**实验一 - Linux进程管理及其扩展**

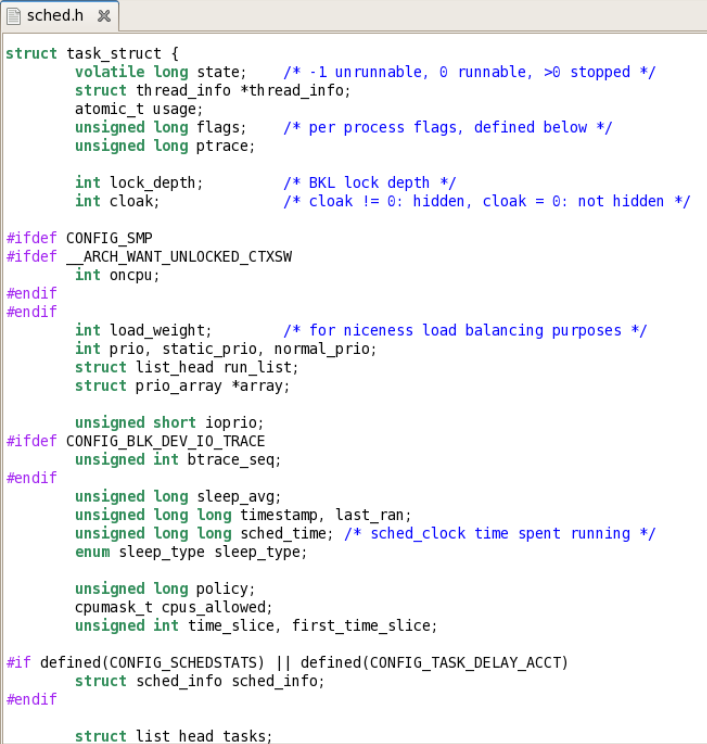
2022年8月10日

1. **实验内容**
2. 阅读并分析Linux内核源代码，了解进程控制块、进程队列等数据结构；
3. 实现一个系统调用，使得可以根据指定的参数隐藏进程，使用户无法使用ps或top观察到进程状态。具体要求如下：
4. 实现系统调用int hide(pid\_t pid, int on)，在进程pid有效的前提下，如果on置1，进程被隐藏，用户无法通过ps或top观察到进程状态；如果on置0且此前为隐藏状态，则恢复正常状态。
5. 考虑权限问题，只有根用户才能隐藏进程。
6. 设计一个新的系统调用int hide\_user\_processes(uid\_t uid, char \*binname)，参数uid为用户ID号，当binname参数为NULL时，隐藏该用户的所有进程；否则，隐藏二进制映像名为binname的用户进程。该系统调用应与hide系统调用共存。
7. 在/proc目录下创建一个文件/proc/hidden，该文件可读可写，对应一个全局变量hidden\_flag，当hidden\_flag为0时，所有进程都无法隐藏，即便此前进程被hide系统调用要求隐藏。只有当hidden\_flag为1时，此前通过hide调用要求被屏蔽的进程才隐藏起来。
8. 在/proc目录下创建一个文件/proc/hidden\_process，该文件的内容包含所有被隐藏进程的pid，各pid之间用空格分开。
9. **实验目的**

通过实验，加深理解进程控制块、进程队列等概念，了解进程管理的具体实施方法。

1. **设计思路和流程图**
2. 阅读并分析Linux内核源代码，了解进程控制块、进程队列等数据结构。

进程控制块PCB是描述一个进程的数据块，在linux内核中对应的是task\_struct数据结构，定义在linux-2.6.21/include/linux/sched.h文件中，可以看到其中包含了众多成员，包括进程状态、标识符、内存指针、优先级、计数器等等，用于描述、管理一个进程：



对进程的管理，便是修改进程对应的task\_struct中的相应信息，以及修改进程在队列中的顺序，来实现进程的不同状态的切换、修改分配给进程的系统资源、不同进程的执行顺序等等。

1. (1) 实现系统调用int hide(pid\_t pid, int on)，在进程pid有效的前提下，如果on置1，进程被隐藏，用户无法通过ps或top观察到进程状态；如果on置0且此前为隐藏状态，则恢复正常状态。

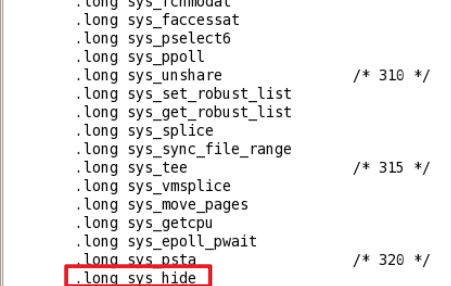
**设计思路：**

linux中的进程显示需要完成对当前系统所有进程的遍历、查找，其实现分别在proc\_pid\_readdir()和proc\_pid\_lookup()函数中实现，我们为每一个进程添加一个用于表示是否隐藏的标识符，当遍历进程读到了这一标识符，则跳过该进程，就实现了对进程的隐藏。

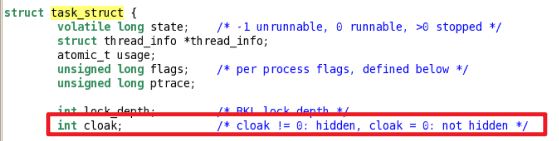
**具体实现：**

**进入linux-2.6.21文件夹：**cd Desktop/linux-2.6.21，实验中的大部分操作需要在该路径下进行。

**新增系统调用：**输入gedit arch/i386/kernel/syscall\_table.S编辑文件，末尾追加.long sys\_hide新增名为sys\_hide的系统调用，系统调用号为321：



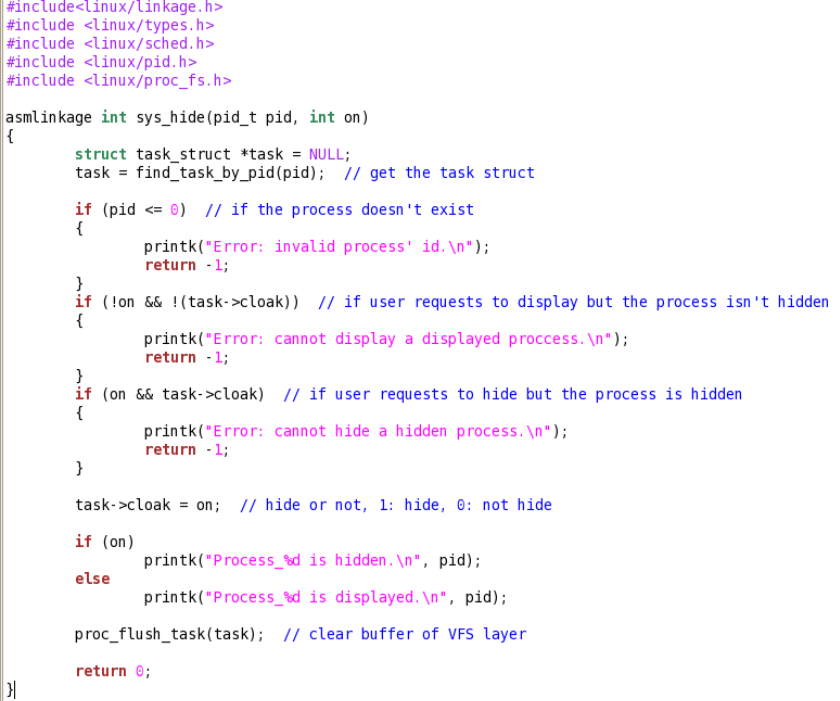
**为进程控制块新增一个用于标记隐藏与否的成员：**输入gedit include/linux/sched.h编辑文件，在task\_struct结构体下新增int类型成员cloak（由于c语言中没有bool类型，所以用int类型表示，cloak = 0时进程不隐藏，否则隐藏），保存文件：



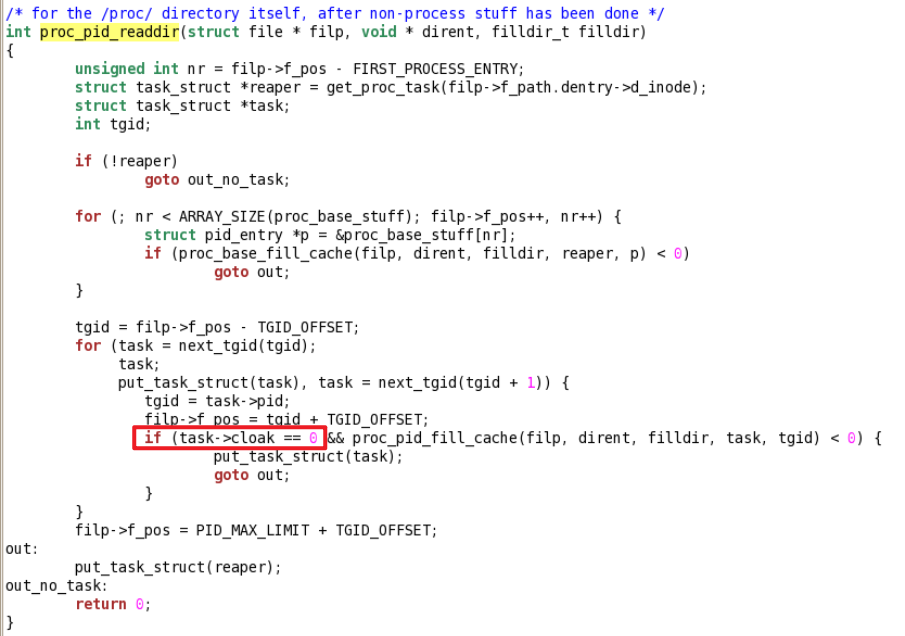
**创建进程时初始化为显示进程（cloak = 0）：**fork系统调用的实现在do\_fork()函数中，其又通过copy\_process()函数来创建子进程，所以在其中实现对cloak的初始化。输入gedit kernel/fork.c编辑文件，在copy\_process()函数下新增p->cloak = 0（p是指向当前所操作进程的指针），保存文件：



