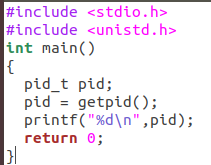
**实验一: 操作系统初步**

1实验要求与步骤

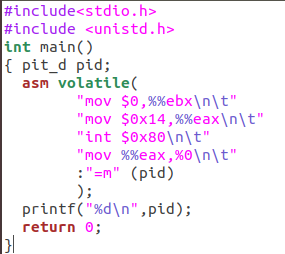
1.1了解系统调用不同的封装形式

1.1.1参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？)。

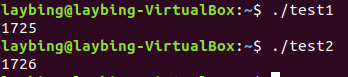
直接调用代码：



汇编中断调用代码：



程序调用结果：



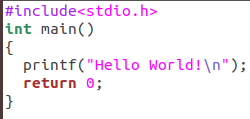
getpid的系统调用号是1725，系统调用的中断向量号是1726。

1.1.2上机完成习题1.13。

printf(“Hello World!\n”)

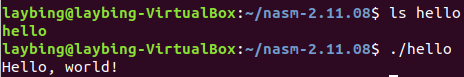
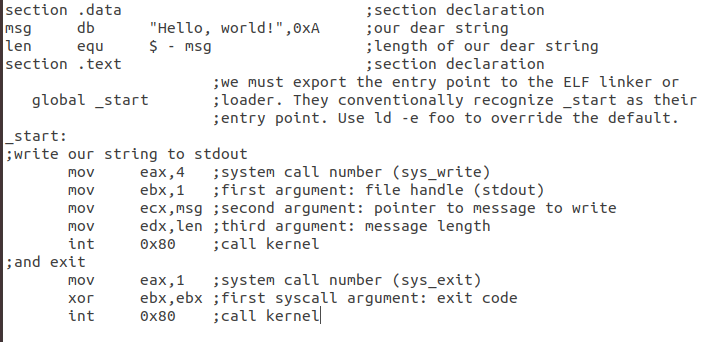
分别用相应的Linux系统调用C函数形式和汇编代码两种形式来实现上述命令

C函数代码及运行结果：

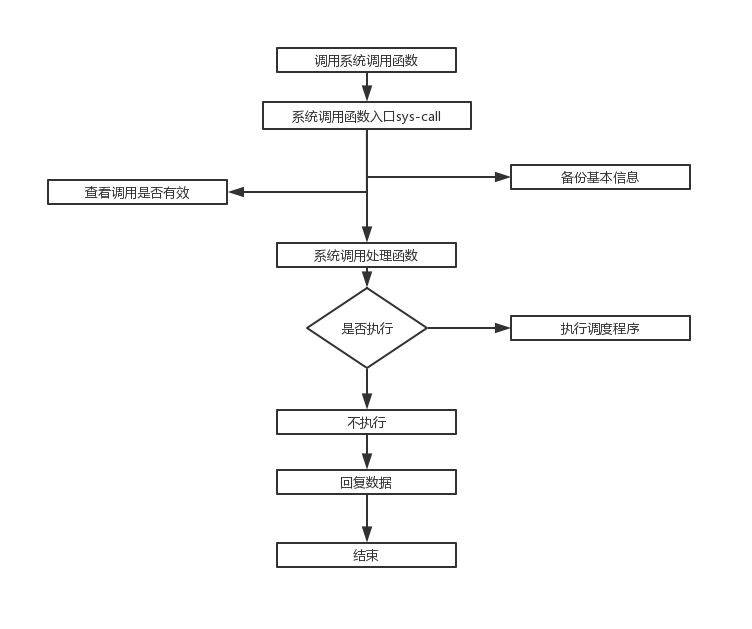


E:\课程\大三下\操作系统\实验\lab1\121.PNG

汇编代码及运行结果：



1.1.3阅读pintos操作系统源代码，画出系统调用实现的流程图。



1.2（并发实验）根据以下代码完成下面的实验。

#include <stdio.h>

#include <stdlib.h>

#include <sys/time.h>

#include <assert.h>

#include "common.h"

Int main(int argc, char \*argv[])

{

if (argc != 2) {

fprintf(stderr, "usage: cpu <string>\n");

exit(1);

}

char \*str = argv[1];

while (1) {

spin(1);

printf("%s\n", str);

}

return 0;

