

北京交通大学

《操作系统》实验报告

学 号： 16281015

姓 名： 王子谦

专 业： 计算机科学与技术

学 院： 计算机与信息技术学院

提交日期： 2019 年 3 月 10 日

《操作系统》实验一

实验题目：

一、（系统调用实验）了解系统调用不同的封装形式。

实验要求

- 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。

实验思路：

1. 本题主要是了解调用 API 方式和汇编中断实现方式。在汇编中断中，通过 EAX 存放所需功能的系统调用号，int 80 实现系统调用。因此 `getpid` 的系统调用号为 0x14, 即为 20. Linux 的系统调用中断向量号为 0x80. 即为 128

源代码：实验指导已给出。

实验结果展示：



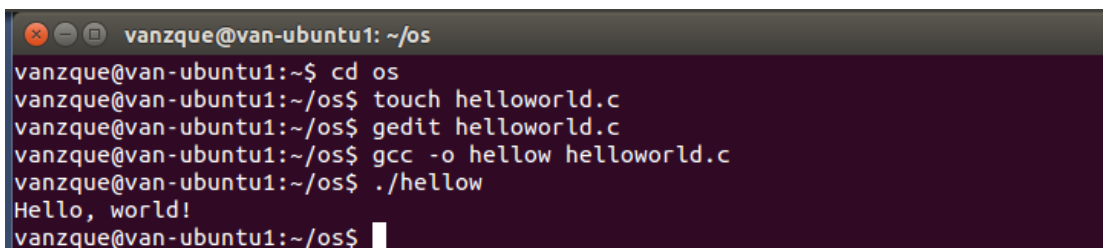
```
vanzque@van-ubuntu1: ~/os
vanzque@van-ubuntu1:~/os$ ls
asm      cpu      cpu.c~  getpid  mem.c   thread.c
asm.c    cpu.c   gepid.c mem     test.c~ thread.c~
vanzque@van-ubuntu1:~/os$ ./getpid  → 调用API
6358
vanzque@van-ubuntu1:~/os$ ./asm      → 系统中断调用
6359
vanzque@van-ubuntu1:~/os$
```

2. 本题需要知道的是 linux 中所涉及到的系统调用 `write` 与 `exit`。系统调用号分别为 4 与 1。

C 语言源代码:

```
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    char *hello = "Hello, world!\n";
    write(1, hello, strlen(hello));
    return 0;
}
```

实验结果展示:


```
vanzque@van-ubuntu1: ~/os
vanzque@van-ubuntu1:~$ cd os
vanzque@van-ubuntu1:~/os$ touch helloworld.c
vanzque@van-ubuntu1:~/os$ gedit helloworld.c
vanzque@van-ubuntu1:~/os$ gcc -o hellow helloworld.c
vanzque@van-ubuntu1:~/os$ ./hellow
Hello, world!
vanzque@van-ubuntu1:~/os$
```

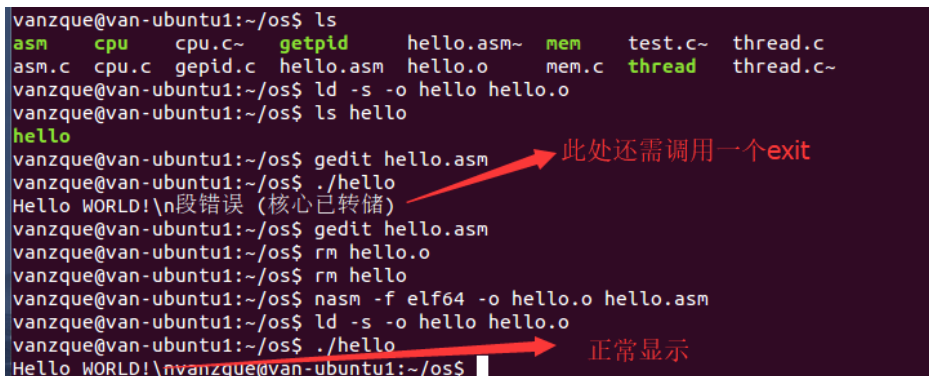
汇编源代码:

```
[section .data]

strHello    db "Hello WORLD!\n"
strLen      equ $-strHello

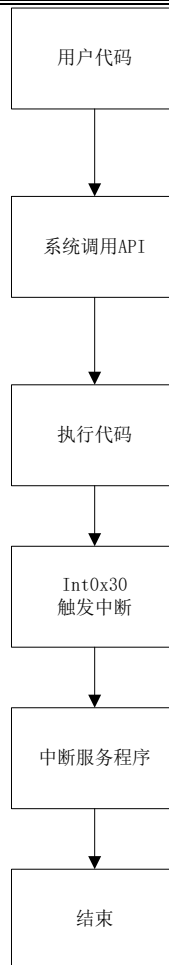
[section .text]
global _start

_start:
    mov edx, strLen
    mov ecx, strHello
    mov ebx, 0
    mov eax, 4 ;sys_write
    int 0x80
    mov ebx, 1
    mov eax, 1 ;sys_ext
    int 0x80
```