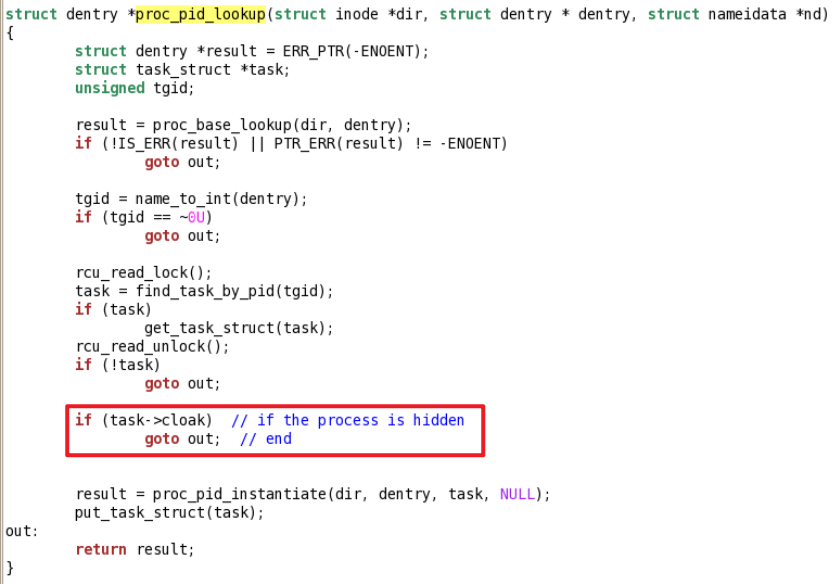
**设计如何实现进程隐藏：**功能在名为sys\_hide()的函数中实现，需要传入的参数有：进程号pid（pid\_t类型）和是否隐藏on（int类型），使用find\_task\_by\_pid()函数并由进程号获取该进程的task\_struct。完成对错误输入的判断：确保进程号有效（pid>0），不能隐藏已隐藏的进程（on与cloak不能同时非零），不能显示未隐藏的进程（on与cloak不能同时为零），并使用printk()输出错误信息到日志缓冲区。更新进程的cloak为传入值，并输出操作完成的提示。具体实现如下：



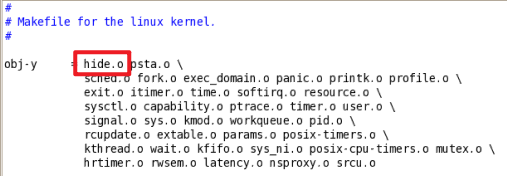
**修改proc\_pid\_readdir()函数：**在对进程的遍历中新增判断条件：cloak=0，即只有不隐藏进程时才有效。输入gedit fs/proc/base.c编辑文件，修改如下，保存文件：



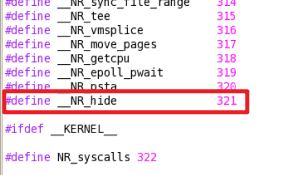
**修改proc\_pid\_lookup()函数：**在对进程的查找中新增判断条件：cloak=0，即只有不隐藏进程时才有效。输入gedit fs/proc/base.c编辑文件，修改如下，保存文件：



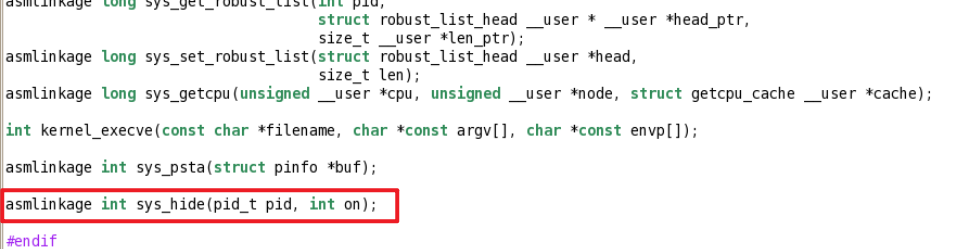
**使hide.c在编译时可见**：输入gedit kernel/Makefile编辑文件，在obj-y之后添加hide.o项，保存文件：



**新增系统调用号宏定义：**输入gedit include/asm-i386/unistd.h编辑文件，新增定义\_\_NR\_hide 321，并修改NR\_syscalls项的定义值为322，保存文件：



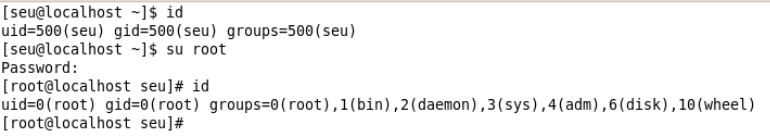
**新增系统调用sys\_hide()的声明：**输入gedit include/linux/syscalls.h编辑文件，末尾添加函数的声明，保存文件：



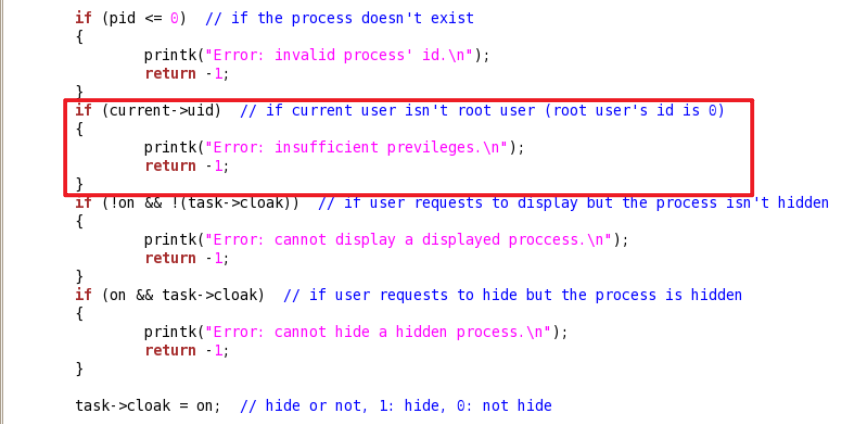
**设计系统调用测试程序：**与2.(2)部分合并进行测试，见下文。

1. (2) 考虑权限问题，只有根用户才能隐藏进程。

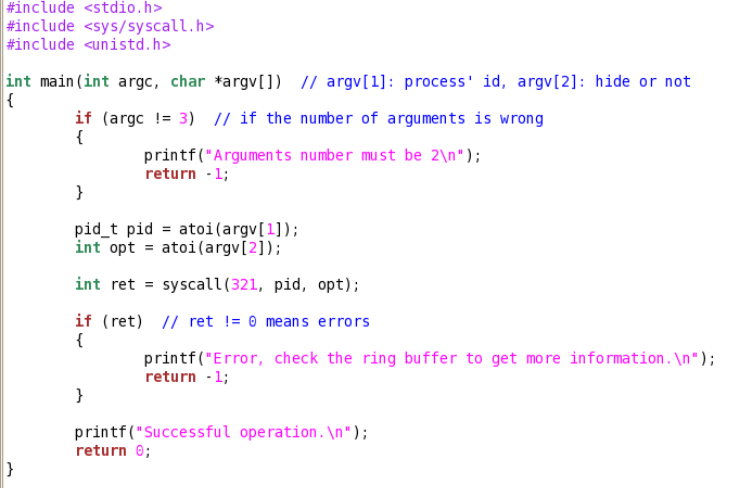
命令行中输入id可以查看当前用户uid，先在普通用户下输入id命令，显示uid=500，切换至根用户，显示uid=0：



为确保只有根用户才能执行隐藏进程的操作，在实现系统调用的hide.c文件中新增uid=0的判断，若非根用户则输出错误并返回：



**设计系统调用测试程序：**需要编写一个c文件，在其中调用321号系统调用，也即sys\_hide()。测试程序同样需要用户传入两个参数：进程号pid（pid\_t类型）与要进行的操作opt（int类型），通过主函数从命令行传入。则需要对用户所输入参数数量进行判断，必须为2（即argc=3），否则提示错误程序退出。通过系统调用函数syscall()调用sys\_hide()并传入以上两个参数，获取syscall()的返回值，若为-1（在sys\_hide()中，为成功执行返回值为-1，否则为0），则提示用户执行失败，查看日志缓冲区获取详细错误信息，若成功执行则打印操作成功。具体实现如下：



1. (3) 设计一个新的系统调用int hide\_user\_processes(uid\_t uid, char \*binname)，参数uid为用户ID号，当binname参数为NULL时，隐藏该用户的所有进程；否则，隐藏二进制映像名为binname的用户进程。该系统调用应与hide系统调用共存。

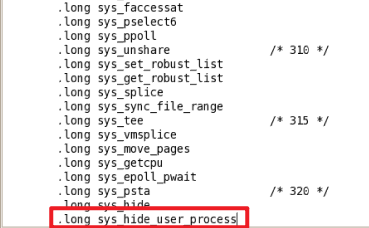
**设计思路：**

首先需要对当前用户的权限进行判断：非根用户无权限执行；再传入的参数进行判断：若binname不为NULL，那么隐藏uid用户的所有名为binname的进程：使用for\_each\_process()函数遍历所有进程，依次获取每一个进程的用户号uid和进程名comm（二者均为task\_struct下的成员），将程序的uid、comm与用户输入的uid、binname进行匹配，将符合的进程的cloak项置为1；

