Operating System Bonus: OpenEuler

Susie Glitter

2025 年 7 月 7 日

为支持国产操作系统，在此使用 OpenEuler 重新完成操作系统的八个大作业

1 环境搭建

1.1 OpenEuler 虚拟机安装

由于大作业需要比较古老的内核(4 开头?),我使用了最古老的openEuler 20.03 LTS SP3

使用 VMware 安装虚拟机，硬盘大小给了 32G，内存大小需要 2G，否则安装时会卡死

下面最好在 root 用户中进行操作

1.2 网络问题

ping 一下看看能不能上网，大概率是不行

这个链接指示修改 ifcfg-ens33 文件，使其开机时启动网卡，再 reboot，终于通网了

1.3 图形化界面安装

再用 yum 安装 dde 图形化界面，可以加上-y 自动确认

```
sudo yum update
sudo yum install dde
sudo systemctl set-default graphical.target
```

```
4 sudo reboot
```

可能要安两个小时

1.4 安装 kernel-devel

这一步是为了获得 make 所需的 build 文件夹内容，openEuler 不自带这个文件夹

```
1 yum install kernel-devel-$(uname -r)
```

1.5 安装 JDK

这个链接教了怎么安装 JDK 和配置环境
编译为字节码的指令和书上有所区别，需要“javac 完整文件名”
到此，我们就可以在 openEuler 上面完成操作系统的八个大作业了

2 大作业一：Linux 内核模块

2.1 任务一：Jiffies

```
[gg@localhost jiffies]$ sh jiffies.sh
+ make
make -C /lib/modules/4.19.90-2401.1.0.0233.oe1.x86_64/build M=/home/gg/Desktop/os/ch2/ji
ffies modules
make[1]: 进入目录“/usr/src/kernels/4.19.90-2401.1.0.0233.oe1.x86_64”
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: 离开目录“/usr/src/kernels/4.19.90-2401.1.0.0233.oe1.x86_64”
+ sudo dmesg -c
[ 1297.995467] /proc/jiffies created
[ 1301.034508] /proc/jiffies removed
+ sudo insmod jiffies.ko
+ sudo dmesg
[ 2994.198927] /proc/jiffies created
+ cat /proc/jiffies
jiffies:4297660998
+ sleep 1s
+ cat /proc/jiffies
jiffies:4297662003
+ sleep 2s
+ cat /proc/jiffies
jiffies:4297664009
+ sudo rmmod jiffies.ko
+ sudo dmesg
[ 2994.198927] /proc/jiffies created
[ 2997.244779] /proc/jiffies removed
[gg@localhost jiffies]$
```

2.2 任务二：seconds

```
[gg@localhost seconds]$ sh seconds.sh
+ make
make -C /lib/modules/4.19.90-2401.1.0.0233.oe1.x86_64/build M=/home/gg/Desktop/os/ch2/se
conds modules
make[1]: 进入目录“/usr/src/kernels/4.19.90-2401.1.0.0233.oe1.x86_64”
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: 离开目录“/usr/src/kernels/4.19.90-2401.1.0.0233.oe1.x86_64”
+ sudo dmesg -c
[ 3186.959971] /proc/seconds created
[ 3199.000343] /proc/seconds removed
+ sudo insmod seconds.ko
+ sudo dmesg
[ 3210.769840] /proc/seconds created
+ cat /proc/seconds
seconds:4297877
+ sleep 2s
+ cat /proc/seconds
seconds:4297879
+ sleep 10s
+ cat /proc/seconds
seconds:4297889
+ sudo rmmod seconds.ko
+ sudo dmesg
[ 3210.769840] /proc/seconds created
[ 3222.808391] /proc/seconds removed
[gg@localhost seconds]$
```

3 大作业二：Shell 与 Linux 内核模块

3.1 任务一：Shell osh>

测试用例 1：命令执行与 &

```
[gg@localhost osh]$ gcc osh.c -o osh
[gg@localhost osh]$ ./osh
osh>ls -l
总用量 28
-rwxrwxr-x 1 gg gg 17984  5月 30 14:16 osh
-rw-r--r-- 1 gg gg  5946  4月 15 01:43 osh.c
osh>ls -l &
osh>总用量 28
-rwxrwxr-x 1 gg gg 17984  5月 30 14:16 osh
-rw-r--r-- 1 gg gg  5946  4月 15 01:43 osh.c
exit
[gg@localhost osh]$
```

