] Run JOB 1 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 9 (of 50) ]
[ time 91 ] Run JOB 1 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 8 (of 50) ]
[ time 92 ] Run JOB 1 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 7 (of 50) ]
[ time 93 ] Run JOB 1 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 6 (of 50) ]
[ time 94 ] Run JOB 1 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 5 (of 50) ]
[ time 95 ] Run JOB 1 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 4 (of 50) ]
[ time 96 ] Run JOB 1 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 3 (of 50) ]
[ time 97 ] Run JOB 1 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 2 (of 50) ]
[ time 98 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 1 (of 50) ]
[ time 99 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 0 (of 50) ]
[ time 100 ] FINISHED JOB 1

Final statistics:
  Job  0: startTime   0 - response   0 - turnaround  90
  Job  1: startTime   0 - response  10 - turnaround 100

  Avg  1: startTime n/a - response 5.00 - turnaround 95.00
```

答：队列数设为 1 `./mlfq.py -l 0,50,0:0,50,0 -n 1 -q 10 -c`

8.5

给定一个系统，其最高队列中的时间片长度为 10ms,你需要如何频繁地将工作推回到最高优先级级别（带有-B 标志），以保证一个长时间运行（并可能饥饿）的工作得到至少 5%的 CPU?

```
.....
[ time 382 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 16 (of 200) ]
[ time 383 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 15 (of 200) ]
[ time 384 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 14 (of 200) ]
[ time 385 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 13 (of 200) ]
[ time 386 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 12 (of 200) ]
[ time 387 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 11 (of 200) ]
[ time 388 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 10 (of 200) ]
[ time 389 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 9 (of 200) ]
[ time 390 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 8 (of 200) ]
[ time 391 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 7 (of 200) ]
[ time 392 ] Run JOB 1 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 0 (of 100) ]
[ time 393 ] FINISHED JOB 1
[ time 393 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 6 (of 200) ]
[ time 394 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 5 (of 200) ]
[ time 395 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 4 (of 200) ]
[ time 396 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 3 (of 200) ]
[ time 397 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 2 (of 200) ]
[ time 398 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 1 (of 200) ]
[ time 399 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 0 (of 200) ]
```

```
[ time 400 ] FINISHED JOB 0

Final statistics:
  Job  0: startTime   0 - response   0 - turnaround 400
  Job  1: startTime   0 - response  10 - turnaround 393
  Job  2: startTime   0 - response  15 - turnaround 368

