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。**

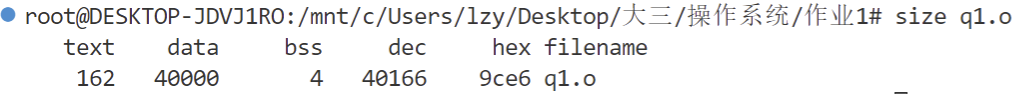
**（2）请了解readelf、objdump命令的使用，用这些命令查看（1）中所写程序的data和bss段，截图展示。**

**（3）请说明（1）中所写程序是否用到了栈。**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | #include<stdio.h>  #include<stdlib.h>  **int** array[100\*100]={1};  **int** main(){  **static** **int** bss\_1;  **char** \* str[100];  **void** \* now = (**void** \*)**malloc**(**sizeof**(**int**));  **return** 0;  } |

本程序中，array变量在data段，bss\_1变量在bss段，指针数组str位于栈上，now指针在栈上，在运行时分配给now的空间位于堆上。

首先执行size命令，可以观察到：



其次执行

其中text、data、bss段都分配空间给了一些变量。随后执行objdump -t q1.o指令：

表格

描述已自动生成

可以观察到，bss\_1变量是个静态局部变量，位于bss段。Array是已初始化的全局变量，位于data段。

在这段代码中，指针数组str和now指针在栈上，因为其为局部变量，位于栈中，运行时才会向其分配内存。

**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**

**系统环境：**



**函数选择：**

经过调研发现，gettimeofday和clock\_gettime两种函数中clock\_gettime更适合测量系统调用的时间开销，因为clock\_gettime函数可以精准到纳秒级别，而gettimeofday只能达到微秒级别。

**三种系统调用的实现：**

通过glibc库原生提供的getpid()函数完成调用，代码如下：

1. #include<stdio.h>
2. #include<sys/stat.h>
3. #include<unistd.h>
4. #include<time.h>
5. **int** main(){
6. **struct** timespec t1 = {0, 0};
7. **struct** timespec t2 = {0, 0};
9. pid\_t pid;
11. clock\_gettime(CLOCK\_REALTIME, &t1);
13. pid = getpid();
15. clock\_gettime(CLOCK\_REALTIME, &t2);
16. **double** duration=(t2.tv\_sec - t1.tv\_sec)\* 1000000000 + (t2.tv\_nsec - t1.tv\_nsec);
17. printf("time = %.4lf ns\n",duration);
19. printf("current process's pid(getpid):%d %d\n",pid,getpid());
20. **return** 0;
21. }

执行结果：

文本

描述已自动生成

通过syscall函数获取进程id代码如下：

1. #include<stdio.h>
2. #include<sys/stat.h>
3. #include <sys/syscall.h>
4. #include <sys/types.h>
5. #include<unistd.h>
6. #include<time.h>
7. **int** main(){
8. **struct** timespec t1 = {0, 0};
9. **struct** timespec t2 = {0, 0};
10. clock\_gettime(CLOCK\_REALTIME, &t1);
11. pid\_t pid;
12. pid=syscall(SYS\_gettid);
13. clock\_gettime(CLOCK\_REALTIME, &t2);
15. **double** duration=(t2.tv\_sec - t1.tv\_sec)\* 1000000000 + (t2.tv\_nsec - t1.tv\_nsec);
16. printf("time = %.4lf ns\n",duration);
18. printf("current process's pid(SYS\_gettid):%d %d\n",pid,getpid());
19. **return** 0;
20. }

执行结果：

文本, 信件

描述已自动生成

通过汇编方式获得进程ID方式如下：

64位内联汇编

1. #include<stdio.h>
2. #include<sys/stat.h>
3. #include<unistd.h>
4. #include<time.h>
5. **int** main(){
6. **struct** timespec t1 = {0, 0};
7. **struct** timespec t2 = {0, 0};
8. clock\_gettime(CLOCK\_REALTIME, &t1);
9. pid\_t pid;
10. asm **volatile**(
11. "mov $39, %%eax\n\t"
12. "syscall\n\t"
13. :"=a"(pid)
14. /\*第一个冒号后的限定字符串用于描述指令中的“输出”操作数。
15. 刮号中的pid将操作数与C语言的变量联系起来。
16. "a":寄存器EAX。
17. "b":寄存器EBX。
18. "c":寄存器ECX。
19. "d":寄存器EDX。
20. \*/
21. );
23. clock\_gettime(CLOCK\_REALTIME, &t2);
24. **double** duration=(t2.tv\_sec - t1.tv\_sec)\* 1000000000 + (t2.tv\_nsec - t1.tv\_nsec);
25. printf("time = %.4lf ns\n",duration);
27. printf("current process's pid(ASM):%d %d\n",pid,getpid());
28. **return** 0;
29. }

32位内联汇编

1. #include<stdio.h>
2. #include<sys/stat.h>
3. #include<unistd.h>
4. #include<time.h>
5. **int** main(){
6. **struct** timespec t1 = {0, 0};
7. **struct** timespec t2 = {0, 0};
8. clock\_gettime(CLOCK\_REALTIME, &t1);
9. pid\_t pid;
10. asm **volatile**(
11. "mov $20, %%eax\n\t"
12. "int $0x80\n\t"
13. :"=a"(pid)
14. );
16. clock\_gettime(CLOCK\_REALTIME, &t2);
17. **double** duration=(t2.tv\_sec - t1.tv\_sec)\* 1000000000 + (t2.tv\_nsec - t1.tv\_nsec);
18. printf("time = %.4lf ns\n",duration);
20. printf("current process's pid(ASM):%d %d\n",pid,getpid());
21. **return** 0;
22. }

执行结果

文本

中度可信度描述已自动生成

**运行时间的对比分析：**

|  |  |
| --- | --- |
| **方式** | **时间（五次平均/ns）** |
| **gblic库函数调用** | **1864** |
| **Syscall系统调用** | **1293** |
| **64位内联汇编** | **344** |
| **32位内联汇编** | **452** |
| 编译时没有开任何优化开关 | |

**分析：**

三种方法的运行时间差异符合预期：

对于gblic库的库函数getpid()调研后发现其本质是一个封装好的系统调用，其实先和syscall的实现类似，那么其速度的确应该慢于syscall。对于32位和64位内联汇编语言来讲，64位汇编语言肯定是快于32位的，因为这是一台64位机，32位api会被转为64位api再执行。总的来讲，汇编语言肯定要快于高层次的系统调用，所以三种方法都符合预期。