

# 操作系统课程设计说明文档

## Project04

吕林

[lvlin3896@qq.com](mailto:lvlin3896@qq.com)

前言：本次课程设计修改的 Minix 版本为 3.1.8。本次课程设计包含 6 个小问题，第  $i$  个小问题是第  $i+1$  个问题的子问题，难度依次递增，总难度比以往都有所下降。要完成后面的问题，需要阅读更多的 Minix 源代码，并做更多的尝试。如果你能够完成所有的问题，我们会给予加分奖励。每个问题的测试程序在附件中可以找到，也可以在我的 public（用户名 lvlin，密码 public）上下载，最后的检查就使用这些测试程序，同学们在编写完成后可以下载测试程序完成自测。在最终检查前，同学们需要提交自己的代码和报告，并在约定的时间带着你的机器来找我演示，并向我讲解你的实现思路。检查时间为 16-17 周，地点为电信群楼 3-321 室。请大家认真阅读报告，并独立完成作业，能做多少算多少。严禁抄袭，**抄袭者和被抄袭者都将面临最严厉的处罚。**

### 一、为 Minix 增加新的系统调用

增加新的系统调用，接口为：

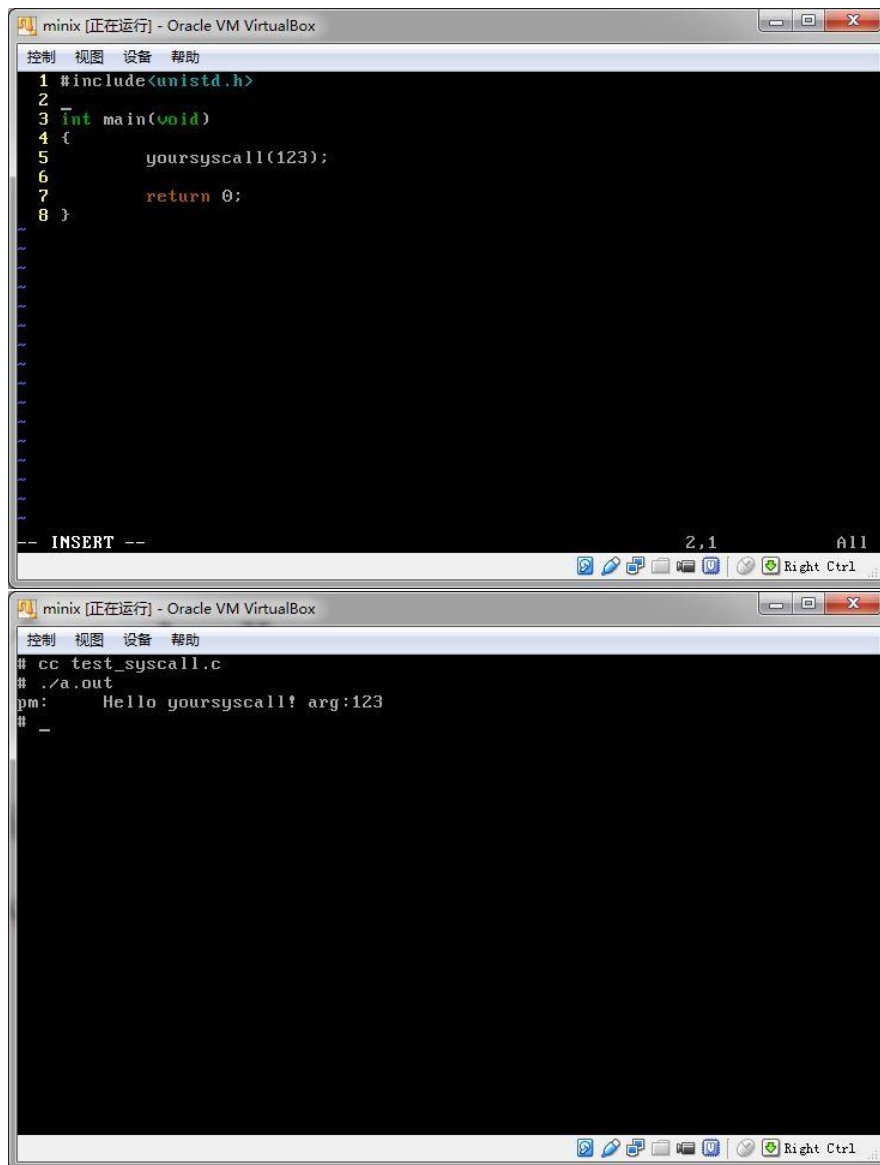
```
void yoursyscall(void);
```

新的系统调用由 pm 服务器来处理，pm 在收到请求后输出一句话即可。

具体的增加过程和注意事项请参阅附件中的文档。

运行示例：





The image shows two screenshots of a Minix VM window titled "minix [正在运行] - Oracle VM VirtualBox".

