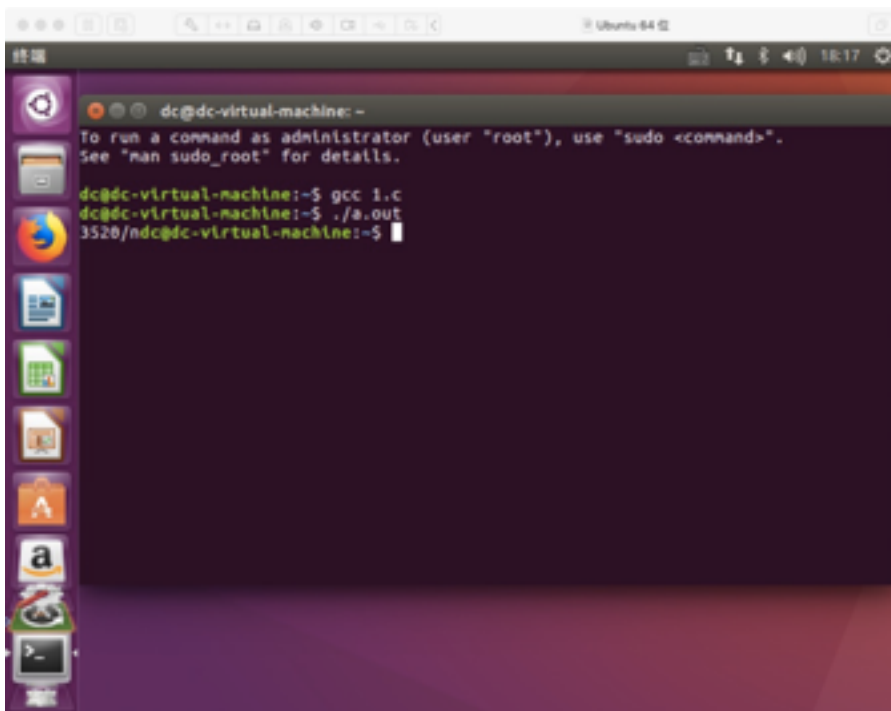


一、（系统调用实验）了解系统调用不同的封装形式。

要求：1、参考下列网址中的程序。阅读分别运行用API接口函数getpid()直接调用和汇编中断调用两种方式调用Linux操作系统的同一个系统调用getpid的程序(请问getpid的系统调用号是多少？linux系统调用的中断向量号是多少？)。2、上机完成习题1.13。3、阅读pintos操作系统源代码，画出系统调用实现的流程图。

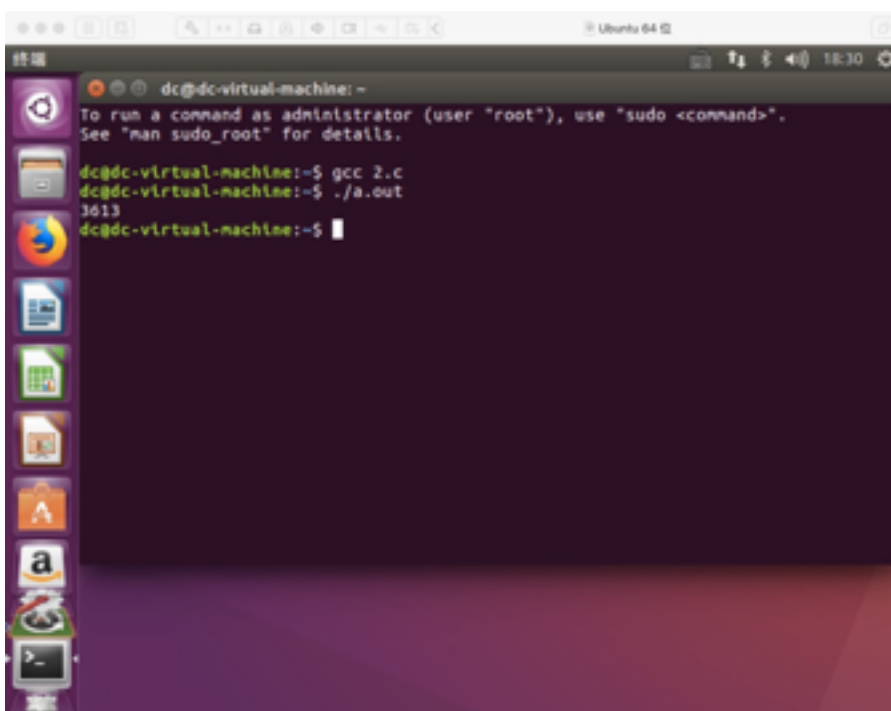
1.

