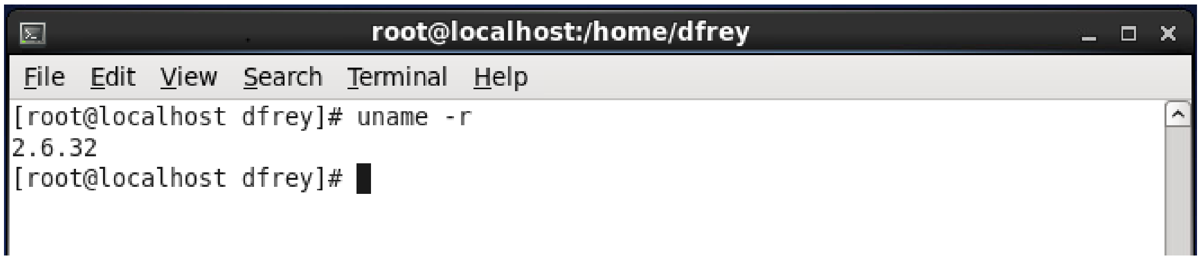
CS 4500 Project 1 – System Calls and Processes

1-1. **Project Members**: Daniel Frey, Justin Hale
We have neither given nor received unauthorized assistance on this work.

1-2. **VM**: jhale-virtual-machine
instructor Account Password: zhuang

1-3.

Task 0: This task was rather straight forward. The main issue with this part was getting the VM to boot into this version, however, the note on the assignment sheet initially helped. To not have to remember to select this every time, the default option was changed to make the VM boot into 2.6.32. Instead of typing out the instructions each time to compile the kernel, we made a script that executed these instructions. Furthermore, we made an alias so that we could simply type 'recompile' to compile the kernel.
Below is a screenshot to show the new version of the kernel was successful.


root@localhost:/home/dfrey
File Edit View Search Terminal Help

```
[root@localhost dfrey]# uname -r
2.6.32
[root@localhost dfrey]# 
```

Task 1: This task, too, was straight forward. It was mainly following the assignment sheet to get this set up. This task allowed us to get familiar with creating and implementing system calls for the following tasks. No real issues were involved with this task other than accidentally mistyping something.
*To test this system call, use *./test_syscall* in the *instructor* directory.

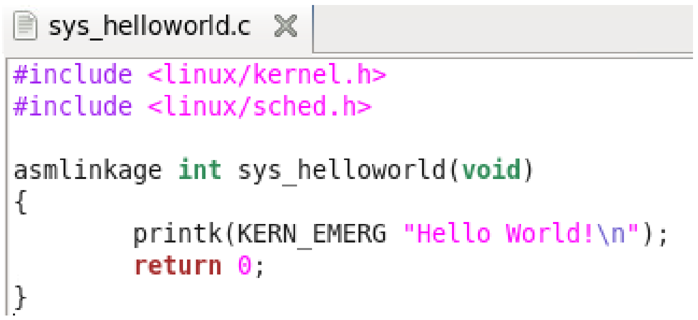
Task 2: To solve this problem we decided that we must use *current* as indicated in the assignment sheet. After learning how to use current, printing out the name and pid were simply done by accessing those elements of the struct. For printing out the same information for parent processes until init, we created a for loop that iterated over each parent process to display the necessary information. We learned how simple it was to loop over processes.
*****To test this system call, use *./test_print_self* in the *instructor* directory.

Task 3: To solve this problem we looked up the Linux functions on how to get the task_struct of a specific pid. To do this, we used the functions find_task() and find_vpid(). The latter finds the process, and the former gets the task_struct for that process. Together with these we were able to output the necessary information. To make sure that we had no issues with invalid pids, we checked that the task_struct was not null for a given process. From this task we learned how to efficiently get the task_struct of a running process.

The test program for Task 3 required more than the previous ones. The test program uses the command *pgrep bash* to get the arbitrary pid. The output from that command is obtained, converted to an integer, and then passed to the system call.
*To use this system call, use *./test_print_other* in the *instructor* directory.

