第 4 章

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。

```
zlz@zlz-virtual-machine:~$ ./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)
```

根据指令，两个进程只使用 cpu，没有进行 I/O 等操作，每个时间 cpu 都被进程利用，对应的 cpu 利用率应该是 100%。（通过搜索，我们指定的过程是“5：100”，这意味着它应该由 5 条指令组成，并且每条指令是 CPU 指令的几率是 100%。而两个 5：100 则是两个进程，前 5 个时间进程 0 在运行，而进程 1 处于就绪状态，后 5 个时间进程 1 在运行，进程 0 已经完成，cpu 利用率为 100%）。

```
zlz@zlz-virtual-machine:~$ ./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
```

由-c 可知，结果和猜想一致。

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。

```
zlz@zlz-virtual-machine:~$ ./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
  io_done
```

进程 0 有四条指令，需要 4 个时间，进程 1 有一条指令，但是发生了 I/O 请求，所以会有发出 I/O 请求，进入内核态的一个时间，等待 I/O 的时间，最后完成 I/O，回到用户态的一个时间，所以总时间应该为 4+2+waiting time。

```
zlz@zlz-virtual-machine:~$ ./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       WAITING                           1
 11*          DONE   RUN:io_done             1
```

由-c 可知，确实和猜想一致，这里的 waiting time 为 5 个时间。

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。**

```
zlz@zlz-virtual-machine:~$ ./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)
```

我们可以看到进程 0 先执行，发出 I/O 请求，然后进程 1 运行。交换顺序很

重要，在执行 I/O 时，I/O 请求等待的时间中，进程 0 会进入堵塞状态，cpu 会切换到进程 1，而进程 1 只需要 4 个时间，在等待时间中就可以完成，所以最终的时间相当于只消耗了进程 0 单独运行所需要的时间，即 7 个时间。

```
zlz@zlz-virtual-machine:~$ ./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        WAITING          DONE                         1
  7*   RUN:io_done          DONE            1
```

由-c 可知，结果和我们预测的一样，这说明了执行顺序很重要，合适的执行顺序对 cpu 效率和利用率的提高可能很有帮助。

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)

```
zlz@zlz-virtual-machine:~$ ./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        WAITING         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，根据题目上所描述的-S 决定了进程在进行 I/O 操作时，系统不会切换到另一个进程，而是等待进程完成，所以先执行的进程 0 发生 I/O 一直占用 cpu，即使在等待时间中仍没有发生切换，所以总的占用 cpu 的时间为 11 个，和第二题一致。

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。

```
zlz@zlz-virtual-machine:~$ ./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        WAITING          DONE                           1
  7*   RUN:io_done          DONE             1
```

使用标记，该标记是在等待 I/O 时切换到另一个进程，所以在执行进程 0 时，发生 I/O，进程 0 堵塞，会切换到进程 1 执行，在等待时间内就可以完成进程 1，然后切换回进程 0，执行完进程 0，所以占用 cpu 的总时间为 7 个，和第三问一致。这种方法可以最大限度地使用 cpu，cpu 利用率高。

第五章

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<stdlib.h>
#include<unistd.h>


int main(int argc,char *argv[]){
        int x=100;
        int rc=fork();
        if(rc<0){//fork failed
                fprintf(stderr,"fork failed\n");
        }else if(rc==0){//child(new process)
                printf("child(pid:%d),x=%d\n",(int)getpid(),x);
        }else {//parent(original process)
                printf("parent(pid:%d),x=%d\n",(int)getpid(),x);
        }
}
```

```
zlz@zlz-virtual-machine:~$ gedit fork.c
zlz@zlz-virtual-machine:~$ gcc fork.c -o fork
zlz@zlz-virtual-machine:~$ ./fork
parent(pid:3242),x=100
child(pid:3243),x=100
```

可以发现子进程和父进程的变量都为 100

```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){//fork failed
                fprintf(stderr,"fork failed\n");
        }else if(rc==0){//child(new process)
                x+=50;
                printf("child(pid:%d),x=%d\n",(int)getpid(),x);
        }else {//parent(original process)
                x-=50;
                printf("parent(pid:%d),x=%d\n",(int)getpid(),x);
        }
}
```

```
zlz@zlz-virtual-machine:~$ ./fork
parent(pid:3325),x=50
child(pid:3326),x=150
```

当在父进程和子进程中分别修改变量时，我发现父进程和子进程变量相互独立，互不影响。

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?

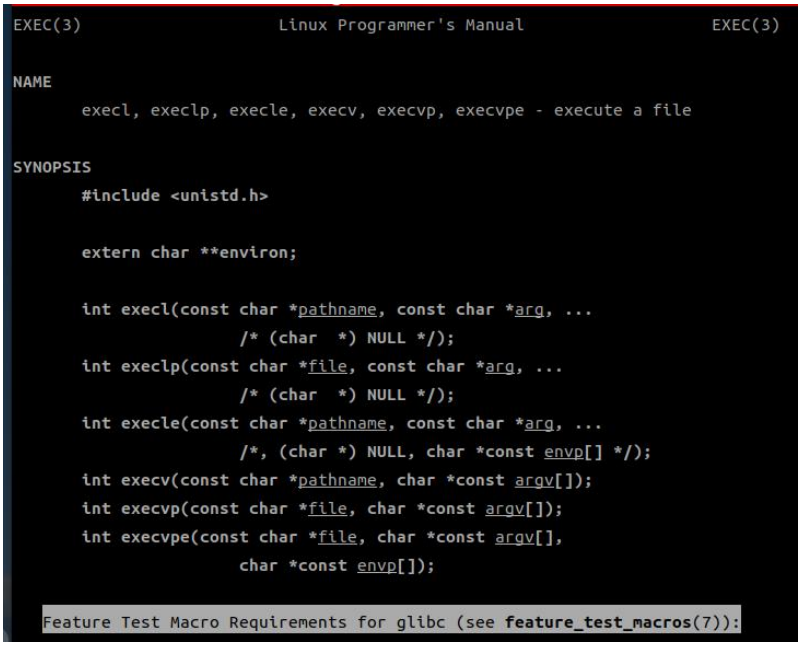
```c
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<fcntl.h>

int main(int argc,char *argv[]){
        int rc=fork();
        int fd=open("./fork.txt",O_RDWR);
        if(rc<0){//fork failed
                fprintf(stderr,"fork failed\n");
                exit(1);
        }else if(rc==0){//child(new process
                printf("child(pid:%d),fd:%d\n",(int)getpid(),fd);
                char s1[]="Child process";
                write(fd,s1,sizeof(s1));
        }else {//parent(original process)
                printf("parent(pid:%d),fd:%d\n",(int)getpid(),fd);
                char s2[]="Parent process";
                write(fd,s2,sizeof(s2));
        }
        close(fd);
        return 0;
}
```

```
zlz@zlz-virtual-machine:~$ ./fork
parent(pid:3738),fd:3
child(pid:3739),fd:3
```

子进程和父进程都可以访问 open（）返回的文件描述符 fd，当它们并发写入时，文件被较后执行的子进程覆盖，所以显示 Child process

而当我设置父进程 sleep 一段时间，让父进程较后执行，可以看到最后文件被父进程所覆盖，显示 Parent process

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?