程序一运行结果：



```
dc@dc-virtual-machine: ~  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
dc@dc-virtual-machine:~$ gcc 1.c  
dc@dc-virtual-machine:~$ ./a.out  
3520/ndc@dc-virtual-machine:~$
```

程序二运行结果：



```
dc@dc-virtual-machine: ~  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
dc@dc-virtual-machine:~$ gcc 2.c  
dc@dc-virtual-machine:~$ ./a.out  
3613  
dc@dc-virtual-machine:~$
```

getpid系统调用号：39

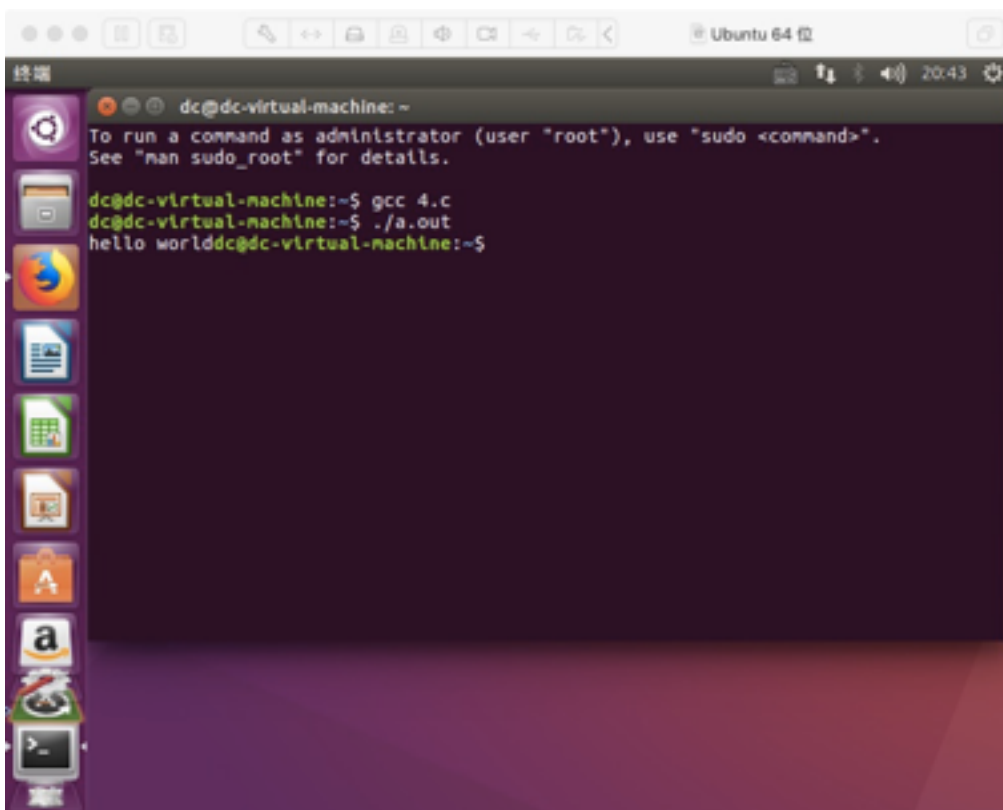
linux系统调用的中断向量号80

2.

c语言代码：

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    pid_t pid;
    pid = getpid();
    printf("%d\n",pid)
}
```

运行代码：

A screenshot of a terminal window titled "Ubuntu 64 位" with a standard Ubuntu desktop background. The terminal shows the following commands and output:

```
dc@dc-virtual-machine: ~
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

dc@dc-virtual-machine:~$ gcc 4.c
dc@dc-virtual-machine:~$ ./a.out
hello world
dc@dc-virtual-machine:~$
```

汇编代码如下：