2-1. **Source code for Task 1**

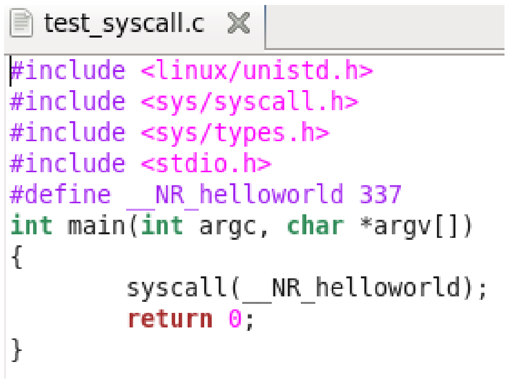
Kernel:



sys_helloworld.c

```c
#include <linux/kernel.h>
#include <linux/sched.h>

asmlinkage int sys_helloworld(void)
{
        printk(KERN_EMERG "Hello World!\n");
        return 0;
}
```

Test:

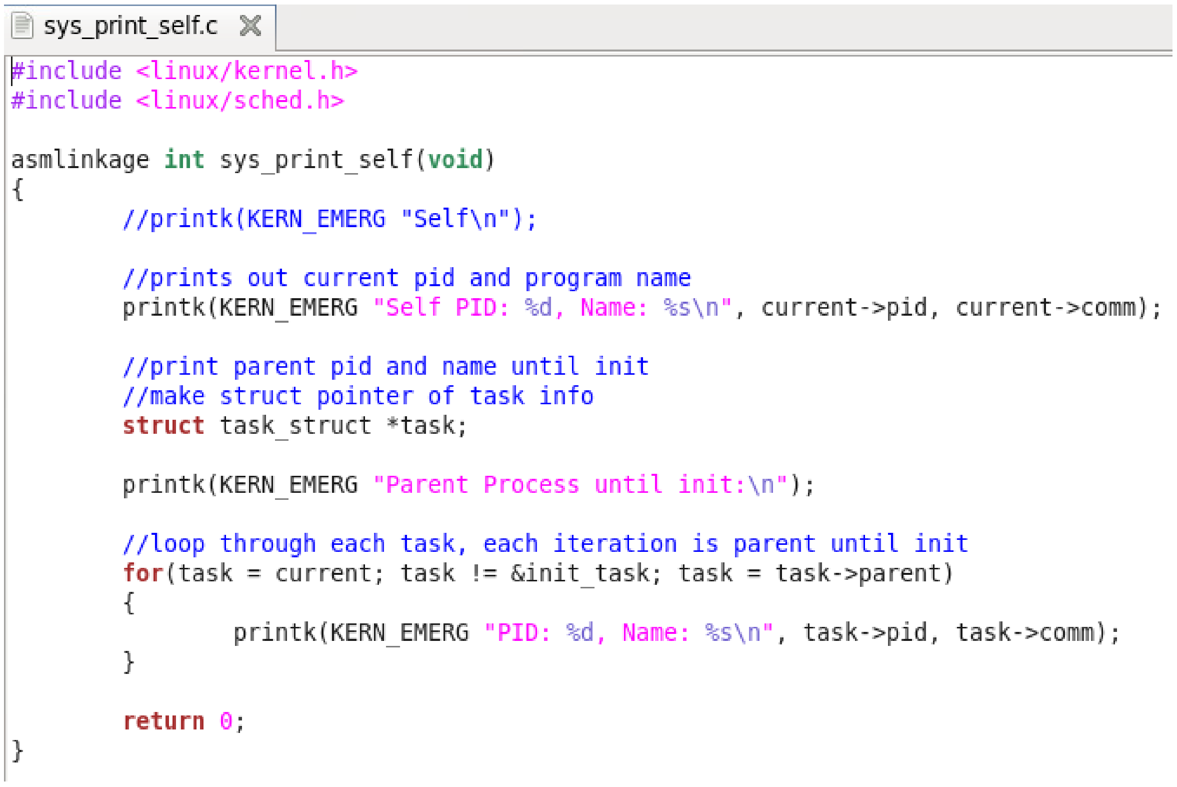


test_syscall.c

```c
#include <linux/unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <stdio.h>
#define __NR_helloworld 337
int main(int argc, char *argv[])
{
        syscall(__NR_helloworld);
        return 0;
}
```

