

Assignment 1

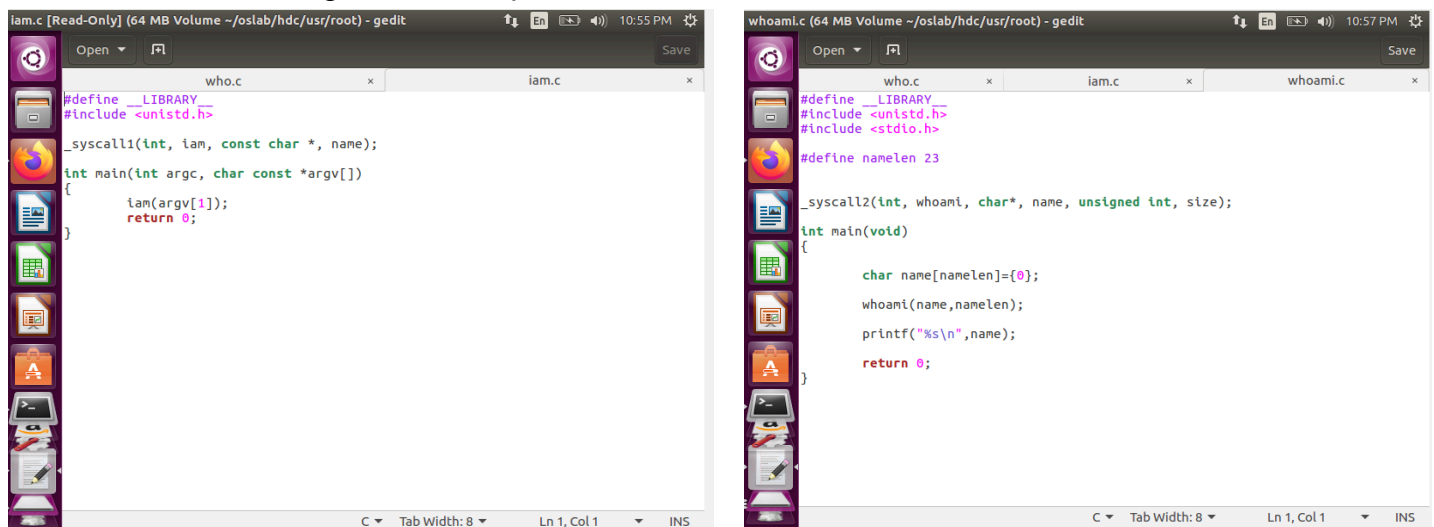
Group members: Johnathan Otte & Sam Decker

Goal

This lab aimed to understand how system calls are called, implemented, and interact with other parts of an operating system. A system call provides the kernel's services to user programs through APIs. First, we needed to modify some files, such as `unistd.h`, `system_calls.h`, and other files to add each system call, `iam` and `whoami`, so they are recognized. Each of the system calls will be called from `who.c` as well as have their own APIs.