测试用例 2：历史记录

```
[gg@localhost osh]$ ./osh
osh>!!
No commands in history.
osh>ls -l
总用量 28
-rwxrwxr-x 1 gg gg 17984  5月 30 14:16 osh
-rw-r--r-- 1 gg gg  5946  4月 15 01:43 osh.c
osh>!!
总用量 28
-rwxrwxr-x 1 gg gg 17984  5月 30 14:16 osh
-rw-r--r-- 1 gg gg  5946  4月 15 01:43 osh.c
osh>
```

测试用例 3：文件重定向（额外实现同时重定向输入输出）

```
[gg@localhost osh]$ ./osh
osh>cat in.txt
3
4
2
1
5
osh>sort <in.txt >out.txt
osh>cat out.txt
1
2
3
4
5
osh>
```

测试用例 4：管道通信（额外实现同时使用管道与重定向）

```
[gg@localhost osh]$ ./osh
osh>sort <in.txt | tee >out.txt
osh>cat out.txt
1
2
3
4
5
osh>
```

3.2 任务二：写入/proc 文件

用批处理获取第一个进程（一般是 bash）的 pid，写入/proc/pid，再从/proc/pid 中读取信息

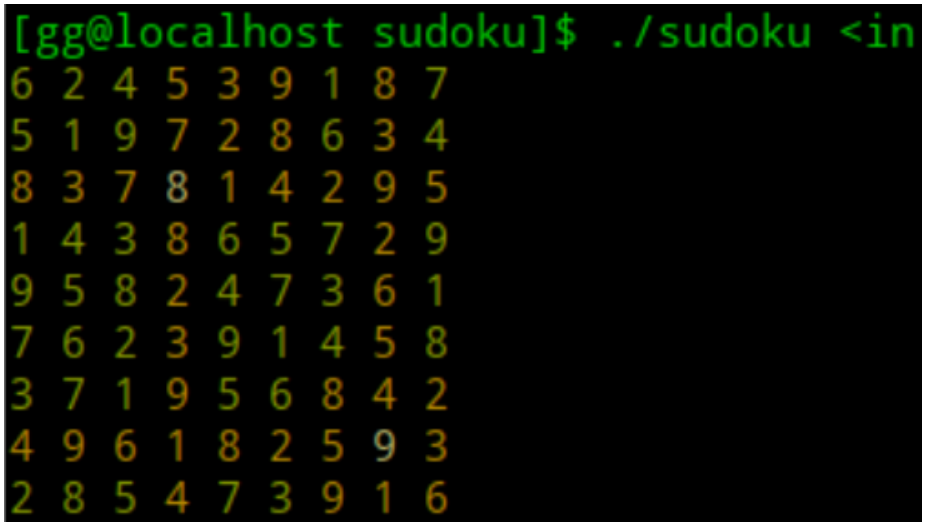
```
1 proc_name="pid"
2 pid=`ps | awk 'NR==2 {print $1}'`
3 echo ${pid} > /proc/${proc_name}
4 cat /proc/${proc_name}
```

```
[gg@localhost pid]$ sh pid.sh
+ make
make -C /lib/modules/4.19.90-2401.1.0.0233.oe1.x86_64/build M=/home/gg/Desktop/os/ch3/pi
d modules
make[1]: 进入目录"/usr/src/kernels/4.19.90-2401.1.0.0233.oe1.x86_64"
  Building modules, stage 2.
  MODPOST 1 modules
make[1]: 离开目录"/usr/src/kernels/4.19.90-2401.1.0.0233.oe1.x86_64"
+ sudo dmesg -c
[ 5072.948295] /proc/pid created
[ 5072.979957] /proc/pid removed
+ sudo insmod pid.ko
+ sudo dmesg
[ 5087.437096] /proc/pid created
++ awk 'NR==2 {print $1}'
++ ps
+ pid=29379
+ echo pid: 29379
pid: 29379
+ echo 29379
+ cat /proc/pid
command=[bash], pid=[29379], state=[1]
+ sudo rmmod pid.ko
+ sudo dmesg
[ 5087.437096] /proc/pid created
[ 5087.473455] /proc/pid removed
[gg@localhost pid]$
```

