|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
|  |  |  |  | Embedded system Q's  <http://linuxdevicedrivercinterviewqs.blogspot.com/2015/03/linux-device-driverembedded-c-interview.html>  Refer linux materials : <https://www.javatpoint.com/linux-signals>  OS concepts in GFG: <https://www.geeksforgeeks.org/interrupts/> | | | |
|  | 1. What is userspace and kernel space ?   **Memory get's divided into two distinct areas:**  **1. what runs in us and ks**   * **The user space**, which is a set of locations where normal user processes run (i.e everything other than the kernel). The role of the kernel is to manage applications running in this space from messing with each other, and the machine. * **The kernel space**, which is the location where the code of the kernel is stored, and executes under.   kernel space) is the most privileged  2. Accessibility  Processes running under the user space have access only to a limited part of memory, whereas the kernel has access to all of the memory. Processes running in user space also *don't* have access to the kernel space.    User space processes can *only access a small part of the kernel* via an interface exposed by the kernel - **the system calls**.     1. Why we need US and KS ?   US restricts user programs so they can't mess with memory (and other resources) owned by other programs or by the OS kernel.       1. How system calls enter kernel and which function it calls in kernel ?   From the previous part we know that system call concept is very similar to an interrupt. Furthermore, system calls are implemented as software interrupts. So, when the processor handles a syscall instruction from a user application, this instruction causes an **exception** which transfers control to an **exception handler**. As we know, all exception handlers (or in other words kernel [C](https://en.wikipedia.org/wiki/C_%28programming_language%29) functions that will react on an exception) are placed in the kernel code. But how does the Linux kernel search for the address of the necessary system call handler for the related system call? The Linux kernel contains a special table called the **system call table**. The system call table is represented by the **sys\_call\_table array** in the Linux kernel which is defined in the [arch/x86/entry/syscall\_64.c](https://github.com/torvalds/linux/blob/16f73eb02d7e1765ccab3d2018e0bd98eb93d973/arch/x86/entry/syscall_64.c) source code file. Let's look at its implementation:    *From <*[*https://0xax.gitbooks.io/linux-insides/content/SysCall/linux-syscall-2.html*](https://0xax.gitbooks.io/linux-insides/content/SysCall/linux-syscall-2.html)*>   Read this :* [*https://0xax.gitbooks.io/linux-insides/content/SysCall/linux-syscall-2.html*](https://0xax.gitbooks.io/linux-insides/content/SysCall/linux-syscall-2.html)     1. What is Device driver ?   A software designed to interact with h/w , without which h/w device fails to work .  **A DD communicated with h/w by means of computer bus on which h/w is connected.**  A DD acts as translator b/w OS and h/w.  Tyes :  Char driver : byte oriented  Block driver : block size data oriented  N/w driver: data in packets    LDD2  1- What is basic structure of kernel module ?  Headers: linux/module.h  linux/kernel.h  linux/init.h    init function -> this func is excuted 1st when we inset LDD module into kernel (insmode and modeprobe)  Exit function -> function will execute last when LDD is unoaded from kernel (rmmod)  Module License  Module Author  Module description  Module Version    Modeinfo is used to see all this info    2- Write a simple kernel module and use diff printk to print logs.      LDD3   1. What us function pointer and call back function ?  syntax of a function taking argument as function pointer and returning a function pointer ? 2. Explain abt  modue\_param , variable sysfs entry will be made , we can change value of variable at sysfs entry   Module\_param\_array  Module\_param\_cb , whenever we set value of a variable it will call this callback function mentioned in  struct kernel\_param\_ops to indicate the value is changed.      