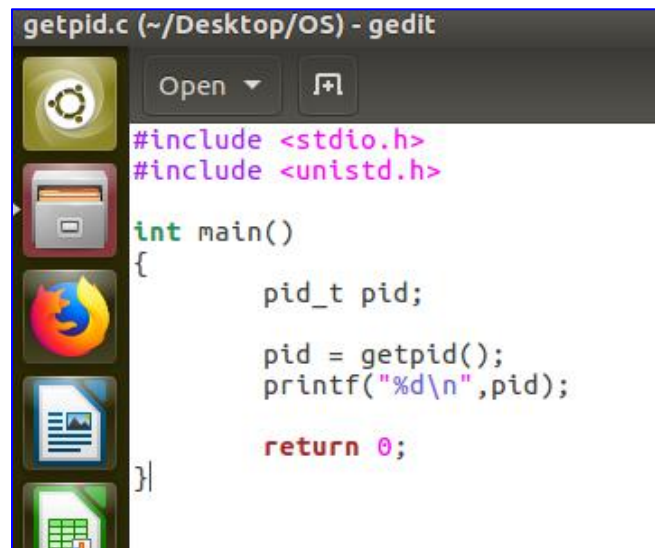


实验一：操作系统初步

袁少随 16281054 安全 1601

一、（系统调用实验）了解系统调用不同的封装形式。

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？)



```
getpid.c (~/Desktop/OS) - gedit
Open  [icon]

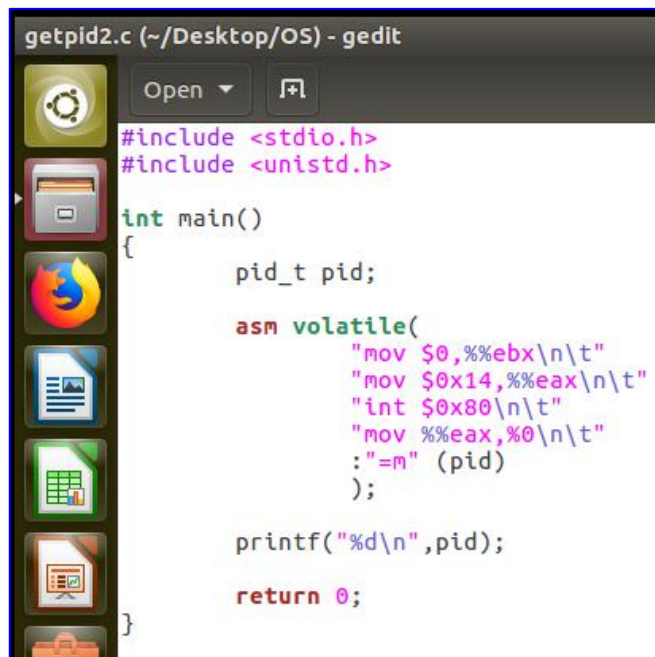
#include <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid;

    pid = getpid();
    printf("%d\n",pid);

    return 0;
}
```

用 API 接口函数 `getpid()` 直接调用程序代码



```
getpid2.c (~/Desktop/OS) - gedit
Open  [icon]

#include <stdio.h>
#include <unistd.h>

int main()
{
    pid_t pid;

    asm volatile(
        "mov $0,%%ebx\n\t"
        "mov $0x14,%%eax\n\t"
        "int $0x80\n\t"
        "mov %%eax,%0\n\t"
        : "=r" (pid)
        );

    printf("%d\n",pid);

    return 0;
}
```

汇编中断调用程序代码

```
yuan@ubuntu: ~/Desktop/OS
yuan@ubuntu:~$ cd Desktop
yuan@ubuntu:~/Desktop$ ls
badfile  OS  security
yuan@ubuntu:~/Desktop$ cd OS
yuan@ubuntu:~/Desktop/OS$ gcc -o getpid getpid.c
yuan@ubuntu:~/Desktop/OS$ ./getpid
10129
yuan@ubuntu:~/Desktop/OS$ gcc -o getpid2 getpid2.c
yuan@ubuntu:~/Desktop/OS$ ./getpid2
10168
yuan@ubuntu:~/Desktop/OS$
```