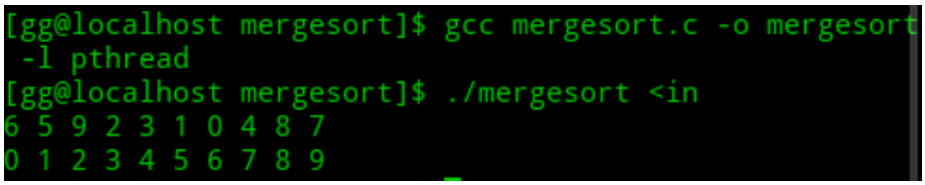
4 大作业三：多线程编程

4.1 任务一：多线程数独校验

编译记得加上 -l pthread

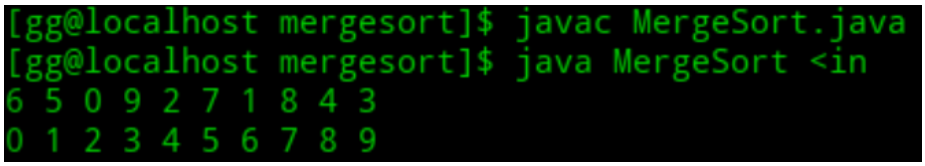


4.2 任务二：多线程归并排序



4.3 任务三：java 实现两种多线程排序

归并排序



快速排序

```
[gg@localhost quicksort]$ javac QuickSort.java
[gg@localhost quicksort]$ java QuickSort <in
6 5 0 9 2 7 1 8 4 3
0 1 2 3 4 5 6 7 8 9
```

5 大作业四：调度算法

5.1 任务一：实现五种调度算法

若 make 失败，删除所有.o 文件重试

fcfs

```
[root@localhost posix]# ./fcfs schedule.txt
Scheduling algorithm: fcfs
t=0     Running task = [T1] [4] [20]    for 20 units.
t=20    Running task = [T2] [3] [25]    for 25 units.
t=45    Running task = [T3] [3] [25]    for 25 units.
t=70    Running task = [T4] [5] [15]    for 15 units.
t=85    Running task = [T5] [5] [20]    for 20 units.
t=105   Running task = [T6] [1] [10]    for 10 units.
t=115   Running task = [T7] [3] [30]    for 30 units.
t=145   Running task = [T8] [10] [25]   for 25 units.
```

sjf

```
[root@localhost posix]# ./sjf schedule.txt
Scheduling algorithm: sjf
t=0     Running task = [T6] [1] [10]    for 10 units.
t=10    Running task = [T4] [5] [15]    for 15 units.
t=25    Running task = [T1] [4] [20]    for 20 units.
t=45    Running task = [T5] [5] [20]    for 20 units.
t=65    Running task = [T2] [3] [25]    for 25 units.
t=90    Running task = [T3] [3] [25]    for 25 units.
t=115   Running task = [T8] [10] [25]   for 25 units.
t=140   Running task = [T7] [3] [30]    for 30 units.
```

rr

```
[root@localhost posix]# ./rr schedule.txt
Scheduling algorithm: rr
t=0     Running task = [T1] [4] [20]    for 10 units.
t=10    Running task = [T2] [3] [25]    for 10 units.
t=20    Running task = [T3] [3] [25]    for 10 units.
t=30    Running task = [T4] [5] [15]    for 10 units.
t=40    Running task = [T5] [5] [20]    for 10 units.
t=50    Running task = [T6] [1] [10]    for 10 units.
t=60    Running task = [T7] [3] [30]    for 10 units.
t=70    Running task = [T8] [10] [25]   for 10 units.
t=80    Running task = [T1] [4] [10]    for 10 units.
t=90    Running task = [T2] [3] [15]    for 10 units.
t=100   Running task = [T3] [3] [15]    for 10 units.
t=110   Running task = [T4] [5] [5]     for 5 units.
t=115   Running task = [T5] [5] [10]    for 10 units.
t=125   Running task = [T7] [3] [20]    for 10 units.
t=135   Running task = [T8] [10] [15]   for 10 units.
t=145   Running task = [T2] [3] [5]     for 5 units.
t=150   Running task = [T3] [3] [5]     for 5 units.
t=155   Running task = [T7] [3] [10]    for 10 units.
t=165   Running task = [T8] [10] [5]    for 5 units.
```