Implementation Details

The first thing that was implemented was the `iam.c` and `whoami.c`.



```
iam.c [Read-Only] (64 MB Volume ~/oslab/hdc/usr/root) - gedit
who.c x iam.c x
#define _LIBRARY_
#include <unistd.h>

_syscall1(int, iam, const char *, name);

int main(int argc, char const *argv[])
{
    iam(argv[1]);
    return 0;
}

C Tab Width: 8 Ln 1, Col 1 INS

whoami.c (64 MB Volume ~/oslab/hdc/usr/root) - gedit
who.c x iam.c x whoami.c x
#define _LIBRARY_
#include <unistd.h>
#include <stdio.h>

#define namelen 23

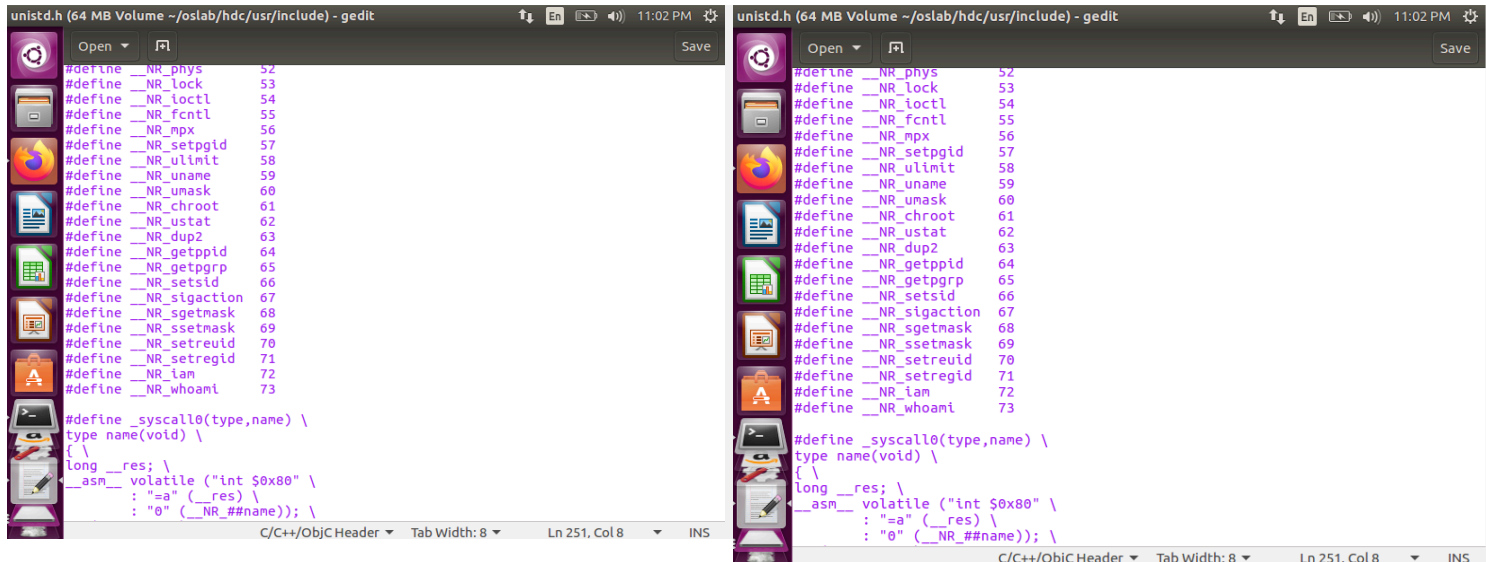
_syscall2(int, whoami, char*, name, unsigned int, size);

int main(void)
{
    char name[namelen]={0};
    whoami(name,namelen);
    printf("%s\n",name);
    return 0;
}

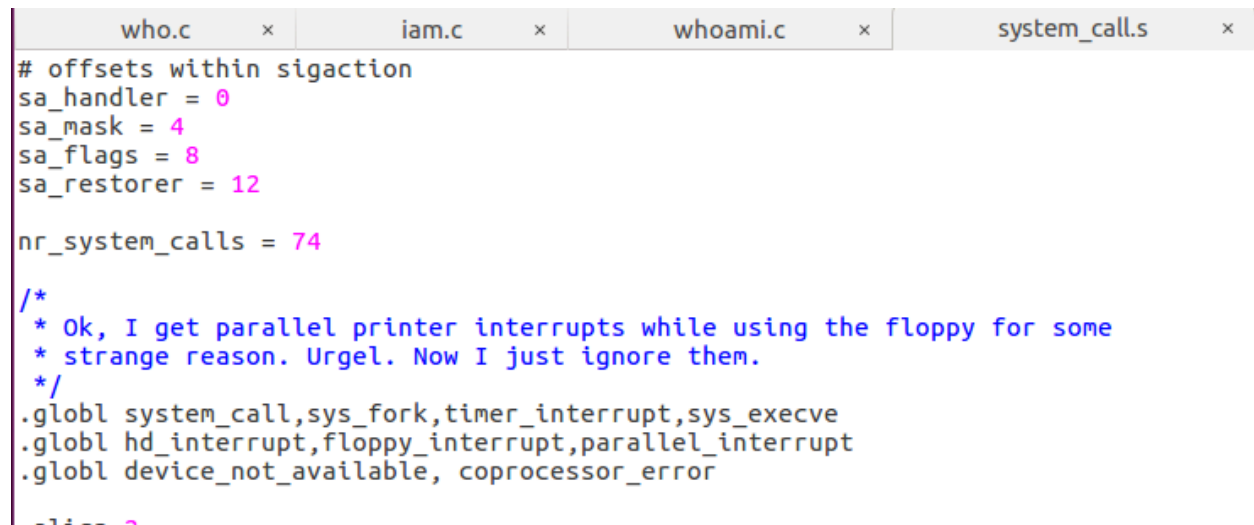
C Tab Width: 8 Ln 1, Col 1 INS
```