参数可以单独传入，也可以按数组的方式传入，并可以传入环境变量，路径，可执行文件名。

execl：通过指定文件路径执行一个文件，参数列表以可变参数形式传递，以 NULL 结束。

execlp：通过在 PATH 环境变量中查找文件名来执行一个文件，参数列表以

可变参数形式传递，以 NULL 结束。

execle：通过指定文件路径执行一个文件，并提供新的环境变量，参数列表以可变参数形式传递，最后一个参数是指向环境变量数组的指针，数组以 NULL 结束。

execv：通过指定文件路径执行一个文件，参数列表以数组形式传递。

execvp：通过在 PATH 环境变量中查找文件名来执行一个文件，参数列表以数组形式传递。

execvpe：类似于 execvp，但可以提供新的环境变量，最后一个参数是指向环境变量数组的指针，数组以 NULL 结束。

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

int main() {
    int rc = fork();
    if (rc < 0) {
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) {
        printf("execl\n");
        execl("/bin/ls","ls",NULL);
    } else {
        int wc = wait(NULL);
    }
    return 0;
}

```

```
zlz@zlz-virtual-machine:~$ gedit fork.c
zlz@zlz-virtual-machine:~$ gcc fork.c -o fork
zlz@zlz-virtual-machine:~$ ./fork
execl
1
1001
1001.c
1001.i
1001.o
1001.s
1003
1003.c
1004
1004.c
1011.i
1.c
```

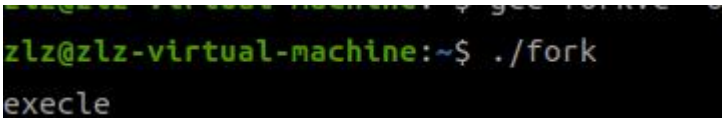
用 execl()运行程序/bin/ls

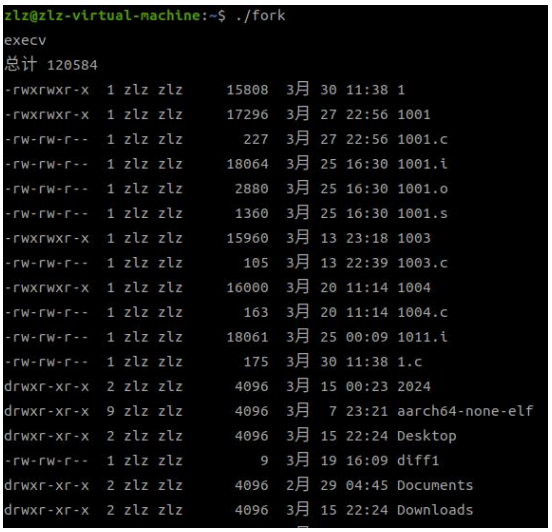
```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main() {
    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
        execl("/bin/ls",arg,NULL);
        //execlp
        execlp("ls",arg,NULL);
        //execle
        execle("/bin/ls",arg,NULL,envp);
        //execv
        execv("/bin/ls",argv);
        //execvp
        execvp("ls",argv);
        //execvpe
        execvpe("ls",argv,envp);
    } else {
        int wc = wait(NULL);
    }
    return 0;
}
```

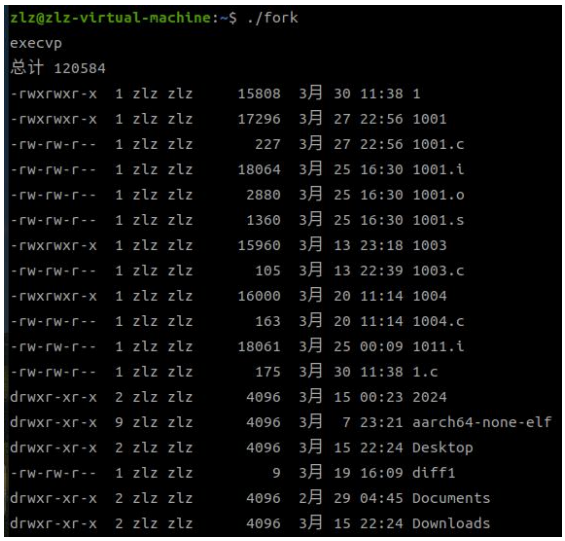
分别得到下列结果

```
zlz@zlz-virtual-machine:~$ ./fork
execl
1
1001
1001.c
1001.i
1001.o
1001.s
1003
1003.c
1004
1004.c
1011.i
1.c
```

```
zlz@zlz-virtual-machine:~$ ./fork
execlp
1
1001
1001.c
1001.i
1001.o
1001.s
1003
1003.c
1004
1004.c
1011.i
1.c
```







多种 exec()调用的传递参数不同，传递方式不同，比较灵活，可以实现多种功能。

第七章

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。

假设任务

A B C

200 200 200

SJF（最短任务优先）：

	周转时间	响应时间
A	200	0
B	400	200
C	600	400

Aver 400 200
FIFO(先进先出）:
周转时间 响应时间
A 200 0
B 400 200
C 600 400
Aver 400 200
对 SJF 和 FIFO，平均周转时间为 400，平均响应时间为 200。

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

假设任务
A B C
100 200 300
SJF（最短任务优先）：
周转时间 响应时间
A 100 0
B 300 100
C 600 300
Aver 1000/3 400/3
FIFO(先进先出）:
周转时间 响应时间
A 100 0
B 300 100
C 600 300
Aver 1000/3 400/3
对 SJF 和 FIFO，平均周转时间为 1000/3，平均响应时间为 400/3。

3. 现在做同样的事情，但采用 RR 调度程序，时间片为 1。

假设任务
A B C
200 200 200
RR(轮转）：
周转时间 响应时间
A 598 0
B 599 1
C 600 2
Aver 599 1
对于 RR,平均周转时间为 599，平均响应时间为 1，明显看出使用轮转策略可以带来高效的响应，但是会极大增长了周转时间，执行效率低

4. 对于什么类型的工作负载，SJF 提供与 FIFO 相同的周转时间?

1.作业到达时间不一致

2.作业到达时间存在一致，列表中执行顺序按作业长度非严格递增，即先到达的先执行的顺序，就可以保证相同的周转时间

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?

所有作业的执行时间相等且等于量子长度

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?

1.如果所有工作长度都增加,那么除了第一个工作的响应时间(即最短工作)不变外，其他响应时间都增加。

2.如果部分工作长度增加，那么比增加后工作仍长的工作响应时间会增加，如果增加后工作时间后，工作顺序发生了变化，那么部分工作的响应时间可能会变短，总体应该仍变长

可以用模拟程序来预测趋势，这段代码由 chatgpt 编写，可以模拟 SFJ 的过程，通过修改工作长度来模拟趋势

这段代码先创建了完整的进程结构体，按照工作长度进行排序，排序后依此计算每个工作的响应时间和周转时间，最终计算平均。

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

typedef struct {
    int id;
    int arrival_time;
    int burst_time;
    int waiting_time;
    int turnaround_time;
} Process;

void sjf(Process processes[], int n) {
    int total_waiting_time = 0;
    int total_turnaround_time = 0;

    // Sort processes by burst time
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (processes[j].burst_time > processes[j + 1].burst_time) {
                Process temp = processes[j];
                processes[j] = processes[j + 1];
                processes[j + 1] = temp;
            }
        }
    }

    // Calculate waiting time and turnaround time
    processes[0].waiting_time = 0;
    processes[0].turnaround_time = processes[0].burst_time;
    for (int i = 1; i < n; i++) {
        processes[i].waiting_time = processes[i - 1].turnaround_time;
        processes[i].turnaround_time = processes[i].waiting_time + processes[i].burst_time;
    }

    // Calculate total waiting time and total turnaround time
    for (int i = 0; i < n; i++) {
        total_waiting_time += processes[i].waiting_time;
        total_turnaround_time += processes[i].turnaround_time;
    }

    // Print results
    printf("Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].arrival_time,
               processes[i].burst_time, processes[i].waiting_time, processes[i].turnaround_time);
    }

    // Print average waiting time and average turnaround time
    printf("Average Waiting Time: %.2f\n", (float)total_waiting_time / n);
    printf("Average Turnaround Time: %.2f\n", (float)total_turnaround_time / n);
}

int main() {
    Process processes[] = {
        {1, 0, 5},
        {2, 0, 3},
        {3, 0, 8},
        {4, 0, 6},
        {5, 0, 1},
    };
    int n = sizeof(processes) / sizeof(processes[0]);

    sjf(processes, n);

    return 0;
}
```

```
zlz@zlz-virtual-machine:~$ ./fork
Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
5               0               1               0               1
2               0               3               1               4
1               0               5               4               9
4               0               6               9               15
3               0               8               15              23
Average Waiting Time: 5.80
Average Turnaround Time: 10.40
```

```
zlz@zlz-virtual-machine:~$ ./fork
Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
5               0               3               0               3
2               0               5               3               8
4               0               6               8               14
1               0               8               14              22
3               0               10              22              32
Average Waiting Time: 9.40
Average Turnaround Time: 15.80
```

可以明显看出响应时间的增加。

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?

