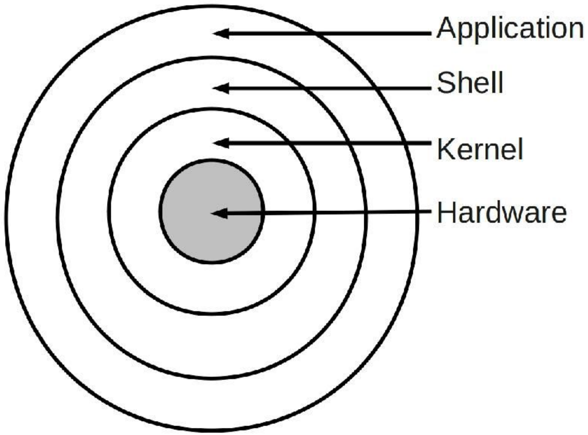
CENG 322
LAB 9

Kernel Code

Kernel

- A kernel is a central component of an operating system.
- It acts as an interface between the user applications and the hardware.
- The main aim of the kernel is to manage the communication between the software (user level applications) and the hardware (CPU, disk memory etc).



The main tasks:

- Process management
- Device management
- Memory management

Kernel Types

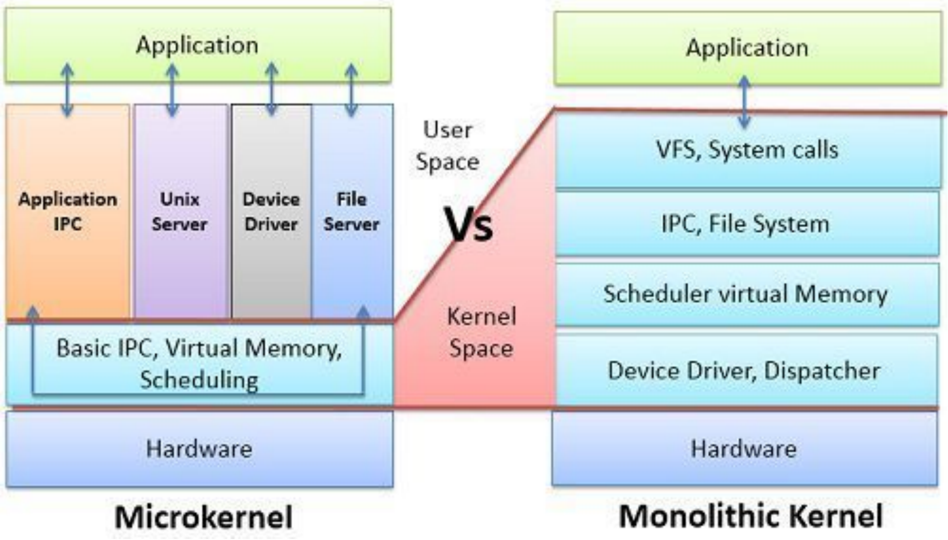
Anciently, all the basic system services like process and memory management, interrupt handling etc. were packaged into a single module in kernel space.

This type of architecture led to some serious drawbacks:

- Size of kernel, which is huge.
- Poor maintainability (i.e. bug fixing or addition of new features resulted in recompilation of the whole kernel which could consume hours)

What are the types? (Mainly)