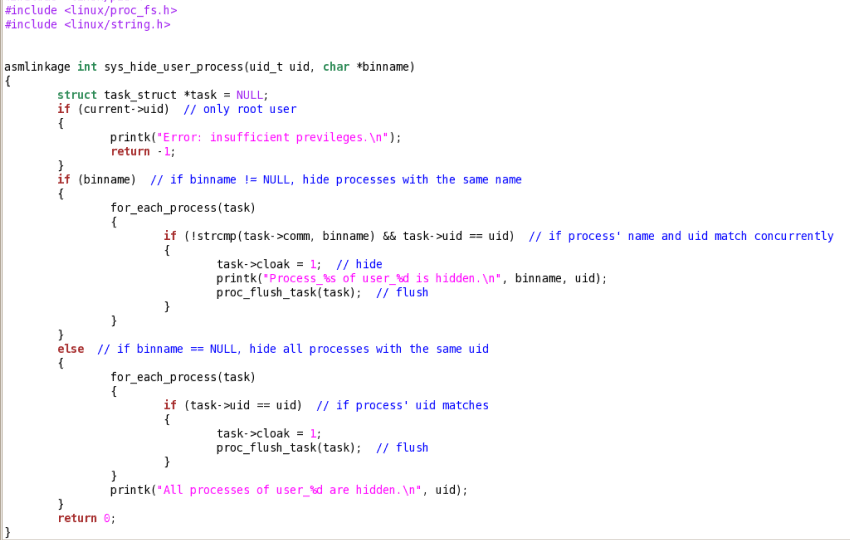
若用户没有指定binname，那么隐藏uid用户的所有进程：遍历过程与前面相同，但是只匹配uid，将符合的进程的cloak置为1进行隐藏。

**具体实现：**

**新增系统调用：**输入gedit arch/i386/kernel/syscall\_table.S编辑文件，末尾追加.long sys\_hide\_user\_process新增名为sys\_hide\_user\_process的系统调用，系统调用号为322：

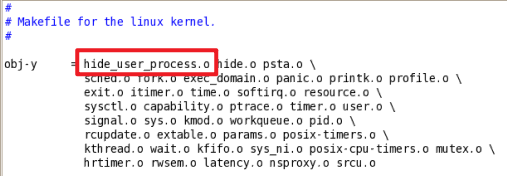


**实现系统调用sys\_hide\_user\_process：**输入gedit kernel/hide\_user\_process.c创建文件，按照前面的设计编写程序，在发生错误与成功执行后都要打印相应的提示内容，具体实现如下：

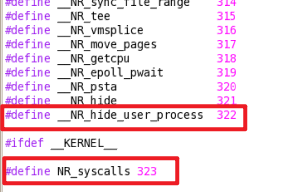


其中使用了strcmp()函数来比较两个字符串，需要注意的是，在相同时返回0，字符串1<字符串2时返回负数，大于时返回正数。

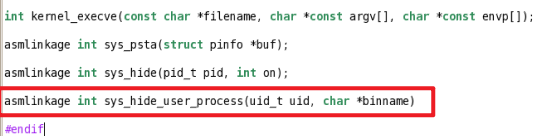
**使hide.c在编译时可见**：输入gedit kernel/Makefile编辑文件，在obj-y之后添加hide\_user\_process.o项，保存文件：



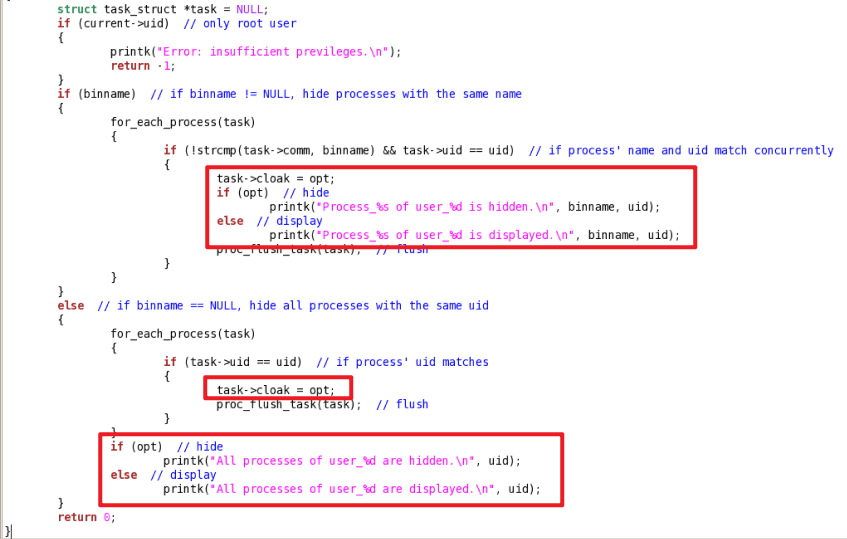
**新增系统调用号宏定义：**输入gedit include/asm-i386/unistd.h编辑文件，新增定义\_\_NR\_hide\_user\_process 322，并修改NR\_syscalls项的定义值为323，保存文件：



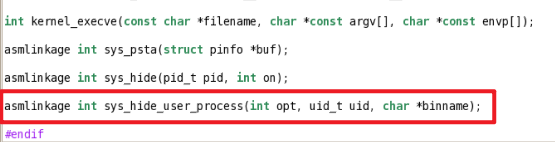
**新增系统调用sys\_hide()的声明：**输入gedit include/linux/syscalls.h编辑文件，末尾添加函数的声明，保存文件：



**问题提出：**该系统调用当前只实现了将用户的进程隐藏，但如果需要将这些进程显示，则需要一项一项来执行显示操作，较为麻烦，所以修改系统调用函数，新增一个参数opt（int类型），控制显示与隐藏，具体修改如下：



函数参数改变，需要在函数的声明中相应地改动，避免函数重定义的错误。输入gedit include/linux/syscalls.h编辑文件，末尾添加修改的函数的声明，保存文件：

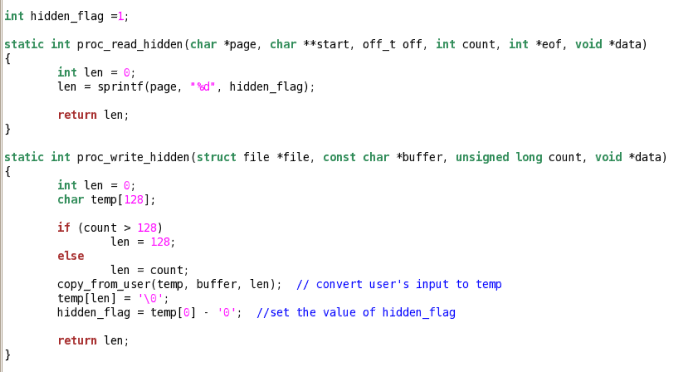


**设计系统调用测试程序：**与2.(5)部分合并进行测试，见下文。

1. (4) 在/proc目录下创建一个文件/proc/hidden，该文件可读可写，对应一个全局变量hidden\_flag，当hidden\_flag为0时，所有进程都无法隐藏，即便此前进程被hide系统调用要求隐藏。只有当hidden\_flag为1时，此前通过hide调用要求被屏蔽的进程才隐藏起来。

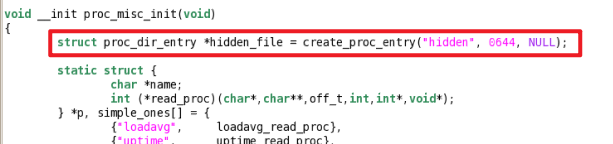
**定义hidden\_flag：**首先需要定义一个全局变量hidden\_flag，写在fs/proc/路径下的一个新建的头文件hidden\_flag.h中（因为需要调用hidden\_flag的文件都在fs/proc/下），在需要使用到时，包含该头文件即可。

**实现回调函数：**输入gedit fs/proc/proc\_misc.c编辑文件，在proc\_misc.c文件中实现回调函数，注意要包含头文件”hidden\_flag.h”，保存文件：

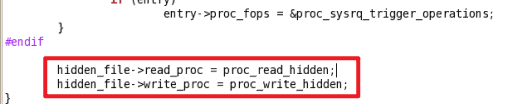


对写函数稍作解释：用户输入的值先写进buffer，对长度进行判断后再传递至temp（char类型数组）中，然后用户所输入的需要写入文件的值应为temp数组的首元素，即temp[0]，需要减去’0’的原因是写入的内容是char类型，减去’0’得到一个整形值，即要赋给hidden\_flag的值。

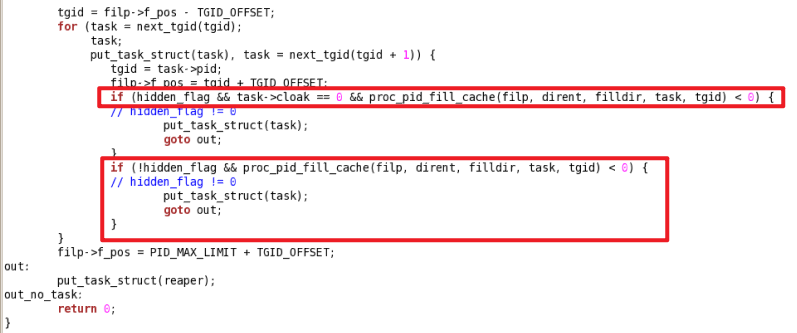
**创建hidden文件：**这里选择将创建hidden文件的代码插入到proc\_misc\_init()函数中，具体实现如下：



create\_proc\_entry()函数的三个参数分别为文件名、访问权限（0644表示用户具有读写权限，组用户和其他用户具有只读权限）和父目录（NULL表示/proc根目录）。