终端运行命令及结果

getpid 的系统调用号是 20;linux 系统调用的中断向量号是 0X80;

2、上机完成习题 1.13

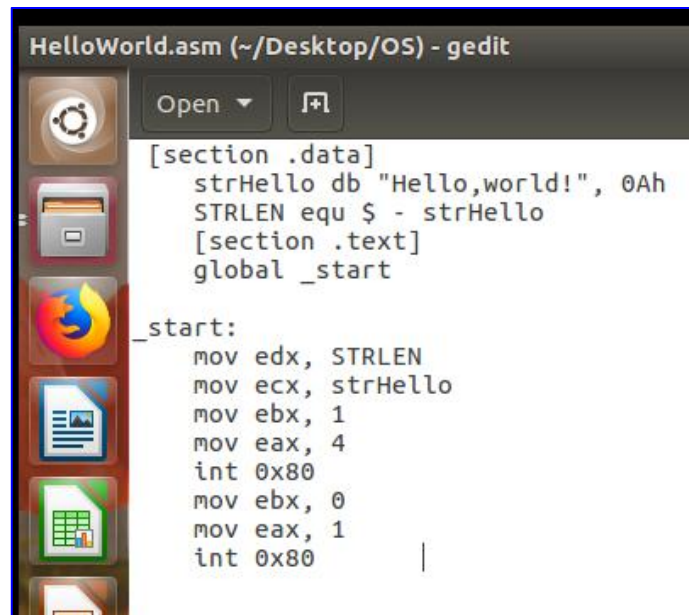
```
HelloWorld.c (~/Desktop/OS) - gedit
#include <unistd.h>

int main()
{
    write(1,"Hello world!\n",13);
    return 0;
}
```

系统调用的 C 函数形式输出“Hello World!”程序

```
yuan@ubuntu:~/Desktop/OS$ gcc -o HelloWorld HelloWorld.c
yuan@ubuntu:~/Desktop/OS$ ./HelloWorld
Hello world!
yuan@ubuntu:~/Desktop/OS$
```

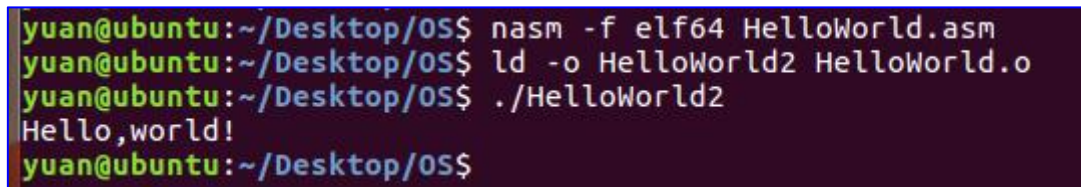
编译运行程序结果



```
[section .data]
    strHello db "Hello,world!", 0Ah
    STRLEN equ $ - strHello
[section .text]
global _start

_start:
    mov edx, STRLEN
    mov ecx, strHello
    mov ebx, 1
    mov eax, 4
    int 0x80
    mov ebx, 0
    mov eax, 1
    int 0x80
```