  Avg  2: startTime n/a - response 8.33 - turnaround 387.00
```

答：发生“饥饿”的原因是MLFQ的规则当一个作业在其时间片内主动放弃CPU，那么它的优先级不变。这样当有很多交互程序时，每个交互程序运行一段时间后（不足一个时间片）主动放弃CPU发起I/O，I/O持续时间内另一个作业刚好也运行不足一个时间片，随后当第一个作业完成I/O时第二个作业发出I/O，两个作业的优先级都不会降低且一直切换霸占CPU导致其他作业被“饿死”。所以需要每隔一段时间boost提升所有作业到最高优先级队列。此时长时间运行作业可以占用CPU一个时间片，随后降低优先级，继续“饥饿”直到下一次boost又占用CPU一个时间片。若最高队列时间片长度为10ms，切要保证一个长时间运行作业得到至少5%的CPU，那么在一个boost时间内它就要获得5%的CPU，这个时间也是一个时间片长度，所以T(boost) = T(time slice) / 0.05 = 10 / 0.05 = 200 ms。综上，将作业推回到最高优先级的时间频率不能大于200 ms。

`./mlfq.py -l 0,200,0:0,100,5:0,100,5 -q 10 -n 3 -i 5 -B 100 -c`

第九章

9.1

计算 3 个工作在随机种子为 1、2 和 3 时的模拟解。

- 随机数种子为1

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/09.第九章-调度:比例份额$ ./lottery.py -j 3 -s 1
ARG jlist
ARG jobs 3
ARG maxlen 10
ARG maxticket 100
ARG quantum 1
ARG seed 1

Here is the job list, with the run time of each job:
  Job 0 ( length = 1, tickets = 84 )
  Job 1 ( length = 7, tickets = 25 )
  Job 2 ( length = 4, tickets = 44 )
```

```

Here is the set of random numbers you will need (at most):
Random 651593
Random 788724
Random 93859
Random 28347
Random 835765
Random 432767
Random 762280
Random 2106
Random 445387
Random 721540
Random 228762
Random 945271
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/09.第九章-调度:比例份额$ ./lottery.py -j 3 -s 1 -c
ARG jlist
ARG jobs 3
ARG maxlen 10
ARG maxticket 100
ARG quantum 1
ARG seed 1

Here is the job list, with the run time of each job:
  Job 0 ( length = 1, tickets = 84 )
  Job 1 ( length = 7, tickets = 25 )
  Job 2 ( length = 4, tickets = 44 )


** Solutions **

Random 651593 -> Winning ticket 119 (of 153) -> Run 2
  Jobs:  (  job:0 timeleft:1 tix:84 )   (  job:1 timeleft:7 tix:25 )   (* job:2 timeleft:4 tix:44 )
Random 788724 -> Winning ticket 9 (of 153) -> Run 0
  Jobs:  (* job:0 timeleft:1 tix:84 )   (  job:1 timeleft:7 tix:25 )   (  job:2 timeleft:3 tix:44 )
--> JOB 0 DONE at time 2
Random 93859 -> Winning ticket 19 (of 69) -> Run 1
  Jobs:  (  job:0 timeleft:0 tix:--- )   (* job:1 timeleft:7 tix:25 )   (  job:2 timeleft:3 tix:44 )
Random 28347 -> Winning ticket 57 (of 69) -> Run 2
```

```
  Jobs:  (  job:0 timeleft:0 tix:--- )   (  job:1 timeleft:6 tix:25 )   (* job:2 timeleft:3 tix:44 )
Random 835765 -> Winning ticket 37 (of 69) -> Run 2
  Jobs:  (  job:0 timeleft:0 tix:--- )   (  job:1 timeleft:6 tix:25 )   (* job:2 timeleft:2 tix:44 )
Random 432767 -> Winning ticket 68 (of 69) -> Run 2
  Jobs:  (  job:0 timeleft:0 tix:--- )   (  job:1 timeleft:6 tix:25 )   (* job:2 timeleft:1 tix:44 )
--> JOB 2 DONE at time 6
Random 762280 -> Winning ticket 5 (of 25) -> Run 1
  Jobs:  (  job:0 timeleft:0 tix:--- )   (* job:1 timeleft:6 tix:25 )   (  job:2 timeleft:0 tix:--- )
Random 2106 -> Winning ticket 6 (of 25) -> Run 1
  Jobs:  (  job:0 timeleft:0 tix:--- )   (* job:1 timeleft:5 tix:25 )   (  job:2 timeleft:0 tix:--- )
Random 445387 -> Winning ticket 12 (of 25) -> Run 1
  Jobs:  (  job:0 timeleft:0 tix:--- )   (* job:1 timeleft:4 tix:25 )   (  job:2 timeleft:0 tix:--- )
Random 721540 -> Winning ticket 15 (of 25) -> Run 1
  Jobs:  (  job:0 timeleft:0 tix:--- )   (* job:1 timeleft:3 tix:25 )   (  job:2 timeleft:0 tix:--- )
Random 228762 -> Winning ticket 12 (of 25) -> Run 1
  Jobs:  (  job:0 timeleft:0 tix:--- )   (* job:1 timeleft:2 tix:25 )   (  job:2 timeleft:0 tix:--- )
Random 945271 -> Winning ticket 21 (of 25) -> Run 1
  Jobs:  (  job:0 timeleft:0 tix:--- )   (* job:1 timeleft:1 tix:25 )   (  job:2 timeleft:0 tix:--- )
--> JOB 1 DONE at time 12
```

随机数序列为：651593,788724, 93859, 28347,835765,432767,762280,2106,445387,721540,228762,945271

根据作业长度及随机数序列推得作业票号变化：

Job 0(0-83) Job 1(84-108) Job 2(109-152)