**修改隐藏显示条件：**在前面需要对隐藏进程进行过滤的函数proc\_pid\_readdir()和proc\_pid\_lookup()中需要补充条件hidden\_flag!=0，输入gedit fs/proc/base.c编辑文件，具体改动如下，并添加hidden\_flag.h头文件，保存文件：

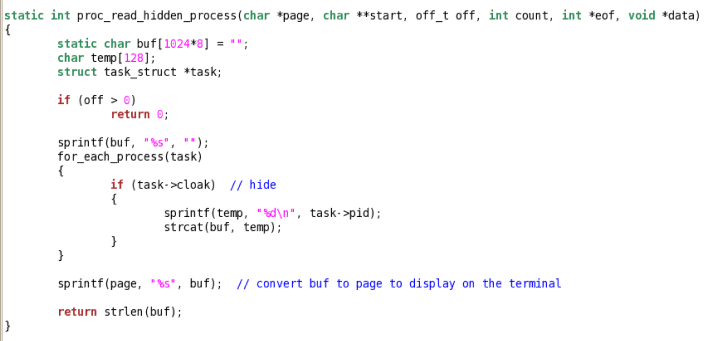




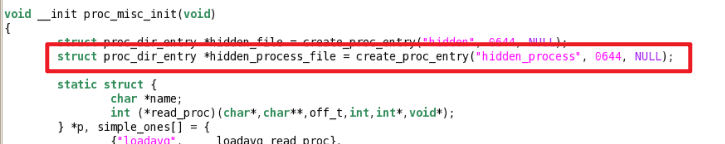
**设计系统调用测试程序：**与2.(5)部分合并进行测试，见下文。

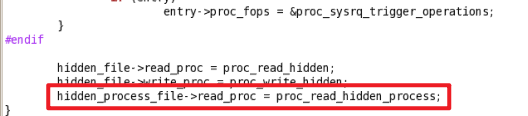
1. (5) 在/proc目录下创建一个文件/proc/hidden\_process，该文件的内容包含所有被隐藏进程的pid，各pid之间用空格分开。

**实现回调函数：**在proc\_misc.c文件中定义回调函数，因为只要求其可读，所以只写读函数即可，具体实现如下：



**创建hidden\_process文件：**修改proc\_misc\_init()函数，具体实现如下，保存文件：

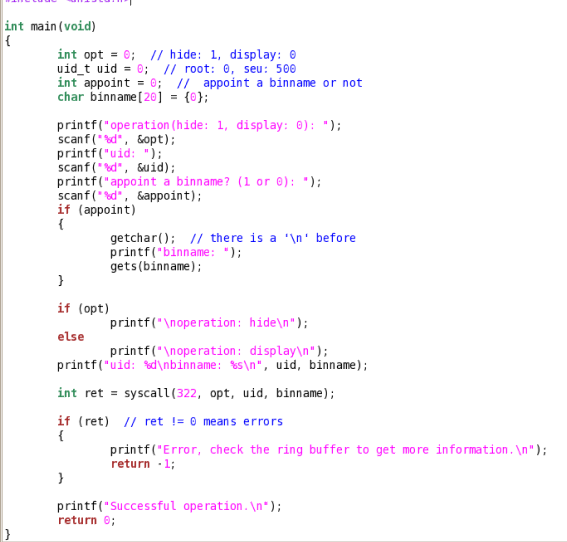




这里的hidden\_process文件只要求可读，所以其实访问权限可以设为0444，即文件只有可读权限。

**设计系统调用测试程序：**编写一个c文件，在其中调用322号系统调用，也即sys\_hide\_user\_process()。

用户在提示下依次输入：隐藏或显示的操作opt（int类型）、用户号uid（int类型，传入系统调用函数后是uid\_t类型）、是否指定进程名appoint（int类型），若用户要指定进程名，则再输入进程名binname，否则binname为NULL。通过系统调用函数syscall()调用sys\_hide\_user\_process ()并传入opt、uid、binname3个参数，获取syscall()的返回值，若为-1（在sys\_hide\_user\_process ()中，为成功执行返回值为-1，否则为0），则提示用户执行失败，查看日志缓冲区获取详细错误信息，若成功执行则打印操作成功。具体实现如下：



1. **主要数据结构及其说明**

使用的数据结构主要有task\_struct，是linux操作系统下的进程控制块，其中包含了用于描述、管理、控制一个进程的众多成员，比如进程名、进程号、对应的用户、多个指针、寄存器等等，在报告开头有写。在实验中为其新增了一个成员cloak，用于描述进程是否被隐藏。

1. **源程序并附上注释**

* **/kernel/hide.c**

1. #include<linux/linkage.h>
2. #include <linux/types.h>
3. #include <linux/sched.h>
4. #include <linux/pid.h>
5. #include <linux/proc\_fs.h>
7. asmlinkage **int** sys\_hide(pid\_t pid, **int** on)
8. {
9. **struct** task\_struct \*task = NULL;
10. task = find\_task\_by\_pid(pid);  // 由pid获取task\_struct
12. **if** (pid <= 0)  // pid<0无效，报错退出
13. {
14. printk("Error: invalid process' id.\n");
15. **return** -1;
16. }
17. **if** (current->uid)  // 当前用户uid不为0（不是根用户），报错退出
18. {
19. printk("Error: insufficient previleges.\n");
20. **return** -1;
21. }
22. **if** (!on && !(task->cloak))  // on和cloak都为0（用户要显示一个未隐藏的进程），报错退出
23. {
24. printk("Error: cannot display a displayed proccess.\n");
25. **return** -1;
26. }
27. **if** (on && task->cloak)  // on和cloak都为1（用户要隐藏一个已隐藏的进程），报错退出
28. {
29. printk("Error: cannot hide a hidden process.\n");
30. **return** -1;
31. }
33. task->cloak = on;  // 设定cloak与on相同，1为隐藏，0为显示
35. **if** (on)  // 输出对应提示信息
36. printk("Process\_%d is hidden.\n", pid);
37. **else**
38. printk("Process\_%d is displayed.\n", pid);
40. proc\_flush\_task(task);  // 清理VFS层缓冲
42. **return** 0;
43. }

* **/kernel/hide.c**

1. #include <linux/linkage.h>
2. #include <linux/types.h>
3. #include <linux/sched.h>
4. #include <linux/pid.h>
5. #include <linux/proc\_fs.h>
6. #include <linux/string.h>
8. asmlinkage **int** sys\_hide\_user\_process(**int** opt, uid\_t uid, **char** \*binname)
9. {
10. **struct** task\_struct \*task = NULL;
11. **if** (current->uid)  // 用户uid不为0（非根用户），报错退出
12. {
13. printk("Error: insufficient previleges.\n");
14. **return** -1;
15. }
16. **if** (binname != NULL)  // 进程名非空，则隐藏指定进程
17. {
18. for\_each\_process(task)  // 遍历进程列表
19. {
20. **if** (!strcmp(task->comm, binname) && task->uid == uid)  // 进程名与uid都和输入的参数匹配
21. {
22. task->cloak = opt;  // 设定cloak与opt相等，1为隐藏，0为显示
23. **if** (opt)  // 输出相应提示信息
24. printk("Process\_%s of user\_%d is hidden.\n", binname, uid);
25. **else**
26. printk("Process\_%s of user\_%d is displayed.\n", binname, uid);
27. proc\_flush\_task(task);  // 清理VFS层缓冲
28. }
29. }
30. }
31. **else**  // 进程名为空，则隐藏该用户所有进程
32. {
33. for\_each\_process(task)
34. {
35. **if** (task->uid == uid)  // uid匹配
36. {
37. task->cloak = opt;  // 设定cloak
38. proc\_flush\_task(task);  // 清理缓冲区
39. }
40. }
41. **if** (opt)  // 输出相应提示信息
42. printk("All processes of user\_%d are hidden.\n", uid);
43. **else**
44. printk("All processes of user\_%d are displayed.\n", uid);
45. }
46. **return** 0;
47. }