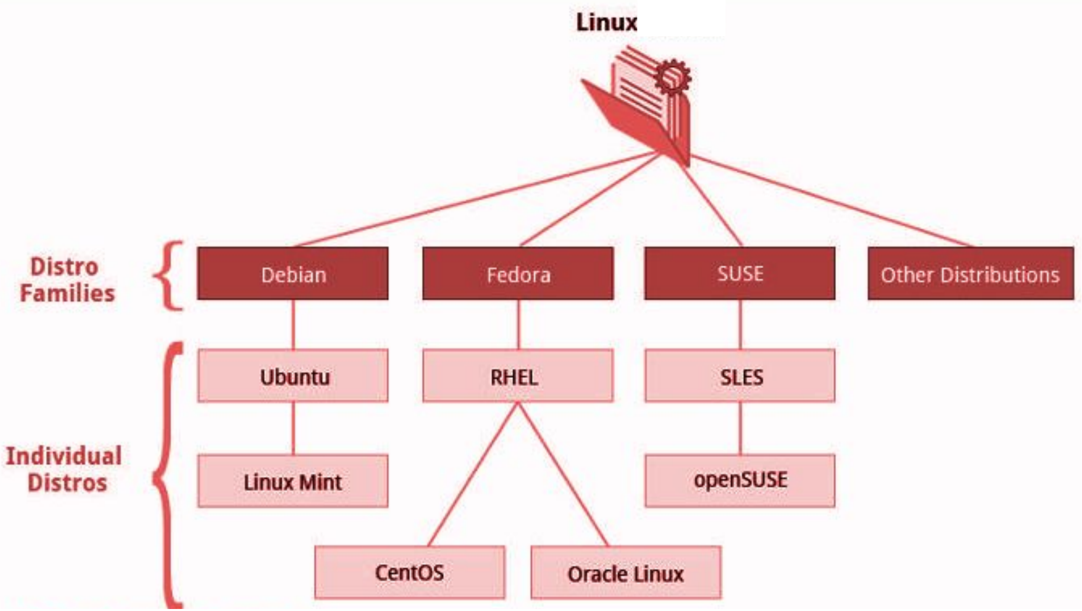
- Monolithic Kernel: OS runs as a single program in KM. (Linux)
- Micro Kernel: User and kernel services are implemented in different address space. (Mac OS X)



Linux

Is Linux a kernel?

What is Linux distribution?



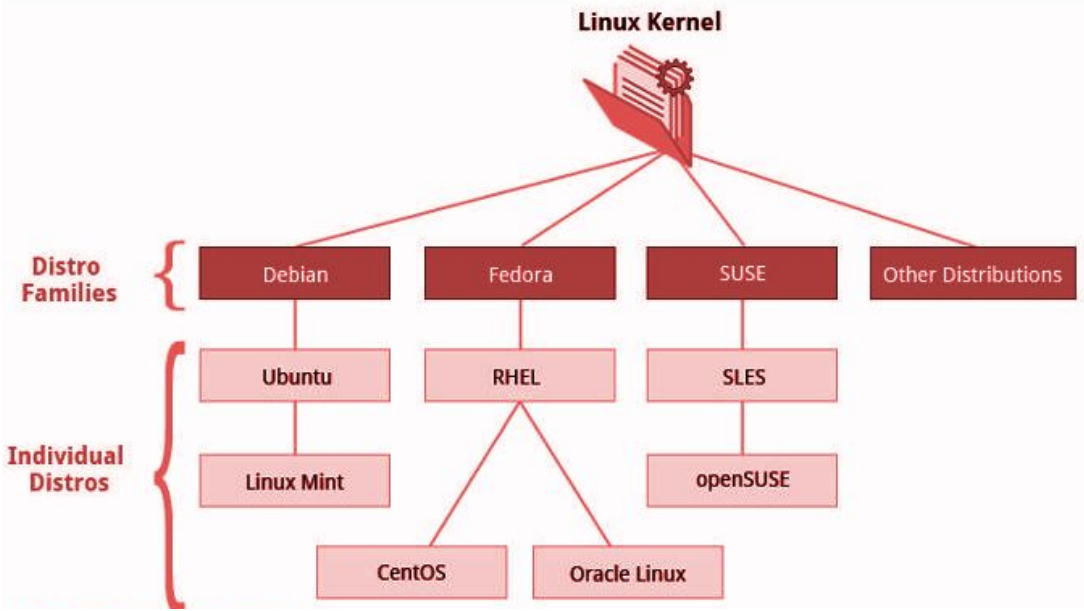
Linux

Is Linux a kernel?

- YES. Linux is just a kernel, not an OS!

What is Linux distribution?