LDD4 - Major and Minor number    Note:   * To linux everythng is a file. A h/w device also looks like a file and normal open, read write operations can b used on them. * **The driver will create a special file for every hardware device, we can commicate with the h/w with this special file.**  1. **How Application will communicate with h/w devives ?**  * **Application open** device file . The device file is created by device driver. * This device file will find corresponding device driver with the help of major and minor number. * Then device driver will talk to h/w through the bus on which h/w is connected.      1. **What is major and minor number ?**   Major : identifies the driver associated with the device. It can be shared with multiple device drivers  Minor: Many devices may uses same major number, we need to assign a number to each devices using same major number and i.e Minor number .  The device driver uses the minor number **<minor>**to distinguish individual physical or logical devices.  Checjk /proc/devices    API: register\_chrdev\_region(); / alloc\_chrdev\_region();  MKDEV(int major, int minor) , MINOR(dev\_t dev); MAJOR(dev\_t dev);  Dev\_t dev -> 32 bit -> 12 bit = Major (4096 = 4k), 20 bit= Minior(max: 1,048,576 = 1M)    LDD5 - Device File  NOTE-   * Whenever we access device file, kernel recognize the I/O request and passes it to device driver . Such as reading data from a serial port or sending data to h/w. * ls -l /dev   **/dev/null** acts as a byte sink; any write request to **/dev/null** will succeed, but the data are written will be ignored.   * the size field in the **ls -l** listing is replaced by two numbers, separated by a comma. 1st- major and 2nd-minor        * Manually : Mknod -m <permission> <name> <device type> <major> <minor> * Automatically : create a class : class\_create(); * create device : device\_create(); * Udev is a device manager which adds/remove the device files from /dev       LDD6: CDEV     * Cdev is a element of inode strct(inode strcut represents file in kernel). * cdev stuct represents char driver * Initialze: fops , major and minor no. , owner * Dynamic: void cdev\_init(struct cdev \*cdev, struct file\_operations \*fops); * Static:  |  | | --- | | **struct cdev \*my\_cdev = cdev\_alloc( );**  **my\_cdev->ops = &my\_fops;** |      * Once the **cdev**structure is set up with file\_operations and owner, the final step is to tell the kernel about it with a call to:   **int cdev\_add(struct cdev \*dev, dev\_t num, unsigned int count);**  After a call to **cdev\_add()**, your device is immediately alive. All functions you defined (through the file\_operations structure) can be called.      What are file operartions ? 1. fops struct is defined in  [***<linux/fs.h>***](https://elixir.bootlin.com/linux/latest/source/include/linux/fs.h)   1. Open , writ, read, release       LDD7 : how user application invokes kernel device driver     * The user Program will communicate with the kernel space program using the device file. * Kmalloc - GFP\_ATOMIC – Allocation will not sleep      * Copy\_from\_user :   **unsigned long copy\_from\_user(void \*to, const void \_\_user \*from, unsigned long  n);**     * Copy\_to user   **unsigned long copy\_to\_user(const void \_\_user \*to, const void \*from, unsigned long  n);**      LDD8 : IOCTL     * IOCTL is i/p and o/p control. * This is used in specific operations of a device for which kernel doesn’t have any system calls.   Example:    **Some examples of ICOTL**    **Change baude rate of serial port**  **Adjust volume**  **Reading or writing device registers**  **file:** "include/uapi/linux/rtc.h"  #define RTC\_ALM\_SET \_IOW('p', 0x07, struct rtc\_time) /\* **Set alarm time** \*/  #define RTC\_ALM\_READ \_IOR('p', 0x08, struct rtc\_time) /\* **Read alarm time** \*/  #define RTC\_RD\_TIME \_IOR('p', 0x09, struct rtc\_time) /\* **Read RTC time** \*/  #define RTC\_SET\_TIME \_IOW('p', 0x0a, struct rtc\_time) /\* **Set RTC tim**e \*/  #define RTC\_IRQP\_READ \_IOR('p', 0x0b, unsigned long) /\* **Read IRQ rate** \*/  #define RTC\_IRQP\_SET \_IOW('p', 0x0c, unsigned long) /\* **Set IRQ rate** \*/  #define RTC\_EPOCH\_READ \_IOR('p', 0x0d, unsigned long) /\* **Read epoch** \*/  #define RTC\_EPOCH\_SET \_IOW('p', 0x0e, unsigned long) /\* **Set epoch** \*/    #define RTC\_WKALM\_SET \_IOW('p', 0x0f, struct rtc\_wkalrm)/\* **Set wakeup alarm**\*/  #define RTC\_WKALM\_RD \_IOR('p', 0x10, struct rtc\_wkalrm)/\* **Get wakeup alarm**\*/    Q1- What are the API IOCTL need ?  