* **测试程序EX\_1-2.c**

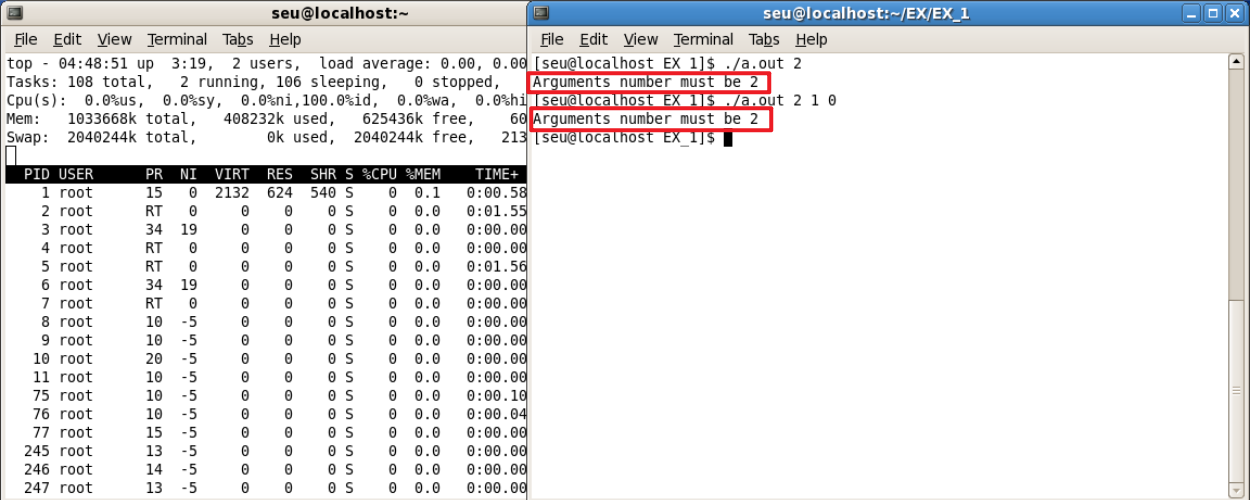
1. #include <stdio.h>
2. #include <sys/syscall.h>
3. #include <unistd.h>
5. **int** main(**int** argc, **char** \*argv[])  // argv[1]: pid, argv[2]: 隐藏或显示
6. {
7. **if** (argc != 3)  // 参数数目错误
8. {
9. printf("Arguments number must be 2\n");
10. **return** -1;
11. }
13. pid\_t pid = atoi(argv[1]);
14. **int** state = atoi(argv[2]);  // 1为隐藏，0为显示
16. **int** ret = syscall(321, pid, state);  // 系统调用
18. **if** (ret)  // ret != 0则出现错误
19. {
20. printf("Error, check the ring buffer to get more information.\n");  // 让用户检查日志缓冲
21. **return** -1;
22. }
24. printf("Successful operation.\n");
25. **return** 0;
26. }

* **测试程序EX\_1-5.c**

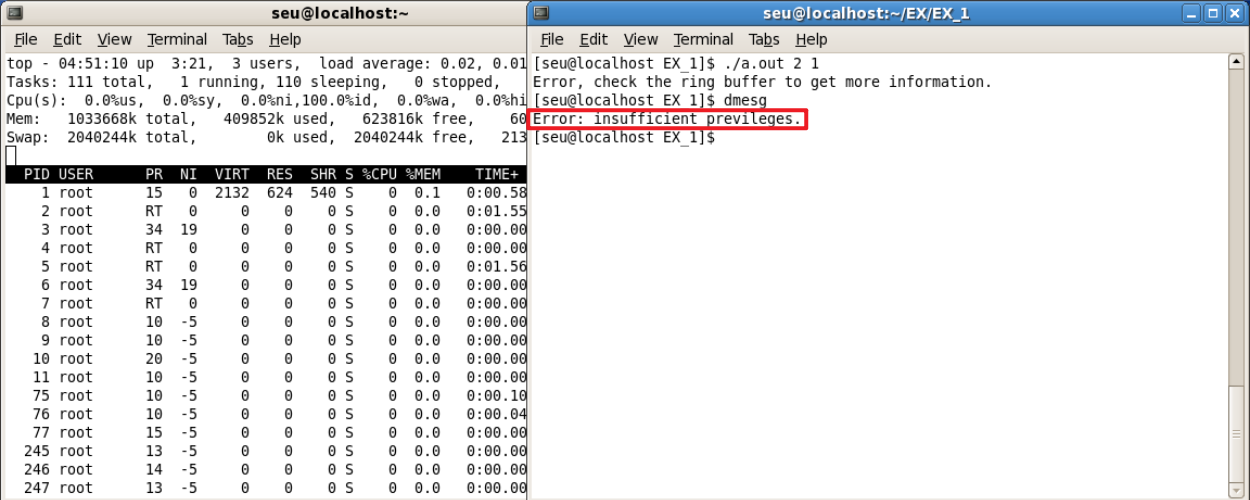
1. #include <stdio.h>
2. #include <sys/syscall.h>
3. #include <unistd.h>
5. **int** main(**void**)
6. {
7. **int** opt = 0;  // 隐藏: 1, 显示y: 0
8. uid\_t uid = 0;  // root: 0, seu: 500
9. **int** appoint = 0;  //  指定进程名: 1，不指定: 0
10. **char** binname[20] = {0};
12. // 输入参数
13. printf("operation(hide: 1, display: 0): ");
14. scanf("%d", &opt);
15. printf("uid: ");
16. scanf("%d", &uid);
17. printf("appoint a binname? (1 or 0): ");
18. scanf("%d", &appoint);
19. **if** (appoint)  // 若指定进程名，接下来用户输入
20. {
21. getchar();  // 吞掉前面的一个换行符
22. printf("binname: ");
23. gets(binname);
24. }
26. // 打印前面用户所输入的参数
27. **if** (opt)
28. printf("\noperation: hide\n");
29. **else**
30. printf("\noperation: display\n");
31. printf("uid: %d\nbinname: %s\n", uid, binname);
33. **int** ret = 0;  // 系统调用返回值
34. **if** (appoint)
35. {
36. ret = syscall(322, opt, uid, binname);
37. }
38. **else**
39. {
40. ret = syscall(322, opt, uid, NULL);  // 若不指定，则直接传入NULL
41. }
43. **if** (ret)  // ret != 0则出现错误
44. {
45. printf("Error, check the ring buffer to get more information.\n");
46. **return** -1;
47. }
49. printf("Successful operation.\n");
50. **return** 0;
51. }
52. **程序运行结果及分析**
53. (2) 测试

编译测试文件EX\_1-2.c，开启新终端，输入top查看系统中当前运行的进程状态。

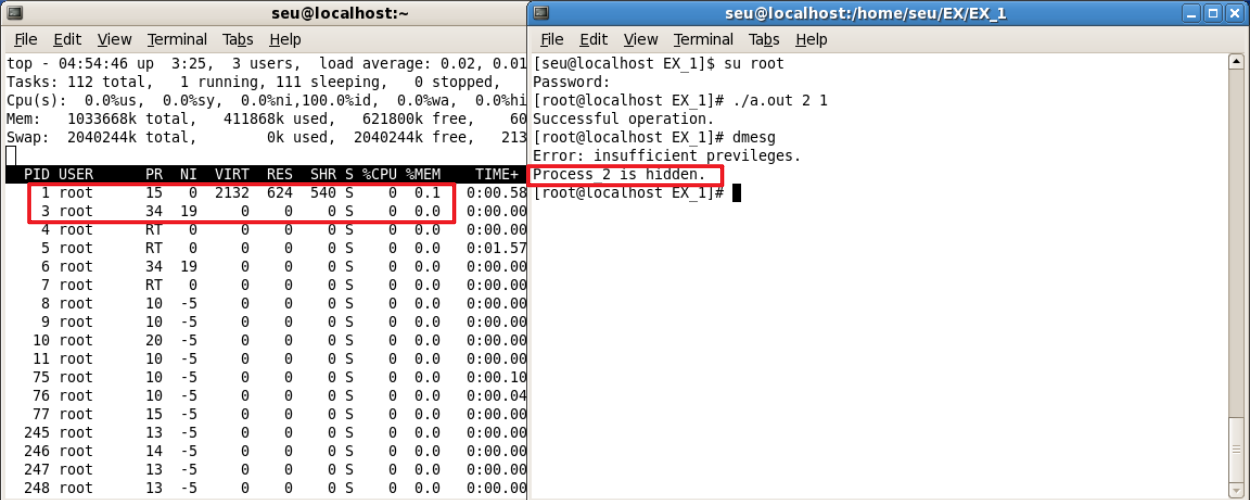
**错误参数数目输入：**终端输入./a.out 2和./a.out 2 1 0均会报错，通过：



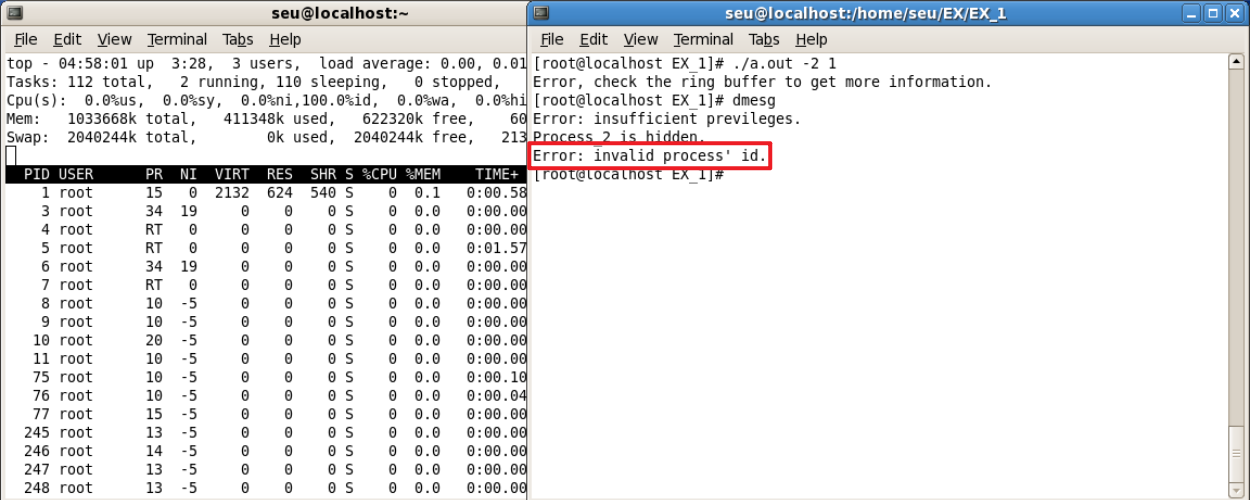
**普通用户执行：**终端输入./a.out 2 1，报错，输入dmesg查看日志缓冲区，提示权限不足，通过：