The top screenshot shows a C program being edited in a text editor. The code is as follows:

```
1 #include<unistd.h>
2
3 int main(void)
4 {
5     yoursyscall(123);
6
7     return 0;
8 }
```

The bottom screenshot shows the same window after compilation and execution. The terminal output is:

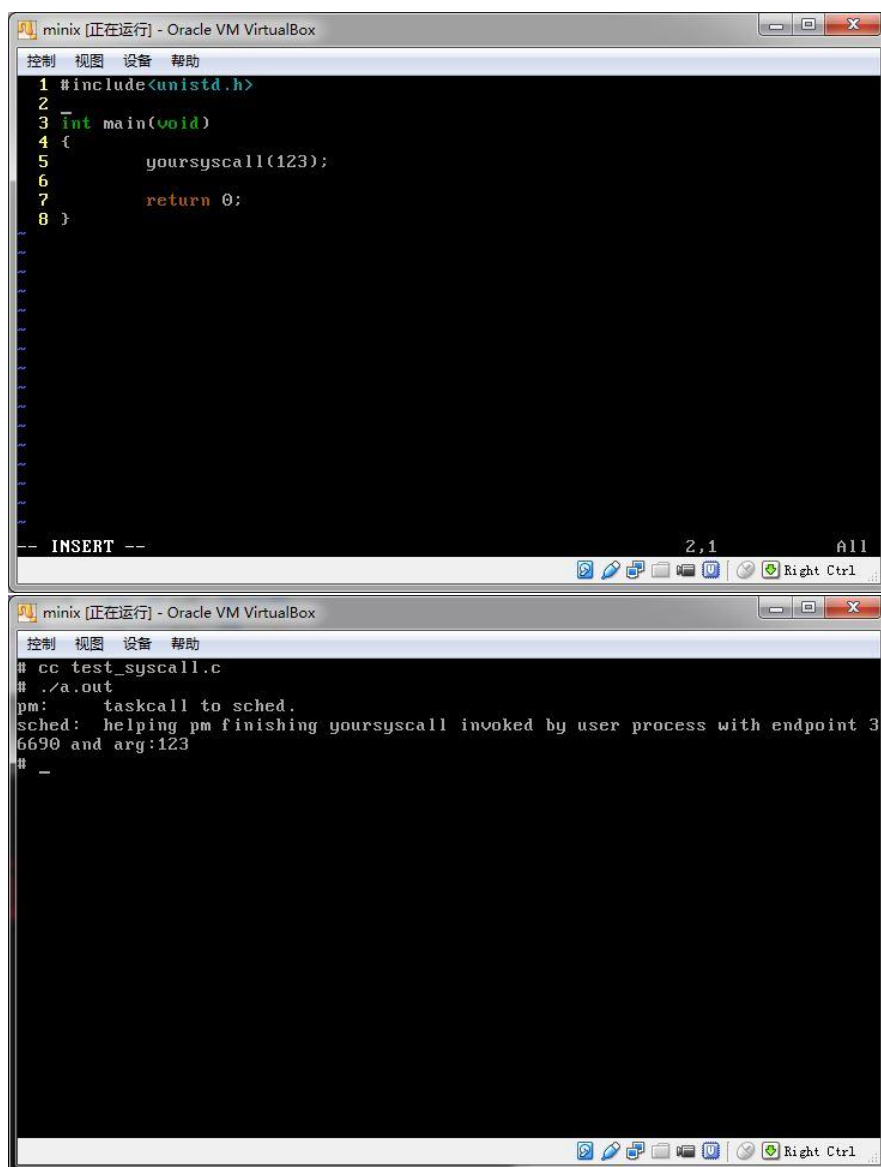
```
# cc test_syscall.c
# ./a.out
pm: Hello yoursyscall! arg:123
#
```