2-2. **Source code for Task 2**

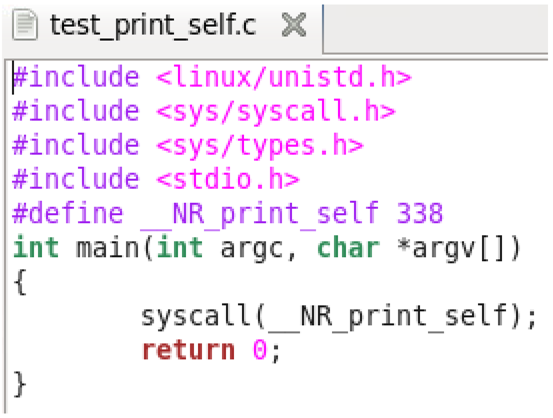
Kernel:



sys_print_self.c

```c
#include <linux/kernel.h>
#include <linux/sched.h>

asmlinkage int sys_print_self(void)
{
        //printk(KERN_EMERG "Self\n");

        //prints out current pid and program name
        printk(KERN_EMERG "Self PID: %d, Name: %s\n", current->pid, current->comm);

        //print parent pid and name until init
        //make struct pointer of task info
        struct task_struct *task;

        printk(KERN_EMERG "Parent Process until init:\n");

        //loop through each task, each iteration is parent until init
        for(task = current; task != &init_task; task = task->parent)
        {
                printk(KERN_EMERG "PID: %d, Name: %s\n", task->pid, task->comm);
        }

        return 0;
}
```