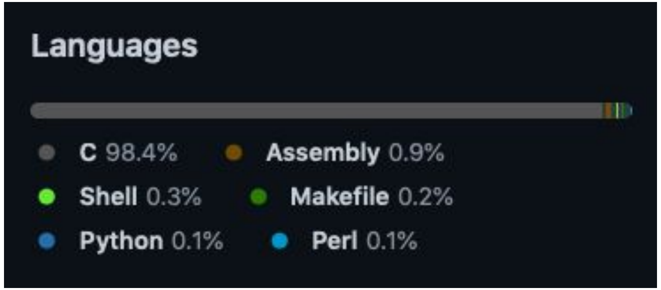
- It is an OS composed of the Linux kernel, GNU tools, additional software and a package manager.
- Kernel + Additional Programs



Linux

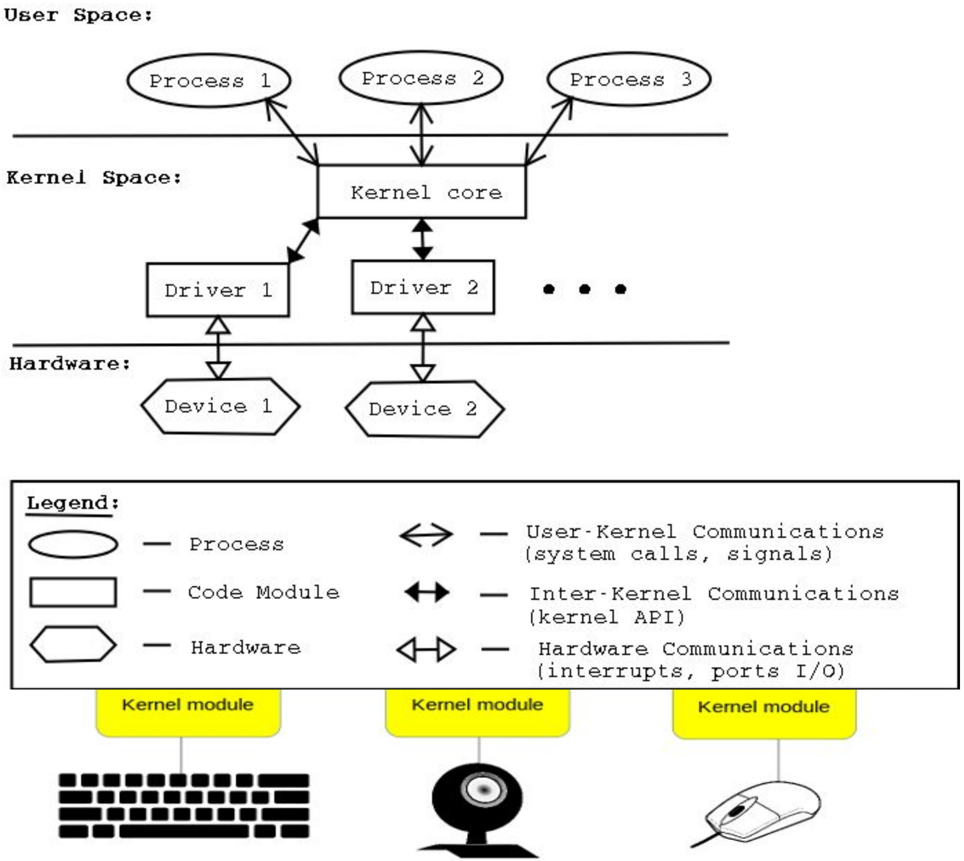
Why are Linux distributions UNIX-like OS?

- Before Linux kernel, Richard Stallman created the GNU project, the first free software project, in 1983.
- The GNU project implemented many of the popular Unix utilities like cat, grep, ls, shell (bash) along with developing their own compilers (GCC) and editors (Emacs).
- Back in the 80s UNIX was super expensive, thereof Linus Torvalds developed a new kernel that was like UNIX. With the GNU tools (especially shell commands), it is behaved like UNIX.
- Check source code: [https://github.com/torvalds/linu](https://github.com/torvalds/linux)



Kernel Module

- Modules are pieces of code that can be loaded and unloaded into the kernel.
- They extend the functionality of the kernel without the need to reboot the system. (e.g. device driver allows the kernel to access hardware connected to the system)
- Without modules, we would have to add new functionality directly into the kernel image (requires rebuild and reboot the kernel).