Job 0(null) Job 1(0-24) Job 2(25-68)

Job 0(null) Job 1(0-24) Job 2(null)

又因为随机数对总票数取余就是"中奖"的作业，中奖号码序列为：119，9，19，57，37，68，5，6，12，15，12，21

运行作业序列：2 0 1 2 2 2 1 1 1 1 1 1

- 随机数种子为2

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/09.第九章-调度:比例份额$ ./lottery.py -j 3 -s 2
ARG jlist
ARG jobs 3
ARG maxlen 10
ARG maxticket 100
ARG quantum 1
ARG seed 2

Here is the job list, with the run time of each job:
  Job 0 ( length = 9, tickets = 94 )
  Job 1 ( length = 8, tickets = 73 )
  Job 2 ( length = 6, tickets = 30 )


Here is the set of random numbers you will need (at most):
Random 605944
Random 606802
Random 581204
Random 158383
Random 430670
Random 393532
Random 723012
Random 994820
Random 949396
Random 544177
Random 444854
Random 268241
Random 35924
Random 27444
Random 464894
Random 318465
Random 380015
Random 891790
Random 525753
Random 560510
Random 236123
Random 23858
Random 325143
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/09.第九章-调度:比例份额$ ./lottery.py -j 3 -s 2 -c
ARG jlist
```

```
ARG jobs 3
ARG maxlen 10
ARG maxticket 100
ARG quantum 1
ARG seed 2

Here is the job list, with the run time of each job:
  Job 0 ( length = 9, tickets = 94 )
  Job 1 ( length = 8, tickets = 73 )
  Job 2 ( length = 6, tickets = 30 )


** Solutions **

Random 605944 -> Winning ticket 169 (of 197) -> Run 2
  Jobs:  (  job:0 timeleft:9 tix:94 )   (  job:1 timeleft:8 tix:73 )   (* job:2 timeleft:6 tix:30 )
Random 606802 -> Winning ticket 42 (of 197) -> Run 0
  Jobs:  (* job:0 timeleft:9 tix:94 )   (  job:1 timeleft:8 tix:73 )   (  job:2 timeleft:5 tix:30 )
Random 581204 -> Winning ticket 54 (of 197) -> Run 0
  Jobs:  (* job:0 timeleft:8 tix:94 )   (  job:1 timeleft:8 tix:73 )   (  job:2 timeleft:5 tix:30 )
Random 158383 -> Winning ticket 192 (of 197) -> Run 2
  Jobs:  (  job:0 timeleft:7 tix:94 )   (  job:1 timeleft:8 tix:73 )   (* job:2 timeleft:5 tix:30 )
Random 430670 -> Winning ticket 28 (of 197) -> Run 0
  Jobs:  (* job:0 timeleft:7 tix:94 )   (  job:1 timeleft:8 tix:73 )   (  job:2 timeleft:4 tix:30 )
Random 393532 -> Winning ticket 123 (of 197) -> Run 1
  Jobs:  (  job:0 timeleft:6 tix:94 )   (* job:1 timeleft:8 tix:73 )   (  job:2 timeleft:4 tix:30 )
Random 723012 -> Winning ticket 22 (of 197) -> Run 0
  Jobs:  (* job:0 timeleft:6 tix:94 )   (  job:1 timeleft:7 tix:73 )   (  job:2 timeleft:4 tix:30 )
Random 994820 -> Winning ticket 167 (of 197) -> Run 2
  Jobs:  (  job:0 timeleft:5 tix:94 )   (  job:1 timeleft:7 tix:73 )   (* job:2 timeleft:4 tix:30 )
Random 949396 -> Winning ticket 53 (of 197) -> Run 0
  Jobs:  (* job:0 timeleft:5 tix:94 )   (  job:1 timeleft:7 tix:73 )   (  job:2 timeleft:3 tix:30 )
Random 544177 -> Winning ticket 63 (of 197) -> Run 0
  Jobs:  (* job:0 timeleft:4 tix:94 )   (  job:1 timeleft:7 tix:73 )   (  job:2 timeleft:3 tix:30 )
Random 444854 -> Winning ticket 28 (of 197) -> Run 0
```