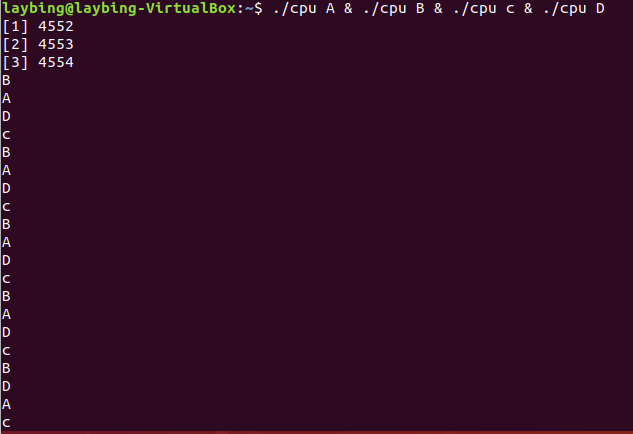
}

1.2.1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）

E:\课程\大三下\操作系统\实验\lab1\2.1.PNG

1.2.2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。



运行结果顺序与程序执行顺序不是对应的，并且每轮运行结果的顺序会有改变。

并发环境下，由于程序的封闭性被打破，程序与计算不再一一对应，一个程序副本可以有多个计算。

1.3（内存分配实验）根据以下代码完成实验。

#include <unistd.h>

#include <stdio.h>

#include <stdlib.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

}

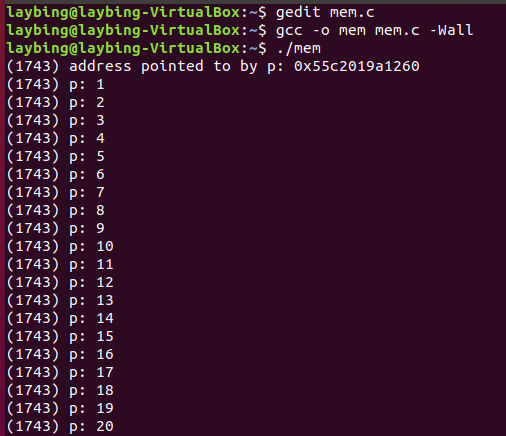
return 0;

}

1.3.1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)

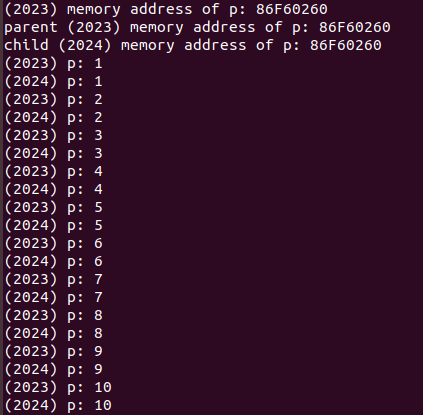
程序执行结果：



首先，通过malloc()函数申请了部分内存。然后打印出了[内存地址](https://www.baidu.com/s?wd=%E5%86%85%E5%AD%98%E5%9C%B0%E5%9D%80&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd)，再将数字0赋值给最新申请的内存地址中。最后，每延迟一秒，打印该程序的进程号，该进程号是唯一的，以及循环递增存储在该地址上的值。

1.3.2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 &

程序执行结果：



每个进程都是从相同的地址开始分配内存，独立地去更新该地址的数值的。

操作系统中真正发生的事情是虚拟内存。每个进程都访问它们自己的私有的虚拟地址空间，操作系统将这些虚拟地址空间以某种方式映射到机器的物理内存。一个运行的程序引用的内存不会影响其他进程的地址空间；就正在运行的程序而言，它独自占有所有的物理内存。然而，事实却是物理内存是一个由操作系统管理的共享资源。

1.4（共享的问题）根据以下代码完成实验。

#include <stdio.h>

#include <stdlib.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);

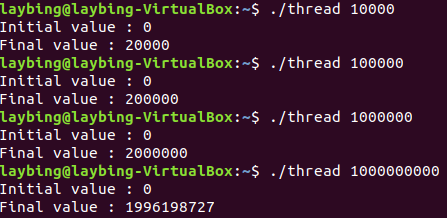
printf("Final value : %d\n", counter);

return 0;

}

1.4.1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）E:\课程\大三下\操作系统\实验\lab1\1.4a.PNG

程序结果输出两个worker函数循环中共享的计数器的增长次数。

1.4.2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。）

E:\课程\大三下\操作系统\实验\lab1\1.4b2.PNG

不断增大输入的值，当输入的值为1000000000，程序给出的结果不是2000000000，而是1995508327。又一次运行，我们不仅得到了一个不一样的错误的结果。

解释：上面程序共享的计数器递增的关键部分包涵三条指令：一个是从内存加载计数器的值到寄存器，一个时增加，一个是将它存入内存。因为这三条指令不是原子执行的（即一次执行所有），所有出了问题。

1.4.3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。

全局变量是各个线程共享的，线程并发执行时访问共享变量会导致原子性问题，可见性问题，有序性问题等。