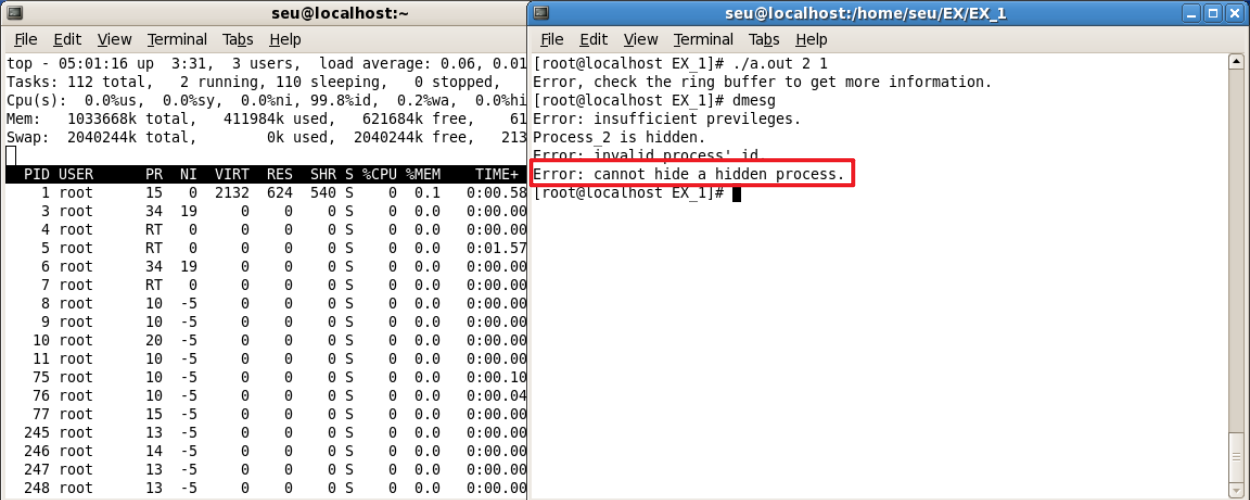
**根用户执行：**终端输入su root切换为根用户，输入./a.out 2 1隐藏进程2，提示操作成功，并在左侧终端中观察到进程2消失，输入dmesg看到提示进程2被隐藏，通过：



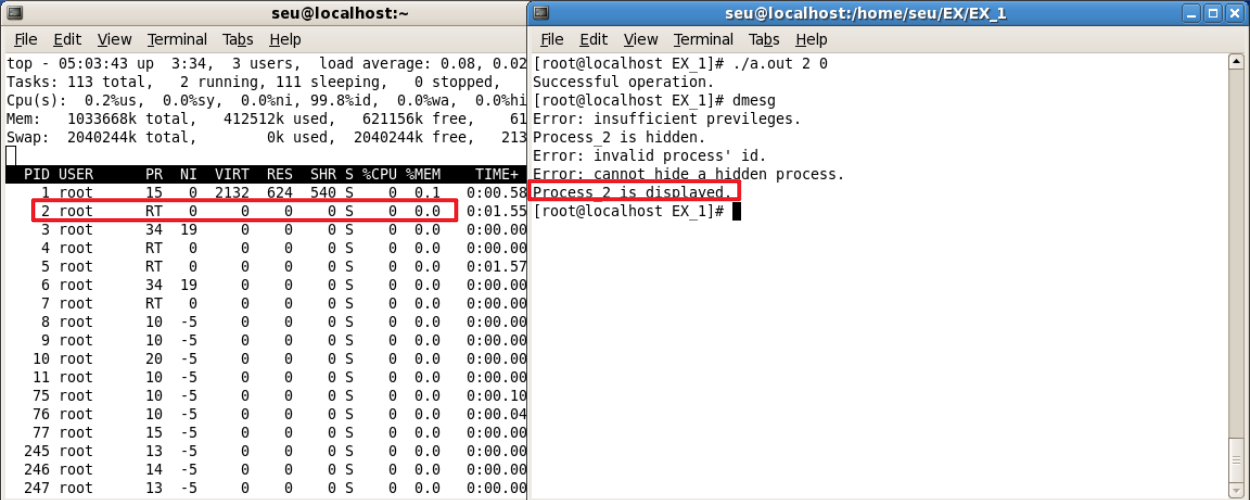
**错误pid输入：**输入./a.out -2 1，报错，输入dmesg看到pid错误的提示，通过：



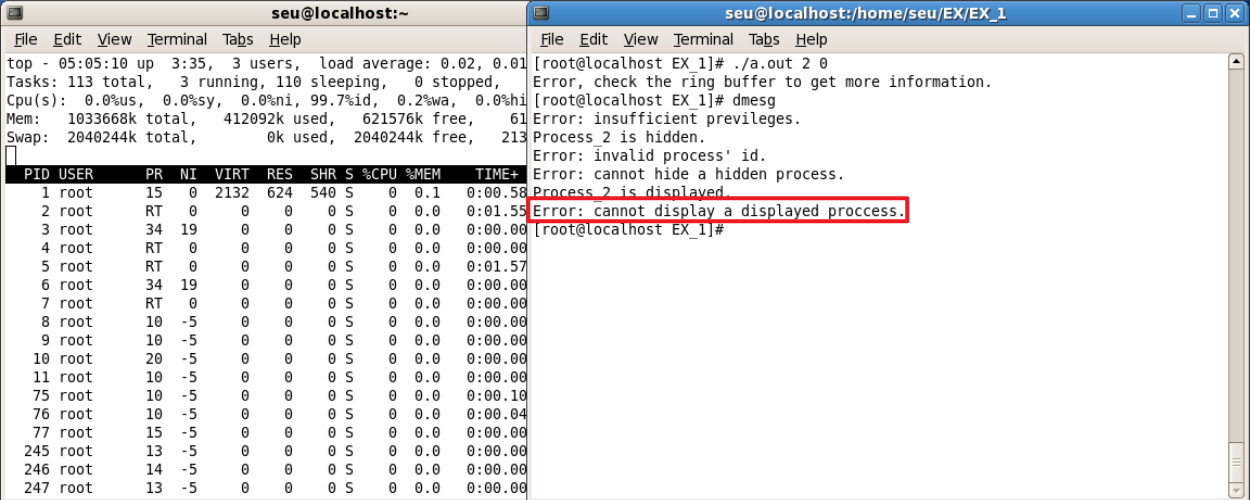
**隐藏已隐藏进程：**输入./a.out 2 1隐藏进程2（进程2此时为隐藏状态），报错，输入dmesg看到错误信息，通过：



**显示隐藏进程：**输入./a.out 2 0，提示成功，并在左侧终端看到进程2出现，输入dmesg看到成功信息，通过：



**显示未隐藏程序：**输入./a.out 2 0，报错，输入dmesg看到错误信息，通过：



测试完成。

1. (5) 测试

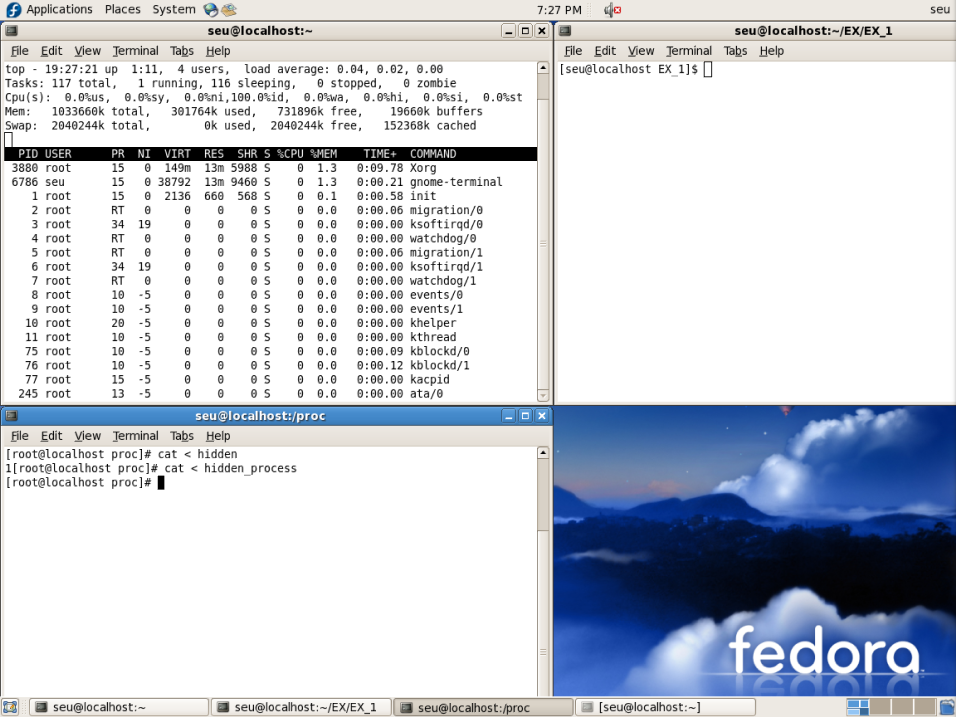
开启终端进入/proc/路径下，输入ls可以看到hidden和hidden\_process两个文件。