随着量子长度的增加，RR 的响应时间会变长。这是因为随着量子长度的增加，每个进程在 CPU 上执行的时间片段变长，导致平均等待时间增加，最终导致平均响应时间的增大。

假设有 N 个事件，量子长度为 t0，那么平均响应时间为(0+t0+...+t0)/n=(n-1)*t0/2,最坏情况的响应时间是最坏一个工作的响应时间，为(n-1)*t0。

第八章

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。

（1）根据此任务的要求，我们使用如下指令

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

其中-n 为队列个数，设置为 2，-j 为工作个数，设置为 2，-m 为每项作业长度上限，设置为 100，-M 为 I/O 频率，由于关闭 I/O，其频率为 0，设置为 0。

```
Usage: mlfq.py [options]
Options:
  -h, --help            show this help message and exit
  -s SEED, --seed=SEED  the random seed
  -n NUMQUEUES, --numQueues=NUMQUEUES
                        number of queues in MLFQ (if not using -Q)
  -q QUANTUM, --quantum=QUANTUM
                        length of time slice (if not using -Q)
  -Q QUANTUMLIST, --quantumList=QUANTUMLIST
                        length of time slice per queue level,
                        specified as x,y,z,... where x is the
                        quantum length for the highest-priority
                        queue, y the next highest, and so forth
  -j NUMJOBS, --numJobs=NUMJOBS
                        number of jobs in the system
  -m MAXLEN, --maxlen=MAXLEN
                        max run-time of a job (if random)
  -M MAXIO, --maxio=MAXIO
                        max I/O frequency of a job (if random)
  -B BOOST, --boost=BOOST
                        how often to boost the priority of all
                        jobs back to high priority (0 means never)
  -i IOTIME, --iotime=IOTIME
                        how long an I/O should last (fixed constant)
  -S, --stay            reset and stay at same priority level
                        when issuing I/O
  -l JLIST, --jlist=JLIST
                        a comma-separated list of jobs to run,
                        in the form x1,y1,z1:x2,y2,z2:... where
                        x is start time, y is run time, and z
                        is how often the job issues an I/O request
  -c                    compute answers for me
```

```
zlz@zlz-virtual-machine:~$ ./mlfq.py -n 2 -j 2 -m 100 -M 0
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.
```

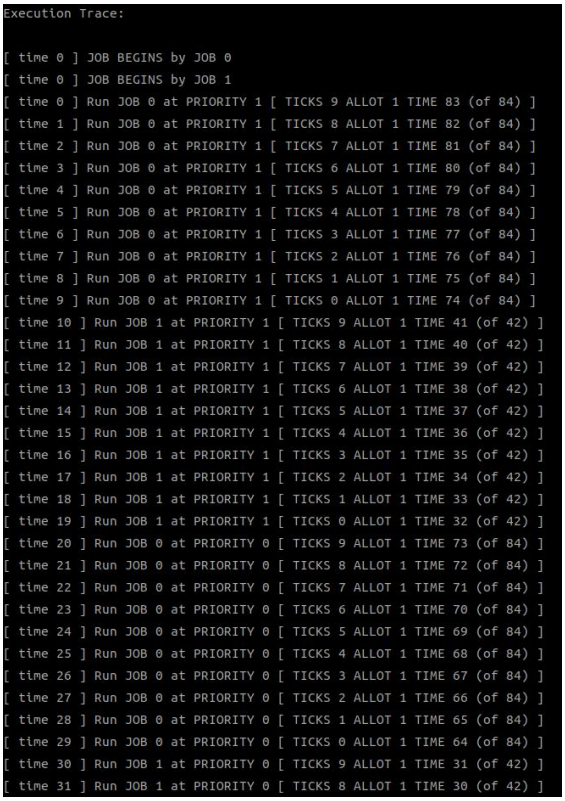
工作列表：
工作 0(A)，运行时间 84
工作 1(B)，运行时间 42
时间片长度是 10
重新加入最高优先级队列的时间 S（巫毒常量）：0
（由于只有两个进程，且无 I/O 操作，所以可以忽略重新加入最高队列）

时间	队列 0	队列 1	cpu
0-10	AB		A
10-20	B	A	B

20-30	AB	A
30-40	AB	B
40-50	AB	A
50-60	AB	B
60-70	AB	A
70-80	AB	B
80-90	AB	A
90-92	AB	B
92-126	A	A

平均响应时间为 5,平均响应时间为 109

通过-c 我们可以检验我们的想法， Avg 1: startTime n/a - response 5.00 - turnaround 109.00 和我们想法一致

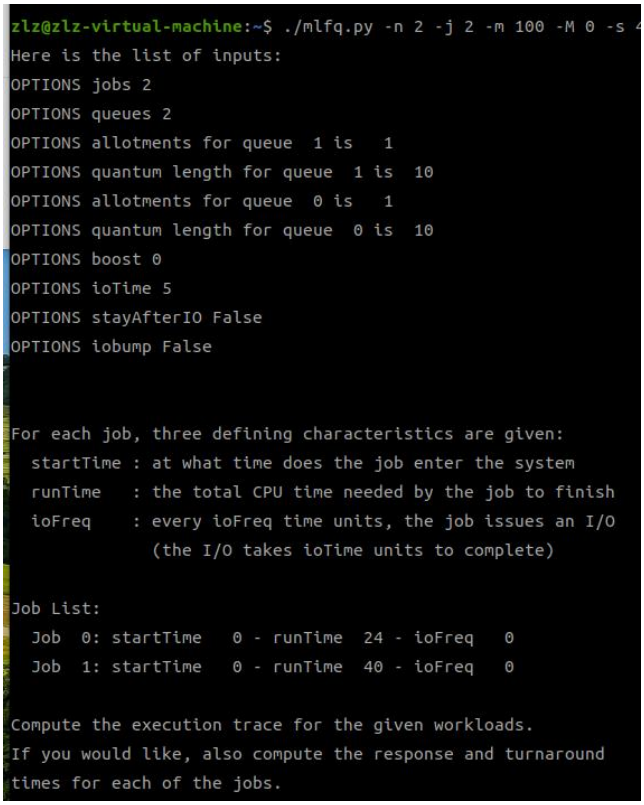


```
[ time 32 ] Run JOB 1 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 29 (of 42) ]
[ time 33 ] Run JOB 1 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 28 (of 42) ]
[ time 34 ] Run JOB 1 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 27 (of 42) ]
[ time 35 ] Run JOB 1 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 26 (of 42) ]
[ time 36 ] Run JOB 1 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 25 (of 42) ]
[ time 37 ] Run JOB 1 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 24 (of 42) ]
[ time 38 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 23 (of 42) ]
[ time 39 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 22 (of 42) ]
[ time 40 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 63 (of 84) ]
[ time 41 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 62 (of 84) ]
[ time 42 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 61 (of 84) ]
[ time 43 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 60 (of 84) ]
[ time 44 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 59 (of 84) ]
[ time 45 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 58 (of 84) ]
[ time 46 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 57 (of 84) ]
[ time 47 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 56 (of 84) ]
[ time 48 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 55 (of 84) ]
[ time 49 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 54 (of 84) ]
[ time 50 ] Run JOB 1 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 21 (of 42) ]
[ time 51 ] Run JOB 1 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 20 (of 42) ]
[ time 52 ] Run JOB 1 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 19 (of 42) ]
[ time 53 ] Run JOB 1 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 18 (of 42) ]
[ time 54 ] Run JOB 1 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 17 (of 42) ]
[ time 55 ] Run JOB 1 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 16 (of 42) ]
[ time 56 ] Run JOB 1 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 15 (of 42) ]
[ time 57 ] Run JOB 1 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 14 (of 42) ]
[ time 58 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 13 (of 42) ]
[ time 59 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 12 (of 42) ]
[ time 60 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 53 (of 84) ]
[ time 61 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 52 (of 84) ]
[ time 62 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 51 (of 84) ]
[ time 63 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 50 (of 84) ]
[ time 64 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 49 (of 84) ]
[ time 65 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 48 (of 84) ]
[ time 66 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 47 (of 84) ]
[ time 67 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 46 (of 84) ]
```

```
[ time 68 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 45 (of 84) ]
[ time 69 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 44 (of 84) ]
[ time 70 ] Run JOB 1 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 11 (of 42) ]
[ time 71 ] Run JOB 1 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 10 (of 42) ]
[ time 72 ] Run JOB 1 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 9 (of 42) ]
[ time 73 ] Run JOB 1 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 8 (of 42) ]
[ time 74 ] Run JOB 1 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 7 (of 42) ]
[ time 75 ] Run JOB 1 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 6 (of 42) ]
[ time 76 ] Run JOB 1 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 5 (of 42) ]
[ time 77 ] Run JOB 1 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 4 (of 42) ]
[ time 78 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 3 (of 42) ]
[ time 79 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 2 (of 42) ]
[ time 80 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 43 (of 84) ]
[ time 81 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 42 (of 84) ]
[ time 82 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 41 (of 84) ]
[ time 83 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 40 (of 84) ]
[ time 84 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 39 (of 84) ]
[ time 85 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 38 (of 84) ]
[ time 86 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 37 (of 84) ]
[ time 87 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 36 (of 84) ]
[ time 88 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 35 (of 84) ]
[ time 89 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 34 (of 84) ]
[ time 90 ] Run JOB 1 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 1 (of 42) ]
[ time 91 ] Run JOB 1 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 0 (of 42) ]
[ time 92 ] FINISHED JOB 1
[ time 92 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 33 (of 84) ]
[ time 93 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 32 (of 84) ]
[ time 94 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 31 (of 84) ]
[ time 95 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 30 (of 84) ]
[ time 96 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 29 (of 84) ]
[ time 97 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 28 (of 84) ]
[ time 98 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 27 (of 84) ]
[ time 99 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 26 (of 84) ]
[ time 100 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 25 (of 84) ]
[ time 101 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 24 (of 84) ]
[ time 102 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 23 (of 84) ]
[ 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
```