```
.section .data
message:
    .ascii "hello world!\n"
    length = . - message

.section .text
```

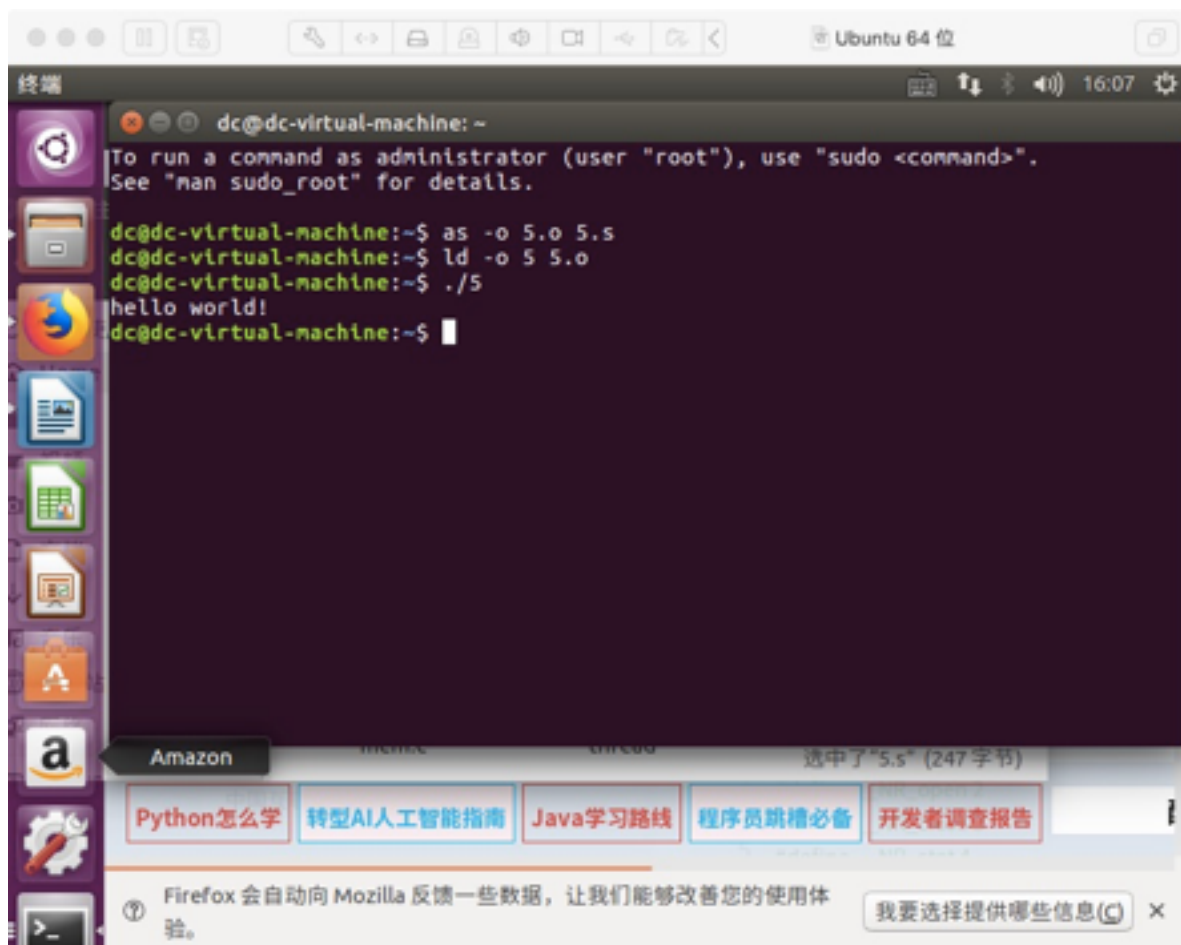
```

.global _start    # must be declared for linker
_start:
    movq $1, %rax    # 'write' syscall number
    movq $1, %rdi    # file descriptor, stdout
    lea message(%rip), %rsi # relative addressing string message
    movq $length, %rdx
    syscall

