第四章

作业

补充：模拟作业

模拟作业以模拟器的形式出现，你运行它以确保理解某些内容。模拟器通常是 Python 程序，它们让你能够生成不同的问题（使用不同的随机种子），也让程序为你解决问题（带-c 标志），以便你检查答案。使用-h 或--help 标志运行任何模拟器，将提供有关模拟器所有选项的更多信息。

每个模拟器附带的 README 文件提供了有关如何运行它的更多详细信息，其中详细描述了每个标志。

程序 process-run.py 让你查看程序运行时进程状态如何改变，是在使用 CPU（例如，执行相加指令）还是执行 I/O（例如，向磁盘发送请求并等待它完成）。详情请参阅 README 文件。

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。

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 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)
```

使用率为 100%，两个进程都只使用 CPU 不使用 IO，0 进程结束后就执行 1 进程。

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 process-run.py -l 5:1
00,5:100 -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
```

This result is not too interesting: the process is simple in the RUN state and then finishes, using the CPU the whole time and thus keeping the CPU busy the entire run, and not doing any I/Os.

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 I/O 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。

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 process-run.py -l 4:1
00,1:0
Produce a trace of what would happen when you run these processes:
Process 0
  cpu
  cpu
  cpu
  cpu

Process 1
  io
  io_done

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)

guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 process-run.py -l 4:1
00,1:0 -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
```

与 io 等待时间有关，总时间=cpu(4)+io(2)+等待时间，这里的等待时间是 5，总时间为 11。

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。

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 process-run.py -l 1:0
,4:100
Produce a trace of what would happen when you run these processes:
Process 0
  io
  io_done

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)

guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 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
```

在等待进程 0 的 io 的时间，执行了进程 1 的 CPU，总时间变成了 7。交换顺序很重要，可以提高 cpu 利用率和效率。

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 I/O 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）

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 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
```

使用标记来阻止切换。这种情况下系统将等待 io 操作结束再进行下一个进程，即使可以使用 CPU。时间与第二题相同，cpu 利用率很低。

5．现在，运行相同的进程，但切换行为设置，在等待 I/O 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。

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 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
```

使用标记来允许切换。这种情况下系统在等待 io 操作时执行下一个进程。时间与第三题相同，提高了 cpu 利用率。

第五章

1．编写一个调用 fork()的程序。在调用 fork()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 <stdlib.h>
#include <unistd.h>
int main(int argc,char *argv[])
{
int x=100;
int rc = fork();
if(rc<0){
        fprintf(stderr,"fork failed\n");
        exit(1);}
else if(rc==0){
        printf("child (pid:%d),x=%d\n",(int)getpid(),x);}
else{
        printf("parent of (pid:%d),x=%d\n",(int)getpid(),x);
        sleep(1);
}
return 0;
}
```

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ ./five1.o
parent of (pid:2302),x=100
child (pid:2303),x=100
```

子进程中的值与父进程相同。

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main(int argc,char *argv[])
{
int x=100;
int rc = fork();
if(rc<0){
        fprintf(stderr,"fork failed\n");
        exit(1);}
else if(rc==0){
        x=500;
        printf("child (pid:%d),x=%d\n",(int)getpid(),x);}
else{
        x=1000;
        printf("parent of (pid:%d),x=%d\n",(int)getpid(),x);
        sleep(1);
}
return 0;
}
```

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ ./five1.2.o
parent of (pid:2009),x=1000
child (pid:2010),x=500
```

当他们分别改变 x 的值时，各自改变，互不干扰。

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 <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
int main(int argc,char *argv[])
{
int rc = fork();
int fd=open("./five2.txt",O_RDWR);
if(rc<0){
        fprintf(stderr,"fork failed\n");
        exit(1);}
else if(rc==0){
        printf("child,fd:%d\n",fd);
        char s1[]="child";
        write(fd,s1,sizeof(s1));}
else{
        printf("parent,fd:%d\n",fd);
        char s2[]="parent";
        write(fd,s2,sizeof(s2));
        sleep(1);
}
close(fd);
return 0;
}
```

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ gcc five2.c -o five2.o
guoruiling@guoruiling-virtual-machine:~/os/hw1$ ./five2.o
parent,fd:3
child,fd:3
```