priority

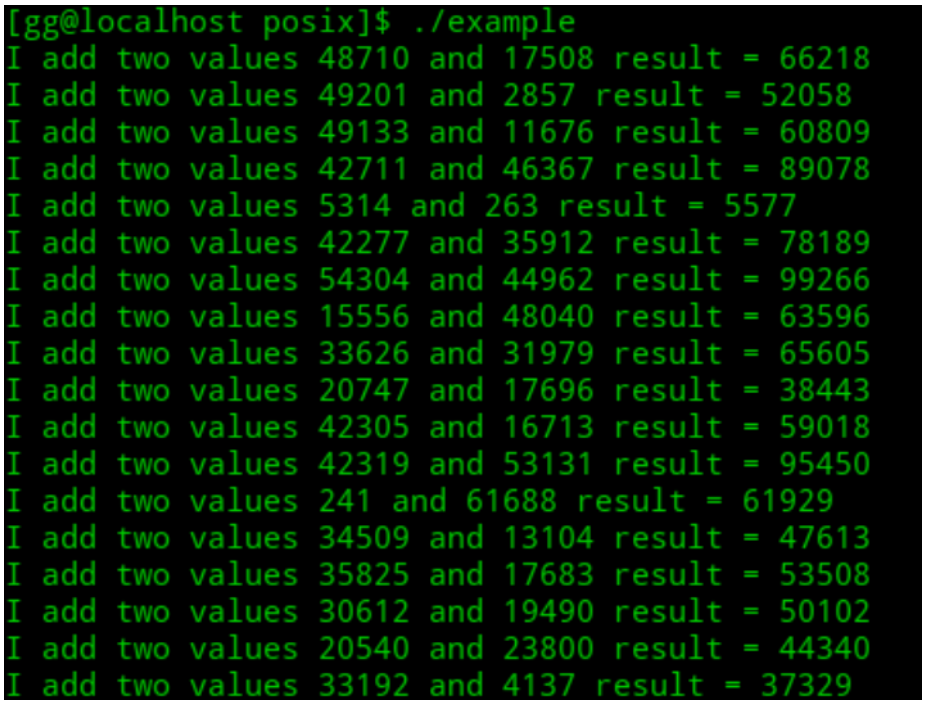
```
[root@localhost posix]# ./priority schedule.txt
Scheduling algorithm: priority
t=0     Running task = [T8] [10] [25]   for 25 units.
t=25    Running task = [T4] [5] [15]    for 15 units.
t=40    Running task = [T5] [5] [20]    for 20 units.
t=60    Running task = [T1] [4] [20]    for 20 units.
t=80    Running task = [T2] [3] [25]    for 25 units.
t=105   Running task = [T3] [3] [25]    for 25 units.
t=130   Running task = [T7] [3] [30]    for 30 units.
t=160   Running task = [T6] [1] [10]    for 10 units.
```

