Procfs Kernel Module

Procfs Kernel Module

- Hello World for /proc
- Steps
  - Create entry in module_init
  - Create file system functions
    - Open
    - Read
    - Close
  - Delete entry in module_cleanup

Headers and Globals

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple module featuring proc read");

#define ENTRY_NAME "helloworld"
#define PERMS 0644
#define PARENT NULL
static struct file_operations fops;

static char *message;
static int read_p;
```

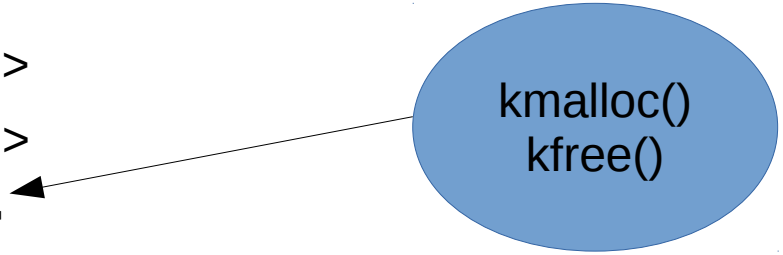
Headers and Globals

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple module featuring proc read");

#define ENTRY_NAME "helloworld"
#define PERMS 0644
#define PARENT NULL
static struct file_operations fops;

static char *message;
static int read_p;
```

File System Structure and Calls

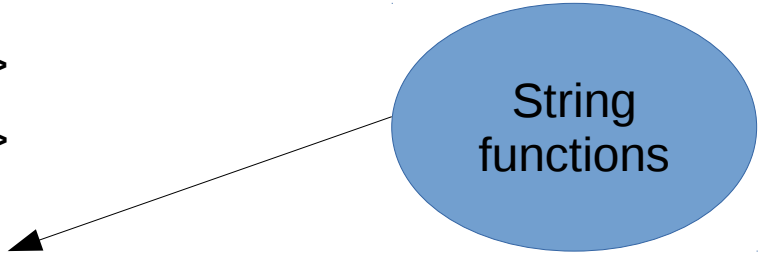
Headers and Globals

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple module featuring proc read");

#define ENTRY_NAME "helloworld"
#define PERMS 0644
#define PARENT NULL
static struct file_operations fops;

static char *message;
static int read_p;
```



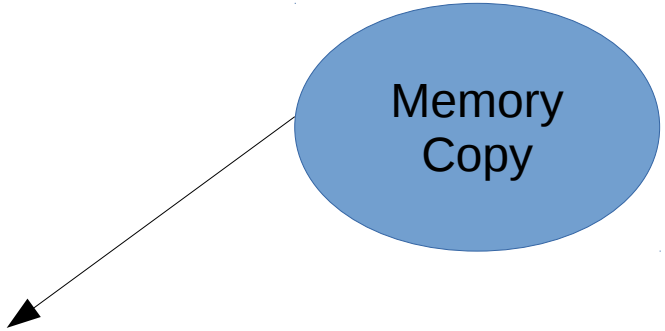
Headers and Globals

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple module featuring proc read");

#define ENTRY_NAME "helloworld"
#define PERMS 0644
#define PARENT NULL
static struct file_operations fops;

static char *message;
static int read_p;
```



Headers and Globals

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple module featuring proc read");

#define ENTRY_NAME "helloworld"
#define PERMS 0644
#define PARENT NULL
static struct file_operations fops;

static char *message;
static int read_p;
```



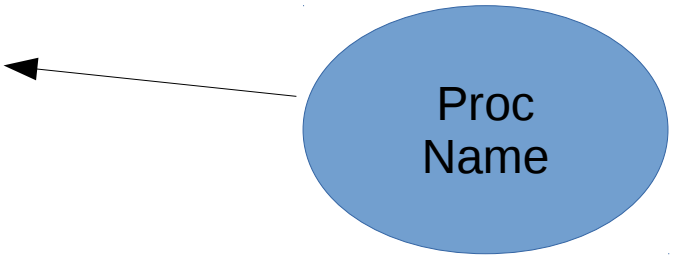
Headers and Globals

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple module featuring proc read");

#define ENTRY_NAME "helloworld"
#define PERMS 0644
#define PARENT NULL
static struct file_operations fops;

static char *message;
static int read_p;
```

Module Descriptions

Headers and Globals

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple module featuring proc read");

#define ENTRY_NAME "helloworld"
#define PERMS 0644
#define PARENT NULL
static struct file_operations fops;

static char *message;
static int read_p;
```



Headers and Globals

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple module featuring proc read");

#define ENTRY_NAME "helloworld"
#define PERMS 0644
#define PARENT NULL
static struct file_operations fops;

static char *message;
static int read_p;
```

Proc Permissions

Headers and Globals

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple module featuring proc read");

#define ENTRY_NAME "helloworld"
#define PERMS 0644
#define PARENT NULL
static struct file_operations fops;

static char *message;
static int read_p;
```

Proc
Parent Directory

Headers and Globals

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple module featuring proc read");

#define ENTRY_NAME "helloworld"
#define PERMS 0644
#define PARENT NULL
static struct file_operations fops;

static char *message;
static int read_p;
```