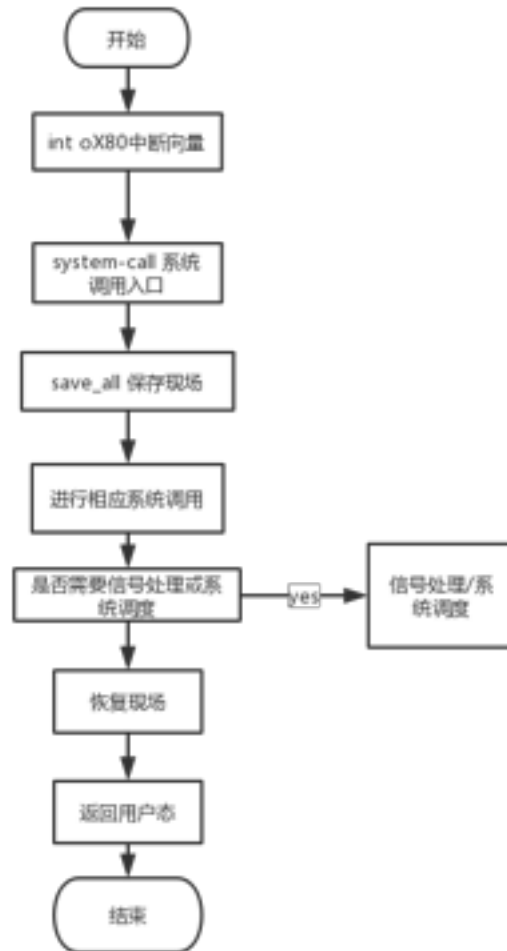
    movq $60, %rax    # 'exit' syscall number
    xor %rdi, %rdi    # set rdi to zero
    syscall

```

运行.s 文件



### 3.流程图



二、（并发实验）根据以下代码完成下面的实验。

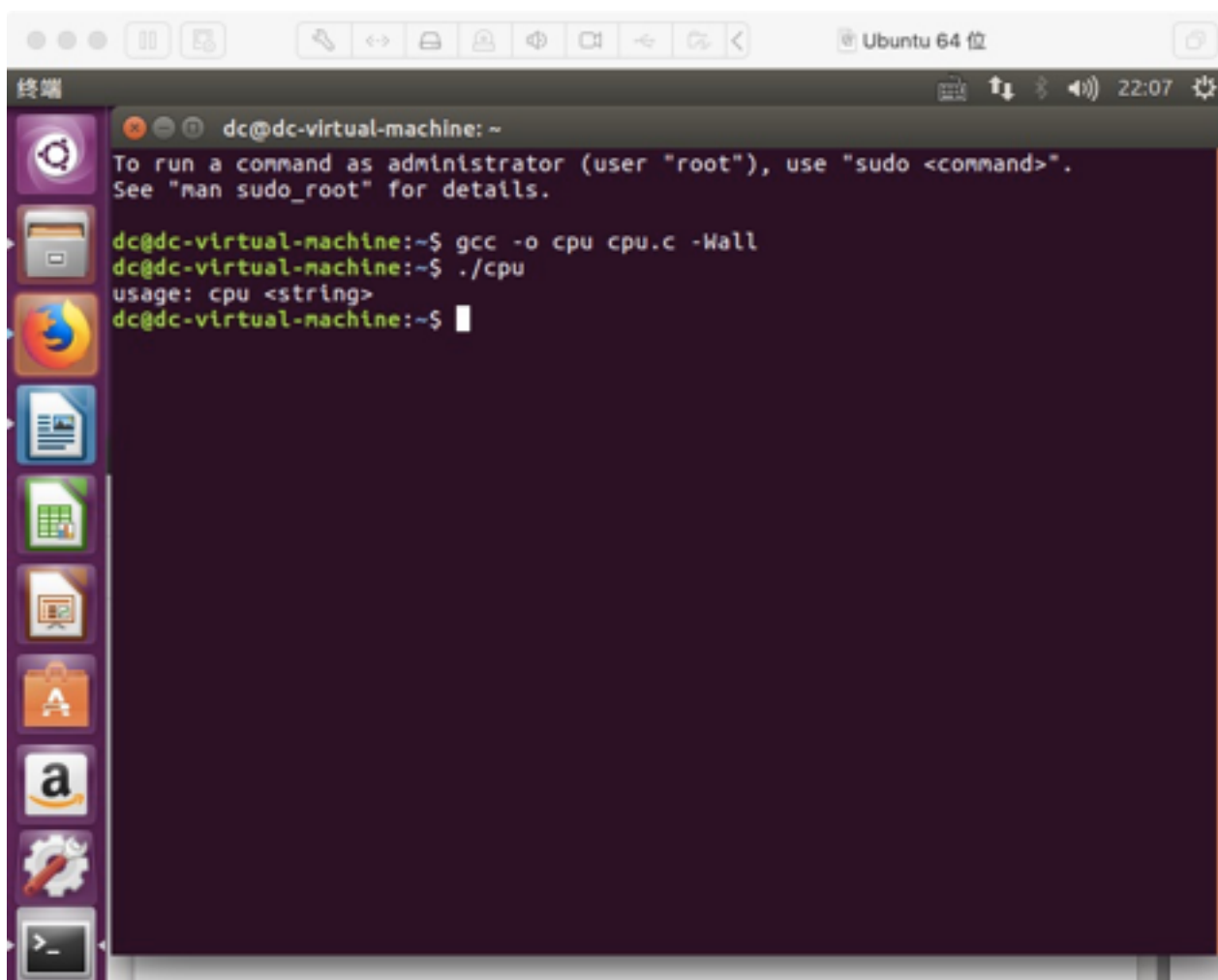
要求：

1. 编译运行该程序（cpu.c），观察输出结果，说明程序功能。  
(编译命令： gcc -o cpu cpu.c -Wall) (执行命令： ./cpu)
- 2、再次按下面的运行并观察结果：执行命令： ./cpu A & ; ./cpu B & ; ./cpu C & ; ./cpu D &程序cpu运行了几次？他们运行的顺序有何特点和规律？请结合操作系统的特征进行解释。

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/time.h>
4 #include <assert.h>
5 #include "common.h"
6
7 int
8 main(int argc, char *argv[])
9 {
```

```
10 if (argc != 2) {
11     fprintf(stderr, "usage: cpu <string>\n");
12     exit(1);
13 }
14 char *str = argv[1];
15 while (1) {
16     spin(1);
17     printf("%s\n", str);
18 }
19. return 0;
20.
```