Writing Kernel Module

Kernel Code

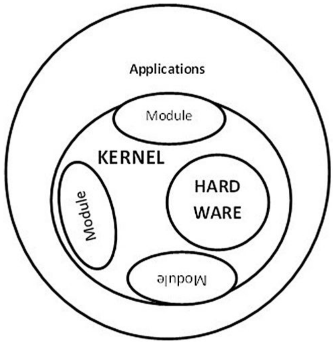
- Risk data loss and system corruption in the case of a any kernel code mistake !
- If you have a fault, it will lock up the entire system.
- Using a virtual machine is safer and eliminates risks.
- No access to the standard library (e.g. printk - replacement for printf and kmalloc - similar to malloc).
- There is no garbage collection. After unloading module, we have to cleaning up.
- Required root privileges.

Kernel Module

- Kernel modules typically act as APIs for user space programs.
- To discover what modules are already loaded within your current kernel use the command lsmod.
- For information about module: modinfo.
- Loaded modules are stored within the file /proc/modules, check: sudo cat /proc/modules
- To search for the video module: sudo lsmod | grep video
- Built-in kernel modules: more /lib/modules/$(uname -r)/modules.builtin | head -10

Writing Kernel Module

- **Static Modules**
    - Those are compiled as part of the base kernel and it is available at any time (compiled in).
    - If I compile “asus” device driver as a static module along with the kernel, I will be adding an extra size to the kernel image permanently. Suppose if I modify “asus“ device driver then need to re-compile the entire kernel to build it. Every time needs to rebooted for the changes to take effect.



- **Dynamic Modules**
    - Those are compiled as modules separately and loaded based on user demand. These are also called as Loadable Kernel Modules(LKM).
    - It relies on the presence of libraries, and the library is only loaded once no matter how many programs use it.
    - If we build “asus” module as dynamically(LKM), do not need to rebuild the kernel. It can be compiled separately. It can be loaded into the kernel at runtime without having the machine to reboot.

Writing Kernel Module

Dynamic Module Commands

- insmod: Inserting a module into the kernel. Kernel object (suffixed with .ko) can be installed with an option.

  $ insmod my_module.ko my_option=1

- rmmod: Removing a module. The kernel will not remove a module currently being used, -w option is removing once the use-count has decreased to zero.

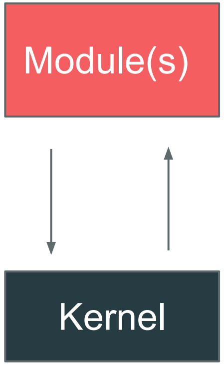
  $ rmmod my_module

- modprobe: Intelligent version of insmod and rmmod, it considers dependencies (if that particular module is dependent on any other module) and loads them.

  $ modprobe module_name parameter=value

- dmesg: displays kernel-related messages.

  $ dmesg



Writing Kernel Module

- Step 1/4: Checking kernel version and installing required packages.

- Check kernel version:

```
sudo uname -a
```

- Install the essential development tools and the kernel headers necessary for module development:

```
apt-get install build-essential linux-headers-'uname -r'
```

- User space header(s) can not be included in this module (e.g. stdio.h, stdlib.h...) headers.
- Linux kernel headers are located under usr/include/linux/.

Writing Kernel Module

- Step 2/4: Creating a code file (.c).

```
#include <linux/init.h>  // for the macros
#include <linux/module.h>  // used by all modules
#include <linux/kernel.h>  // for KERN_INFO
```

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("A. Yigit");
MODULE_DESCRIPTION("A simple example Linux module." );
MODULE_VERSION("0.01");
```

Check: CENG322_module.c

Writing Kernel Module

- Step 2/4: Creating a code file (.c).

- Printk displays the message in the kernel log (/var/log/kern.log).

```
#define KERN_EMERG   "<0>"  /* system is unusable */
#define KERN_ALERT   "<1>"  /* action must be taken immediately */
#define KERN_CRIT    "<2>"  /* critical conditions */
#define KERN_ERR     "<3>"  /* error conditions */
#define KERN_WARNING "<4>"  /* warning conditions */
#define KERN_NOTICE  "<5>"  /* normal but significant condition */
#define KERN_INFO    "<6>"  /* informational */
#define KERN_DEBUG   "<7>"  /* debug-level messages */
```

```
static int __init my_init(void) {
 printk(KERN_INFO "Hello, World! \n");
 return 0;
}
static void __exit my_exit(void) {
 printk(KERN_INFO "Goodbye, World! \n");
}
module_init(my_init);
module_exit(my_exit);
```

Check: CENG322_module.c

Writing Kernel Module

- Step 3/4: Compiling the code using a makefile.