访问成功，子进程和父进程都可以访问返回的文件描述符。

```
1  child^@^@
```

并发写入文件时，发现文件被后写入的子进程写入的内容覆盖。

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 execvP()。

为什么同样的基本调用会有这么多变种？

```
EXEC(3)                    Linux Programmer's Manual                    EXEC(3)

NAME
       execl, execlp, execle, execv, execvp, execvpe - execute a file

SYNOPSIS
       #include <unistd.h>

       extern char **environ;

       int execl(const char *pathname, const char *arg, ...
                       /* (char  *) NULL */);
       int execlp(const char *file, const char *arg, ...
                       /* (char  *) NULL */);
       int execle(const char *pathname, const char *arg, ...
                       /*, (char *) NULL, char *const envp[] */);
       int execv(const char *pathname, char *const argv[]);
       int execvp(const char *file, char *const argv[]);
       int execvpe(const char *file, char *const argv[],
                       char *const envp[]);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       execvpe(): _GNU_SOURCE
```

使用 man 指令查看操作手册，发现 exec 的所有变体，它们的参数传递方式和传递的参数不同，方便以不同的形式使用，更加便捷。

以 execl 为例，运行程序 ls。

```
1  #include <stdio.h>
 1 #include <stdlib.h>
 2 #include <unistd.h>
 3
 4 int main(int argc,char *argv[])
 5 {
 6 int x=100;
 7 int rc = fork();
 8 if(rc<0){
 9         fprintf(stderr,"fork failed\n");
10         exit(1);}
11 else if(rc==0){
12         printf("execl\n");
13         execl("/bin/ls","ls",NULL);}
14 else{
15         sleep(1);
16 }
17 return 0;
18 }
```

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ ./five2.4.o
execl
five1.2.c  five1.c  five2.4.c  five2.c  five2.txt
five1.2.o  five1.o  five2.4.o  five2.o  process-run.py
```

以下分别以六种形式执行。

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

int main(int argc,char *argv[])
{
int x=100;
int rc = fork();
if(rc<0){
        fprintf(stderr,"fork failed\n");
        exit(1);}
else if(rc==0){
        printf("execl\n");

        const char* arg;
        char *const argv[]={"ls","-l",NULL};
        char *const envp[]={"","",NULL};

        execl("/bin/ls",arg,NULL);
        execlp("ls",arg,NULL);
        execle("/bin/ls",arg,NULL,envp);
        execv("/bin/ls",argv);
        execvp("ls",argv);
        execvpe("ls",argv,envp);
}
else{
        sleep(1);
}
return 0;
}
```

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ ./five2.4.o
execl
five1.2.c  five1.c  five2.4.c  five2.c  five2.txt
five1.2.o  five1.o  five2.4.o  five2.o  process-run.py
```

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ ./five2.4.o
execlp
five1.2.c  five1.c  five2.4.c  five2.c  five2.txt
five1.2.o  five1.o  five2.4.o  five2.o  process-run.py
```

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ ./five2.4.o
execle
five1.2.c  five1.c  five2.4.c  five2.c  five2.txt
five1.2.o  five1.o  five2.4.o  five2.o  process-run.py
```

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ ./five2.4.o
execv
total 100
-rw-rw-r-- 1 guoruiling guoruiling   345  3月 31 22:18 five1.2.c
-rwxrwxr-x 1 guoruiling guoruiling 16216  3月 31 22:18 five1.2.o
-rw-rw-r-- 1 guoruiling guoruiling   328  3月 31 22:14 five1.c
-rwxrwxr-x 1 guoruiling guoruiling 16216  3月 31 22:14 five1.o
-rw-rw-r-- 1 guoruiling guoruiling   504  4月  5 17:29 five2.4.c
-rwxrwxr-x 1 guoruiling guoruiling 16376  4月  5 17:29 five2.4.o
-rw-rw-r-- 1 guoruiling guoruiling   430  3月 31 22:35 five2.c
-rwxrwxr-x 1 guoruiling guoruiling 16328  3月 31 22:35 five2.o
-rw-rw-r-- 1 guoruiling guoruiling     7  4月  5 16:56 five2.txt
-rw-rw-r-- 1 guoruiling guoruiling 13352  3月 31 19:36 process-run.py
```