（2）我们可以改变随机数种子来检验其他事件是否满足，这里选用-s 4



工作列表：
工作 0(A)，运行时间 24
工作 1(B)，运行时间 40
时间片长度是 10
重新加入最高优先级队列的时间 S（巫毒常量）：0
（由于只有两个进程，且无 I/O 操作，所以可以忽略重新加入最高队列）

时间	队列 0	队列 1	cpu
0-10	AB		A
10-20	B	A	B
20-30		AB	A
30-40		AB	B
40-44		AB	A
44-64		B	B

平均周转时间为 54，平均响应时间为 5

```
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 23 (of 24) ]
[ time 1 ] Run JOB 0 at PRIORITY 1 [ TICKS 8 ALLOT 1 TIME 22 (of 24) ]
[ time 2 ] Run JOB 0 at PRIORITY 1 [ TICKS 7 ALLOT 1 TIME 21 (of 24) ]
[ time 3 ] Run JOB 0 at PRIORITY 1 [ TICKS 6 ALLOT 1 TIME 20 (of 24) ]
[ time 4 ] Run JOB 0 at PRIORITY 1 [ TICKS 5 ALLOT 1 TIME 19 (of 24) ]
[ time 5 ] Run JOB 0 at PRIORITY 1 [ TICKS 4 ALLOT 1 TIME 18 (of 24) ]
[ time 6 ] Run JOB 0 at PRIORITY 1 [ TICKS 3 ALLOT 1 TIME 17 (of 24) ]
[ time 7 ] Run JOB 0 at PRIORITY 1 [ TICKS 2 ALLOT 1 TIME 16 (of 24) ]
[ time 8 ] Run JOB 0 at PRIORITY 1 [ TICKS 1 ALLOT 1 TIME 15 (of 24) ]
[ time 9 ] Run JOB 0 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 14 (of 24) ]
[ time 10 ] Run JOB 1 at PRIORITY 1 [ TICKS 9 ALLOT 1 TIME 39 (of 40) ]
[ time 11 ] Run JOB 1 at PRIORITY 1 [ TICKS 8 ALLOT 1 TIME 38 (of 40) ]
[ time 12 ] Run JOB 1 at PRIORITY 1 [ TICKS 7 ALLOT 1 TIME 37 (of 40) ]
[ time 13 ] Run JOB 1 at PRIORITY 1 [ TICKS 6 ALLOT 1 TIME 36 (of 40) ]
[ time 14 ] Run JOB 1 at PRIORITY 1 [ TICKS 5 ALLOT 1 TIME 35 (of 40) ]
[ time 15 ] Run JOB 1 at PRIORITY 1 [ TICKS 4 ALLOT 1 TIME 34 (of 40) ]
[ time 16 ] Run JOB 1 at PRIORITY 1 [ TICKS 3 ALLOT 1 TIME 33 (of 40) ]
[ time 17 ] Run JOB 1 at PRIORITY 1 [ TICKS 2 ALLOT 1 TIME 32 (of 40) ]
[ time 18 ] Run JOB 1 at PRIORITY 1 [ TICKS 1 ALLOT 1 TIME 31 (of 40) ]
[ time 19 ] Run JOB 1 at PRIORITY 1 [ TICKS 0 ALLOT 1 TIME 30 (of 40) ]
[ time 20 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 13 (of 24) ]
[ time 21 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 12 (of 24) ]
[ time 22 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 11 (of 24) ]
[ time 23 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 10 (of 24) ]
[ time 24 ] Run JOB 0 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 9 (of 24) ]
[ time 25 ] Run JOB 0 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 8 (of 24) ]
[ time 26 ] Run JOB 0 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 7 (of 24) ]
[ time 27 ] Run JOB 0 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 6 (of 24) ]
[ time 28 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 5 (of 24) ]
[ time 29 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 4 (of 24) ]
[ time 30 ] Run JOB 1 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 29 (of 40) ]
[ time 31 ] Run JOB 1 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 28 (of 40) ]
[ time 32 ] Run JOB 1 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 27 (of 40) ]
[ time 33 ] Run JOB 1 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 26 (of 40) ]
[ time 34 ] Run JOB 1 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 25 (of 40) ]
[ time 35 ] Run JOB 1 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 24 (of 40) ]
[ time 36 ] Run JOB 1 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 23 (of 40) ]
[ time 37 ] Run JOB 1 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 22 (of 40)
[ time 38 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 21 (of 40)
[ time 39 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 20 (of 40)
[ time 40 ] Run JOB 0 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 3 (of 24) ]
[ time 41 ] Run JOB 0 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 2 (of 24) ]
[ time 42 ] Run JOB 0 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 1 (of 24) ]
[ time 43 ] Run JOB 0 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 0 (of 24) ]
[ time 44 ] FINISHED JOB 0
[ time 44 ] Run JOB 1 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 19 (of 40)
[ time 45 ] Run JOB 1 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 18 (of 40)
[ time 46 ] Run JOB 1 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 17 (of 40)
[ time 47 ] Run JOB 1 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 16 (of 40)
[ time 48 ] Run JOB 1 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 15 (of 40)
[ time 49 ] Run JOB 1 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 14 (of 40)
[ time 50 ] Run JOB 1 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 13 (of 40)
[ time 51 ] Run JOB 1 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 12 (of 40)
[ time 52 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 11 (of 40)
[ time 53 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 10 (of 40)
[ time 54 ] Run JOB 1 at PRIORITY 0 [ TICKS 9 ALLOT 1 TIME 9 (of 40) ]
[ time 55 ] Run JOB 1 at PRIORITY 0 [ TICKS 8 ALLOT 1 TIME 8 (of 40) ]
[ time 56 ] Run JOB 1 at PRIORITY 0 [ TICKS 7 ALLOT 1 TIME 7 (of 40) ]
[ time 57 ] Run JOB 1 at PRIORITY 0 [ TICKS 6 ALLOT 1 TIME 6 (of 40) ]
[ time 58 ] Run JOB 1 at PRIORITY 0 [ TICKS 5 ALLOT 1 TIME 5 (of 40) ]
[ time 59 ] Run JOB 1 at PRIORITY 0 [ TICKS 4 ALLOT 1 TIME 4 (of 40) ]
[ time 60 ] Run JOB 1 at PRIORITY 0 [ TICKS 3 ALLOT 1 TIME 3 (of 40) ]
[ time 61 ] Run JOB 1 at PRIORITY 0 [ TICKS 2 ALLOT 1 TIME 2 (of 40) ]
[ time 62 ] Run JOB 1 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 1 (of 40) ]
[ time 63 ] Run JOB 1 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 0 (of 40) ]
[ time 64 ] FINISHED JOB 1

Final statistics:
  Job  0: startTime   0 - response   0 - turnaround  44
  Job  1: startTime   0 - response  10 - turnaround  64

  Avg  1: startTime n/a - response 5.00 - turnaround 54.00
```

