Part 1

Task 1

Kernel modules

Kernel modules are pieces of code that can be loaded and unloaded into the kernel upon demand.
They extend the functionality of the kernel without the need to reboot the system.
For example, one type of module is the device driver, which allows the kernel to access hardware connected to the system.
(Without modules, we would have to build monolithic kernels and add new functionality directly into the kernel image, Besides having larger kernels, this has the disadvantage of requiring us to rebuild and reboot the kernel every time we want new functionality.)

Pros and cons

pros

- The kernel does not have to rebuild your kernel as often. This saves time and prevents the possibility of introducing an error in rebuilding and reinstalling the base kernel.
- Using modules can save memory, because they are loaded only when the system is actually using them.
- It is easier to diagnose system problems. Modules are much faster to maintain and debug. What would require a full reboot to do with a filesystem driver built into the kernel can be done with a few quick commands using modules. It is possible to try out different parameters or even change the code repeatedly in rapid succession, without waiting for a boot.[1]

---

[1] https://eng.libretexts.org/Bookshelves/Computer_Science/Operating_Systems/Linux_-_The_Penguin_Marches_On_(McClanahan)/06%3A_Kernel_Module_Management/2.04%3A_Kernel_Modules#:~:text=and%20remediate%20accordingly.-,Linux%20Kernel%20Modules,as%20built%2Din%20or%20loadable.

Cons

- It may lose stability. If there is a module that does something bad, the kernel can crash, as modules should have full permissions.
- Security is compromised. A module can do anything, so one could easily write an evil module to crash things.
- Coding can be more difficult, as the module cannot reference kernel procedures without kernel symbols.

Relationship Between Kernel and Kernel Module

The kernel has complete control over the system, and modules interact with the kernel to provide specific functions. The kernel can load or unload these modules dynamically.

Real-world kernel modules

`ip_tables` - Offers network packet filtering capabilities, used to set up, maintain, and inspect packet filter rules.

`snd_hda_intel` - Deals with Intel High Definition Audio, controlling sound on most modern PCs.

`crypto` - Kernel cryptographic API, provides a variety of cryptographic cipher algorithms.

Task 2

Adding module to kernel:

```
bezjac@bezjac-virtual-machine:~/test1$ sudo insmod hello_check_point_module.ko
bezjac@bezjac-virtual-machine:~/test1$ sudo dmesg | tail
```

```
[ 4863.197550] hello_check_point_module:
[ 4863.212707] hello_check_point_module:
ing kernel
[ 4863.243306] Hello, World!
```

Removing Module from kernel:

```
bezjac@bezjac-virtual-machine:~/test1$ sudo rmmod hello_check_point_module.ko
bezjac@bezjac-virtual-machine:~/test1$ sudo dmesg | tail
```

```
[ 5052.439585] Goodbye, World!
```

Part 2

Task 3

hello_check_point_module.c

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

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Yehuda");
MODULE_DESCRIPTION("A simple example Linux module.");
MODULE_VERSION("0.02");

// messagge  file level scope
static char *message = "World";
// module_param  = used to declare parameters that can be passed to a module.
//charp = pointer to char.
//S_IRUGO = reading premession (by anyone).
module_param(message, charp, S_IRUGO);
MODULE_PARM_DESC(message, "A name to display in /var/log/kern.log");

static int __init lkm_example_init(void) {
    printk(KERN_INFO "Hello, %s!\n", message);
    return 0;
}

static void __exit lkm_example_exit(void) {
    printk(KERN_INFO "Goodbye, %s!\n", message);
}

module_init(lkm_example_init);
module_exit(lkm_example_exit);
```

```
yehuda@yehuda-virtual-machine:~/kernelLern$ sudo rmmod hello_check_point_module.ko
yehuda@yehuda-virtual-machine:~/kernelLern$ sudo insmod hello_check_point_module.ko message="Team"
yehuda@yehuda-virtual-machine:~/kernelLern$ sudo rmmod hello_check_point_module.ko
yehuda@yehuda-virtual-machine:~/kernelLern$ sudo dmesg | tail
[ 1632.207963] e1000: ens33 NIC Link is Up 1000 Mbps Full Duplex, Flow Control: None
[ 2193.479137] audit: type=1326 audit(1690971788.457:218): auid=1000 uid=1000 gid=1000 ses=2 subj=snap.notepad-plus-plus.notep
ad-plus-plus pid=4591 comm="wineserver" exe="/snap/notepad-plus-plus/386/wine-platform/wine-devel/bin/wineserver" sig=0 arch=c
000003e syscall=203 compat=0 ip=0x7f87a5fb684b code=0x50000
[ 2665.319656] Hello, !
[ 2739.157159] Goodbye, !
[ 2742.430355] Hello, World!
[ 2780.945970] Goodbye, World!
[ 2817.536703] Hello, Team!
[ 2892.182859] Goodbye, Team!
[ 2905.262801] Hello, Team!
[ 2912.351789] Goodbye, Team!
yehuda@yehuda-virtual-machine:~/kernelLern$
```