```
  Jobs:  (* job:0 timeleft:3 tix:94 )   (  job:1 timeleft:7 tix:73 )   (  job:2 timeleft:3 tix:30 )
Random 268241 -> Winning ticket 124 (of 197) -> Run 1
  Jobs:  (  job:0 timeleft:2 tix:94 )   (* job:1 timeleft:7 tix:73 )   (  job:2 timeleft:3 tix:30 )
Random 35924 -> Winning ticket 70 (of 197) -> Run 0
  Jobs:  (* job:0 timeleft:2 tix:94 )   (  job:1 timeleft:6 tix:73 )   (  job:2 timeleft:3 tix:30 )
Random 27444 -> Winning ticket 61 (of 197) -> Run 0
  Jobs:  (* job:0 timeleft:1 tix:94 )   (  job:1 timeleft:6 tix:73 )   (  job:2 timeleft:3 tix:30 )
--> JOB 0 DONE at time 14
Random 464894 -> Winning ticket 55 (of 103) -> Run 1
  Jobs:  (  job:0 timeleft:0 tix:--- )   (* job:1 timeleft:6 tix:73 )   (  job:2 timeleft:3 tix:30 )
Random 318465 -> Winning ticket 92 (of 103) -> Run 2
  Jobs:  (  job:0 timeleft:0 tix:--- )   (  job:1 timeleft:5 tix:73 )   (* job:2 timeleft:3 tix:30 )
Random 380015 -> Winning ticket 48 (of 103) -> Run 1
  Jobs:  (  job:0 timeleft:0 tix:--- )   (* job:1 timeleft:5 tix:73 )   (  job:2 timeleft:2 tix:30 )
Random 891790 -> Winning ticket 16 (of 103) -> Run 1
  Jobs:  (  job:0 timeleft:0 tix:--- )   (* job:1 timeleft:4 tix:73 )   (  job:2 timeleft:2 tix:30 )
Random 525753 -> Winning ticket 41 (of 103) -> Run 1
  Jobs:  (  job:0 timeleft:0 tix:--- )   (* job:1 timeleft:3 tix:73 )   (  job:2 timeleft:2 tix:30 )
Random 560510 -> Winning ticket 87 (of 103) -> Run 2
  Jobs:  (  job:0 timeleft:0 tix:--- )   (  job:1 timeleft:2 tix:73 )   (* job:2 timeleft:2 tix:30 )
Random 236123 -> Winning ticket 47 (of 103) -> Run 1
  Jobs:  (  job:0 timeleft:0 tix:--- )   (* job:1 timeleft:2 tix:73 )   (  job:2 timeleft:1 tix:30 )
Random 23858 -> Winning ticket 65 (of 103) -> Run 1
  Jobs:  (  job:0 timeleft:0 tix:--- )   (* job:1 timeleft:1 tix:73 )   (  job:2 timeleft:1 tix:30 )
--> JOB 1 DONE at time 22
Random 325143 -> Winning ticket 3 (of 30) -> Run 2
  Jobs:  (  job:0 timeleft:0 tix:--- )   (  job:1 timeleft:0 tix:--- )   (* job:2 timeleft:1 tix:30 )
--> JOB 2 DONE at time 23
```

随机数序列为：
605944,606802,581204,158383,430670,393532,723012,994820,949396,544177,444854,268241,35924,27444,464894,318465,380015,891790,525753,560510,236123,23858,325143

根据作业长度及随机数序列推得作业票号变化：

Job 0(0-93) Job 1(94-166) Job 2(167-196)

Job 0(null) Job 1(0-72) Job 2(73-102)

Job 0(null) Job 1(null) Job 2(0-29)

又因为随机数对总票数取余就是“中奖”的作业，中奖号码序列为：
169,42,54,192,28,123,22,167,53,63,28,124,70,61,55,92,48,16,41,87,47,65,

运行作业序列：2 0 0 2 0 1 0 2 0 0 0 1 0 0 1 2 1 1 1 2 1 1 2

- 随机数种子为3

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/09.第九章-调度:比例份额$ ./lottery.py -j 3 -s 3