3. 将如何配置调度程序参数，像轮转调度程序那样工作?

当工作在同一级队列时，作业会依此轮转执行，因此只需要设置一个队列，就可以像轮转调度程序那样工作。

命令为./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。

需要保证一个长时间运行的工作可以分配到至少 5%的 cpu（时间片长度为 10ms），那么就是说每 200ms 至少有一个时间片由此工作执行，那么应该每 200s 将一个工作重新投入到最高优先级队列。

命令为-B 200

第九章

1. 计算 3 个工作在随机种子为 1、2 和 3 时的模拟解。

(1) 随机数种子 seed=1

```
zlz@zlz-virtual-machine:~$ ./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
```

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

可以通过-c 验证我们的想法

```
zlz@zlz-virtual-machine:~$ ./lottery.py -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: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
```

(2)　随机数种子 seed=2

```
zlz@zlz-virtual-machine:~$ ./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
```

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
23			6（完成）	325143	30	3	job2

可以通过-c 验证我们的想法

```
zlz@zlz-virtual-machine:~$ ./lottery.py -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
```

(3)随机数种子 seed=3

```
zlz@zlz-virtual-machine:~$ ./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
```

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

可以通过-c 验证我们的想法

```
zlz@zlz-virtual-machine:~$ ./lottery.py -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
```

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）。

使用./lottery.py -l 10:1,10:100 进行模拟

```
zlz@zlz-virtual-machine:~$ ./lottery.py -l 10:1,10:100
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		工作长度：10，10					
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
20	10（完成）			902166	1	0	job0

会导致彩票数特别大的工作几乎一直占用 cpu。在工作 1 完成前，工作 0 可能会运行，但是概率很小，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，持有份额小的工作几乎不执行，响应时间和周转时间很长，长期处于饥饿状态。

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。

选取前十个随机种子进行验证

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
all			85

可以看出，平均提前完成时间为 8.5，可以明显看出不公平现象，大多数进程 0 占用更多的彩票数。

第十五章

1. 用种子 1、2 和 3 运行，并计算进程生成的每个虚拟地址是处于界限内还是界限外？如果在界限内，请计算地址转换。

（1）种子数 seed=1

```
zlz@zlz-virtual-machine:~$ ./relocation.py -s 1

ARG seed 1
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x0000363c (decimal 13884)
  Limit  : 290

Virtual Address Trace
  VA  0: 0x0000030e (decimal:  782) --> PA or segmentation violation?
  VA  1: 0x00000105 (decimal:  261) --> PA or segmentation violation?
  VA  2: 0x000001fb (decimal:  507) --> PA or segmentation violation?
  VA  3: 0x000001cc (decimal:  460) --> PA or segmentation violation?
  VA  4: 0x0000029b (decimal:  667) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.
```

VA 0: 0x0000030e (decimal: 782) --> SEGMENTATION VIOLATION

VA 1: 0x00000105 (decimal: 261) --> VALID: 0x00003741 (decimal: 14145)

VA 2: 0x000001fb (decimal: 507) --> SEGMENTATION VIOLATION

VA 3: 0x000001cc (decimal: 460) --> SEGMENTATION VIOLATION
VA 4: 0x0000029b (decimal: 667) --> SEGMENTATION VIOLATION

（2）种子数 seed=2

```
zlz@zlz-virtual-machine:~$ ./relocation.py -s 2

ARG seed 2
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x00003ca9 (decimal 15529)
  Limit  : 500

Virtual Address Trace
  VA  0: 0x00000039 (decimal:   57) --> PA or segmentation violation?
  VA  1: 0x00000056 (decimal:   86) --> PA or segmentation violation?
  VA  2: 0x00000357 (decimal:  855) --> PA or segmentation violation?
  VA  3: 0x000002f1 (decimal:  753) --> PA or segmentation violation?
  VA  4: 0x000002ad (decimal:  685) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.
```

VA 0: 0x00000039 (decimal: 57) --> VALID: 0x00003ce2 (decimal: 15586)
VA 1: 0x00000056 (decimal: 86) --> VALID: 0x00003cff (decimal: 15615)
VA 2: 0x00000357 (decimal: 855) --> SEGMENTATION VIOLATION
VA 3: 0x000002f1 (decimal: 753) --> SEGMENTATION VIOLATION
VA 4: 0x000002ad (decimal: 685) --> SEGMENTATION VIOLATION

（3）种子数 seed=3

```
zlz@zlz-virtual-machine:~$ ./relocation.py -s 3

ARG seed 3
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x000022d4 (decimal 8916)
  Limit  : 316

Virtual Address Trace
  VA  0: 0x0000017a (decimal:  378) --> PA or segmentation violation?
  VA  1: 0x0000026a (decimal:  618) --> PA or segmentation violation?
  VA  2: 0x00000280 (decimal:  640) --> PA or segmentation violation?
  VA  3: 0x00000043 (decimal:   67) --> PA or segmentation violation?
  VA  4: 0x0000000d (decimal:   13) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.
```

```
VA  0: 0x0000017a (decimal:  378) --> SEGMENTATION VIOLATION
VA  1: 0x0000026a (decimal:  618) --> SEGMENTATION VIOLATION
VA  2: 0x00000280 (decimal:  640) --> SEGMENTATION VIOLATION
VA  3: 0x00000043 (decimal:   67) --> VALID: 0x00002317 (decimal:
8983)
VA  4: 0x0000000d (decimal:   13) --> VALID: 0x000022e1 (decimal:
8929)
```

3. **使用以下标志运行：-s 1 -n 10 -l 100。可以设置基址的最大值是多少，以便地址空间仍然完全放在物理内存中?**

```
zlz@zlz-virtual-machine:~$ ./relocation.py -s 1 -n 10 -l 100

ARG seed 1
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x00000899 (decimal 2201)
  Limit  : 100

Virtual Address Trace
  VA  0: 0x00000363 (decimal:  867) --> PA or segmentation violation?
  VA  1: 0x0000030e (decimal:  782) --> PA or segmentation violation?
  VA  2: 0x00000105 (decimal:  261) --> PA or segmentation violation?
  VA  3: 0x000001fb (decimal:  507) --> PA or segmentation violation?
  VA  4: 0x000001cc (decimal:  460) --> PA or segmentation violation?
  VA  5: 0x0000029b (decimal:  667) --> PA or segmentation violation?
  VA  6: 0x00000327 (decimal:  807) --> PA or segmentation violation?
  VA  7: 0x00000060 (decimal:   96) --> PA or segmentation violation?
  VA  8: 0x0000001d (decimal:   29) --> PA or segmentation violation?
  VA  9: 0x00000357 (decimal:  855) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.
```

经过查询，本题的翻译有误，问的应该是可以设置基址的最大值是多少

物理内存大小为 16k=16384

-s 1 -n 10 -l 100 可知，我们设置的限制寄存器 limit 的值为 100，也就是界限寄存器为 100，那么最大基址寄存器可以设置为 Base，Base+100<=16384，那么 Base<=16284，也就是说基址寄存器的最大值应为 16284

可以通过设置-b 来确定是否正确