```
yehuda@yehuda-virtual-machine:~/kernelLern$ modinfo hello_check_point_module.ko
filename:       /home/yehuda/kernelLern/hello_check_point_module.ko
version:        0.02
description:    A simple example Linux module.
author:         Yehuda
license:        GPL
srcversion:     0899ABDC052D010D97BF237
depends:
retpoline:      Y
name:           hello_check_point_module
vermagic:       5.19.0-50-generic SMP preempt mod_unload modversions
parm:           message:A name to display in /var/log/kern.log (charp)
yehuda@yehuda-virtual-machine:~/kernelLern$
```

Task 6

```c
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/ip.h>
#include <linux/icmp.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Bezalel Yehuda");
MODULE_DESCRIPTION("Kernel module to drop ping packets and print src and dest
IPs");
MODULE_VERSION("0.01");

static struct nf_hook_ops *nfho = NULL; //struct containing behavior to register to
netfilter hook.

// hook function
static int hfunc(void *priv, struct sk_buff *skb, const struct nf_hook_state
*state)
{
struct iphdr *iph; // struct to read ip header info from buffer
struct icmphdr *icmph; // struct to read icmp header info from buffer
unsigned int src_ip;
unsigned int dest_ip;

if (!skb) // bufffer is empty, not a ping message - accept packet
return NF_ACCEPT;

iph = ip_hdr(skb); // extract ip header info from buffer
if (iph->protocol != IPPROTO_ICMP) // accept all packets that are not using ICMP
protocol
return NF_ACCEPT;

icmph = icmp_hdr(skb); // extract icmp header info from buffer
if (icmph->type != ICMP_ECHO) // accept all ICMP messages that are not ICMP ECHO
messages
return NF_ACCEPT;

src_ip = (unsigned int)iph->saddr; //source ip address
dest_ip = (unsigned int)iph->daddr; // destination ip address
printk(KERN_INFO "Dropping ping packet. src: %pI4 dest: %pI4\n", &src_ip,
&dest_ip);
```

```
return NF_DROP; // drop ping messages
}

static int __init LKM_init(void)
{
// define behaviour for netfilter hook
nfho = (struct nf_hook_ops*)kcalloc(1, sizeof(struct nf_hook_ops), GFP_KERNEL);
nfho->hook = (nf_hookfn*)hfunc; /// set earlier func as hook func
nfho->hooknum = NF_INET_PRE_ROUTING; // define hook timing
nfho->pf = PF_INET; // define protocol
nfho->priority = NF_IP_PRI_FIRST; // define max priority for rule.

nf_register_net_hook(&init_net, nfho); // register the hook behaviour.

return 0;
}

static void __exit LKM_exit(void)
{
nf_unregister_net_hook(&init_net, nfho); // unregister the hook
kfree(nfho);
}

module_init(LKM_init);
module_exit(LKM_exit);
```

```
➜  ~ ping 172.16.27.132
PING 172.16.27.132 (172.16.27.132): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
Request timeout for icmp_seq 2
Request timeout for icmp_seq 3
Request timeout for icmp_seq 4
```

```
missing - tainting kernel
775.317632] Dropping ping packet. src: 172.16.27.1 dest: 172.16.27.132
776.322118] Dropping ping packet. src: 172.16.27.1 dest: 172.16.27.132
777.326097] Dropping ping packet. src: 172.16.27.1 dest: 172.16.27.132
778.325813] Dropping ping packet. src: 172.16.27.1 dest: 172.16.27.132
779.330848] Dropping ping packet. src: 172.16.27.1 dest: 172.16.27.132
780.333656] Dropping ping packet. src: 172.16.27.1 dest: 172.16.27.132
```