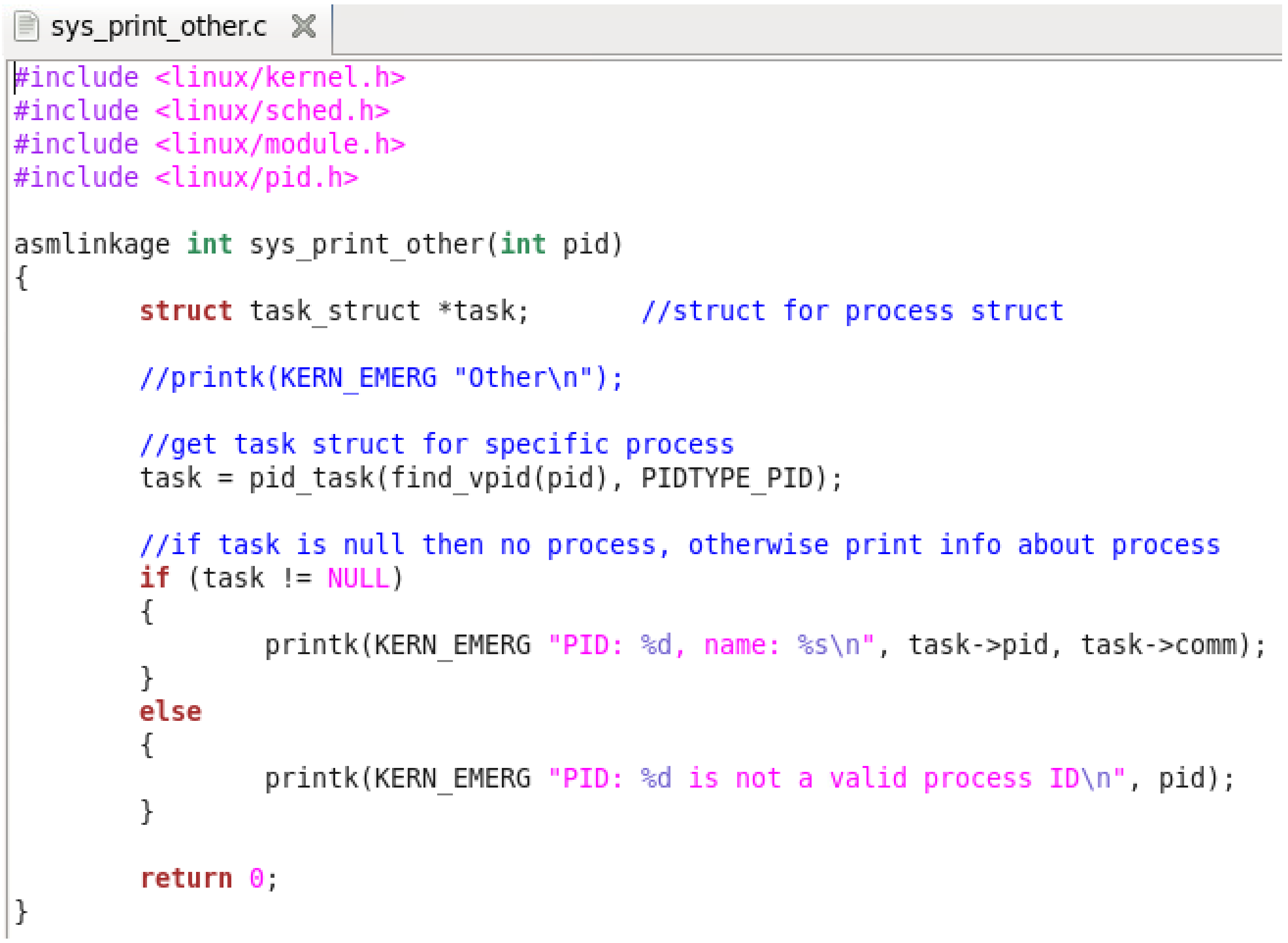
Test:



```c
#include <linux/unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <stdio.h>
#define __NR_print_self 338
int main(int argc, char *argv[])
{
        syscall(__NR_print_self);
        return 0;
}
```