### 三、将 pm 的 printf 改为一个\_taskcall

在上一个的基础上，假设 pm 无法单独完成你的 yoursyscall，他在收到请求获得参数后，对 yoursyscall 的处理方式就是调用\_taskcall 函数，将请求发给 sched 服务器，让 sched 服务器完成你的调用。发给 sched 服务器的信息中应该包含是哪个用户进程发来的请求以及 arg 值是多少。

关于调用\_taskcall 函数向其他服务器（sched 服务器）发送请求，请参阅 nice 系统调用的具体实现，请仔细分析 nice 系统调用在 pm 端与 sched 服务器的交互方式，以及 sched 对 pm 的请求的处理方式。

运行示例：



The image shows two screenshots of a minix VM terminal window. The top screenshot displays a C program named `test_syscall.c` with the following code:

```
1 #include<unistd.h>
2
3 int main(void)
4 {
5     yoursyscall(123);
6
7     return 0;
8 }
```

The bottom screenshot shows the output of compiling and running the program:

```
# cc test_syscall.c
# ./a.out
pm: taskcall to sched.
sched: helping pm finishing yoursyscall invoked by user process with endpoint 3
6690 and arg:123
#
```

## 四、修改 sched 服务器

做到这一步的同学一定会发现，前三步基本上都是照猫画虎，没有什么特殊的地方需要更改。在第四步里面我们要做一些实质性的改动来实现特定的功能。

我先以一个例子来解释我们第四步要干什么事情。

如果我写这样一个程序 `one.c`（以下的程序在附件中能够找到）：





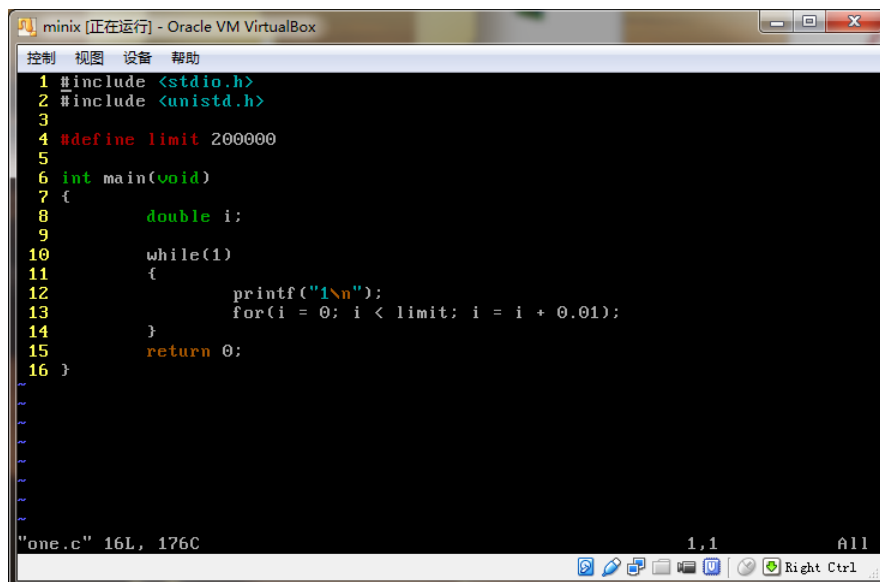
而这个服务器是负责 Minix 调度策略的服务器。Minix 调度机制的具体实现代码在内核中，内核中的代码一行也不允许修改。

其次是，所有进程中，优先执行调用了 yoursyscall 的进程，如果两个或多个进程都调用了 yoursyscall，那么它们都要得到执行，并且参数大的获得的执行机会更多。