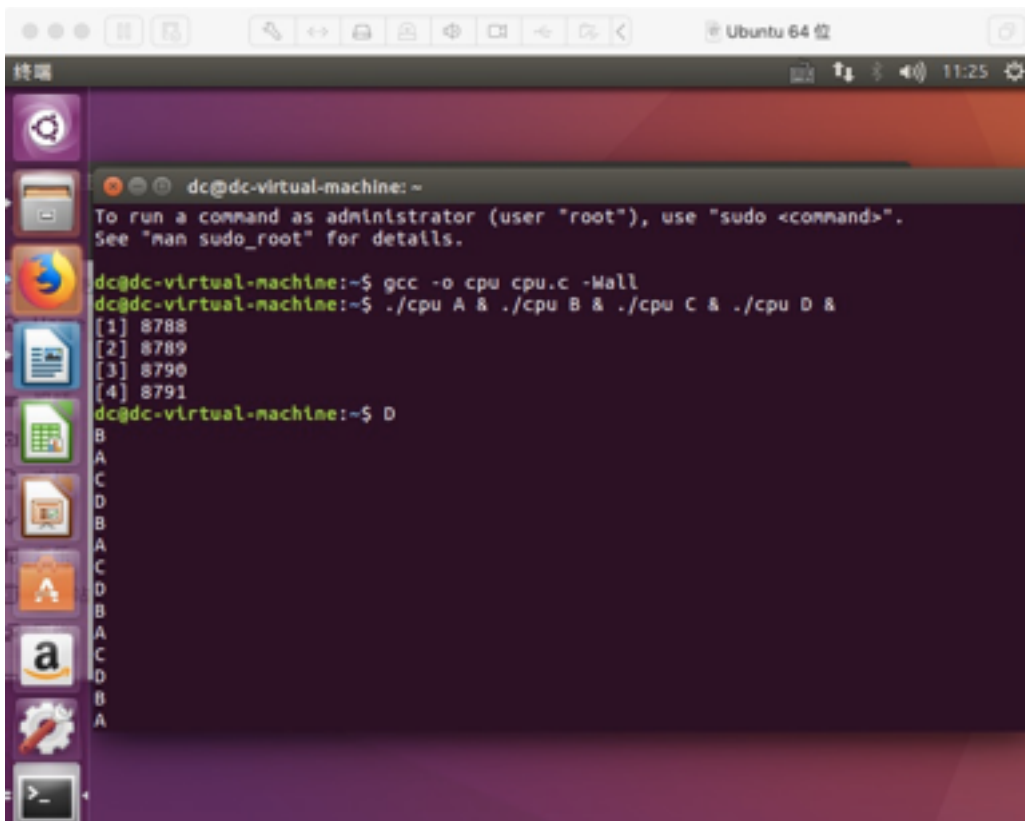
1.运行程序结果：



The screenshot shows a terminal window titled 'Ubuntu 64 位' with a dark background. The prompt is 'dc@dc-virtual-machine: ~'. A message at the top says: 'To run a command as administrator (user "root"), use "sudo <command>". See "man sudo\_root" for details.' The user enters the command 'gcc -o cpu cpu.c -Wall', followed by './cpu'. The output is 'usage: cpu <string>' and the prompt returns to 'dc@dc-virtual-machine:~\$'.

