Procfs Kernel Module

Procfs

- Special file system used for information related to the system and its processes
- Mounted at /proc
- Acts as an interface to the kernel
- To see different processes using procfs, run

```
> ll /proc
```

Procfs and Processes

- /proc/PID/cmdline: the command that originally started the process.
- /proc/PID/cwd: a symlink to the current working directory of the process.
- /proc/PID/exe: a symlink to the original executable file, if it still exists.
- /proc/PID/fd: a directory containing a symbolic link for each open file descriptor.
- /proc/PID/maps: a text file containing information about mapped files and blocks (like heap and stack).
- /proc/PID/status: contains basic information about a process including its run state and memory usage.

- https://en.wikipedia.org/wiki/Procfs

Hello World Procfs Module

- Create a proc entry /proc/hello upon module load
- Support read/write to proc entry
- Remove proc entry upon module exit

- Helpful Tutorial: https://devarea.com/linux-kernel-development-creating-a-proc-file-and-interfacing-with-user-space/#.XisQuHVKg5k

headers and globals

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>              //file system calls
#include <linux/uaccess.h>              //memory copy from kernel <-> userspace

MODULE_LICENSE("Dual BSD/GPL");

#define BUF_LEN 100 //max length of read/write message

static struct proc_dir_entry* proc_entry;           //pointer to proc entry

static char msg[BUF_LEN];               //buffer to store read/write message
static int procfs_buf_len;                          //variable to hold length of message
```

read

```
static ssize_t procfile_read(struct file* file, char * ubuf, size_t count, loff_t *ppos)
{
            printk(KERN_INFO "proc_read\n");
            procfs_buf_len = strlen(msg);

            if (*ppos > 0 || count < procfs_buf_len)    //check if data already read and if space in user buffer
                        return 0;

            if (copy_to_user(ubuf, msg, procfs_buf_len))    //send data to user buffer
                        return -EFAULT;

            *ppos = procfs_buf_len;     //update position

            printk(KERN_INFO "gave to user %s\n", msg);

            return procfs_buf_len;     //return number of characters read
}
```

write

```
static ssize_t procfile_write(struct file* file, const char * ubuf, size_t count, loff_t* ppos)
{
        printk(KERN_INFO "proc_write\n");

  //write min(user message size, buffer length) characters
        if (count > BUF_LEN)
                procfs_buf_len = BUF_LEN;
        else
                procfs_buf_len = count;

        copy_from_user(msg, ubuf, procfs_buf_len);

        printk(KERN_INFO "got from user: %s\n", msg);

        return procfs_buf_len;
}
```

Memory Copying

- Kernel → User

  unsigned long copy_to_user (void __user *to, const void *from, unsigned long size)

- User → Kernel

  unsigned long copy_from_user (void *to, const void __user* from, unsigned long size)

- Needed because
  - User process uses virtual memory
  - Prevents crashing due to inaccessible regions
  - Can handle architecture specific issues

init

```
//make sure this struct is a global variable (not inside of a function)
static struct file_operations procfile_fops = {
    .owner = THIS_MODULE,
    .read = procfile_read,            //fill in callbacks to read/write functions
                    .write = procfile_write,
};

static int hello_init(void)
{
    //proc_create(filename, permissions, parent, pointer to fops)
                    proc_entry = proc_create("hello", 0666, NULL, &procfile_fops);

                    if (proc_entry == NULL)
                                        return -ENOMEM;

                    return 0;
}
```

exit

```
static void hello_exit(void)
{
        proc_remove(proc_entry);
        return;
}
```

Testing

```
> insmod hello_proc.ko
> echo hi > /proc/hello
> cat /proc/hello
> rmmod hello_proc
```

file_operations

file_operations has many functions that can be used to define different behaviours

read: reading from the proc file

write: writing to the proc file

open: run on opening the proc file

many more for more specific uses

my_timer

- Store last access time of /proc/timer
- Compute difference between last access time and current time