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ ./five2.4.o
execvp
total 100
-rw-rw-r-- 1 guoruiling guoruiling   345  3月 31 22:18 five1.2.c
-rwxrwxr-x 1 guoruiling guoruiling 16216  3月 31 22:18 five1.2.o
-rw-rw-r-- 1 guoruiling guoruiling   328  3月 31 22:14 five1.c
-rwxrwxr-x 1 guoruiling guoruiling 16216  3月 31 22:14 five1.o
-rw-rw-r-- 1 guoruiling guoruiling   505  4月  5 17:31 five2.4.c
-rwxrwxr-x 1 guoruiling guoruiling 16376  4月  5 17:31 five2.4.o
-rw-rw-r-- 1 guoruiling guoruiling   430  3月 31 22:35 five2.c
-rwxrwxr-x 1 guoruiling guoruiling 16328  3月 31 22:35 five2.o
-rw-rw-r-- 1 guoruiling guoruiling     7  4月  5 16:56 five2.txt
-rw-rw-r-- 1 guoruiling guoruiling 13352  3月 31 19:36 process-run.py
```

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ ./five2.4.o
execvpe
total 100
-rw-rw-r-- 1 guoruiling guoruiling   345 Mar 31 22:18 five1.2.c
-rwxrwxr-x 1 guoruiling guoruiling 16216 Mar 31 22:18 five1.2.o
-rw-rw-r-- 1 guoruiling guoruiling   328 Mar 31 22:14 five1.c
-rwxrwxr-x 1 guoruiling guoruiling 16216 Mar 31 22:14 five1.o
-rw-rw-r-- 1 guoruiling guoruiling   506 Apr  5 17:34 five2.4.c
-rwxrwxr-x 1 guoruiling guoruiling 16264 Apr  5 17:34 five2.4.o
-rw-rw-r-- 1 guoruiling guoruiling   430 Mar 31 22:35 five2.c
-rwxrwxr-x 1 guoruiling guoruiling 16328 Mar 31 22:35 five2.o
-rw-rw-r-- 1 guoruiling guoruiling     7 Apr  5 16:56 five2.txt
-rw-rw-r-- 1 guoruiling guoruiling 13352 Mar 31 19:36 process-run.py
```

第七章

作业

scheduler.py 这个程序允许你查看不同调度程序在调度指标（如响应时间、周转时间和总等待时间）下的执行情况。详情请参阅 README 文件。

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。

此时两种方法的时间是相同的。

	A	B	C	平均
响应时间	0	200	400	200

周转时间	200	400	600	400

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 ./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
```

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 ./scheduler.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
```

2. 现在做同样的事情，但有不同长度的作业，即 100、200 和 300。

此时两种方法的时间仍然相同。

	A	B	C	平均
响应时间	0	100	300	133.3
周转时间	100	300	600	333.3

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 ./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

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

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 ./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

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

3. 现在做同样的事情，但采用 RR 调度程序，时间片为 1。

	A	B	C	平均
响应时间	0	1	2	1
周转时间	598	599	600	599

```
  [ time 598 ] Run job   1 for 1.00 secs ( DONE at 599.00 )
  [ time 599 ] Run job   2 for 1.00 secs ( DONE at 600.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 598.00  Wait 398.00
  Job   1 -- Response: 1.00  Turnaround 599.00  Wait 399.00
  Job   2 -- Response: 2.00  Turnaround 600.00  Wait 400.00

  Average -- Response: 1.00  Turnaround 599.00  Wait 399.00
```

```
  [ time 598 ] Run job   1 for 1.00 secs ( DONE at 599.00 )
  [ time 599 ] Run job   2 for 1.00 secs ( DONE at 600.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 598.00  Wait 398.00
  Job   1 -- Response: 1.00  Turnaround 599.00  Wait 399.00
  Job   2 -- Response: 2.00  Turnaround 600.00  Wait 400.00

  Average -- Response: 1.00  Turnaround 599.00  Wait 399.00
```