ARG jlist
ARG jobs 3
ARG maxlen 10
ARG maxticket 100
ARG quantum 1
ARG seed 3

Here is the job list, with the run time of each job:
  Job 0 ( length = 2, tickets = 54 )
  Job 1 ( length = 3, tickets = 60 )
  Job 2 ( length = 6, tickets = 6 )


Here is the set of random numbers you will need (at most):
Random 13168
Random 837469
Random 259354
Random 234331
Random 995645
Random 470263
Random 836462
Random 476353
Random 639068
Random 150616
```

```
Random 634861
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/09.第九章-调度:比例份额$ ./lottery.py -j 3 -s 3 -c
ARG jlist
ARG jobs 3
ARG maxlen 10
ARG maxticket 100
ARG quantum 1
ARG seed 3

Here is the job list, with the run time of each job:
  Job 0 ( length = 2, tickets = 54 )
  Job 1 ( length = 3, tickets = 60 )
  Job 2 ( length = 6, tickets = 6 )


** Solutions **

Random 13168 -> Winning ticket 88 (of 120) -> Run 1
  Jobs:  (  job:0 timeleft:2 tix:54 )   (* job:1 timeleft:3 tix:60 )   (  job:2 timeleft:6 tix:6 )
Random 837469 -> Winning ticket 109 (of 120) -> Run 1
  Jobs:  (  job:0 timeleft:2 tix:54 )   (* job:1 timeleft:2 tix:60 )   (  job:2 timeleft:6 tix:6 )
Random 259354 -> Winning ticket 34 (of 120) -> Run 0
  Jobs:  (* job:0 timeleft:2 tix:54 )   (  job:1 timeleft:1 tix:60 )   (  job:2 timeleft:6 tix:6 )
Random 234331 -> Winning ticket 91 (of 120) -> Run 1
  Jobs:  (  job:0 timeleft:1 tix:54 )   (* job:1 timeleft:1 tix:60 )   (  job:2 timeleft:6 tix:6 )
--> JOB 1 DONE at time 4
Random 995645 -> Winning ticket 5 (of 60) -> Run 0
  Jobs:  (* job:0 timeleft:1 tix:54 )   (  job:1 timeleft:0 tix:--- )   (  job:2 timeleft:6 tix:6 )
--> JOB 0 DONE at time 5
Random 470263 -> Winning ticket 1 (of 6) -> Run 2
  Jobs:  (  job:0 timeleft:0 tix:--- )   (  job:1 timeleft:0 tix:--- )   (* job:2 timeleft:6 tix:6 )
Random 836462 -> Winning ticket 2 (of 6) -> Run 2
  Jobs:  (  job:0 timeleft:0 tix:--- )   (  job:1 timeleft:0 tix:--- )   (* job:2 timeleft:5 tix:6 )
Random 476353 -> Winning ticket 1 (of 6) -> Run 2
```

```
  Jobs:  (  job:0 timeleft:0 tix:--- )   (  job:1 timeleft:0 tix:--- )   (* job:2 timeleft:4 tix:6 )
Random 639068 -> Winning ticket 2 (of 6) -> Run 2
  Jobs:  (  job:0 timeleft:0 tix:--- )   (  job:1 timeleft:0 tix:--- )   (* job:2 timeleft:3 tix:6 )
Random 150616 -> Winning ticket 4 (of 6) -> Run 2
  Jobs:  (  job:0 timeleft:0 tix:--- )   (  job:1 timeleft:0 tix:--- )   (* job:2 timeleft:2 tix:6 )
Random 634861 -> Winning ticket 1 (of 6) -> Run 2
  Jobs:  (  job:0 timeleft:0 tix:--- )   (  job:1 timeleft:0 tix:--- )   (* job:2 timeleft:1 tix:6 )
--> JOB 2 DONE at time 11
```