These are the APIs of the two system calls, `whoami` and `iam`. They both contain macros within them that will interact with x80 interrupt that'll allow for the kernel to handle the interrupt as well as place the sys call number into the EAX registers along with all the other parameters for the call.

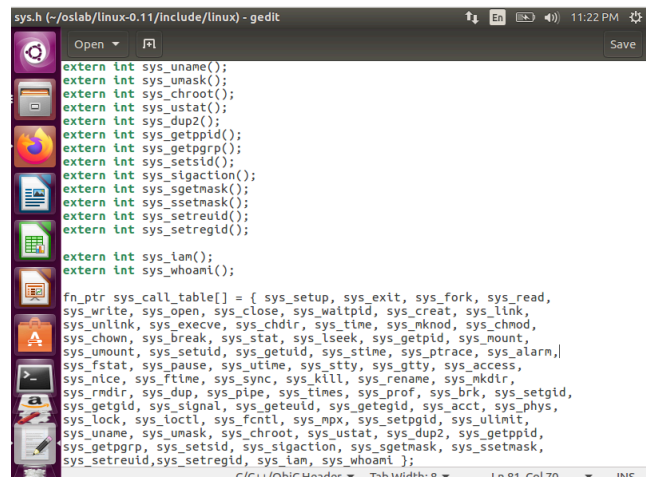
Next, we needed to add the system calls into the `unistd.h` files in the `hdc` and `linux-0.11` directories. Each system call needs a system call number so this is the file where you can add any future system calls. This number is what will be stored into the EAX register.



We had to add to the total system call count within the file system_calls.s in the kernel. This file makes the other functions able to see the system calls due to the .global.



The sys.h file was also modified to add sys_iam and sys_whoami into the sys call table. It uses this array to get the system calls.



The makefile was modified within the Linux-0.11 directory in the kernel in order for our who.c file to be recognized and compiled. This is where you can run the make all in Linux-0.11 to compile your changes with the rest of the Linux code.

```

Makefile (~/.oslab/linux-0.11/kernel) - gedit
Open Save

OBJS = sched.o system_call.o traps.o asm.o fork.o \
      panic.o printk.o vsprintf.o sys.o exit.o \
      signal.o mktime.o who.o

kernel.o: $(OBJS)
$(LD) -m elf_i386 -r -o kernel.o $(OBJS)
sync

clean:
rm -f core *.o *.a tmp_make keyboard.s
for i in *.c;do rm -f `basename $$i .c`.s;done
(cd chr_drv; make clean)
(cd blk_drv; make clean)
(cd math; make clean)

dep:
sed '/\#\#\# Dependencies/q' < Makefile > tmp_make
(for i in *.c;do echo -n `echo $$i | sed 's,\.c,\.s,` " "; \
  $(CPP) -M $$i;done) >> tmp_make
cp tmp_make Makefile
(cd chr_drv; make dep)
(cd blk_drv; make dep)

### Dependencies:
who.s who.o: who.c ../include/linux/kernel.h ../include/unistd.h |
exit.s exit.o: exit.c ../include/errno.h ../include/signal.h \
../include/sys/types.h ../include/sys/wait.h ../include/linux/sched.h \
../include/linux/head.h ../include/linux/fs.h ../include/linux/mm.h \