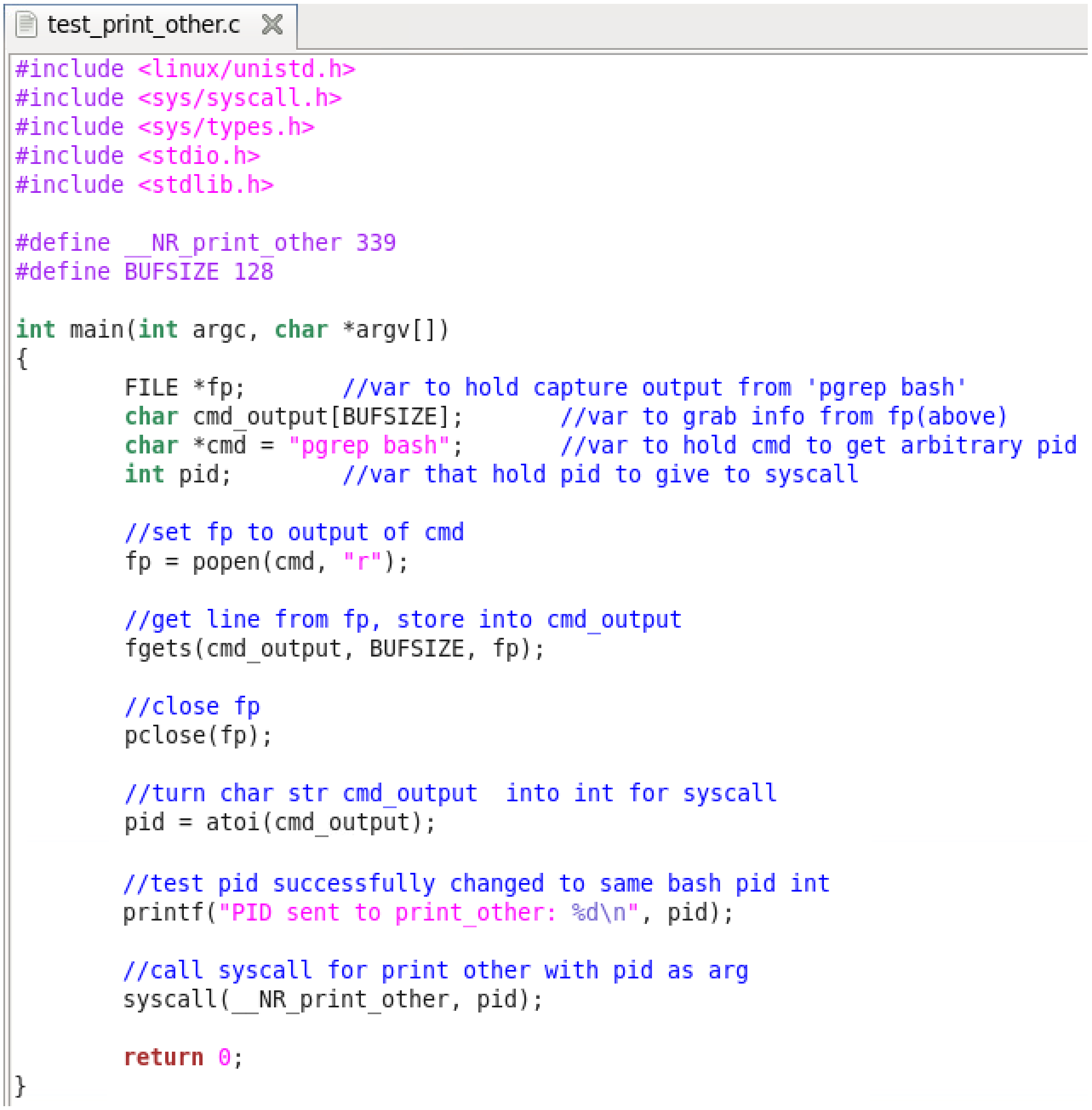
2-3. **Source code for Task 3**

Kernel:



```c
#include <linux/kernel.h>
#include <linux/sched.h>
#include <linux/module.h>
#include <linux/pid.h>

asmlinkage int sys_print_other(int pid)
{
        struct task_struct *task;       //struct for process struct

        //printk(KERN_EMERG "Other\n");

        //get task struct for specific process
        task = pid_task(find_vpid(pid), PIDTYPE_PID);

        //if task is null then no process, otherwise print info about process
        if (task != NULL)
        {
                printk(KERN_EMERG "PID: %d, name: %s\n", task->pid, task->comm);
        }
        else
        {
                printk(KERN_EMERG "PID: %d is not a valid process ID\n", pid);
        }

        return 0;
}
```

Test:


test_print_other.c

```
#include <linux/unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>

#define __NR_print_other 339
#define BUFSIZE 128

int main(int argc, char *argv[])
{
        FILE *fp;       //var to hold capture output from 'pgrep bash'
        char cmd_output[BUFSIZE];       //var to grab info from fp(above)
        char *cmd = "pgrep bash";       //var to hold cmd to get arbitrary pid
        int pid;        //var that hold pid to give to syscall

        //set fp to output of cmd
        fp = popen(cmd, "r");

        //get line from fp, store into cmd_output
        fgets(cmd_output, BUFSIZE, fp);

        //close fp
        pclose(fp);

        //turn char str cmd_output  into int for syscall
        pid = atoi(cmd_output);

        //test pid successfully changed to same bash pid int
        printf("PID sent to print_other: %d\n", pid);

        //call syscall for print other with pid as arg
        syscall(__NR_print_other, pid);

        return 0;
}
```

***Note: source codes attached as files as well.**