随机数序列为：13168,837469, 259354, 234331,995645,470263,836462,476353,639068,150616,634861

根据作业长度及随机数序列推得作业票号变化：

Job 0(0-53) Job 1(54-113) Job 2(114-119)

Job 0(0-53) Job 1(null) Job 2(54-59)

Job 0(null) Job 1(null) Job 2(0-5)

又因为随机数对总票数取余就是"中奖"的作业，中奖号码序列为：

88，109，34，91，5，1，2，1，2，4，1

运行作业序列：1 1 0 1 0 2 2 2 2 2 2

9.2

现在运行两个具体的工作：每个长度为 10,但是一个（工作 0)只有一张彩票，另一个（工作 1)有 100 张（-l 10:1,10:100).彩票数量如此不平衡时会发生什么？在工作 1 完成之前，工作 0 是否会运行？多久？

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/09.第九章-调度:比例份额$ ./lottery.py -l 10:1,10:100 -c
ARG jlist 10:1,10:100
ARG jobs 3
ARG maxlen 10
ARG maxticket 100
ARG quantum 1
ARG seed 0

Here is the job list, with the run time of each job:
```

```
  Job 0 ( length = 10, tickets = 1 )
  Job 1 ( length = 10, tickets = 100 )


** Solutions **

Random 844422 -> Winning ticket 62 (of 101) -> Run 1
  Jobs:  (  job:0 timeleft:10 tix:1 )   (* job:1 timeleft:10 tix:100 )
Random 757955 -> Winning ticket 51 (of 101) -> Run 1
  Jobs:  (  job:0 timeleft:10 tix:1 )   (* job:1 timeleft:9 tix:100 )
Random 420572 -> Winning ticket 8 (of 101) -> Run 1
  Jobs:  (  job:0 timeleft:10 tix:1 )   (* job:1 timeleft:8 tix:100 )
Random 258917 -> Winning ticket 54 (of 101) -> Run 1
  Jobs:  (  job:0 timeleft:10 tix:1 )   (* job:1 timeleft:7 tix:100 )
Random 511275 -> Winning ticket 13 (of 101) -> Run 1
  Jobs:  (  job:0 timeleft:10 tix:1 )   (* job:1 timeleft:6 tix:100 )
Random 404934 -> Winning ticket 25 (of 101) -> Run 1
  Jobs:  (  job:0 timeleft:10 tix:1 )   (* job:1 timeleft:5 tix:100 )
Random 783799 -> Winning ticket 39 (of 101) -> Run 1
  Jobs:  (  job:0 timeleft:10 tix:1 )   (* job:1 timeleft:4 tix:100 )
Random 303313 -> Winning ticket 10 (of 101) -> Run 1
  Jobs:  (  job:0 timeleft:10 tix:1 )   (* job:1 timeleft:3 tix:100 )
Random 476597 -> Winning ticket 79 (of 101) -> Run 1
  Jobs:  (  job:0 timeleft:10 tix:1 )   (* job:1 timeleft:2 tix:100 )
Random 583382 -> Winning ticket 6 (of 101) -> Run 1
  Jobs:  (  job:0 timeleft:10 tix:1 )   (* job:1 timeleft:1 tix:100 )
--> JOB 1 DONE at time 10
Random 908113 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:  (* job:0 timeleft:10 tix:1 )   (  job:1 timeleft:0 tix:--- )
Random 504687 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:  (* job:0 timeleft:9 tix:1 )   (  job:1 timeleft:0 tix:--- )
Random 281838 -> Winning ticket 0 (of 1) -> Run 0
```