汇编代码形式输出“Hello World!”程序



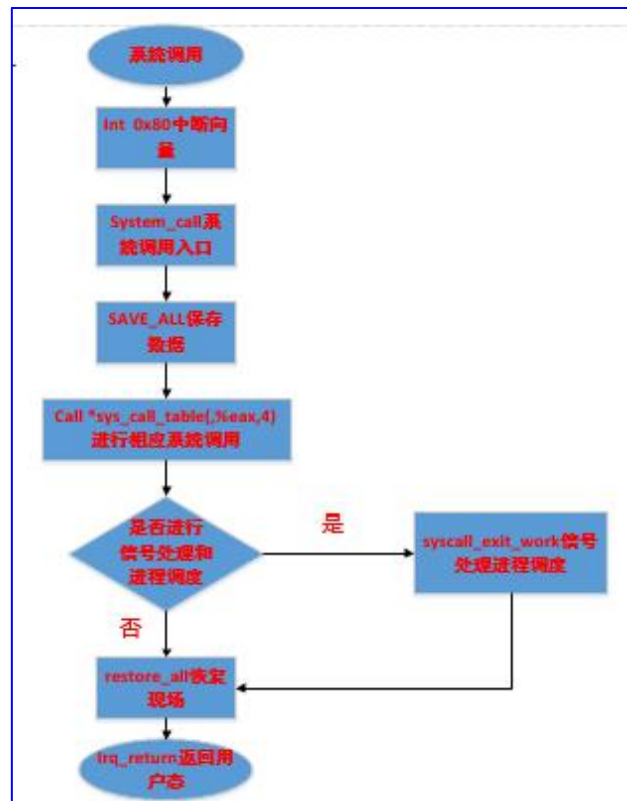
```
yuan@ubuntu:~/Desktop/OS$ nasm -f elf64 HelloWorld.asm
yuan@ubuntu:~/Desktop/OS$ ld -o HelloWorld2 HelloWorld.o
yuan@ubuntu:~/Desktop/OS$ ./HelloWorld2
Hello,world!
yuan@ubuntu:~/Desktop/OS$
```

编译运行程序结果

3、阅读 pintos 操作系统源代码，画出系统调用实现的流程图。

系统调用：

Linux 内核中设置了一组用于实现各种系统功能的子程序，称为系统调用。用户可以通过系统调用命令在自己的应用程序中调用它们。系统调用和普通的函数调用区别在于：系统调用由操作系统核心提供，运行于核心态；而普通的函数调用由函数库或用户自己提供，运行于用户态。



系统调用图

二、（并发实验）根据以下代码完成下面的实验。

- 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。

```

cpu.c (~/Desktop/OS) - gedit
Open  [Icon]

#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <assert.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: cpu <string>\n");
        exit(1);
    }
    char *str = argv[1];
    while (1) {
        sleep(1);
        printf("%s\n", str);
    }
    return 0;
}
  
```

程序（cpu.c）代码

```
yuan@ubuntu:~/Desktop/OS$ gcc -o cpu cpu.c
yuan@ubuntu:~/Desktop/OS$ ./cpu
usage: cpu <string>
yuan@ubuntu:~/Desktop/OS$
```

编译并运行程序

程序功能：

- (1) 当程序参数不为 2 个时，输出 "usage: cpu <string>"，提示错误；
- (2) 当程序输入参数符合要求时，一直打印该参数。

执行命令：./cpu A & ; ./cpu B & ; ./cpu C & ; ./cpu D &结果图：

```
yuan@ubuntu:~/Desktop/OS$ gcc -o cpu cpu.c
yuan@ubuntu:~/Desktop/OS$ ./cpu A & ./cpu B & ./cpu C & ./cpu D
[1] 18612
[2] 18613
[3] 18614
B
C
A
D
C
B
B
A
D
C
B
B
A
D
C
B
A
```