```
obj-m += CENG322_module.o
all:
 make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
 make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

- obj-y: static compilation will link to kernel image.
  obj-m: dynamic compilation can be loaded into kernel image.
- Kernel object(.ko) will be created after compiling.

Check: Makefile

Writing Kernel Module

- Step 4/4: Inserting the module into the kernel.

```
sudo insmod CENG322_module.ko
```

- printk function doesn't output to the console but rather the kernel log

```
sudo dmesg
```

- Checking the kernel modules again:

```
lsmod | grep "CENG322"
```

- Removing the kernel:

```
sudo rmmod CENG322_module
```

- If you run dmesg again, you'll see "Goodbye, World!" in the logs. You can also use lsmod again to confirm it was unloaded.

Passing Arguments

- Declare the variables as global and then use the module_param() macro, (defined in include/linux/moduleparam.h).
- At runtime, insmod will fill the variables, like insmod mymodule.ko myvariable=5.
- The module_param() macro takes 3 arguments; name, type and permissions for the corresponding file in sysfs.
- For arrays of integers or strings see module_param_array() and module_param_string().

```
int myint = 3;
module_param(myint, int, 0);

int myintarray[2];
module_param_array(myintarray, int, NULL, 0);

short myshortarray[4];
int count;
module_param_array(myshortarray, short, &count, 0);
```

- Check: CENG322_modparameter.c

Passing Arguments

- You can find modes in the <linux/stat.h> header.

    - S_IRUSR, S_IWUSR, S_IXUSR – owner permission
    - S_IRGRP, S_IWGRP, S_IXGRP – group permission
    - S_IROTH, S_IWOTH, S_IXOTH – other permission

- Check all permissions: https://www.gnu.org/software/libc/manual/html_node/Permission-Bits.html

Communication with User App

- Write a kernel module that shows a message if user space application reads data from /proc.
- Message will be "You read data!".
- Overriding read function.

```
static struct proc_dir_entry* proc_entry;

static ssize_t custom_read(struct file* file, char __user* user_buffer,
size_t count, loff_t* offset)
{
  printk(KERN_INFO "Calling our own custom read method.");
  char message[] = "You read data!\n";
  int message_length = strlen(greeting);
  if (*offset > 0)
   return 0;
  copy_to_user(user_buffer, message, message_length);
  *offset = message_length;

 return message_length;
}
```

- Check: CENG322_comuserapp.c

Communication with User App

```
static struct file_operations fops =
{
 .owner = THIS_MODULE,
 .read = custom_read
};

// Custom init and exit methods
static int __init custom_init(void) {
 proc_entry = proc_create("iytecommodule", 0666, NULL, &fops);
 printk(KERN_INFO "Communication driver loaded.");
 return 0;
}
```

```
static void __exit custom_exit(void) {
 proc_remove(proc_entry);
 printk(KERN_INFO "Goodbye ...");
}
module_init(custom_init);
module_exit(custom_exit);
```