第三点是，在优先执行调用 yoursyscall 进程的时候，请不要阻碍系统进程的运行，也不要阻碍捕捉键盘进程的运行。

最后一点是，为了完成这个任务，请熟读 sched 及内核中关于调度部分的代码。搞清楚究竟进程调度是如何实现的。

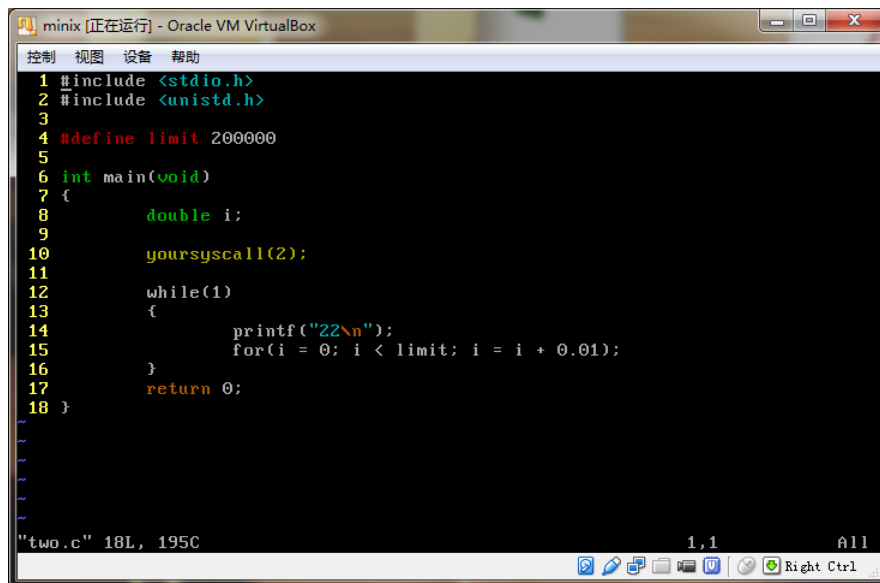
修改后的测试程序及运行示例（我会拿这个程序去检测大家的作业，所以务必自己测一下），首先是 one.c，它不调用 yoursyscall，与原来的测试程序相同：



```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 #define limit 200000
5
6 int main(void)
7 {
8     double i;
9
10    while(1)
11    {
12        printf("1\n");
13        for(i = 0; i < limit; i = i + 0.01);
14    }
15    return 0;
16 }
```

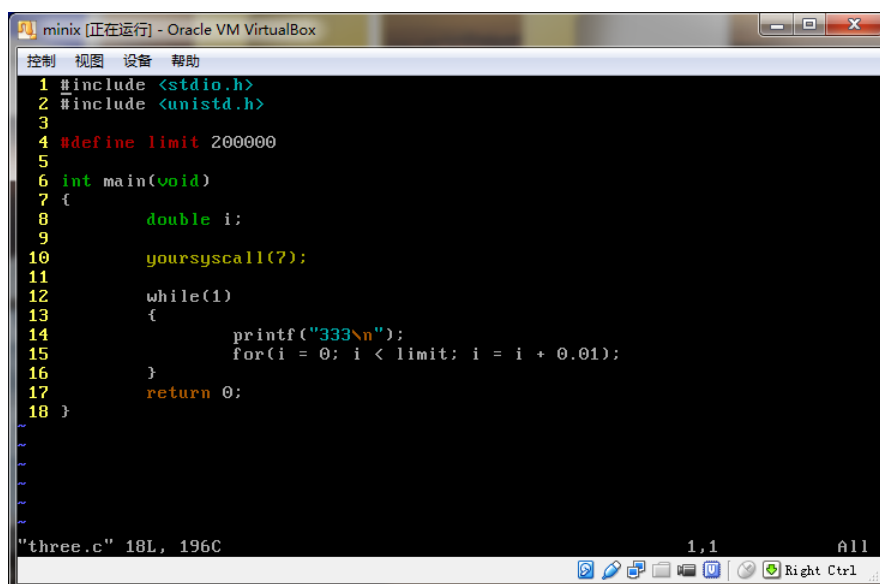
"one.c" 16L, 176C 1,1 All Right Ctrl

two.c 调用 yoursyscall(2):



```
minix [正在运行] - Oracle VM VirtualBox
控制 视图 设备 帮助
1 #include <stdio.h>
2 #include <unistd.h>
3
4 #define limit 2000000
5
6 int main(void)
7 {
8     double i;
9
10    yoursyscall(2);
11
12    while(1)
13    {
14        printf("22\n");
15        for(i = 0; i < limit; i = i + 0.01);
16    }
17    return 0;
18 }
"two.c" 18L, 195C
1,1 All
Right Ctrl
```

three.c 调用 yoursyscall(7):



```
minix [正在运行] - Oracle VM VirtualBox
控制 视图 设备 帮助
1 #include <stdio.h>
2 #include <unistd.h>
3
4 #define limit 2000000
5
6 int main(void)
7 {
8     double i;
9
10    yoursyscall(7);
11
12    while(1)
13    {
14        printf("333\n");
15        for(i = 0; i < limit; i = i + 0.01);
16    }
17    return 0;
18 }
"three.c" 18L, 196C
1,1 All
Right Ctrl
```

do.sh 与之前保持一致:

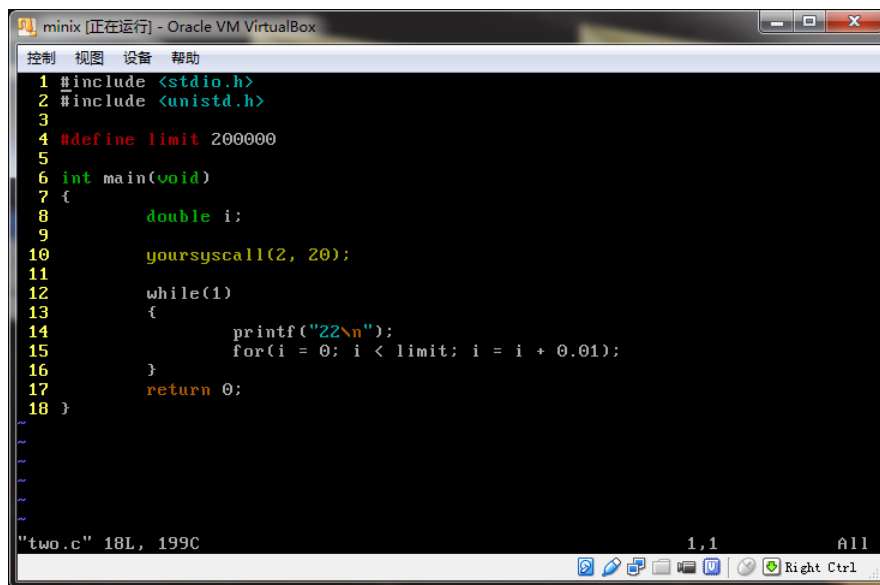








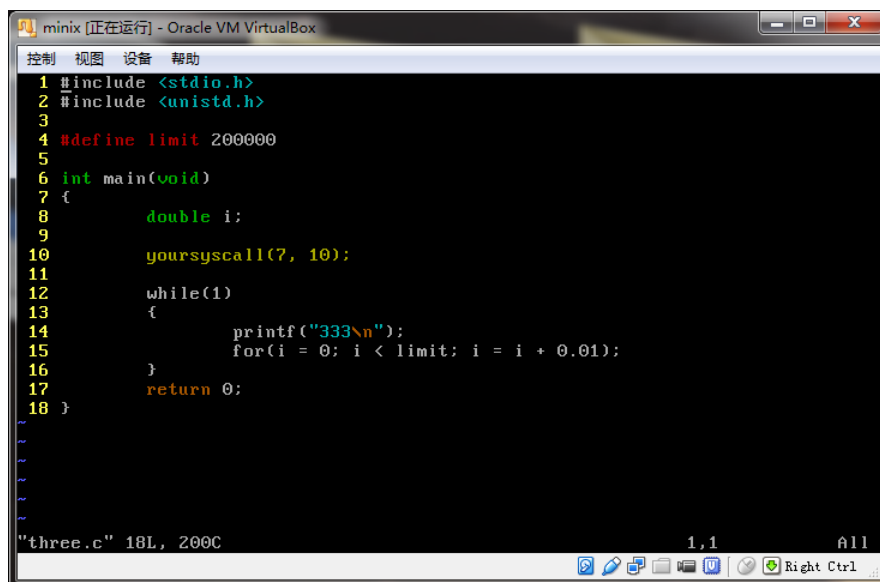
two.c 调用 yoursyscall(2, 20), 调用后 20 秒被强行终止:



```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 #define limit 200000
5
6 int main(void)
7 {
8     double i;
9
10    yoursyscall(2, 20);
11
12    while(1)
13    {
14        printf("22\n");
15        for(i = 0; i < limit; i = i + 0.01);
16    }
17    return 0;
18 }
```