运行的顺序无规律，特点随机。

解释理由：

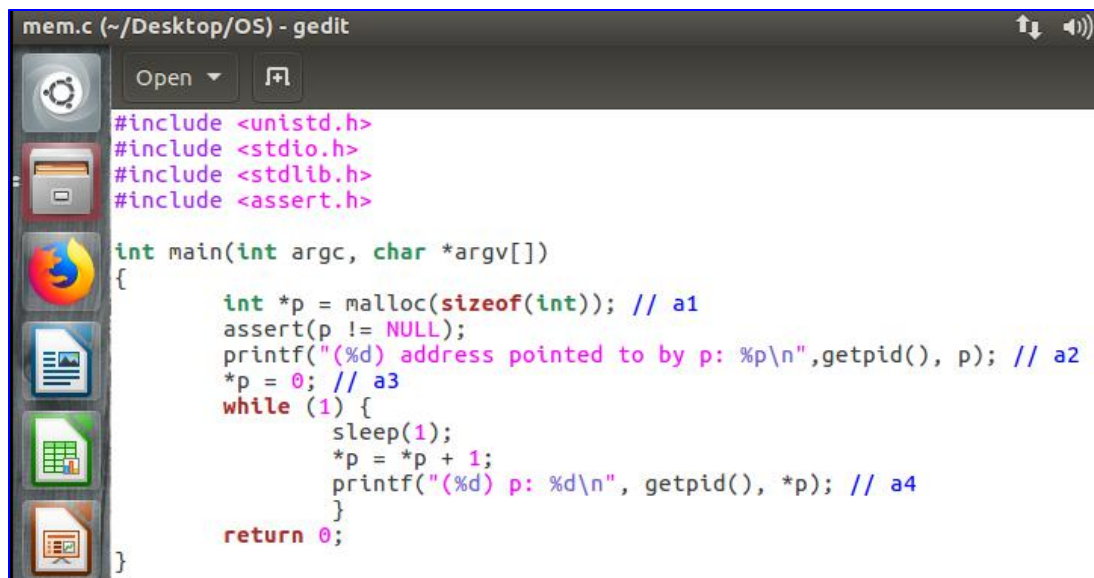
- (1) 四个进程是并发运行的，不是顺序执行的
- (2) 操作系统是多道批处理系统
- (3) 此电脑处理器是多核的，可同时执行多个任务

三、（内存分配实验）根据以下代码完成实验。

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)

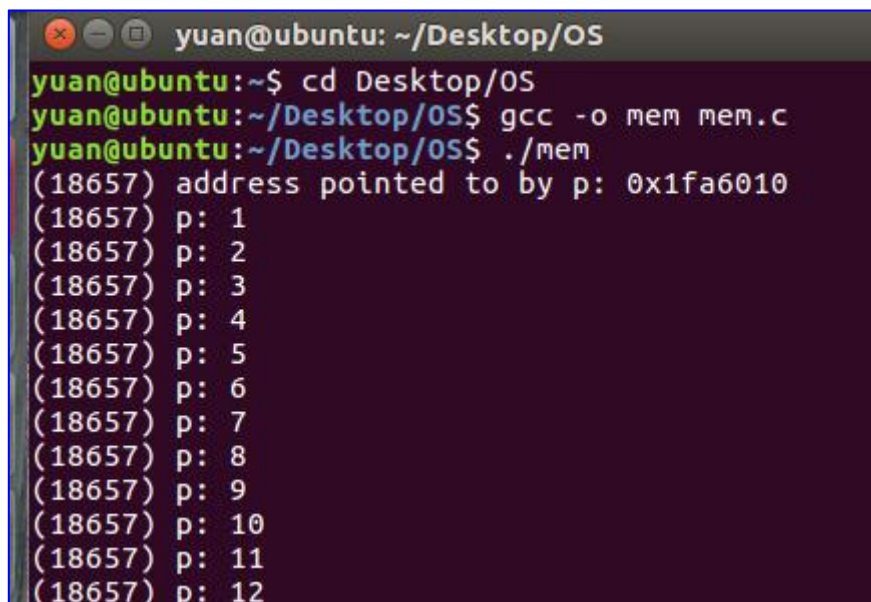
2、再次按下面的命令运行并观察结果。两个分别运行的程序分配的内存地址是否相同？是否共享同一块物理内存区域？为什么？命令： ./mem & ./mem &



```
mem.c (~/Desktop/OS) - gedit
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

int main(int argc, char *argv[])
{
    int *p = malloc(sizeof(int)); // a1
    assert(p != NULL);
    printf("(%d) address pointed to by p: %p\n", getpid(), p); // a2
    *p = 0; // a3
    while (1) {
        sleep(1);
        *p = *p + 1;
        printf("(%d) p: %d\n", getpid(), *p); // a4
    }
    return 0;
}
```