该程序实现的是，调Spin()一个反复检查时间并一旦运行一秒钟就返回的函数Spin()。然后，它打印出用户在命令行上输入的字符串，并一直重复。如果输入的参数不为两个，便输出usage: cpu <string>。

2.按下面的运行执行命令：./cpu A & ; ./cpu B & ; ./cpu C & ; ./cpu D



我认为cpu一共运行了四次，这个程序反反复复检查时间，直到一秒钟过去。第二次过去后，代码会用户传入的输入字符串，然后继续。程序会永远运行。虽然看起来cpu进行了多次的运行，但是操作系统在硬件的帮助下实现了虚拟化cpu，所以系统看起来具有大量虚拟CPU。把单个CPU转换为看起来无限数量的CPU，从而让许多程序看起来一次运行。

三、（内存分配实验）根据以下代码完成实验。

要求：

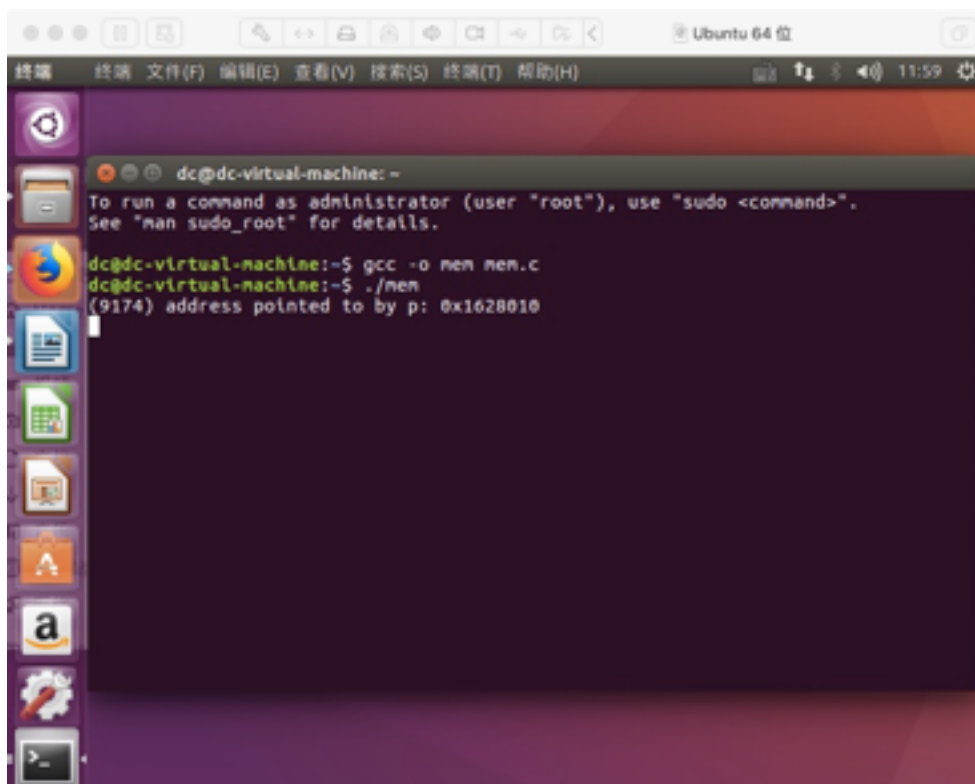
1. 阅读并编译运行该程序(mem.c)，观察输出结果，说明程序功能。(命令： gcc -o mem mem.c -Wall)

2、再次按下面的命令运行并观察结果。两个分别运行的程序分配的内存地址是否相同？是否共享同一块物理内存区域？为什么？命令：`./mem & ./mem &` 进程码 指针

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include "common.h"
5
6 int
7 main(int argc, char *argv[])
8 {
9     int *p = malloc(sizeof(int)); // a1
10    assert(p != NULL);
11    printf("(%d) address pointed to by p: %p\n",
12    getpid(), p); // a2
13    *p = 0; // a3
```