Points to
Proc File
Definitions

Headers and Globals

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/slab.h>
#include <linux/string.h>
#include <linux/uaccess.h>

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple module featuring proc read");

#define ENTRY_NAME "helloworld"
#define PERMS 0644
#define PARENT NULL
static struct file_operations fops;

static char *message;
static int read_p;
```

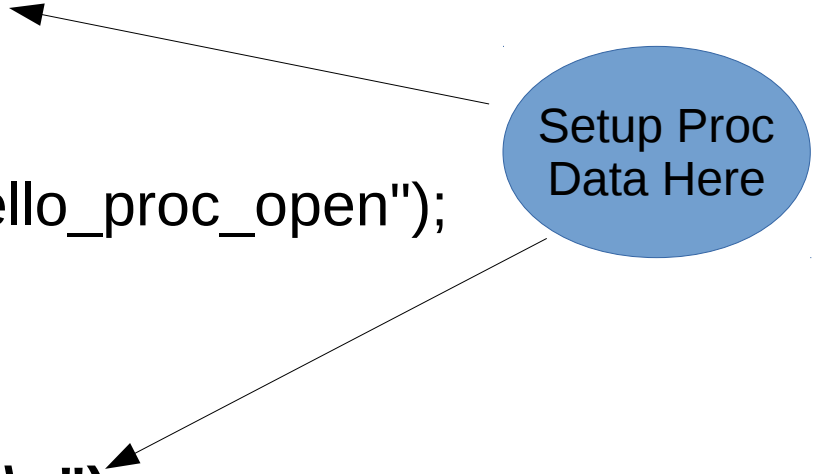
Message to Display in Proc

Creation

```
int hello_proc_open(struct inode *sp_inode, struct file *sp_file) {
	printk(KERN_INFO "proc called open\n");

	read_p = 1;
	message = kmalloc(sizeof(char) * 20, __GFP_RECLAIM |
__GFP_WAIT | __GFP_IO | __GFP_FS);
	if (message == NULL) {
		printk(KERN_WARNING “hello_proc_open");
		return -ENOMEM;
	}
	strcpy(message, "Hello, World!\n");
	return 0;
}
```

Creation

```
int hello_proc_open(struct inode *sp_inode, struct file *sp_file) {
        printk(KERN_INFO "proc called open\n");

        read_p = 1;
        message = kmalloc(sizeof(char) * 20, __GFP_RECLAIM |
__GFP_IO | __GFP_FS);
        if (message == NULL) {
                printk(KERN_WARNING “hello_proc_open");
                return -ENOMEM;
        }
        strcpy(message, "Hello, World!\n");
        return 0;
}
```



kmalloc()

- Takes
  - Number of bytes to allocate
  - A flag on how to allocate it
- Remember to restrict kernel memory allocation
  - Can block important functions
  - Can crash kernel if improperly handled
  - Kernel has limited access to memory

kmalloc()

- Flags

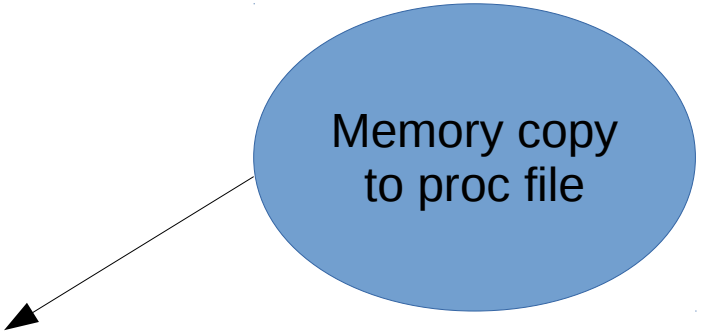
__GFP_RECLAIM	Allocator can sleep
__GFP_HIGH	Allocator can access emergency pools
__GFP_IO	Allocator can start disk I/O
__GFP_FS	Allocator can start filesystem I/O
__GFP_COLD	Allocator should use cache cold pages
__GFP_NOWARN	Allocator will not print failure warnings
__GFP_REPEAT	Allocator will repeat if it fails (can still fail)
__GFP_NOFAIL	Allocator will repeat if it fails (can not fail)
__GFP_NORETRY	Allocator will never retry if it fails
__GFP_NO_GROW	Used by the slab
__GFP_COMP	Used by hugetlb

Read

```
ssize_t hello_proc_read(struct file *sp_file, char __user *buf,
size_t size, loff_t *offset) {
	int len = strlen(message);

	read_p = !read_p;
	if (read_p)
		return 0;

	printk(KERN_INFO "proc called read\n");
	copy_to_user(buf, message, len);
	return len;
}
```

Read

```
ssize_t hello_proc_read(struct file *sp_file, char __user *buf,
size_t size, loff_t *offset) {
        int len = strlen(message);

        read_p = !read_p;
        if (read_p)
                return 0;

        printk(KERN_INFO "proc called read\n");
        copy_to_user(buf, message, len);
        return len;
}
```



Read