程序 (mem.c) 代码



```
yuan@ubuntu: ~/Desktop/OS
yuan@ubuntu:~$ cd Desktop/OS
yuan@ubuntu:~/Desktop/OS$ gcc -o mem mem.c
yuan@ubuntu:~/Desktop/OS$ ./mem
(18657) address pointed to by p: 0x1fa6010
(18657) p: 1
(18657) p: 2
(18657) p: 3
(18657) p: 4
(18657) p: 5
(18657) p: 6
(18657) p: 7
(18657) p: 8
(18657) p: 9
(18657) p: 10
(18657) p: 11
(18657) p: 12
```

编译并运行程序及结果

程序功能:

首先申请一一个 int 大小的内存地址, 并打印进程号和内存地址。之后就是对该内存地址保存的值进行循环累加操作。

```

yuan@ubuntu:~/Desktop/OS$ ./mem & ./mem
[1] 18664
(18665) address pointed to by p: 0x256f010
(18664) address pointed to by p: 0xdc0010
(18665) p: 1
(18664) p: 1
(18664) p: 2
(18665) p: 2
(18665) p: 3
(18664) p: 3
(18664) p: 4
(18665) p: 4
(18664) p: 5
(18665) p: 5
(18665) p: 6
(18664) p: 6
(18665) p: 7
(18664) p: 7
(18665) p: 8
(18664) p: 8
(18664) p: 9
(18665) p: 9
(18665) p: 10

```

编译并运行程序及结果

答：

两个进程申请分配的内存地址不一样，对于每个进程而言，每个进程该分配的内存空间进行累加操作，互不影响。

每个进程的内存空间只是虚拟内存空间，每次访问内存空间的某个地址，都需要把地址翻译为实际物理内存地址。所有进程共享同一物理内存，每个进程只把自己目前需要的虚拟内存空间映射并存储到物理内存上。由于每个进程的虚拟地址是独立于其他进程的，通过页表将虚拟地址转换为真实地址，不论两个进程申请的虚拟内存地址是否相同，真实的物理地址是不一样的，所以两个进程对地址上的数值操作都是独立的。

四、（共享的问题）根据以下代码完成实验。

- 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。

```
thread.c (~/Desktop/OS) - gedit
Open  [icon]

#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

volatile int counter = 0;
int loops;
void *worker(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        counter++;
    }
    return NULL;
}
int main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: threads <value>\n");
        exit(1);
    }
    loops = atoi(argv[1]);
    pthread_t p1, p2;
    printf("Initial value : %d\n", counter);

    pthread_create(&p1, NULL, worker, NULL);
    pthread_create(&p2, NULL, worker, NULL);
    pthread_join(p1, NULL);
    pthread_join(p2, NULL);
    printf("Final value : %d\n", counter);
    return 0;
}
```

程序（thread.c）代码

```
yuan@ubuntu:~/Desktop/OS$ gcc -o thread thread.c -lpthread
yuan@ubuntu:~/Desktop/OS$ ./thread 1000
Initial value : 0
Final value : 2000
yuan@ubuntu:~/Desktop/OS$ ./thread 100
Initial value : 0
Final value : 200
yuan@ubuntu:~/Desktop/OS$ ./thread 10000
Initial value : 0
Final value : 20000
yuan@ubuntu:~/Desktop/OS$ ./thread 100000
Initial value : 0
Final value : 200000
yuan@ubuntu:~/Desktop/OS$
```

编译并运行程序及结果

程序功能：

程序创建两个线程，对同一个变量执行累加操作；

程序结果：

为输入参数的两倍；

现象解释：

两个线程在同一个进程中，访问的是共享的变量。而且两个线程之间互不影响，则程序执行参数的 2 倍的累加操作。