作业1

1.1 一个C程序可以编译成目标文件或可执行文件。目标文件和可执行文件通常包含text、data、bss、rodata段，程序执行时也会用到堆（heap）和栈（stack）。
（1）请写一个C程序，使其包含data段和bss段，并在运行时包含堆的使用。请说明所写程序中哪些变量在data段、bss段和堆上。
（2）请了解readelf、objdump命令的使用，用这些命令查看（1）中所写程序的data和bss段，截图展示。
（3）请说明（1）中所写程序是否用到了栈。

提交内容：所写C程序、问题解答、截图等。

（1）所写C程序如下图所示。

```c
#include <stdio.h>
#include <stdlib.h>
int a = 1;
int b;
int main(){
    int *p = (int *)malloc(sizeof(int));
    int c = 2;
    return 0;
}
```

a为初始化的全局变量，位于.data段。
b为未初始化的全局变量，位于.bss段。
malloc分配的内存空间位于堆中，即指针p所指内容位于堆。
（2）使用objdump命令，截取.data段和.bss段如下图所示。

```
ubuntu@ubuntu:~/Desktop$ objdump -t hw1 | grep ".data"
00000000000006f0 l    d  .rodata        0000000000000000              .rodata
0000000000201000 l    d  .data  0000000000000000              .data
0000000000201000  w      .data  0000000000000000              data_start
0000000000201014 g       .data  0000000000000000              _edata
0000000000201000 g       .data  0000000000000000              __data_start
0000000000201008 g     O .data  0000000000000000              .hidden __dso_hand
le
00000000000006f0 g     O .rodata        0000000000000004              _IO_stdin_
used
0000000000201010 g     O .data  0000000000000004              a
0000000000201018 g     O .data  0000000000000000              .hidden __TMC_END_
_
ubuntu@ubuntu:~/Desktop$ objdump -t hw1 | grep ".bss"
0000000000201014 l    d  .bss   0000000000000000              .bss
0000000000201014 l     O .bss   0000000000000001              completed.7698
0000000000201018 g     O .bss   0000000000000004              b
0000000000201020 g       .bss   0000000000000000              _end
0000000000201014 g       .bss   0000000000000000              __bss_start
ubuntu@ubuntu:~/Desktop$
```

由图可知，初始化的全局变量a，位于.data段；未初始化的全局变量b，位于.bss段。
（3）用到了栈。main函数的调用、函数内的局部变量c均使用到了栈。c保存在main函数的函数栈中。

1.2 Linux 下常见的3种系统调用方法包括有：
（1）通过glibc提供的库函数
（2）使用syscall函数直接调用相应的系统调用
（3）通过int 80指令（32位系统）或者syscall指令（64位系统）的内联汇编调用

请研究Linux(kernel>=2.6.24) getpid这一系统调用的用法，使用上述3种系统调用方法来执行，并记录和对比3种方法的运行时间，并尝试解释时间差异结果。

提示：gettimeofday和clock_gettime是Linux下用来测量耗时的常用函数，请调研这两个函数，选择合适函数来测量一次系统调用的时间开销。

提交内容：所写程序、执行结果、结果分析、系统环境（uname -a)等。

代码如下：

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>
struct timespec tv0,tv1,tv2,tv3;
long sec1,sec2,sec3,nsec1,nsec2,nsec3;
int main(){
    //glibc
    clock_gettime(CLOCK_REALTIME,&tv0);
    int pid1 = getpid();
    clock_gettime(CLOCK_REALTIME,&tv1);
    nsec1 =  (tv1.tv_nsec-tv0.tv_nsec);
    //syscall
    clock_gettime(CLOCK_REALTIME,&tv0);
    int pid2 = syscall(SYS_getpid);
    clock_gettime(CLOCK_REALTIME,&tv2);
    nsec2 =  (tv2.tv_nsec-tv0.tv_nsec);
    //int 0x80
    clock_gettime(CLOCK_REALTIME,&tv0);
    int pid3;
```

```
25      asm volatile(
26          "mov $0x14,%%eax\n\t"
27          "int $0x80\n\t"
28          "mov %%eax,%0\n\t"
29          :"=m"(pid3)
30      );
31      clock_gettime(CLOCK_REALTIME,&tv3);
32      nsec3 =  (tv3.tv_nsec-tv0.tv_nsec);
33      //time
34      printf("pid1 is %d\n",pid1);
35      printf("pid2 is %d\n",pid2);
36      printf("pid3 is %d\n",pid3);
37      printf("time elapse of 1 is %ld ns\n",nsec1);
38      printf("time elapse of 2 is %ld ns\n",nsec2);
39      printf("time elapse of 3 is %ld ns\n",nsec3);
40      return 0;
41  }
42
```

运行比对结果如下：

```
ubuntu@ubuntu:~/Desktop$ gcc -g hw1.c -o hw1
ubuntu@ubuntu:~/Desktop$ ./hw1
pid1 is 5555
pid2 is 5555
pid3 is 5555
time elapse of 1 is 2997 ns
time elapse of 2 is 289 ns
time elapse of 3 is 646 ns
```

结果分析：

glibc进行系统调用耗时最长，syscall耗时最短。glibc有函数封装调用，理应更耗时，与预期相符。

理论上内联式汇编应当最快，因为省去了函数封装，直接在汇编层面进行系统调用，与预期不是很相符。猜测是编译环境和编译器优化的问题。

系统环境：

```
ubuntu@ubuntu:~/Desktop$ uname -a
Linux ubuntu 5.4.0-124-generic #140~18.04.1-Ubuntu SMP Fri Aug 5 11:43:34 UTC 20
22 x86_64 x86_64 x86_64 GNU/Linux
```