```
zlz@zlz-virtual-machine:~$ ./relocation.py -s 1 -n 10 -l 100 -b 16285

ARG seed 1
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x00003f9d (decimal 16285)
  Limit  : 100

Error: address space does not fit into physical memory with those base/bounds values.
Base + Limit: 16385   Psize: 16384
```

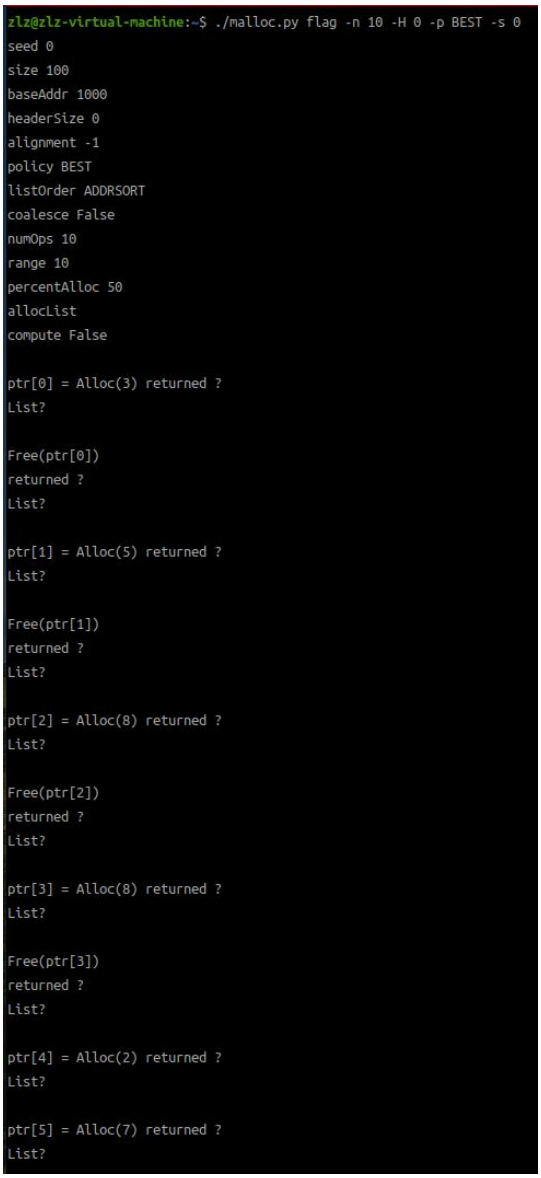
尝试将基址寄存器设置为 16285，有报错信息，不正确

```
zlz@zlz-virtual-machine:~$ ./relocation.py -s 1 -n 10 -l 100 -b 16284

ARG seed 1
ARG address space size 1k
ARG phys mem size 16k

Base-and-Bounds register information:

  Base   : 0x00003f9c (decimal 16284)
  Limit  : 100

Virtual Address Trace
  VA  0: 0x00000089 (decimal:  137) --> PA or segmentation violation?
  VA  1: 0x00000363 (decimal:  867) --> PA or segmentation violation?
  VA  2: 0x0000030e (decimal:  782) --> PA or segmentation violation?
  VA  3: 0x00000105 (decimal:  261) --> PA or segmentation violation?
  VA  4: 0x000001fb (decimal:  507) --> PA or segmentation violation?
  VA  5: 0x000001cc (decimal:  460) --> PA or segmentation violation?
  VA  6: 0x0000029b (decimal:  667) --> PA or segmentation violation?
  VA  7: 0x00000327 (decimal:  807) --> PA or segmentation violation?
  VA  8: 0x00000060 (decimal:   96) --> PA or segmentation violation?
  VA  9: 0x0000001d (decimal:   29) --> PA or segmentation violation?

For each virtual address, either write down the physical address it translates to
OR write down that it is an out-of-bounds address (a segmentation violation). For
this problem, you should assume a simple virtual address space of a given size.
```

将基址寄存器设置为 16284，正确，那么最大值确实为 16284

第十七章

1. **首先运行 flag -n 10 -H 0 -p BEST -s 0 来产生一些随机分配和释放。你能预测 malloc()/free() 会返回什么吗？你可以在每次请求后猜测空闲列表的状态吗？随着时间的推移，你对空闲列表有什么发现?**



使用命令./malloc.py flag -n 10 -H 0 -p BEST -s 0

生成 10 个随机操作，头部大小为 0，使用 BEST 策略，随机种子为 0

ptr[0] = Alloc(3) returned 1000 (searched 1 elements)

Free List [Size 1]: [addr:1003 sz:97]

Free(ptr[0])

returned 0

Free List [Size 2]: [addr:1000 sz:3][addr:1003 sz:97]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)

Free List [Size 2]: [addr:1000 sz:3][addr:1008 sz:92]

Free(ptr[1])

returned 0

Free List [Size 3]: [addr:1000 sz:3][addr:1003 sz:5][addr:1008 sz:92]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)

Free List [Size 3]: [addr:1000 sz:3][addr:1003 sz:5][addr:1016 sz:84]

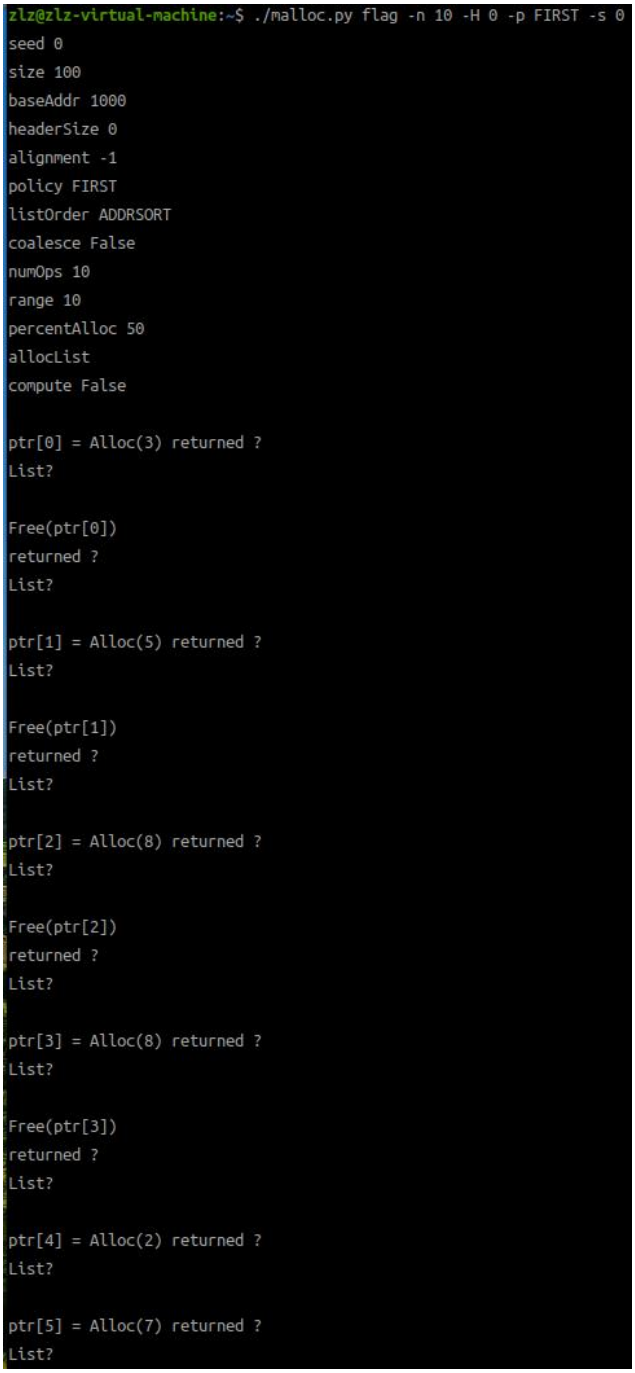
Free(ptr[2])

