###### 4. 系统调用

4.1. 实验目的

建立对系统调用接口的深入认识

掌握系统调用的基本过程

能完成系统调用的全面控制

为后续实验做准备

4.2. 实验内容

此次实验的基本内容是：在 Linux 0.11 上添加两个系统调用，并编写两个简单的应用程序测试它们。

4.2.1. iam()

4.2.2. whoami()

int iam(**const** char \* name);

iam的功能是：将字符串参数 name 的内容拷贝到内核中保存下来。 要求 name 的长度不能超过 23 个字符。返回值是拷贝的字符数。 如果 name 的字符个数超过了 23 ，则返回 -1 ，并置 errno 为 EINVAL。

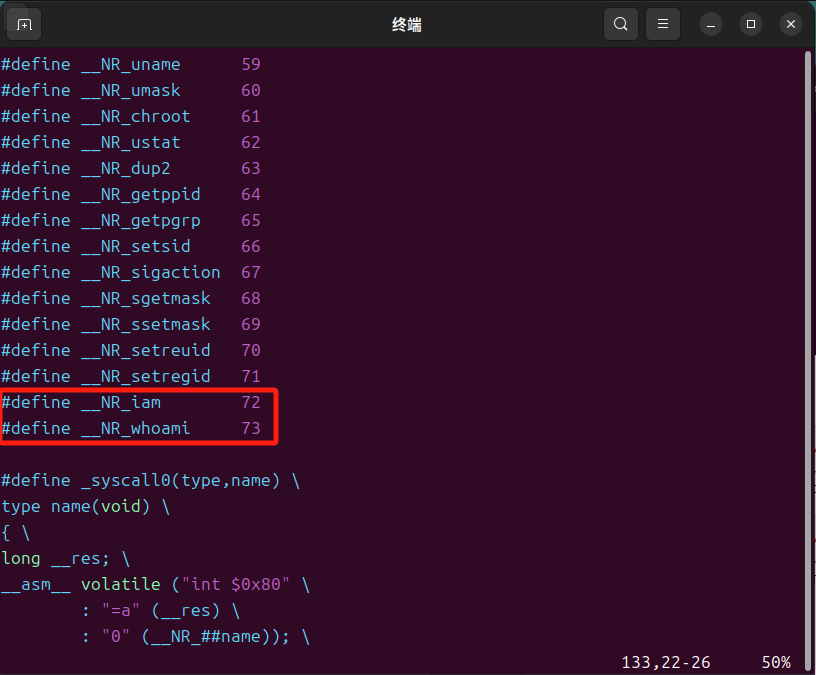
int whoami(char\* name, unsigned int size);

whoami的功能是：将内核中由 iam() 保存的名字拷贝到 name 指向的用户地址空间中， 同时确保不会对 name 越界访存（ name 的大小由 size 说明）。 返回值是拷贝的字符数。如果 size 小于需要的空间，则返回 -1 ，并置 errno 为 EINVAL 。