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：1. 作业到达时间不一致时。

2. 作业到达时间大概一致时，FIFO 列表中执行顺序按照作业长度递增。

3. 部分一致且一致的部分满足条件 2。

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

答：工作的时长一致且等于量子长度，也就是时间片。

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？

答：若所有工作长度都增加，除了第一个任务的响应时间不变，其余任务的响应时间都增加；若部分工作长度增加，长度比该工作长的工作响应时间增加。

```
guorulling@guorulling-virtual-machine:~/os/hw1$ python3 ./scheduler.py -
p SJF -l 200,100,100 -c
ARG policy SJF
ARG jlist 200,100,100

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


** Solutions **

Execution trace:
  [ time   0 ] Run job 1 for 100.00 secs ( DONE at 100.00 )
  [ time 100 ] Run job 2 for 100.00 secs ( DONE at 200.00 )
  [ time 200 ] Run job 0 for 200.00 secs ( DONE at 400.00 )

Final statistics:
  Job   1 -- Response: 0.00  Turnaround 100.00  Wait 0.00
  Job   2 -- Response: 100.00  Turnaround 200.00  Wait 100.00
  Job   0 -- Response: 200.00  Turnaround 400.00  Wait 200.00

  Average -- Response: 100.00  Turnaround 233.33  Wait 100.00
```

```
guorulling@guorulling-virtual-machine:~/os/hw1$ python3 ./scheduler.py -
p SJF -l 100,200,100 -c
ARG policy SJF
ARG jlist 100,200,100

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 = 100.0 )


** Solutions **

Execution trace:
  [ time   0 ] Run job 0 for 100.00 secs ( DONE at 100.00 )
  [ time 100 ] Run job 2 for 100.00 secs ( DONE at 200.00 )
  [ time 200 ] Run job 1 for 200.00 secs ( DONE at 400.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 100.00  Wait 0.00
  Job   2 -- Response: 100.00  Turnaround 200.00  Wait 100.00
  Job   1 -- Response: 200.00  Turnaround 400.00  Wait 200.00

  Average -- Response: 100.00  Turnaround 233.33  Wait 100.00
```

```
guorulling@guorulling-virtual-machine:~/os/hw1$ python3 ./scheduler.py -
p SJF -l 100,200,100 -c
ARG policy SJF
ARG jlist 100,200,100

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 = 100.0 )


** Solutions **

Execution trace:
  [ time   0 ] Run job 0 for 100.00 secs ( DONE at 100.00 )
  [ time 100 ] Run job 2 for 100.00 secs ( DONE at 200.00 )
  [ time 200 ] Run job 1 for 200.00 secs ( DONE at 400.00 )

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 100.00  Wait 0.00
  Job   2 -- Response: 100.00  Turnaround 200.00  Wait 100.00
  Job   1 -- Response: 200.00  Turnaround 400.00  Wait 200.00

  Average -- Response: 100.00  Turnaround 233.33  Wait 100.00
```

7. 随着量子长度的增加，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：假设量子长度为 t，平均响应时间为[0+t+2t+…+(n-1)t]/n=(n-1)t/2,最坏情况的响应时间为(n-1)t。

第八章

作业

程序 mlfq.py 允许你查看本章介绍的 MLFQ 调度程序的行为。详情请参阅 README 文件。

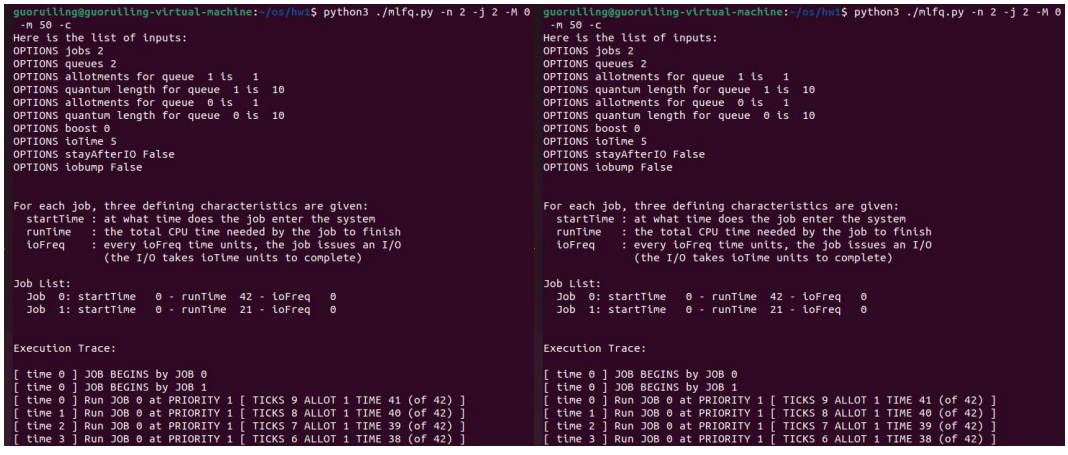
1. 只用两个工作和两个队列运行几个随机生成的问题。针对每个工作计算 MLFQ 的执行记录。限制每项作业的长度并关闭 I/O，让你的生活更轻松。

问题 1：

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 ./mlfq.py -n 2 -j 2 -M 0
 -m 50
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  42 - ioFreq   0
  Job  1: startTime   0 - runTime  21 - 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.
```

工作 0：时间 42，时间配额 1，工作 1：时间 21，时间配额 1。队列 1,2：时间片 10。

时间	队列 0	队列 1	CPU
0-9	AB		A
10-19	B	A	B
20-29		AB	A
30-39		AB	B
40-49		AB	A
50-51		AB	B
51-63		A	A



问题 2：

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 ./mlfq.py -n 2 -j 2 -M 0
 -m 50 -s 2
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  47 - ioFreq   0
  Job  1: startTime   0 - runTime   3 - 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.
```