```
14 while (1) {  
15 Spin(1);  
16 *p = *p + 1;  
17 printf("(%d) p: %d\n", getpid(), *p); // a4  
18 }  
19 return 0;
```

## 1.运行程序结果



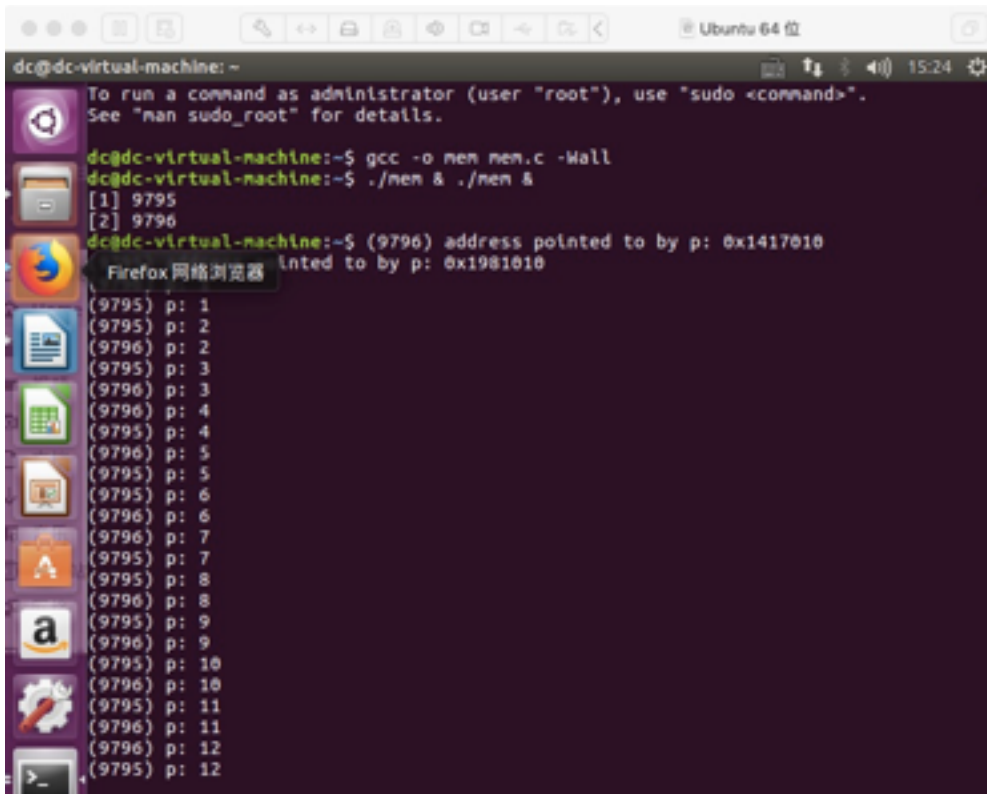
The screenshot shows a terminal window titled "Ubuntu 64 位" with a menu bar containing "终端", "文件(F)", "编辑(E)", "查看(V)", "搜索(S)", "终端(T)", and "帮助(H)". The terminal prompt is "dc@dc-virtual-machine: ~". A message states: "To run a command as administrator (user "root"), use "sudo <command>". See "man sudo\_root" for details." The user enters the command "gcc -o mem mem.c", followed by ". /mem". The output is "(9174) address pointed to by p: 0x1628010". The terminal has a dark purple background and a sidebar on the left with various application icons.

分析运行结果：

程序的功能为：首先，分配一些内存。然后，它印出内存地址，然后将数字0放入新分配的内存的第一个位置。最后循环：延迟一秒并递增存储在p中保存的地址的值。对于每个print语句，它还会打印出正在运行的程序的进程标识符（PID）。该PID在每个运行过程中都是唯一的。

新分配的内存位于地址0x1628010。程序运行时，会慢慢更新值并打印出结果。

## 2.运行结果截图



```
dc@dc-virtual-machine: ~
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

dc@dc-virtual-machine:~$ gcc -o men men.c -Wall
dc@dc-virtual-machine:~$ ./men & ./men &
[1] 9795
[2] 9796
dc@dc-virtual-machine:~$ (9796) address pointed to by p: 0x1417010
Firefox 网络浏览器
(9795) p: 1
(9795) p: 2
(9796) p: 2
(9795) p: 3
(9796) p: 3
(9796) p: 4
(9795) p: 4
(9796) p: 5
(9795) p: 5
(9795) p: 6
(9796) p: 6
(9796) p: 7
(9795) p: 7
(9795) p: 8
(9796) p: 8
(9795) p: 9
(9796) p: 9
(9795) p: 10
(9796) p: 10
(9795) p: 11
(9796) p: 11
(9796) p: 12
(9795) p: 12
```

第一个运行的程序分配到的内存地址为0x417010，第二个运行的程序分配到的内存地址为0x1981010。正在运行的程序都有自己的私有内存，而不是与其他正在运行的程序共享相同的物理内存。因为操作系统虚拟化了内存。每个进程访问自己的私有虚拟地址空间，操作系统以某种方式映射到机器的物理内存。一个正在运行的程序中的内存引用不会影响其他进程的地址空间；而对于运行程序而言，它拥有所有的物理内存。然而，现实是物理内存是由操作系统管理的共享资源。

## 四、（共享的问题）根据以下代码完成实验。

要求：

1. 阅读并编译运行该程序，观察输出结果，说明程序功能。（编译命令：gcc -o thread thread.c -Wall -pthread）（执行命令1：./thread 1000）
2. 尝试其他输入参数并执行，并总结执行结果的有何规律？你能尝试解释它吗？（例如执行命令2：./thread 100000）（或者其他参数。）
3. 提示：哪些变量是各个线程共享的，线程并发执行时访问共享变量会不会导致意想不到的问题。

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "common.h"
4
5 volatile int counter = 0;
6 int loops;
7
8 void *worker(void *arg) {
9     int i;
10    for (i = 0; i < loops; i++) {
11        counter++;
12    }
```