输入 cat < hidden查看hidden文件内容，是1：

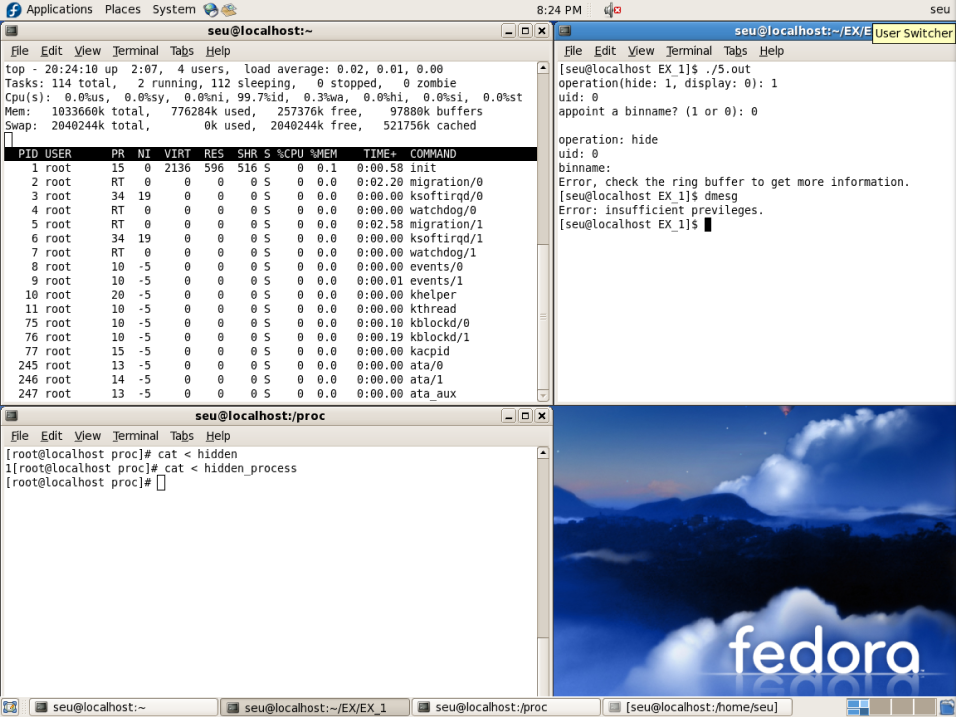


编译测试程序EX\_1-5.c并执行，测试在hidden\_flag=1是能否正常隐藏进程：

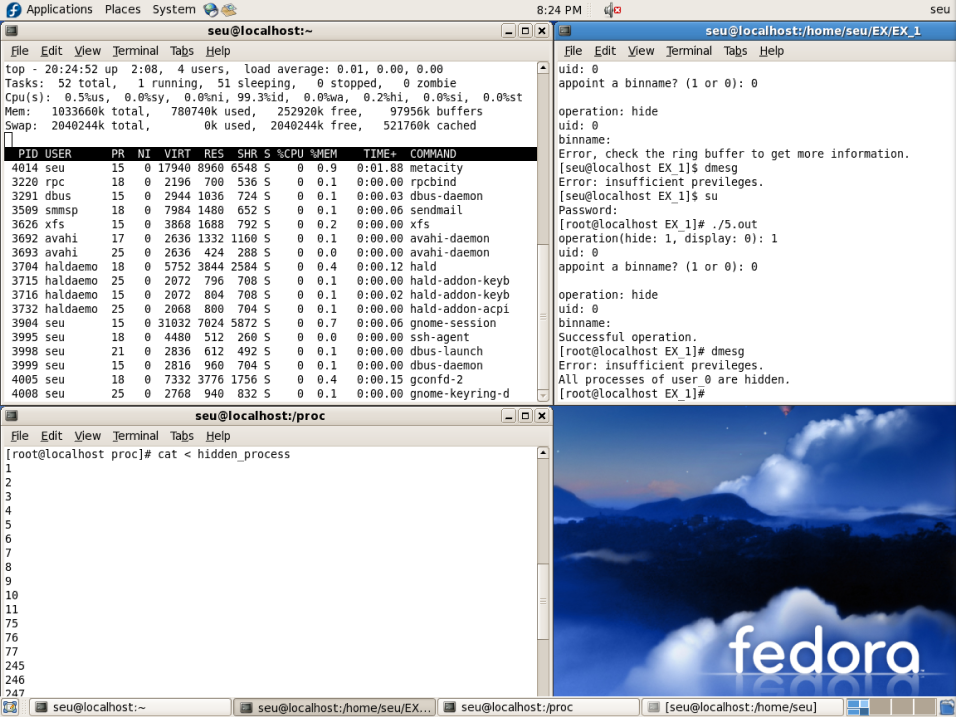
初始，可以看到hidden为1、hidden\_process为空：



隐藏uid为0的用户（root）的所有进程：

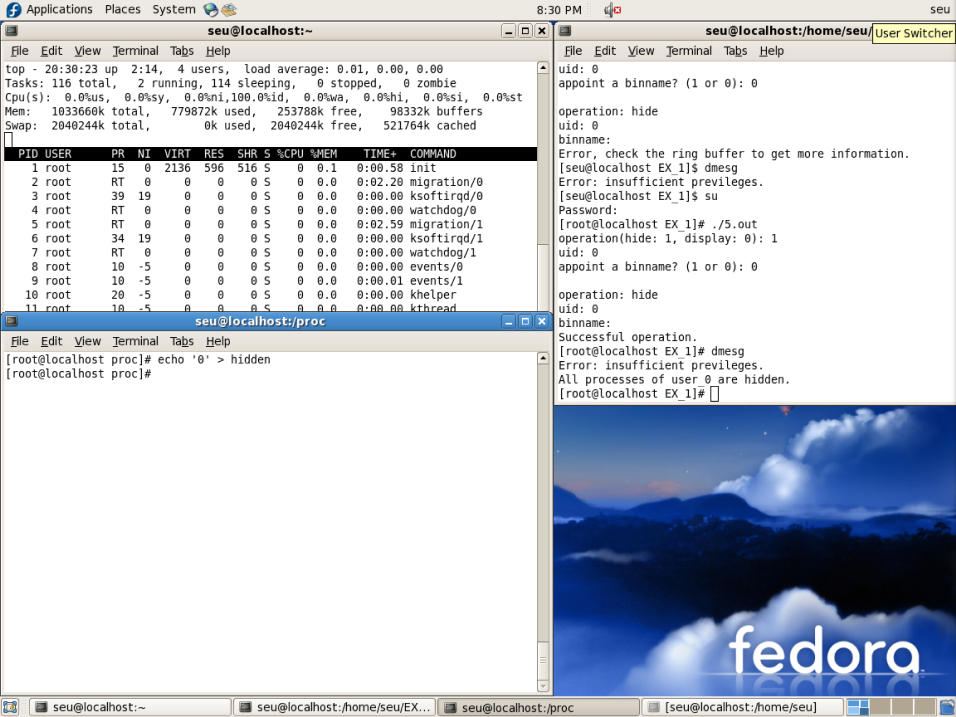


可以看到非根用户会权限不足，不能执行系统调用，切换为根用户再次执行：

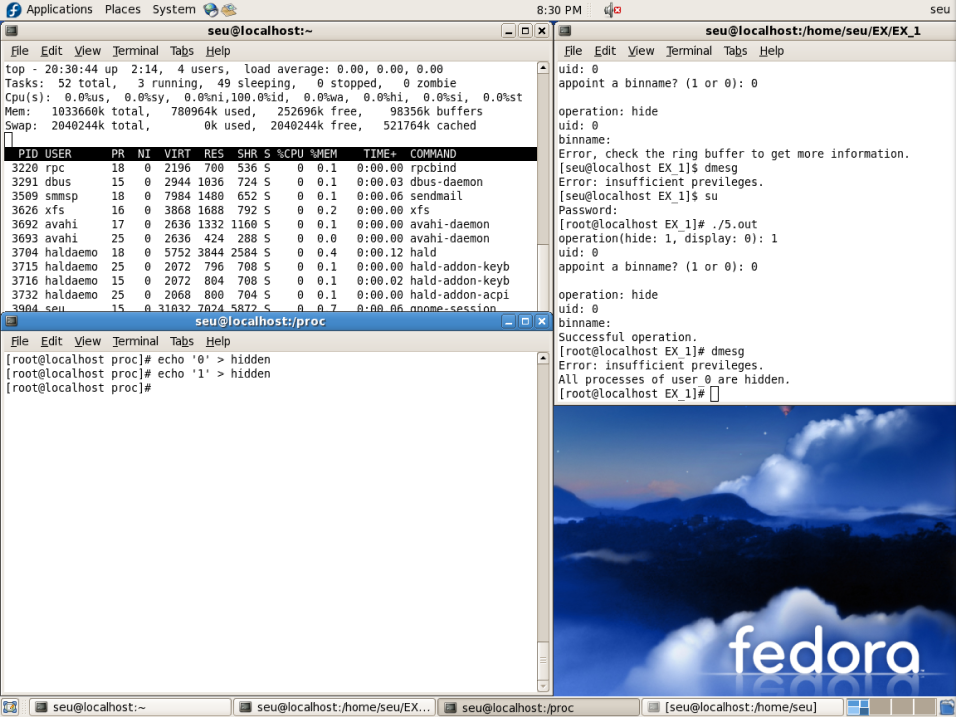


可以看到操作成功，进程列表中的所有根用户的进程被隐藏，输入cat < hidden\_process查看hidden\_process的内容，可以看到这些进程号被写了进来。