```
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]
ptr[3] = Alloc(8) returned 1008 (searched 4 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]
Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]
ptr[4] = Alloc(2) returned 1000 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]
ptr[5] = Alloc(7) returned 1008 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1015 sz:1 ][ addr:1016 sz:84 ]
```

6 次分配空间查找空闲块的次数分别为：1,2,3,4,4,4。

随着不断分配和释放空间，空闲列表中的空闲块逐渐增加，不断碎片化，变为较小的空闲块，且分配搜索空闲块的次数也随之增加。

3. **如果使用首次匹配（-p FIRST）会如何？使用首次匹配时，什么变快了？**



ptr[0] = Alloc(3) returned 1000 (searched 1 elements)

Free List [Size 1]: [addr:1003 sz:97]

Free(ptr[0])

returned 0

Free List [Size 2]: [addr:1000 sz:3][addr:1003 sz:97]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)

Free List [Size 2]: [addr:1000 sz:3][addr:1008 sz:92]

Free(ptr[1])

returned 0

Free List [Size 3]: [addr:1000 sz:3][addr:1003 sz:5][addr:1008 sz:92]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)

```
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]
Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]
ptr[3] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]
Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]
ptr[4] = Alloc(2) returned 1000 (searched 1 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]
ptr[5] = Alloc(7) returned 1008 (searched 3 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1015 sz:1 ][ addr:1016 sz:84 ]
```

两次碎片化程度一样，每次选取的空闲块一致，但在搜索空闲空间时不需要全部遍历完，首次匹配 6 次分配空间所需要的搜索次数分别为：1,2,3,3,1,3，搜寻空闲列表首次匹配更快一点。

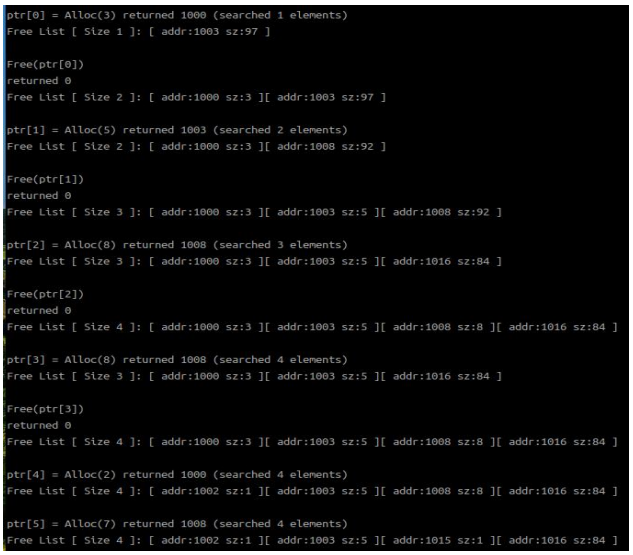
4. 对于上述问题，列表在保持有序时，可能会影响某些策略找到空闲位置所需的时间。使用不同的空闲列表排序（-l ADDRSORT，-l SIZESORT +，-l SIZESORT-）查看策略和列表排序如何相互影响。

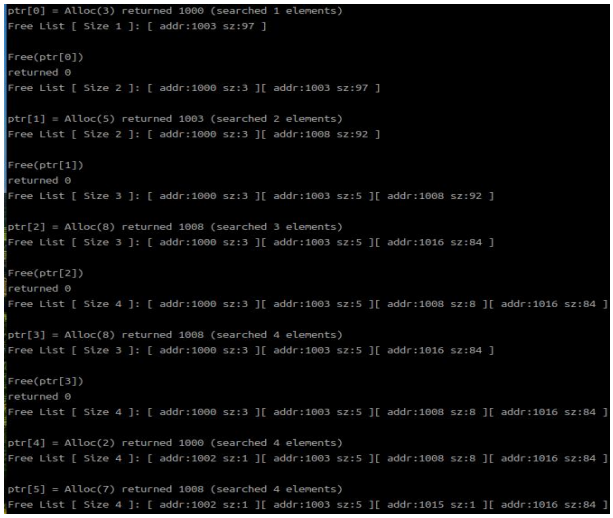
不同的空闲列表排序方式对最优(BEST)和最差(WORST)分配这种分配策略的效率没有影响

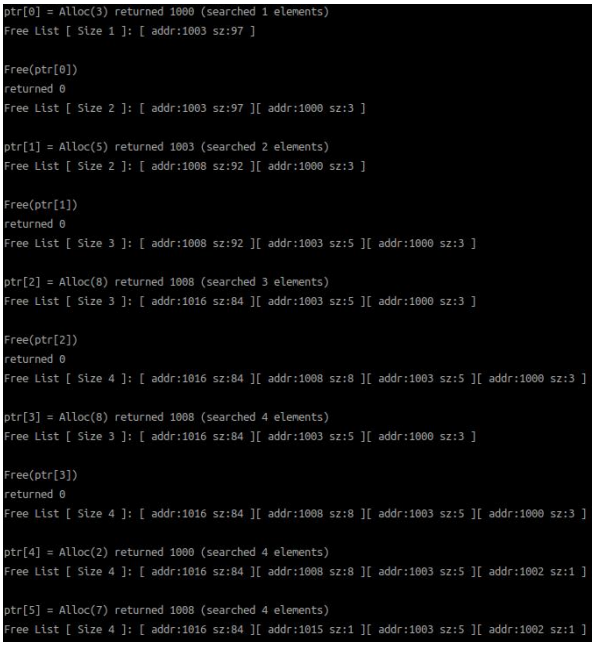
而对于像首次(FIRST)分配这样的分配策略，不同的空闲列表排序方式将会影响其分配策略的效率

这与分配策略的具体思路是相关的：最优和最差分配无论空闲块如何排列，每次分配都需要遍历所有的空闲块，而首次分配只需要找到首个足够大的空闲块，所以与空闲块的排列有关。

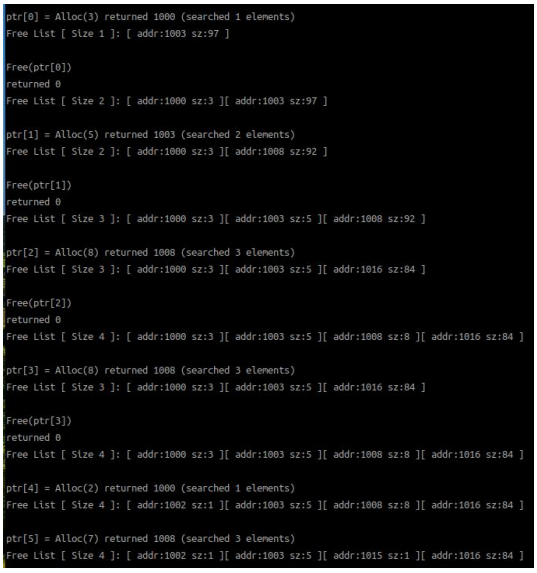
BEST 分配策略：




ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [Size 1]: [addr:1003 sz:97]
Free(ptr[0])
returned 0
Free List [Size 2]: [addr:1000 sz:3][addr:1003 sz:97]
ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [Size 2]: [addr:1000 sz:3][addr:1008 sz:92]
Free(ptr[1])
returned 0
Free List [Size 3]: [addr:1000 sz:3][addr:1003 sz:5][addr:1008 sz:92]
ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [Size 3]: [addr:1000 sz:3][addr:1003 sz:5][addr:1016 sz:84]
Free(ptr[2])
returned 0
Free List [Size 4]: [addr:1000 sz:3][addr:1003 sz:5][addr:1008 sz:8][addr:1016 sz:84]
ptr[3] = Alloc(8) returned 1008 (searched 4 elements)
Free List [Size 3]: [addr:1000 sz:3][addr:1003 sz:5][addr:1016 sz:84]
Free(ptr[3])
returned 0
Free List [Size 4]: [addr:1000 sz:3][addr:1003 sz:5][addr:1008 sz:8][addr:1016 sz:84]
ptr[4] = Alloc(2) returned 1000 (searched 4 elements)
Free List [Size 4]: [addr:1002 sz:1][addr:1003 sz:5][addr:1008 sz:8][addr:1016 sz:84]
ptr[5] = Alloc(7) returned 1008 (searched 4 elements)
Free List [Size 4]: [addr:1002 sz:1][addr:1003 sz:5][addr:1015 sz:1][addr:1016 sz:84]


ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [Size 1]: [addr:1003 sz:97]
Free(ptr[0])
returned 0
Free List [Size 2]: [addr:1003 sz:97][addr:1000 sz:3]
ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [Size 2]: [addr:1008 sz:92][addr:1000 sz:3]
Free(ptr[1])
returned 0
Free List [Size 3]: [addr:1008 sz:92][addr:1003 sz:5][addr:1000 sz:3]
ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [Size 3]: [addr:1016 sz:84][addr:1003 sz:5][addr:1000 sz:3]
Free(ptr[2])
returned 0
Free List [Size 4]: [addr:1016 sz:84][addr:1008 sz:8][addr:1003 sz:5][addr:1000 sz:3]
ptr[3] = Alloc(8) returned 1008 (searched 4 elements)
Free List [Size 3]: [addr:1016 sz:84][addr:1003 sz:5][addr:1000 sz:3]
Free(ptr[3])
returned 0
Free List [Size 4]: [addr:1016 sz:84][addr:1008 sz:8][addr:1003 sz:5][addr:1000 sz:3]
ptr[4] = Alloc(2) returned 1000 (searched 4 elements)
Free List [Size 4]: [addr:1016 sz:84][addr:1008 sz:8][addr:1003 sz:5][addr:1002 sz:1]
ptr[5] = Alloc(7) returned 1008 (searched 4 elements)
Free List [Size 4]: [addr:1016 sz:84][addr:1015 sz:1][addr:1003 sz:5][addr:1002 sz:1]

FIRST 分配策略：


ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [Size 1]: [addr:1003 sz:97]
Free(ptr[0])
returned 0
Free List [Size 2]: [addr:1000 sz:3][addr:1003 sz:97]
ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [Size 2]: [addr:1000 sz:3][addr:1008 sz:92]
Free(ptr[1])
returned 0
Free List [Size 3]: [addr:1000 sz:3][addr:1003 sz:5][addr:1008 sz:92]
ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [Size 3]: [addr:1000 sz:3][addr:1003 sz:5][addr:1016 sz:84]
Free(ptr[2])
returned 0
Free List [Size 4]: [addr:1000 sz:3][addr:1003 sz:5][addr:1008 sz:8][addr:1016 sz:84]
ptr[3] = Alloc(8) returned 1008 (searched 3 elements)
Free List [Size 3]: [addr:1000 sz:3][addr:1003 sz:5][addr:1016 sz:84]
Free(ptr[3])
returned 0
Free List [Size 4]: [addr:1000 sz:3][addr:1003 sz:5][addr:1008 sz:8][addr:1016 sz:84]
ptr[4] = Alloc(2) returned 1000 (searched 1 elements)
Free List [Size 4]: [addr:1002 sz:1][addr:1003 sz:5][addr:1008 sz:8][addr:1016 sz:84]
ptr[5] = Alloc(7) returned 1008 (searched 3 elements)
Free List [Size 4]: [addr:1002 sz:1][addr:1003 sz:5][addr:1015 sz:1][addr:1016 sz:84]