priority_rr

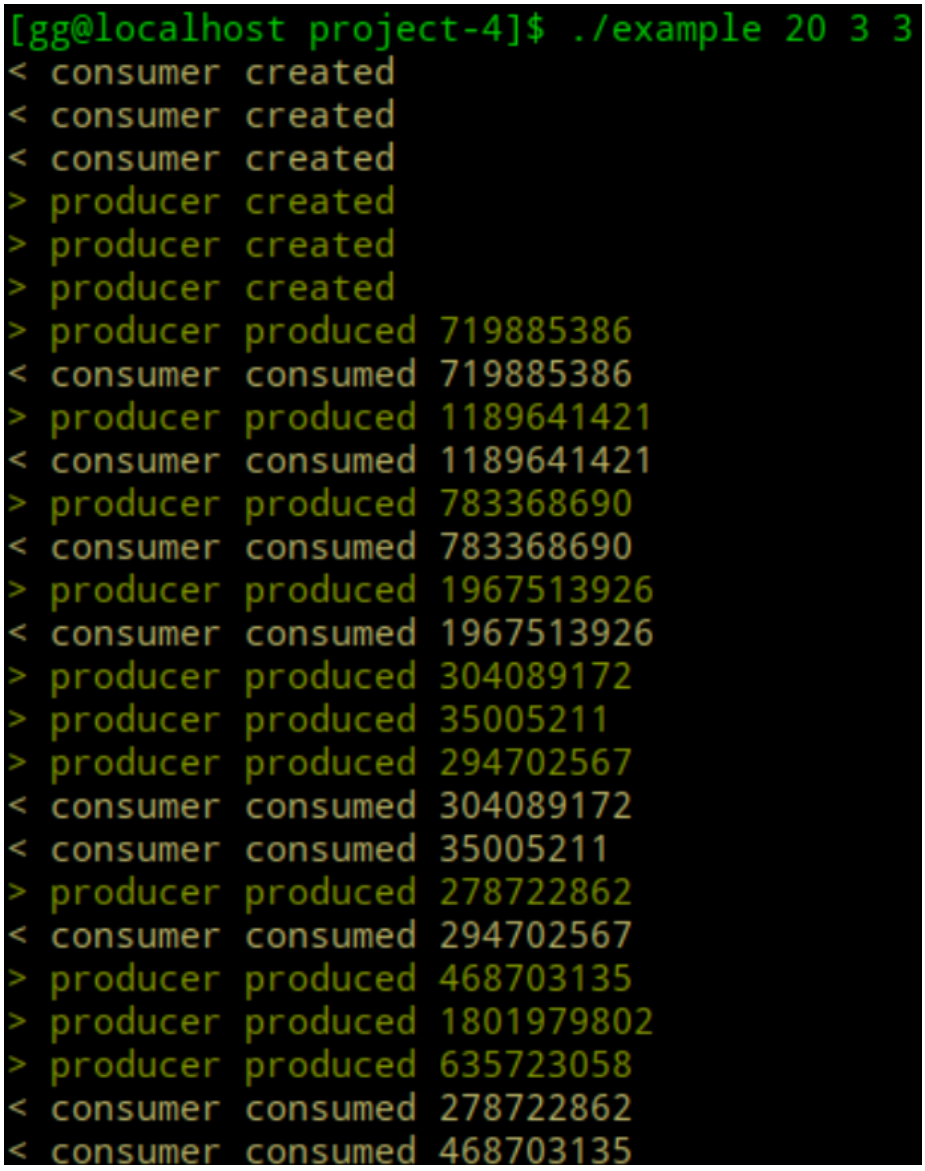
```
[root@localhost posix]# ./priority_rr schedule.txt
Scheduling algorithm: priority_rr
t=0     Running task = [T8] [10] [25]   for 10 units.
t=10    Running task = [T8] [10] [15]   for 10 units.
t=20    Running task = [T8] [10] [5]    for 5 units.
t=25    Running task = [T4] [5] [15]    for 10 units.
t=35    Running task = [T5] [5] [20]    for 10 units.
t=45    Running task = [T4] [5] [5]     for 5 units.
t=50    Running task = [T5] [5] [10]    for 10 units.
t=60    Running task = [T1] [4] [20]    for 10 units.
t=70    Running task = [T1] [4] [10]    for 10 units.
t=80    Running task = [T2] [3] [25]    for 10 units.
t=90    Running task = [T3] [3] [25]    for 10 units.
t=100   Running task = [T7] [3] [30]    for 10 units.
t=110   Running task = [T2] [3] [15]    for 10 units.
t=120   Running task = [T3] [3] [15]    for 10 units.
t=130   Running task = [T7] [3] [20]    for 10 units.
t=140   Running task = [T2] [3] [5]     for 5 units.
t=145   Running task = [T3] [3] [5]     for 5 units.
t=150   Running task = [T7] [3] [10]    for 10 units.
t=160   Running task = [T6] [1] [10]    for 10 units.
```