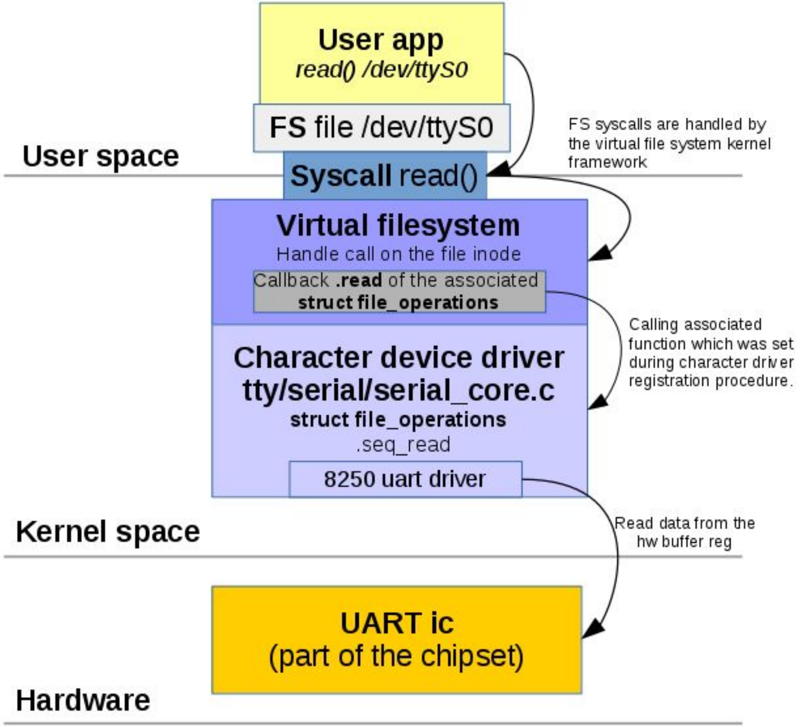
Developing a Device Driver

- Drivers are used to help the hardware devices interact with OS.
- Linux works with an “everything is a file” abstraction.
- Hardware devices are treated like ordinary files, which makes it easier for the software to interact with the device drivers.
- When a device is connected to the system, a device file is created in /dev directory.
- Most Common types of devices in Linux:
  - Character devices: These devices transmit the data character by characters, like a mouse or a keyboard (no buffering, sequentially).
  - Block devices: These devices transfer unit of data storage called a block, USB drives, hard drives, and CD ROMs (randomly).

Developing a Device Driver

```
ls -l /dev/ttyS0
```

- Symbol C, in the beginning, means that this device is a character device.
- Major and Minor numbers are assigned to the device.
- Inside the Linux kernel, every device is identified not by a symbolic name but by a unique number – the device's major number.
- This number assigning by the kernel during device registration. Every device driver can support multiple "sub-devices".
- A serial port adapter may contain two hardware ports. Both of these ports are handled by the same driver, and they share one Major number. But inside this driver, each of these ports is also identified by the unique number, and this is a device Minor number.



Developing a Device Driver

- Create a read-only char device that says how many times you have read from the dev file. Writing will not be supported !
- Simply read in the data and print a message acknowledging that we received it.
- In the multiple-threaded environment, without any protection, concurrent access to the same memory may lead to the race condition.
- In the kernel module, this problem may happen due to multiple instances accessing the shared resources.
- We can use atomic Compare-And-Swap (CAS) to maintain the states, CDEV_NOT_USED and CDEV_EXCLUSIVE_OPEN , to determine whether the file is currently opened by someone or not.
-
- CAS compares the contents of a memory location with the expected value and, only if they are the same, modifies the contents of that memory location to the desired value.

```
enum {
    CDEV_NOT_USED = 0,
    CDEV_EXCLUSIVE_OPEN = 1,
};

/* Is device open? Used to prevent multiple access to device */
static atomic_t already_open = ATOMIC_INIT(CDEV_NOT_USED);
```

- Check: CENG322_devdriver.c

Kernel Building and Custom System Call

- All Linux distributions are based on a predefined kernel.
- In order to disable several options, drivers and change kernel settings, we need to build kernel from scratch.
- It takes a lot of time (hours).
- Firstly download source code of Linux kernel.

The Linux Kernel Archives

About Contact us FAQ Releases Signatures Site news

Protocol	Location
HTTP	https://www.kernel.org/pub/
GIT	https://git.kernel.org/
RSYNC	rsync://rsync.kernel.org/pub/

Latest Release
5.17.4

mainline:	5.18-rc3	2022-04-17	[tarball] [patch] [inc. patch] [view diff] [browse]
stable:	5.17.4	2022-04-20	[tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
stable:	5.16.20 [EOL]	2022-04-13	[tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
longterm:	5.15.35	2022-04-20	[tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
longterm:	5.10.112	2022-04-20	[tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
longterm:	5.4.190	2022-04-20	[tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
longterm:	4.19.239	2022-04-20	[tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
longterm:	4.14.276	2022-04-20	[tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
longterm:	4.9.311	2022-04-20	[tarball] [pgp] [patch] [inc. patch] [view diff] [browse] [changelog]
linux-next:	next-20220422	2022-04-22	[browse]

```
wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.17.4.tar.xz
```

- Check: custom_system_call.txt

Thanks ...