```
ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1003 sz:97 ]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1008 sz:92 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:92 ]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[3] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[4] = Alloc(2) returned 1000 (searched 1 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[5] = Alloc(7) returned 1008 (searched 3 elements)
Free List [ Size 4 ]: [ addr:1002 sz:1 ][ addr:1003 sz:5 ][ addr:1015 sz:1 ][ addr:1016 sz:84 ]
```

```
ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1003 sz:97 ][ addr:1000 sz:3 ]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1008 sz:92 ][ addr:1000 sz:3 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1008 sz:92 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1016 sz:84 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1016 sz:84 ][ addr:1008 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

ptr[3] = Alloc(8) returned 1008 (searched 4 elements)
Free List [ Size 3 ]: [ addr:1016 sz:84 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

Free(ptr[3])
returned 0
Free List [ Size 4 ]: [ addr:1016 sz:84 ][ addr:1008 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

ptr[4] = Alloc(2) returned 1000 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1016 sz:84 ][ addr:1008 sz:8 ][ addr:1003 sz:5 ][ addr:1002 sz:1 ]

ptr[5] = Alloc(7) returned 1008 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1016 sz:84 ][ addr:1015 sz:1 ][ addr:1003 sz:5 ][ addr:1002 sz:1 ]
```

WORST 分配策略：

```
ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1003 sz:97 ]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1008 sz:92 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:92 ]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[3] = Alloc(8) returned 1016 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1024 sz:76 ]

Free(ptr[3])
returned 0
Free List [ Size 5 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1024 sz:76 ]

ptr[4] = Alloc(2) returned 1024 (searched 5 elements)
Free List [ Size 5 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1026 sz:74 ]

ptr[5] = Alloc(7) returned 1026 (searched 5 elements)
Free List [ Size 5 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1033 sz:67 ]
```

```
ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1003 sz:97 ]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1000 sz:3 ][ addr:1008 sz:92 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:92 ]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1016 sz:84 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:84 ]

ptr[3] = Alloc(8) returned 1016 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1024 sz:76 ]

Free(ptr[3])
returned 0
Free List [ Size 5 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1024 sz:76 ]

ptr[4] = Alloc(2) returned 1024 (searched 5 elements)
Free List [ Size 5 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1026 sz:74 ]

ptr[5] = Alloc(7) returned 1026 (searched 5 elements)
Free List [ Size 5 ]: [ addr:1000 sz:3 ][ addr:1003 sz:5 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1033 sz:67 ]
```

```
ptr[0] = Alloc(3) returned 1000 (searched 1 elements)
Free List [ Size 1 ]: [ addr:1003 sz:97 ]

Free(ptr[0])
returned 0
Free List [ Size 2 ]: [ addr:1003 sz:97 ][ addr:1000 sz:3 ]

ptr[1] = Alloc(5) returned 1003 (searched 2 elements)
Free List [ Size 2 ]: [ addr:1008 sz:92 ][ addr:1000 sz:3 ]

Free(ptr[1])
returned 0
Free List [ Size 3 ]: [ addr:1008 sz:92 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

ptr[2] = Alloc(8) returned 1008 (searched 3 elements)
Free List [ Size 3 ]: [ addr:1016 sz:84 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

Free(ptr[2])
returned 0
Free List [ Size 4 ]: [ addr:1016 sz:84 ][ addr:1008 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

ptr[3] = Alloc(8) returned 1016 (searched 4 elements)
Free List [ Size 4 ]: [ addr:1024 sz:76 ][ addr:1008 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

Free(ptr[3])
returned 0
Free List [ Size 5 ]: [ addr:1024 sz:76 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

ptr[4] = Alloc(2) returned 1024 (searched 5 elements)
Free List [ Size 5 ]: [ addr:1026 sz:74 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]

ptr[5] = Alloc(7) returned 1026 (searched 5 elements)
Free List [ Size 5 ]: [ addr:1033 sz:67 ][ addr:1008 sz:8 ][ addr:1016 sz:8 ][ addr:1003 sz:5 ][ addr:1000 sz:3 ]
```