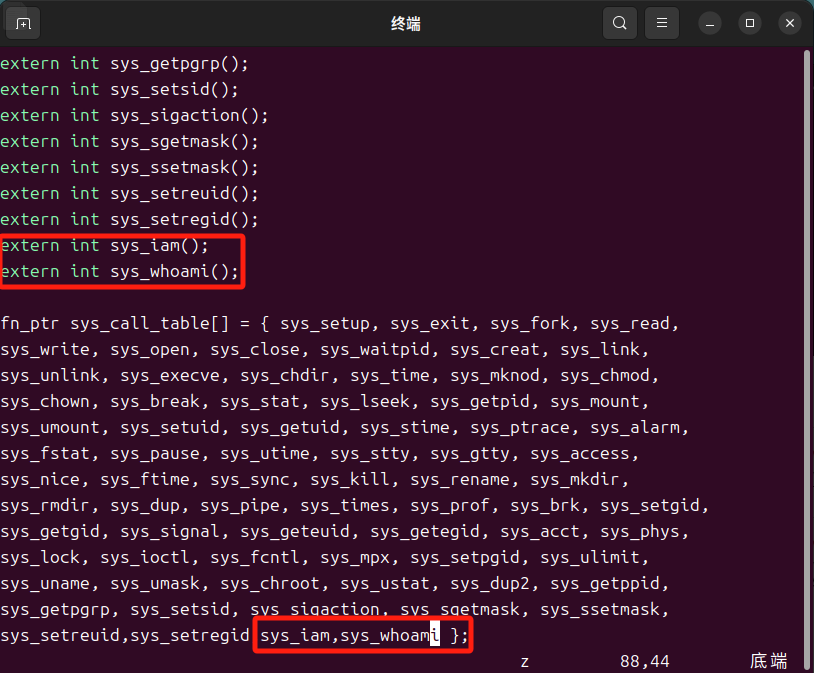
因为具体操作文件重复度高，故一起实现：

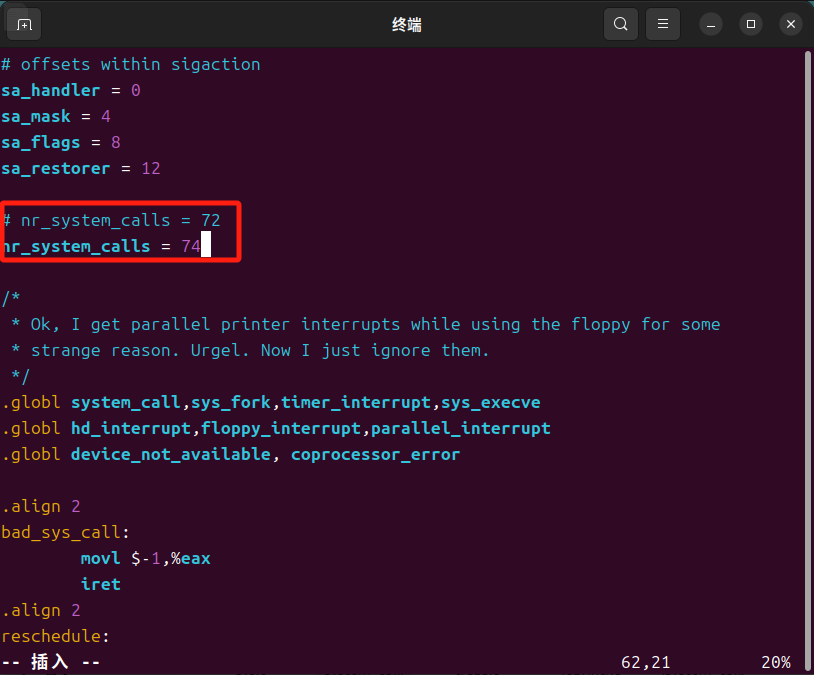
实现方式：

首先打开unistd.h 添加iam（为了方便把whoami也提前加上了）的定义，也就是添加一个能够被指令调用的宏定义，详见下图红标部分：

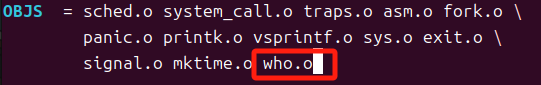


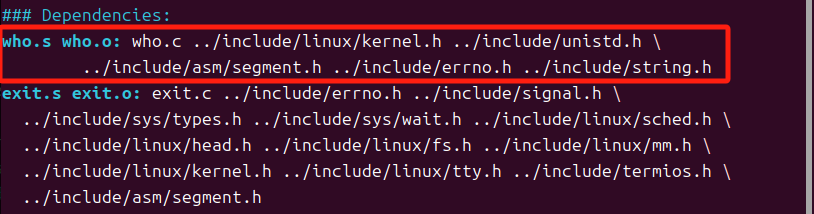
之后，在子目录的sys.h中也加上这两条指令，并且，将../kernel/system\_call.s中的nr\_system\_calls加2，具体操作实现如下图：



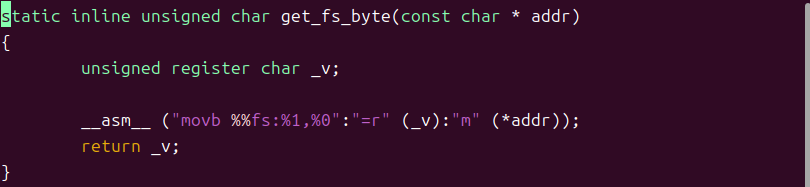


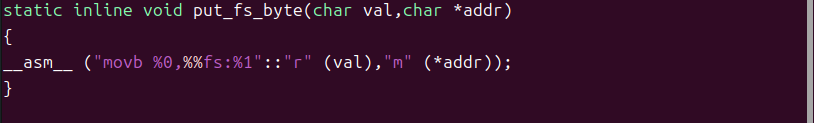
然后，修改./kernel目录下的Makefile，我要在该目录创建的文件名是who.c，所以修改如下：