"two.c" 18L, 199C 1,1 All

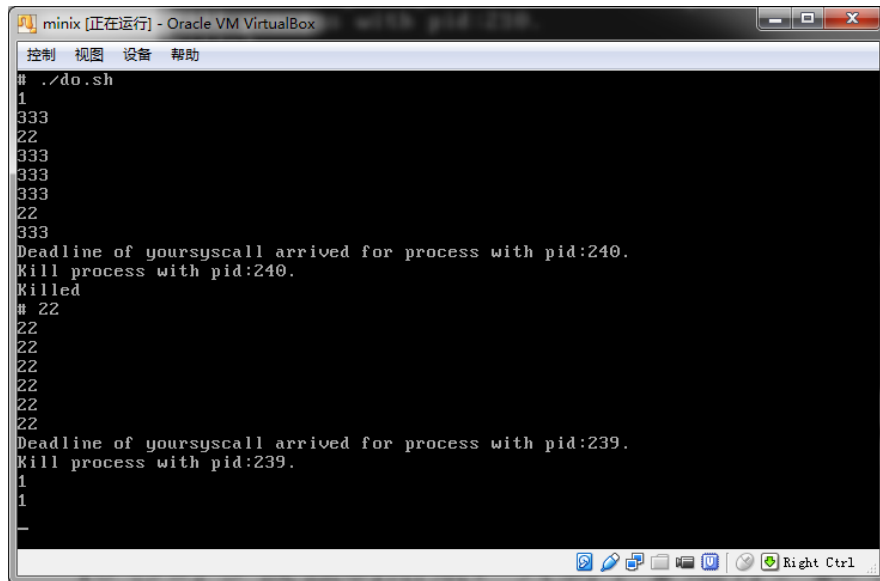
three.c 调用 yoursyscall(7, 10), 调用后 10 秒被强行终止:



```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 #define limit 200000
5
6 int main(void)
7 {
8     double i;
9
10    yoursyscall(7, 10);
11
12    while(1)
13    {
14        printf("333\n");
15        for(i = 0; i < limit; i = i + 0.01);
16    }
17    return 0;
18 }
```

"three.c" 18L, 200C 1,1 All

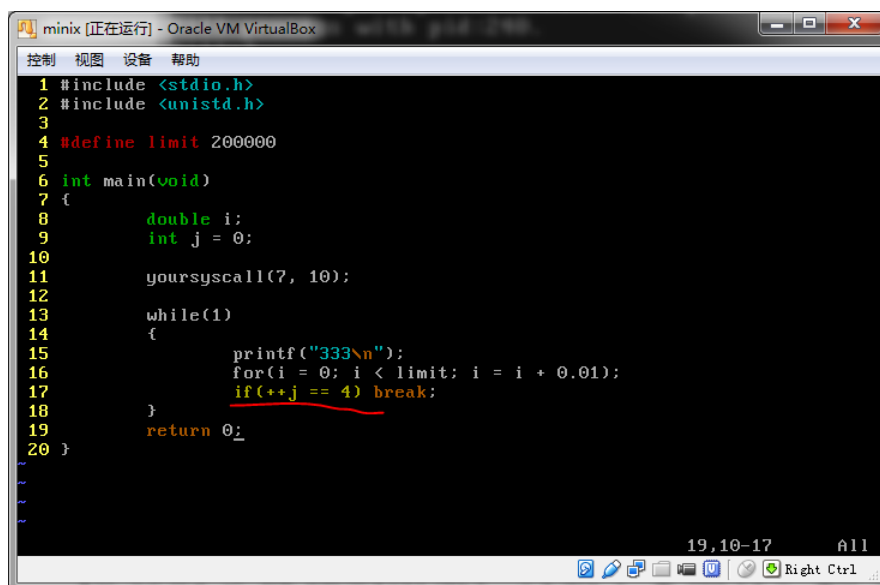
运行 do.sh 得到如下结果:



```
# ./do.sh
1
333
22
333
333
333
22
333
Deadline of yoursyscall arrived for process with pid:240.
Kill process with pid:240.
Killed
# 22
22
22
22
22
22
22
Deadline of yoursyscall arrived for process with pid:239.
Kill process with pid:239.
1
1
1
```