6 大作业五：线程池与生产者消费者问题

6.1 任务一：线程池



6.2 任务二：生产者消费者问题



7 大作业六：银行家算法

7.1 任务一：实现银行家算法

```
[gg@localhost banker]$ ./banker 6 6 7 8 <input2.txt
>*
available array is
6 6 7 8
maximum matrix is
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
allocation matrix is
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
need matrix is
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5

>RQ 0 3 3 3 3
Successfully allocate the resources!

>RQ 1 2 2 2 2
The state is not safe!
```

```
>RL 0 5 5 5 5
0 customer doesn't have this much resources!

>RL 0 1 2 1 2
Successfully release the resources!

>*
available array is
4 5 5 7
maximum matrix is
6 4 7 3
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
allocation matrix is
2 1 2 1
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
need matrix is
4 3 5 2
4 2 3 2
2 5 3 3
6 3 3 2
5 6 7 5
```

8 大作业七：连续内存分配

8.1 任务一：实现连续内存分配

```
[gg@localhost ch9]$ gcc allocator.c -o allocator
[gg@localhost ch9]$ ./allocator
Addresses [0:1048575] Unused
>RQ A 10000 B
Allocate Addresses [0:12287] For Process A
>RQ B 1 W
Allocate Addresses [12288:16383] For Process B
>RQ A 100 B
Allocate Addresses [16384:20479] For Process A
>RQ C 1 W
Allocate Addresses [20480:24575] For Process C
>STAT
Addresses [0:12287] Process A
Addresses [12288:16383] Process B
Addresses [16384:20479] Process A
Addresses [20480:24575] Process C
Addresses [24576:1048575] Unused
>RL A
Deallocate Addresses [0:12287] For Process A
Deallocate Addresses [16384:20479] For Process A
>STAT
Addresses [0:12287] Unused
Addresses [12288:16383] Process B
Addresses [16384:20479] Unused
Addresses [20480:24575] Process C
Addresses [24576:1048575] Unused
>RQ A 1 B
Allocate Addresses [16384:20479] For Process A
>RL A
Deallocate Addresses [16384:20479] For Process A
>RQ A 1 W
Allocate Addresses [24576:28671] For Process A
>RL A
Deallocate Addresses [24576:28671] For Process A
>RQ A 1 F
Allocate Addresses [0:4095] For Process A
>RL A
Deallocate Addresses [0:4095] For Process A
>RL C
Deallocate Addresses [20480:24575] For Process C
>STAT
Addresses [0:12287] Unused
Addresses [12288:16383] Process B
Addresses [16384:1048575] Unused
>
```

9 大作业八：虚拟内存分配器

9.1 任务一：实现 TLB 与页表

```
Virtual address: 49847 Physical address: 31671 Value: -83
Virtual address: 30032 Physical address: 592 Value: 0
Virtual address: 48065 Physical address: 25793 Value: 0
Virtual address: 6957 Physical address: 26413 Value: 0
Virtual address: 2301 Physical address: 35325 Value: 0
Virtual address: 7736 Physical address: 57912 Value: 0
Virtual address: 31260 Physical address: 23324 Value: 0
Virtual address: 17071 Physical address: 175 Value: -85
Virtual address: 8940 Physical address: 46572 Value: 0
Virtual address: 9929 Physical address: 44745 Value: 0
Virtual address: 45563 Physical address: 46075 Value: 126
Virtual address: 12107 Physical address: 2635 Value: -46
Number of Translated Addresses = 1000
Page Faults = 244
Page Fault Rate = 0.2440
TLB Hits = 55
TLB Hit Rate = 0.0550
```

9.2 任务二：实现页表替换

```
Virtual address: 49847 Physical address: 4791 Value: -83
Virtual address: 30032 Physical address: 4944 Value: 0
Virtual address: 48065 Physical address: 18113 Value: 0
Virtual address: 6957 Physical address: 27693 Value: 0
Virtual address: 2301 Physical address: 21245 Value: 0
Virtual address: 7736 Physical address: 13112 Value: 0
Virtual address: 31260 Physical address: 5148 Value: 0
Virtual address: 17071 Physical address: 5551 Value: -85
Virtual address: 8940 Physical address: 5868 Value: 0
Virtual address: 9929 Physical address: 6089 Value: 0
Virtual address: 45563 Physical address: 6395 Value: 126
Virtual address: 12107 Physical address: 6475 Value: -46
Number of Translated Addresses = 1000
Page Faults = 538
Page Fault Rate = 0.5380
TLB Hits = 55
TLB Hit Rate = 0.0550
```

读取的 value 值经对比均与答案一致