```
  Jobs:  (* job:0 timeleft:8 tix:1 )   (  job:1 timeleft:0 tix:--- )
Random 755804 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:  (* job:0 timeleft:7 tix:1 )   (  job:1 timeleft:0 tix:--- )
Random 618369 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:  (* job:0 timeleft:6 tix:1 )   (  job:1 timeleft:0 tix:--- )
Random 250506 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:  (* job:0 timeleft:5 tix:1 )   (  job:1 timeleft:0 tix:--- )
Random 909747 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:  (* job:0 timeleft:4 tix:1 )   (  job:1 timeleft:0 tix:--- )
Random 982786 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:  (* job:0 timeleft:3 tix:1 )   (  job:1 timeleft:0 tix:--- )
Random 810218 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:  (* job:0 timeleft:2 tix:1 )   (  job:1 timeleft:0 tix:--- )
Random 902166 -> Winning ticket 0 (of 1) -> Run 0
  Jobs:  (* job:0 timeleft:1 tix:1 )   (  job:1 timeleft:0 tix:--- )
--> JOB 0 DONE at time 20

```

答：当彩票数量如此不平衡时极有可能作业1一直“中奖”，长时间霸占CPU。在作业1完成之前作业0有可能会运行，每次运行的可能性为1/101，如果能运行，长度每次为一个时间片。这种彩票不平衡的情况使得彩票数多的长时间占用CPU，彩票数少的很难与彩票数多的竞争。

9.3

如果运行两个长度为 100 的工作，都有 100 张彩票（-l 100:100,100:100),调度程序有多不公平？运行一些不同的随机种子来确定（概率上的）答案。不公平性取决于一项工作比另一项工作早完成多少。

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/09.第九章-调度:比例份额$ ./lottery.py -l 100:100,100:100 -s 1 -c
......
--> JOB 1 DONE at time 196
......
--> JOB 0 DONE at time 200
```

```
 cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/09.第九章-调度:比例份额$ ./lottery.py -l 100:100,100:100 -s 3 -c
......
--> JOB 0 DONE at time 196
......
--> JOB 1 DONE at time 200
```

```
cs18@games101vm:~/Desktop/os/book/Operating-Systems-Three-Easy-Pieces-NOTES-main/09.第九章-调度:比例份额$ ./lottery.py -l 100:100,100:100 -s 6 -c
......
--> JOB 1 DONE at time 193
......
--> JOB 0 DONE at time 200
```

定义了一个简单的不公平指标 *U*（unfairness metric），将两个工作完成时刻相除得到 *U* 的值。比如，运行时间 *R* 为 10，第一个工作在时刻 10 完成，另一个在20，*U*=10/20=0.5。如果两个工作几乎同时完成，*U*的值将很接近于 1。在这种情况下，我们的目标是：完美的公平调度程序可以做到 *U*=1。

下图展示了当两个工作的运行时间从 1 到1000 变化时，30 次试验的平均 *U* 值（利用本章末尾的模拟器产生的结果）。可以看出，当工作执行时间很短时，平均不公平度非常糟糕。只有当工作执行非常多的时间片时，彩票调度算法才能得到期望的结果。

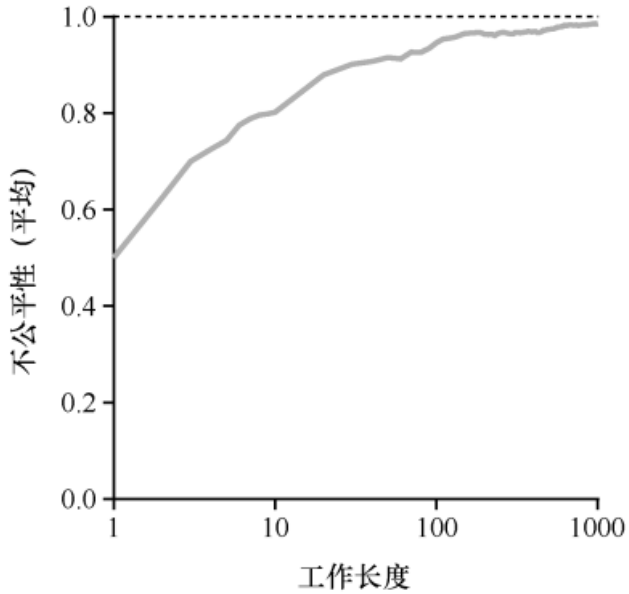

图 9.2　彩票公平性研究

Random seed	T0	T1	U
3	196	200	1.0204
6	200	193	1.0363
9	200	192	1.0417

答：不同随机数种子下（seed = 1,3,6）运行两个长度为 100 的工作，都有 100 张彩票，它们的不公平性U都近似于1，即|U - 1|≈0，所以认为在这种情况下调度程序是比较公平的