实验结果展示:


```
vanzque@van-ubuntu1:~/os$ ls
asm      cpu      cpu.c~  getpid  hello.asm~ mem      test.c~  thread.c
asm.c    cpu.c  gepid.c  hello.asm  hello.o  mem.c    thread  thread.c~
vanzque@van-ubuntu1:~/os$ ld -s -o hello hello.o
vanzque@van-ubuntu1:~/os$ ls hello
hello
vanzque@van-ubuntu1:~/os$ gedit hello.asm
vanzque@van-ubuntu1:~/os$ ./hello
Hello WORLD!\n段错误 (核心已转储)
vanzque@van-ubuntu1:~/os$ gedit hello.asm
vanzque@van-ubuntu1:~/os$ rm hello.o
vanzque@van-ubuntu1:~/os$ rm hello
vanzque@van-ubuntu1:~/os$ nasm -f elf64 -o hello.o hello.asm
vanzque@van-ubuntu1:~/os$ ld -s -o hello hello.o
vanzque@van-ubuntu1:~/os$ ./hello
Hello WORLD!\nvanzque@van-ubuntu1:~/os$
```

实验一



3.

实验题目：

二、（并发实验）根据以下代码完成下面的实验。

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。

实验过程与思路：

1. 该程序功能是间隔时间输出函数参数。如果没有参数输入，报错则结束。

源代码：实验指导已给出。

实验结果展示:

```
vanzque@van-ubuntu1: ~/os
vanzque@van-ubuntu1:~$ cd os
vanzque@van-ubuntu1:~/os$ ./cpu
usage: cpu <string>
```

2. 一次单位时间内 CPU 运行 4 次。以题目要求运行时，则为多个程序并发运行，即为操作系统的并发执行的特性。在宏观上来看是 4 个程序同时运行，在微观上来看是程序之间交替运行的。

实验结果展示:

```
vanzque@van-ubuntu1:~/os$ ./cpu A & ./cpu B & ./cpu C & ./cpu D &
bash: 未预期的符号 ';' 附近有语法错误
vanzque@van-ubuntu1:~/os$ ./cpu A & ./cpu B & ./cpu C & ./cpu D &
[1] 6897
[2] 6898
[3] 6899
[4] 6900
vanzque@van-ubuntu1:~/os$ A
B
C
D
A
B
C
D
A
B
C
D
A
B
C
D
```

实验题目:

(内存分配实验) 根据以下代码完成实验。

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 &

实验过程与思路:

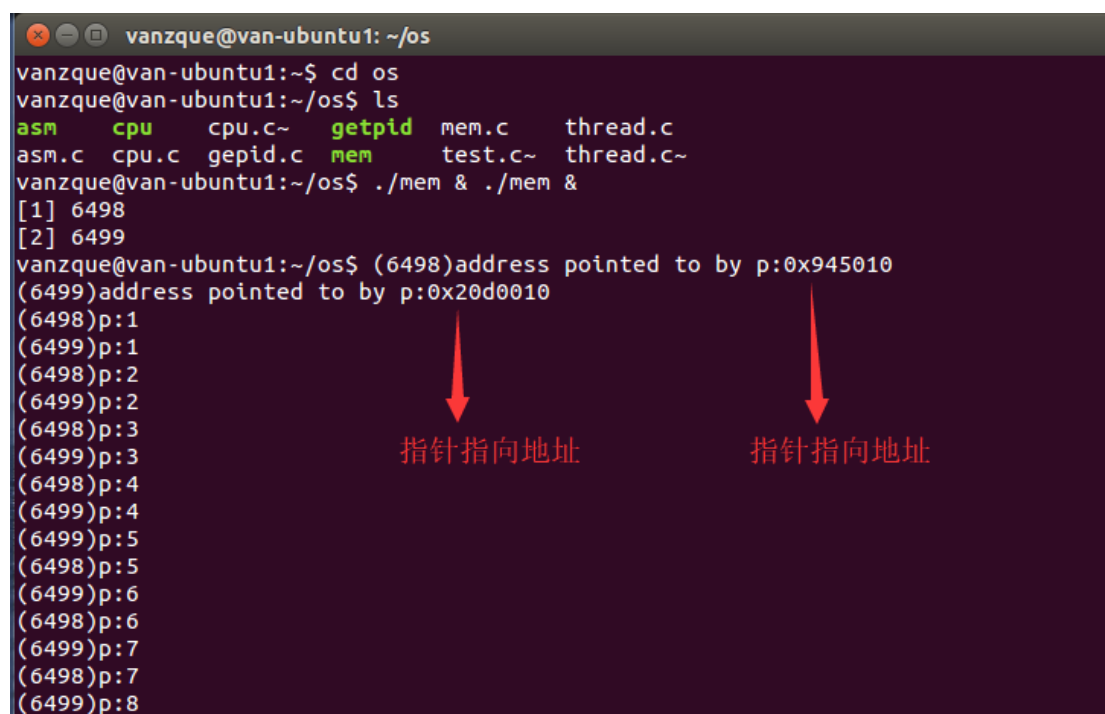
1. **程序功能:** 在该程序中, 给一个指针分配指定空间, 当指针数值为空的时候, 返回指针地址。当数值不为空的时候每隔 1s 时间数值增 1。

源代码：实验指导已给出。

2. 根据实验结果来看，双方的内存地址不同。不在同一物理地址。

根据操作系统的存储管理功能来看，此处体现的是内存保护功能。确保每个用户程序都仅在自己的内存空间内运行，并且互不干扰。

实验结果展示：



```
vanzque@van-ubuntu1: ~/os
vanzque@van-ubuntu1:~$ cd os
vanzque@van-ubuntu1:~/os$ ls
asm      cpu      cpu.c~  getpid  mem.c    thread.c
asm.c    cpu.c    gepid.c mem      test.c~  thread.c~
vanzque@van-ubuntu1:~/os$ ./mem & ./mem &
[1] 6498
[2] 6499
vanzque@van-ubuntu1:~/os$ (6498)address pointed to by p:0x945010
(6499)address pointed to by p:0x20d0010
(6498)p:1
(6499)p:1
(6498)p:2
(6499)p:2
(6498)p:3
(6499)p:3
(6498)p:4
(6499)p:4
(6498)p:5
(6499)p:5
(6498)p:6
(6499)p:6
(6498)p:7
(6499)p:7
(6498)p:8
(6499)p:8
```