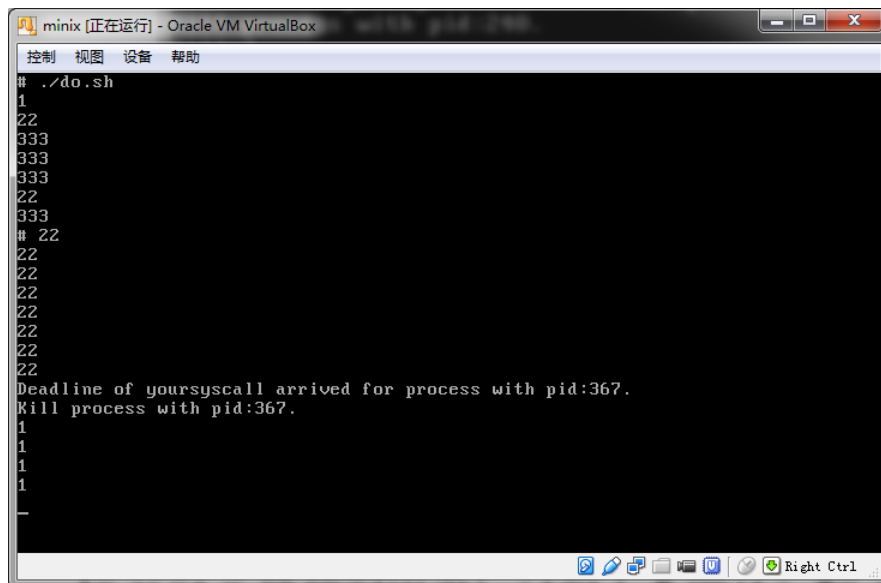
整个运行过程中前 10 秒与不加 deadline 完全一致，只有 2、3 有机会运行且 3 得到更多的运行机会。10 秒时 3 被强制结束，只剩下 2 运行。20 秒时 2 被强制结束，运行机会回到 1 手里。

下一个测试我们测试如果在 deadline 来临之前程序就已经结束了会有怎样的反应，我们修改 three.c:



```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 #define limit 200000
5
6 int main(void)
7 {
8     double i;
9     int j = 0;
10
11     yoursyscall(7, 10);
12
13     while(1)
14     {
15         printf("333\n");
16         for(i = 0; i < limit; i = i + 0.01);
17         if(++j == 4) break;
18     }
19     return 0;
20 }
```

让他输出 4 次就结束，我们让 4 次输出能够在 10 秒内完成。运行 do.sh 得到如下的结果：



```
# ./do.sh
1
22
333
333
333
22
333
# 22
22
22
22
22
22
22
Deadline of yoursyscall arrived for process with pid:367.
Kill process with pid:367.
1
1
1
1
1
```

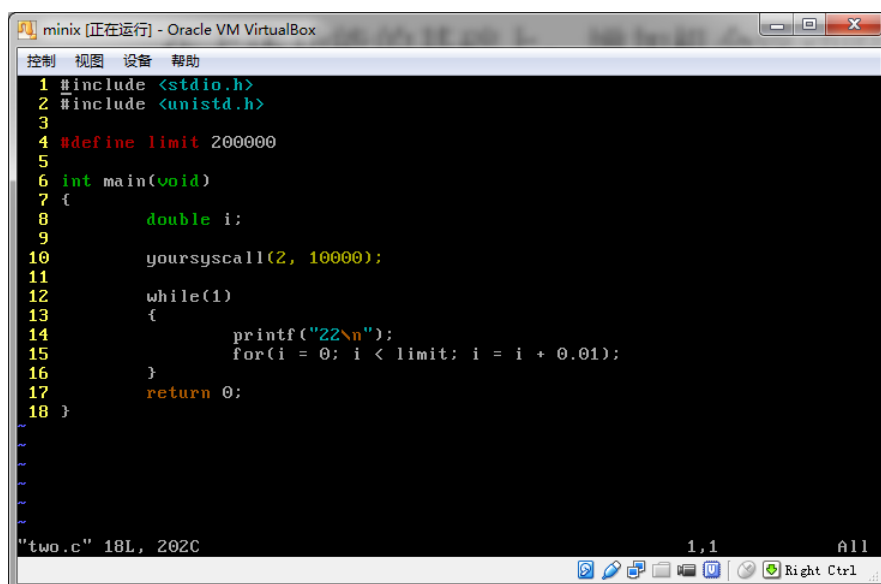
我们发现 3 比 deadline 结束的更早不会引发系统的错误，我们的程序是正确的。