之后，在./kernel目录下创建who.c文件，用于实现whoami和im之间的功能，虽然这两个功能的目的都是将root的名称返回，但是为了实现用户态和内核态之间的数据传输，两者都使用了字符串指针作为参数调用，需要用到include/asm/segment.h中get\_fs\_byte()(获得一个字节的用户空间中的数据)函数和put\_fs\_byte()(将一个字节的数据传到用户空间)，两个函数的具体结构如下图：

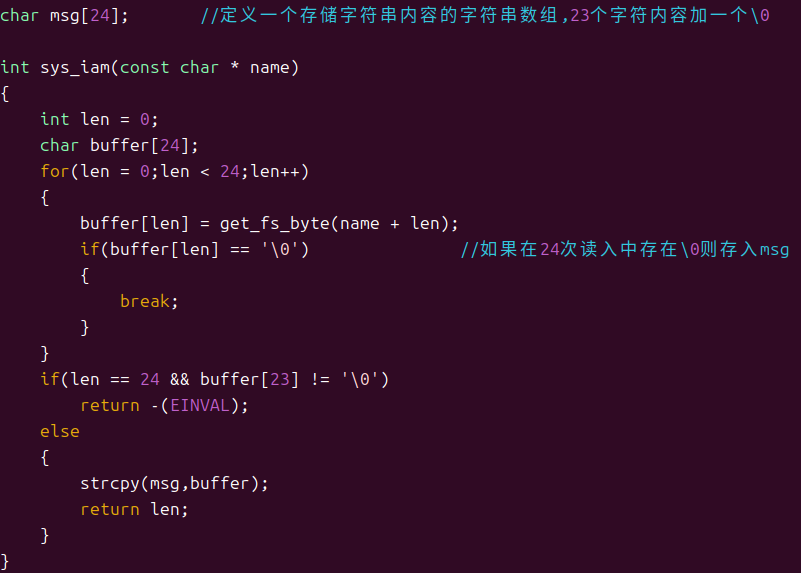


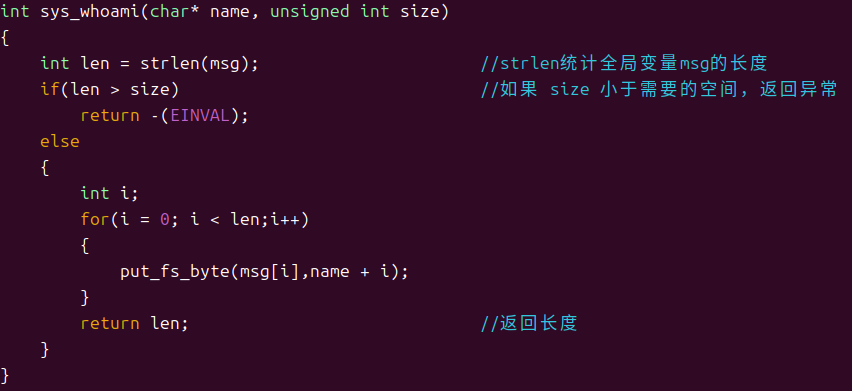


同时，观察到函数需要errno作为失败返回值，所以也要调用include目录下的errno.h。

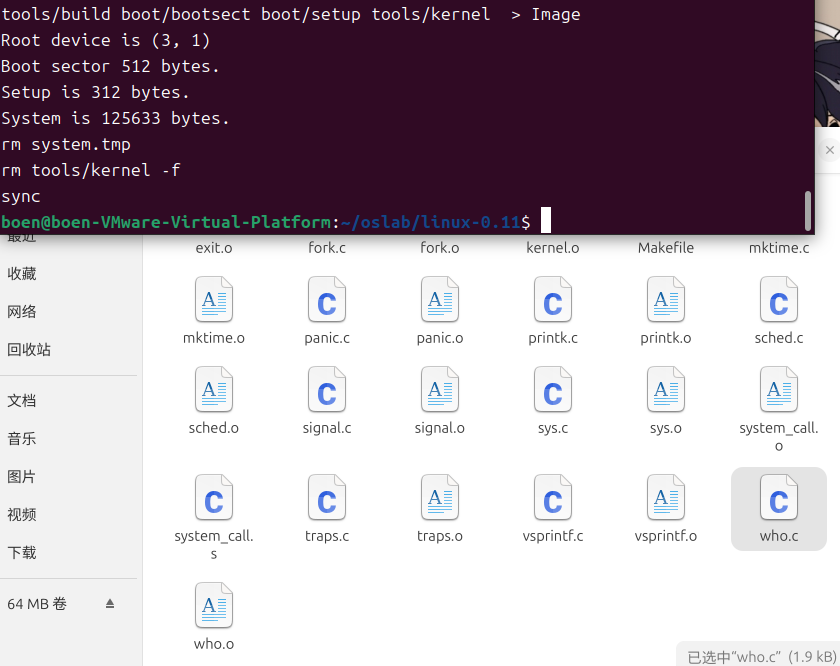
然后，实现对who.c的编写，实际上是在win上面写的，cv到了虚拟机中，具体内容：