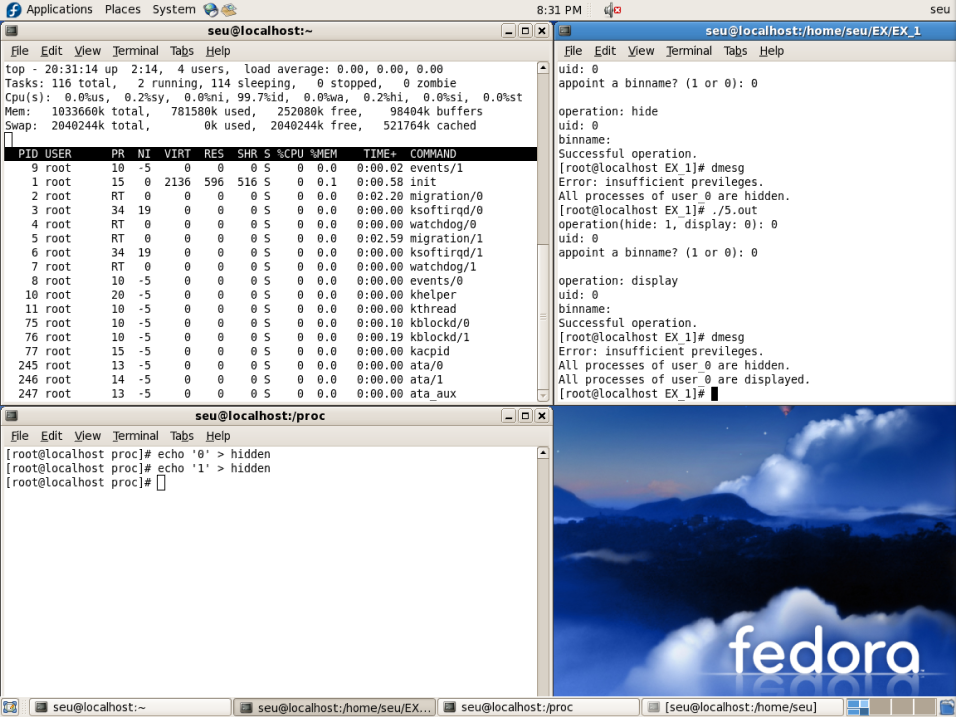
接下来输入echo ‘0’ > hidden，写入0，将hidden\_flag置为0，即不允许隐藏进程，可以看到虽然没有执行显示进程的操作，但root的所有进程都显示了出来：



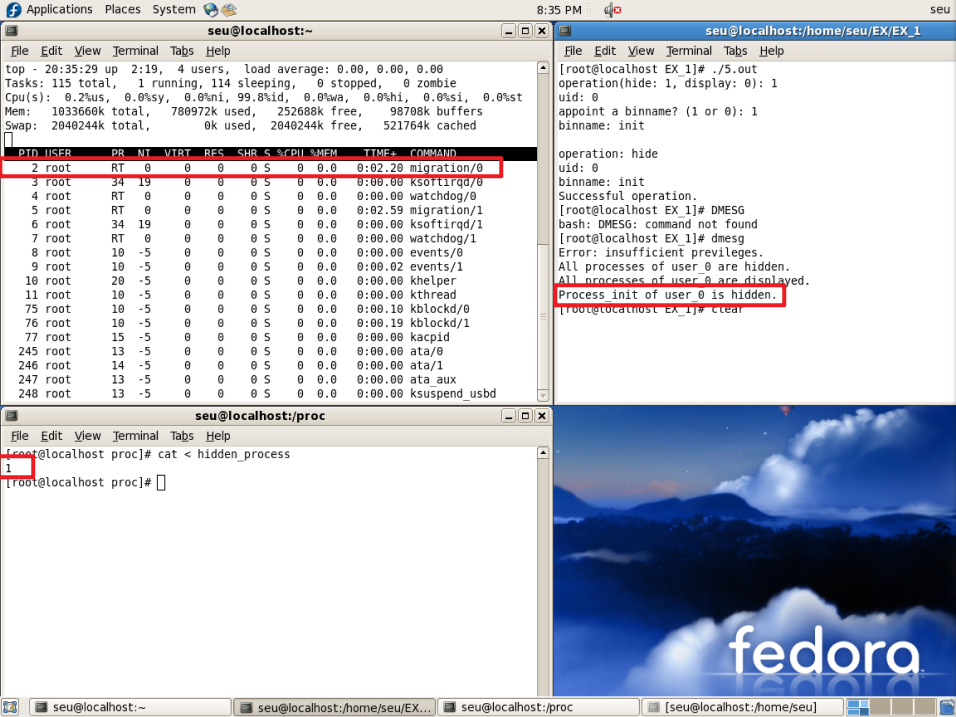
再输入echo ‘1’ > hidden，允许隐藏进程，可以看到root的进程又变为隐藏：



显示uid为0的用户（root）的所有进程：

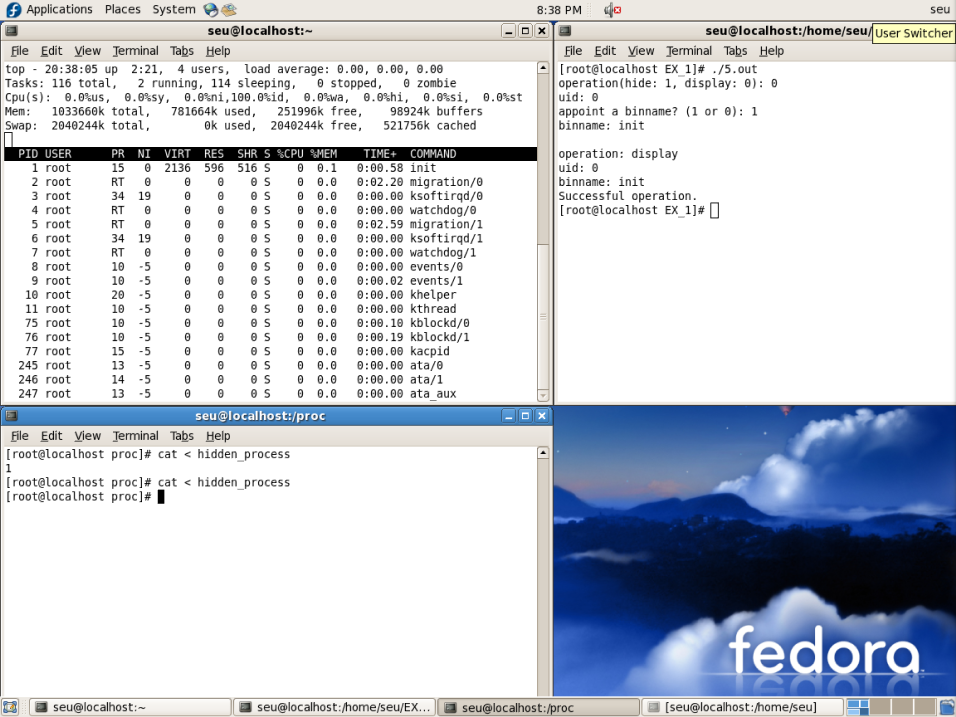


下面隐藏根用户名为init的进程（pid=1）：



可以init看到在进程表中不见了，而且pid 1也写入了hidden\_process文件中。

将init显示出来：



测试完成，各项功能符合要求。

1. **实验体会**
2. 实验中遇到的问题及解决过程

在实验中其实遇到了很多的困难挑战，下面举例来说几个遇到的问题还有是如何解决的。

在一开始实验0时，首先是编译新内核，遇到的错误可以主要分为命令输入错误、权限问题、路径问题，比如一开始sudo一直无法使用，后来找到了su root或su命令来切换到根用户；以及make modules\_install命令中间的下划线一直没有看清，写成了空格；以及对命令功能的理解不到位，比如修改Makefile、修改config的目的都不够清楚，所以只是照着实验手册在做，并不清楚自己所要做的是什么，这样很容易在做的过程中出现问题而且debug也无从下手。

在后面写新的系统调用时，一开始尝试用printf()来输出调试信息，后来发现在内核中不能使用printf()，而是使用printk()来将信息输出到日志缓冲区中并使用dmesg来查看。

在后面需要用到的全局变量定义的时候也遇到了很多问题，中间还不知道什么原因reboot之后不能正常开机，好在可以选择另外一个内核进入系统，而且写过的文件还在。

1. 实验中产生的错误及原因分析

在实验的后半部分，也就是hide\_user\_process()和hidden、hidden\_process文件的读写部分产生的错误较多，下面列举一些。

一开始是在隐藏用户程序系统调用的判断上有较大问题，用到了一个strcmp()函数，最初没有去查找函数的返回值对应信息，自认为是返回1表示两个字符串相等，在这里的判断就出了问题，后面也是一直没有办法正确隐藏某个指定的进程。

另外还遇到了字符串的赋值等问题，从命令行读入字符串需要在定义时写为char str[]的形式，但这样在判断str == NULL时就会始终有问题，后面在测试函数中多加了一个判断，即是否要指定进程名的判断，当不指定时直接向系统调用中传入NULL，避免了这个问题。

到了hidden和hidden\_process文件的读写部分出错就更多了，一开始不知道这样的读写是需要echo和cat来进行的，一直在使用文本编辑器来打开，一直提示冲突，误认为是权限的问题，也走了很多弯路。

另外还有很多语法、逻辑错误，在编程时无法当场发现，到编译时看到报错才知道出现错误，反复修改浪费很多时间。

1. 实验的体会及收获

在这次实验中更深刻的理解了系统内核的结构，也对“模块化”的概念有了更实际的了解，对操作系统的进程调度、管理的了解不再限于课本内容，而是直接操作。也对思路有一些扩展，“隐藏”并不是真正要隐藏，而是“不显示”即可。

1. 对做好今后实验提出建设性建议等。

做好实验需要对所要使用的函数、数据结构有更详细的了解，要知道有哪些成员、函数是能够用到的、用途又是什么。而且扎实的编程基础也是必要的，尽量避免语法、逻辑错误，每次到编译时才发现一些小错误再回去重新修改的效率是相当低的。