## 六、增加机会浮动功能

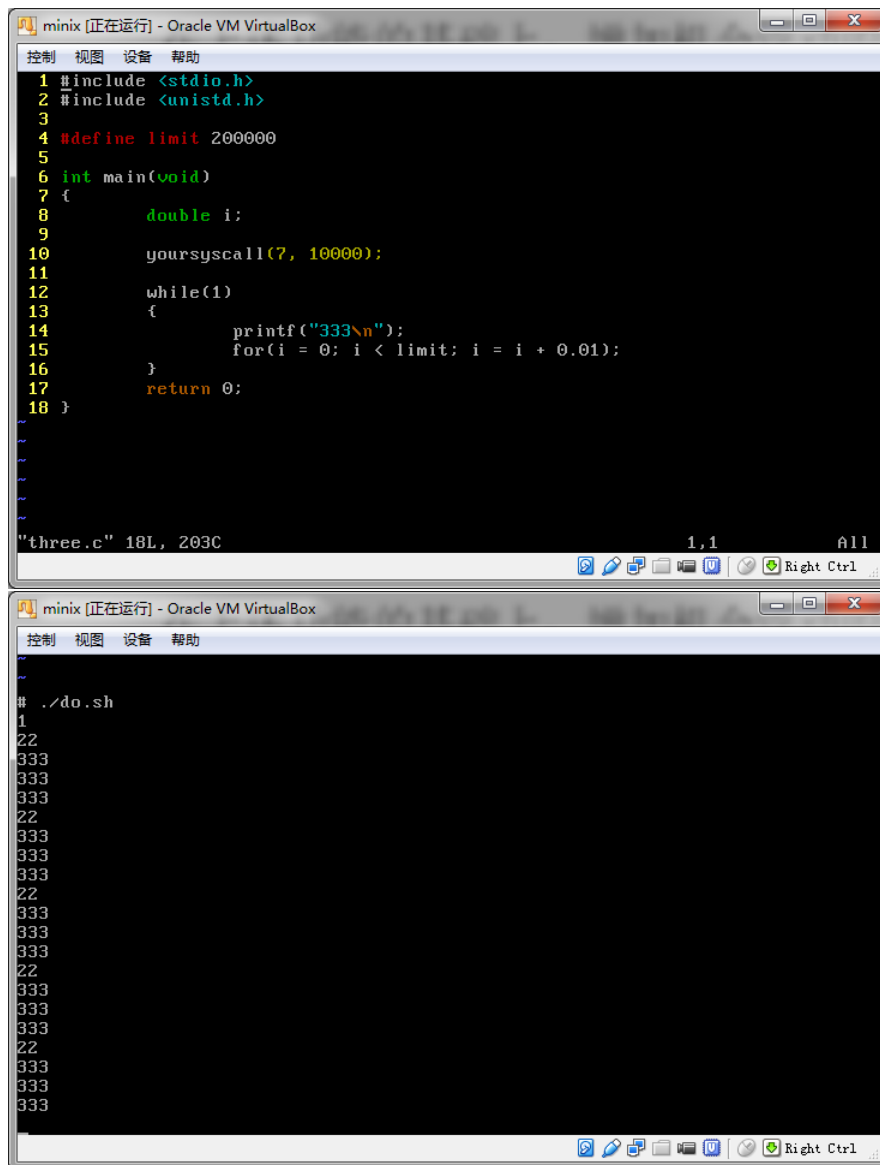
在上述功能的基础上，增加机会浮动的功能，即程序越接近 deadline，它会获得更多的运行机会。

运行示例：

首先我们将 two.c 和 three.c 的 deadline 设置的很大，让 deadline 对运行机会的影响大大减弱，那么得到的应该是第四步的结果：



```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 #define limit 200000
5
6 int main(void)
7 {
8     double i;
9
10    yoursyscall(2, 10000);
11
12    while(1)
13    {
14        printf("22\n");
15        for(i = 0; i < limit; i = i + 0.01);
16    }
17    return 0;
18 }
```



```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 #define limit 200000
5
6 int main(void)
7 {
8     double i;
9
10    yoursyscall(7, 10000);
11
12    while(1)
13    {
14        printf("333\n");
15        for(i = 0; i < limit; i = i + 0.01);
16    }
17    return 0;
18 }
```

```
# ./do.sh
1
22
333
333
333
22
333
333
333
333
22
333
333
333
22
333
333
333
22
333
333
333
```

如果我们将 3 的 arg 也改成 2，那么在长 deadline 情况下，2、3 的输出应该是等个数的：