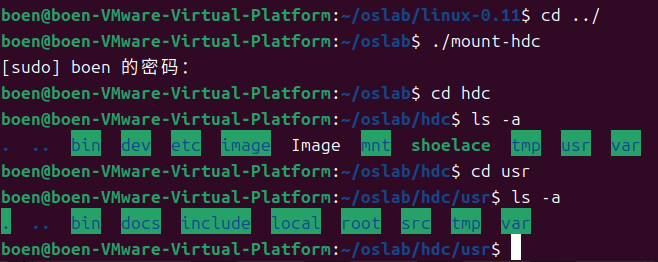


然后，在该目录下make，生成.o文件，之后在../(也就是0.11目录）下make，(或者直接make all）之后进行下一步的任务,make结果如下:



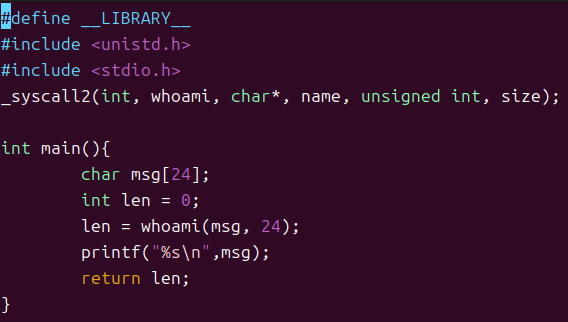
4.2.3. 测试程序

首先，在oslab目录下执行./mount-hdc将生成的镜像文件移动到hdc目录下，然后对hdc目录进行检查：

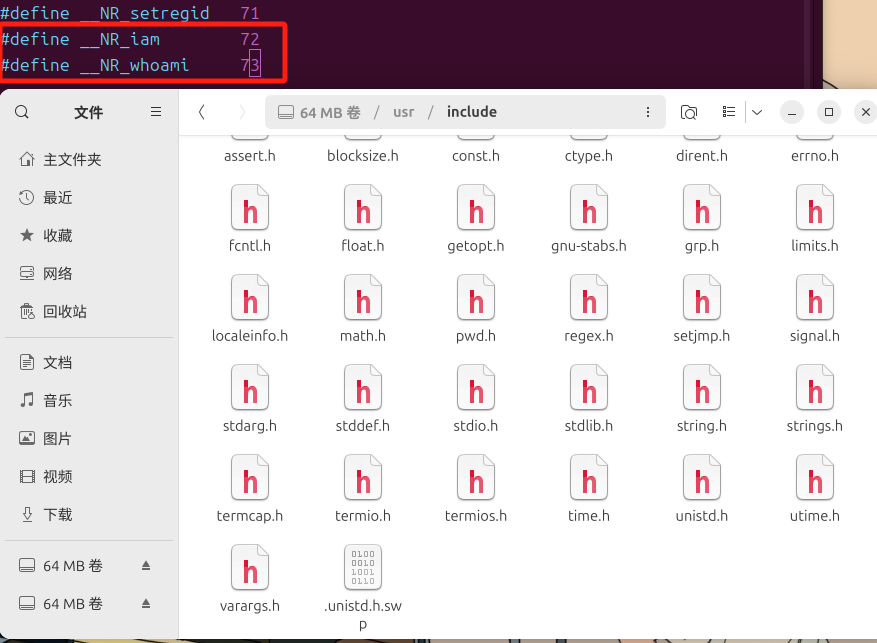


之后，编写iam.c whoami.c，将其与testlab2.c testlab2.sh移动到 hdc/usr/root目录下，两个文件分别内容如下：

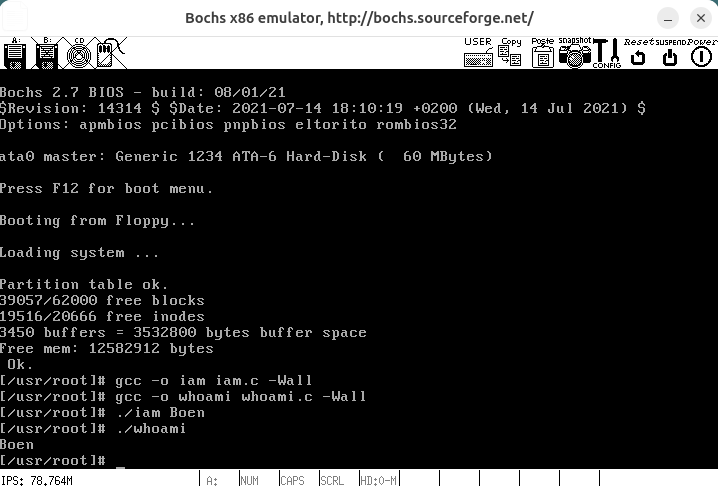


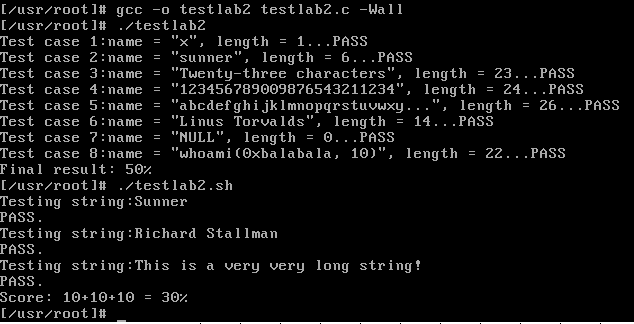


然后，将../usr/include中的unistd.h进行修改，方法于上面相同：



之后通过bochs运行并评分，具体流程以及评分结果如下：





如图，全部满昏！

4.3. 实验报告

在实验报告中回答如下问题：

从 Linux 0.11 现在的机制看，它的系统调用最多能传递几个参数？

    Linux中（Linux-0.11/inlcude/unistd.h）有四个关于系统调用的方法：\_syscall0、\_syscall1、\_syscall2、\_syscall03。

    其中#define \_syscall3(type,name,atype,a,btype,b,ctype,c)包含三个参数。系统调用号存放在 eax 寄存器，而各个参数则分别存放在 ebx、ecx 和 edx 寄存器中。通过执行 `int $0x80` 汇编指令触发系统调用。

    因此Linux 0.11中最多支持系统调用传递3个参数。