实验题目：

（共享的问题）根据以下代码完成实验。

实验要求

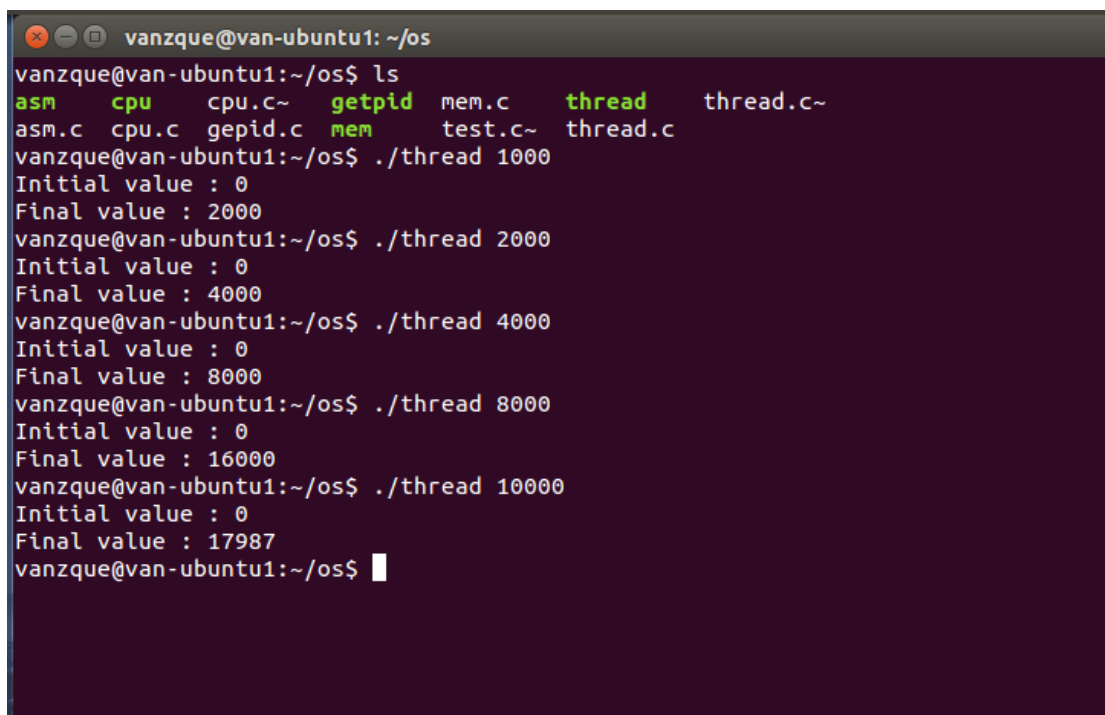
- 1、阅读并编译运行该程序，观察输出结果，说明程序功能。（编译命令：`gcc -o thread thread.c -Wall -pthread`）（执行命令 1：`./thread 1000`）
- 2、尝试其他输入参数并执行，并总结执行结果的有何规律？你能尝试解释它吗？（例如执行命令 2：`./thread 100000`）（或者其他参数。）
- 3、提示：哪些变量是各个线程共享的，线程并发执行时访问共享变量会不会导致意想不到的问题。

实验过程与思路：

1. **程序功能：**在此程序中，原本功能是统计 counter 的次数。但是此处设计了两个线程，他们在变量上拥有同一个共享的变量 counter，因此该程序的功能是查看共享变量对于多线程的影响。输出结果最终数值为原始数值的两倍。

源代码：实验指导已给出。

实验结果展示：



```
vanzque@van-ubuntu1: ~/os
vanzque@van-ubuntu1:~/os$ ls
asm      cpu      cpu.c~  getpid  mem.c   thread  thread.c~
asm.c    cpu.c    gepid.c mem      test.c~ thread.c
vanzque@van-ubuntu1:~/os$ ./thread 1000
Initial value : 0
Final value : 2000
vanzque@van-ubuntu1:~/os$ ./thread 2000
Initial value : 0
Final value : 4000
vanzque@van-ubuntu1:~/os$ ./thread 4000
Initial value : 0
Final value : 8000
vanzque@van-ubuntu1:~/os$ ./thread 8000
Initial value : 0
Final value : 16000
vanzque@van-ubuntu1:~/os$ ./thread 10000
Initial value : 0
Final value : 17987
vanzque@van-ubuntu1:~/os$
```

2. 根据实验现象来看。当参数数值不大时，最终数值是原始数值的两倍，这是由于两个线程共享一个变量导致的。当数值较大时，最终数值出现了比预期结果小的情况。
3. **个人理解：**我认为出现此处情况应该与 CPU 的轮转技术有关，由于 CPU 对于每一个线程都有一定的时间限制，首先时间分给线程 A 执行代码，当线程 A 累加到了一定数值时，此时时间被用完了，而存放在寄存器中的中间变量还没来得及写入实际的物理内存。接着时间分配给线程 B，由于线程 A 算出来的值并没有写回内存，所以实际上此时线程 B 取上一次的数值，并将其写入内存，然后时间结束，又轮到 A 时，A 将其上一次未完成的写入，从而导致数值变低。

实验总结和心得

一开始做实验的时候，主要想到的是对程序的理解。当最后进行总结，完成

实验报告的时候才发觉，每一个实验的单独程序不一定能体现实验的意义，当多个程序进行处理的时候，我们所学的操作系统知识便自然地感受到了，对于操作系统的各个特性都有了进一步的理解与深入。