```

We will look into the who.c file to see how the system calls are called within the file. It is important to remember how in the iam.c, it can take input from the user straight from the command line, and that is assigned to the name variable. Then the sys_iam gets the name from user until it reaches null terminator. Then it copies to the kernel. In sys_whoami it takes gets name and size and copies name from kernel to user. Last it returns the name to user.

```

sys.h
who.c

#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <asm/segment.h>

#define maxlen 23 //22 char and null, don't have to worry about scopes
char kernelname[maxlen]; //len of max

int sys_iam(const char * name)
{
    char tempname[maxlen] = {0};
    int i;
    for (i = 0; i < maxlen; i++)
    {
        char c = get_fs_byte(name + i); //reading the name from user and# ofbytes
        if (c == '\0') //stops after null term
        {
            break;
        }
        tempname[i] = c;
    }

    if (tempname[maxlen-1] != '\0') //any char pass 23 will be null
    {
        return -EINVAL;
    }

    strncpy(kernelname, tempname, maxlen); //copy name in kernel
    return i; //returning size
}

int sys_whoami(char* name, unsigned int size)

```

```

int sys_whoami(char* name, unsigned int size)
{
    int lenneeded = strlen(kernelname) + 1;
    int i;
    for (i = 0; i < lenneeded; i++)
    {
        put_fs_byte(kernelname[i], name + i); //gets name from kernel
    }

    return lenneeded-1;
}

```

The image shows two side-by-side screenshots of the Bochs x86-64 emulator. Both windows show the BIOS boot process, including messages like 'Bochs BIOS - build: 02/13/08', 'Revision: 1.194', and 'Options: apmbios pcibios eltorito rombios32'. The left window shows the user running the command 'iam john' in a shell prompt, which results in 'ENOENT' (No such file or directory). The right window shows the user running the command 'whoami' in a shell prompt, which results in 'john' being displayed. Both windows also show the system's memory and disk status at the bottom.

So we have two APIs in usr/root in bochs. iam takes the input from user and puts it into a pointer, which can be accessed in sys_iam in who.c. whoami will just display the name given and make sure it is the correct length through sys_whoami.

Challenges

One of our first struggles was just knowing what directory we needed to change and learning how system calls interact with the rest of the operating system. Specific spots are the difference of hdc and linux-0.11, but with practice we became familiar with the differences between the two.

We struggled with defining the max length of the name within the who.c. At first we were trying to define it as a regular int maxlen=23, but we figured there was a problem with the scopic because when compiling with bochs it couldn't detect the variable. In class we went over a #define method to set a constant and that seemed to fix the problem.

Taking arguments from the command line was difficult to figure out how to do, but Dr. McKinney told us we needed to look up how to do it and that we did not learn it in Machine Organization. The stackoverflow was pretty straight forward. It was difficult to understand what went into the APIs, we thought it was sys call marcos.

The implementation of `get_fs_byte` and `put_fs_byte` was hard to understand at first on how to exactly use it.

Throughout the whole assignment we used your pdf you gave use "A Heavily Commented Linux Kernel Source Code." to used the whole calling process.

Citations

CybergibbonsCybergibbons 5, et al. "Is It Better to Use `#define` or `Const Int` for Constants?" *Arduino Stack Exchange*, 1 Feb. 1960, arduino.stackexchange.com/questions/506/is-it-better-to-use-define-or-const-int-for-constants.

MelkonMelkon 3711 gold badge22 silver badges1010 bronze badges, et al. "Input from the Execution Line in the Terminal in C." *Stack Overflow*, 1 Nov. 1957, stackoverflow.com/questions/8324066/input-from-the-execution-line-in-the-terminal-in-c.