你能想出办法来扩大这个限制吗？

 在硬件上，可以多设置几个参与传递参数的寄存器供操作系统使用，以扩展\_syscall的范围，或者，可以利用堆栈的方式将所需要的参数压入栈中，通过用户栈和内核栈的切换来获取更多参数，

用文字简要描述向 Linux 0.11 添加一个系统调用 foo() 的步骤。

与上面添加whoami和iam的操作相同，只是名称不同：

首先，在linux-0.11/include/unistd.h中：

        添加系统调用号宏定义'#define \_\_NR\_foo 调用号+1

    然后在./linux/sys.h中：

        1.添加extern int sys\_foo();

        2.在fn\_ptr sys\_call\_table[]中加入sys\_foo

    接着在../kernel/system\_call.s中：

        修改nr\_system\_calls的值，使其加一

    同时，在linux-0.11/kernel中：

        创建foo.c，并在其中实现供用户调用的系统调用函数 sys\_foo()

    在/kernel/Makefile中：

        1.OBJS中加入foo.o

        2.Dependencie下添加foo.s foo.o:foo.c + 所需要的头文件

    最后在用户程序中选择相应的参数调用方式\_syscallN（N是参数个数，回答第一问提及）完成用户程序foo()的调用

4.3.1. 评分标准

将 testlab2.c 在修改过的 Linux 0.11 上编译运行，显示的结果即内核程序的得分。满分50%

只要至少一个新增的系统调用被成功调用，并且能和用户空间交换参数，可得满分

将脚本 testlab2.sh 在修改过的 Linux 0.11 上运行，显示的结果即应用程序的得分。满分30%

实验报告，20%