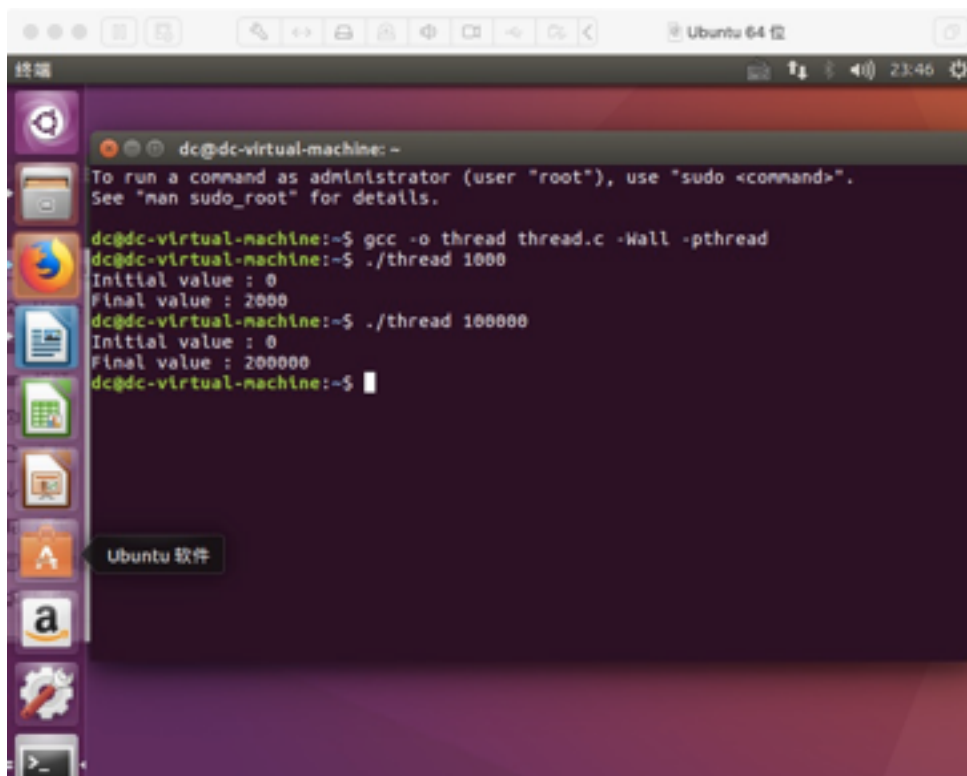


```

13 return NULL;
14 }
15
16 int
17 main(int argc, char *argv[])
18 {
19     if (argc != 2) {
20         fprintf(stderr, "usage: threads <value>\n");
21         exit(1);
22     }
23     loops = atoi(argv[1]);
24     pthread_t p1, p2;
25     printf("Initial value : %d\n", counter);
26
27     Pthread_create(&p1, NULL, worker, NULL);
28     Pthread_create(&p2, NULL, worker, NULL);
29     Pthread_join(p1, NULL);
30     Pthread_join(p2, NULL);
31     printf("Final value : %d\n", counter);
32     return 0;
33 }

```

1.运行结果截图：



```

dc@dc-virtual-machine: ~
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

dc@dc-virtual-machine:~$ gcc -o thread thread.c -Wall -pthread
dc@dc-virtual-machine:~$ ./thread 1000
Initial value : 0
Final value : 2000
dc@dc-virtual-machine:~$ ./thread 100000
Initial value : 0
Final value : 200000
dc@dc-virtual-machine:~$

```

主程序使用Pthread .create()创建两个线程。每个线程开始在一个名为worker()的例程中运行，是循环递增一个统计循环次数的计数器。循环的值确定两个工作程序中的每一个将在循环中递增共享计数器的次

数。两个线程完成时，计数器的最终值为2000，因为每个线程将计数器递增1000次。实际上，当循环的输入值设置为N时，程序的最终输出应该为2N