工作 0：时间 47，时间配额 1，工作 1：时间 3，时间配额 1。队列 1,2：时间片 10。

时间	队列 0	队列 1	CPU
0-9	AB		A
10-12	B	A	B
13-49		A	A

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 ./mlfq.py -n 2 -j 2 -M 0
 -m 50 -s 2 -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  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  47 - ioFreq   0
  Job  1: startTime   0 - runTime   3 - 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 9 ALLOT 1 TIME 46 (of 47) ]
[ time 1 ] Run JOB 0 at PRIORITY 1 [ TICKS 8 ALLOT 1 TIME 45 (of 47) ]
[ time 2 ] Run JOB 0 at PRIORITY 1 [ TICKS 7 ALLOT 1 TIME 44 (of 47) ]
[ time 3 ] Run JOB 0 at PRIORITY 1 [ TICKS 6 ALLOT 1 TIME 43 (of 47) ]
```

```
[ time 12 ] Run JOB 1 at PRIORITY 1 [ TICKS 7 ALLOT 1 TIME 0 (of 3) ]
[ time 13 ] FINISHED JOB 1
[ time 13 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 36 (of 47) ]
```

```
[ time 46 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 3 (of 47) ]
[ time 47 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 2 (of 47) ]
[ time 48 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 1 (of 47) ]
[ time 49 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 0 (of 47) ]
[ time 50 ] FINISHED JOB 0

Final statistics:
  Job  0: startTime   0 - response   0 - turnaround  50
  Job  1: startTime   0 - response  10 - turnaround  13

  Avg  1: startTime n/a - response 5.00 - turnaround 31.50
```

3. 将如何配置调度程序参数，像轮转调度程序那样工作？

答：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。python3 ./mlfq.py -n 1.

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？

答：10ms/5%=200ms，B 的频率至少为 200ms 才能使该工作得到 5%的 CPU。

第九章

作业

lottery.py 这个程序允许你查看彩票调度程序的工作原理。详情请参阅 README 文件。

1. 计算 3 个工作在随机种子为 1、2 和 3 时的模拟解。

1：

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 ./lottery.py -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
guoruiling@guoruiling-virtual-machine:~/os/hw1$
```

份额：84,25,44 时间：1,7,4.

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

2：

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 ./lottery.py -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
```

份额：94,73,30 时间：9,8,6.

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

19		6	4	103	525753	41	job1
20		6	5	103	560510	87	job2
21		7	5	103	236123	47	job1
22		8 完成	5	103	23858	65	job1
23			6 完成	30	325143	3	job2

3：

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 ./lottery.py -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
```

份额：54,60,6 时间：2,3,6.

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

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？一般来说，这种彩票不平衡对彩票调度的行为有什么影响？

```
guoruiling@guoruiling-virtual-machine:~/os/hw1$ python3 ./lottery.py -l 10:1,1
0:100
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 )


Here is the set of random numbers you will need (at most):
Random 844422
Random 757955
Random 420572
Random 258917
Random 511275
Random 404934
Random 783799
Random 303313
Random 476597
Random 583382
Random 908113
Random 504687
Random 281838
Random 755804
Random 618369
Random 250506
Random 909747
Random 982786
Random 810218
Random 902166
```

份额：1,100 时间：10,10

时间	job0	job1	总份额	随机数	模	CPU
1	0	1	101	844422	62	job1
2	0	2	101	757955	51	job1
3	0	3	101	420572	8	job1
4	0	4	101	258917	54	job1
5	0	5	101	511275	13	job1
6	0	6	101	404934	25	job1
7	0	7	101	783799	39	job1
8	0	8	101	303313	10	job1
9	0	9	101	476597	79	job1
10	0	10 完成	101	583382	6	job1
11	1		1	908113	0	job0
12	2		1	504687	0	job0
13	3		1	281838	0	job0
14	4		1	755804	0	job0
15	5		1	618369	0	job0
16	6		1	250506	0	job0
17	7		1	909747	0	job0
18	8		1	982786	0	job0
19	9		1	810218	0	job0
20	10 完成		1	902166	0	job0

导致工作 0 运行的概率很小，模拟中在工作 1 运行前没有运行过。持有彩票份额

小的工作响应时间与周转时间非常长，极有可能饿死。

3. 如果运行两个长度为 100 的工作，都有 100 张彩票（-l100：100,100：100），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。

使用 python3 ./lottery.py -l 100:100,100:100 -s x -c。

种子编号	job0 完成	job1 完成	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
平均数	198.8	192.7	8.5

提前完成的工作平均快 8.5 左右，若以提前完成时间比工作长度为不公平指标，不公平度约为 8.5%。