```
ssize_t hello_proc_read(struct file *sp_file, char __user *buf,
size_t size, loff_t *offset) {
        int len = strlen(message);

        read_p = !read_p;
        if (read_p)
                return 0;

        printk(KERN_INFO "proc called read\n");
        copy_to_user(buf, message, len);
        return len;
}
```

Read loops
until you return 0

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

Close

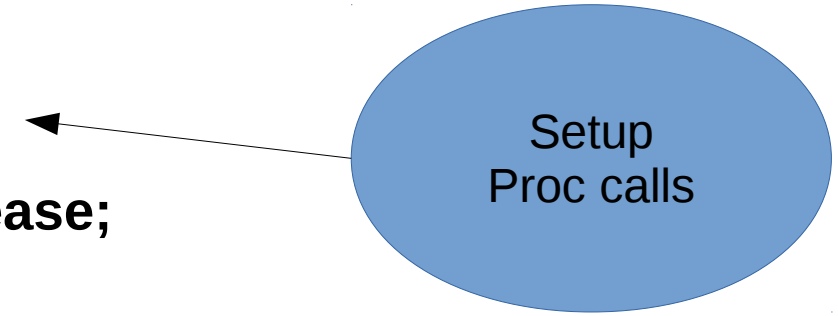
```
int hello_proc_release(struct inode *sp_inode,
struct file *sp_file) {
	printk(KERN_INFO "proc called
release\n");
	kfree(message);
	return 0;
}
```

Init

```
static int hello_init(void) {
	printk(KERN_NOTICE "/proc/%s create\n", ENTRY_NAME);
	fops.open = hello_proc_open;
	fops.read = hello_proc_read;
	fops.release = hello_proc_release;

	if (!proc_create(ENTRY_NAME, PERMS, NULL, &fops)) {
		printk("ERROR! proc_create\n");
		remove_proc_entry(ENTRY_NAME, NULL);
		return -ENOMEM;
	}
	return 0;
}
module_init(hello_proc_init);
```

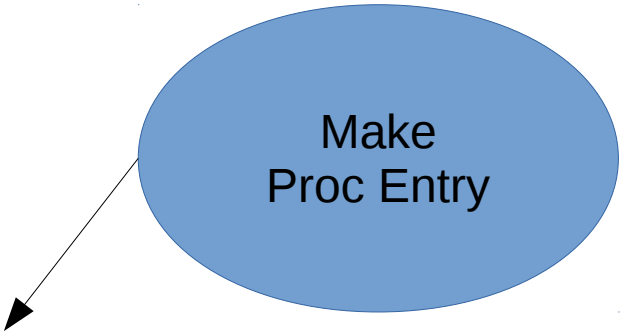
Init

```
static int hello_init(void) {
        printk(KERN_NOTICE "/proc/%s create\n", ENTRY_NAME);
        fops.open = hello_proc_open;
        fops.read = hello_proc_read;
        fops.release = hello_proc_release;

        if (!proc_create(ENTRY_NAME, PERMS, NULL, &fops)) {
                printk("ERROR! proc_create\n");
                remove_proc_entry(ENTRY_NAME, NULL);
                return -ENOMEM;
        }
        return 0;
}
module_init(hello_proc_init);
```



Init

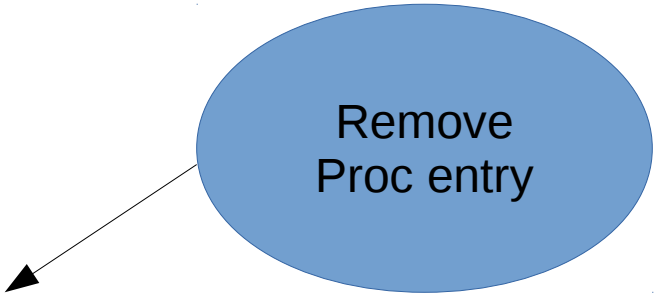
```
static int hello_init(void) {
        printk(KERN_NOTICE "/proc/%s create\n", ENTRY_NAME);
        fops.open = hello_proc_open;
        fops.read = hello_proc_read;
        fops.release = hello_proc_release;

        if (!proc_create(ENTRY_NAME, PERMS, NULL, &fops)) {
                printk("ERROR! proc_create\n");
                remove_proc_entry(ENTRY_NAME, NULL);
                return -ENOMEM;
        }
        return 0;
}
module_init(hello_proc_init);
```



Exit

```
static void hello_exit(void) {
	remove_proc_entry(ENTRY_NAME, NULL);
	printk(KERN_NOTICE "Removing /proc/%s.\n", ENTRY_NAME);
}

module_exit(hello_proc_exit);
```

Exit




```
static void hello_exit(void) {
        remove_proc_entry(ENTRY_NAME, NULL);
        printk(KERN_NOTICE "Removing /proc/%s.\n", ENTRY_NAME);
}

module_exit(hello_proc_exit);
```

Testing

```
$ make
$ sudo insmod hello_proc.ko
$ dmesg | tail
$ cat /proc/helloworld
$ sudo rmmod hello_proc
$ dmesg | tail
```