1- Driver  Create IOCTL command in driver -> #define "ioctl\_name" \_\_IOX('magic number', 'command number', "argument type")  create IOCTL function in driver - int icotl(strcuct file \*file, unsigned int cmd, unsigned long arg);  2- Application:  Create IOCTL cmd in app -> #define "ioctl\_name" \_IOX();  Use Ioctl system call in app -> ioctl(file discriptor, "ioctl cmd", "argument")        Some Real time examples if IOCTL    "arch/m68k/kernel/time.c"  ==>> **copy\_to\_user(argp, &pll, sizeof pll) ? -EFAULT : 0;**  static int rtc\_ioctl(struct device \*dev, unsigned int cmd, unsigned long arg)   * Create IOCTL cmd   "include/uapi/linux/rtc.h" **#define RTC\_EPOCH\_READ \_IOR('p', 0x0d, unsigned long) /\* Read epoch**  #define RTC\_ALM\_SET \_IOW('p', 0x07, struct rtc\_time) /\* **Set alarm time** \*/  #define RTC\_ALM\_READ \_IOR('p', 0x08, struct rtc\_time) /\* **Read alarm time** \*/  #define RTC\_RD\_TIME \_IOR('p', 0x09, struct rtc\_time) /\* **Read RTC time** \*/      -> IOCTL function:  "arch/alpha/kernel/rtc.c"  static int  alpha\_rtc\_ioctl(struct device \*dev, unsigned int cmd, unsigned long arg)  {  switch (cmd) {  case RTC\_EPOCH\_READ:  return put\_user(rtc\_epoch, (unsigned long \_\_user \*)arg);  case RTC\_EPOCH\_SET:  if (arg < 1900)  return -EINVAL;  rtc\_epoch = arg;  return 0;  default:  return -ENOIOCTLCMD;  }  }    -> IOCTL function call in user-application: \_\_ioctl (int fd, unsigned long int request, ...)    *What Is kernel preemption ?*  **kernel preemption** is a property possessed by some [kernels](https://en.wikipedia.org/wiki/Kernel_(operating_system)) (the cores of operating systems), in which the [CPU](https://en.wikipedia.org/wiki/CPU) can be interrupted in the middle of executing kernel [code](https://en.wikipedia.org/wiki/Code) and assigned other tasks (from which it later returns to finish its kernel tasks.    *x*          *LDD9 -> procfs*     1. The folder in root /proc is mount point for procfs, \ 2. It acts as connecting bridge b/w US and KS, us can use information in procfs provided by kernel. 3. Example :  * /proc/meminfo: will give details about memory * /proc/modules : will give details about modules currently that are part of kernel. * /proc/interrupts: registered interrupt requst number <https://www.hitchhikersguidetolearning.com/2021/03/28/looking-at-proc-interrupts/> * /proc/kallsyms - running kernel symbols, including loaded modules * /proc/cpuinfo - info abt cpu on the system * /proc/filesystms - currently active filsystem drivers     Q1- Do proc is read-only ?  - Most of Proc files are read only , some are with write permission    Q2-  **Do proc is used for debugging purpose** ?  - Proc file can also be used to modify kernel behaviors . If file has write permission  - Yes, it is usefull for debugging kernel module.  - While debugging we might have to know th valuse of various variable in that module or the data which module is handling.  - We can create pro entry and dump whatever data we want to look into that proc entry.    API's :  Proc\_mkdir() -> for dir create under proc  Proc\_create() -> for file create under proc  Struct file\_operations proc\_fops{ //open read write release functons }  Q3- How to read and write the data under proc ?  Read : cat /proc/etx/etx\_proc  write: echo "device driver proc" **>** /proc/etx/etx\_proc        LDD11 -> sysfs     1. Sysfs contains information about devices and drivers . 2. Some in sysfs are writable, for configuration and control devices attached to the system. 3. It exports system information of specific devices from KS to US. 4. Sysfs is tied with device driver of the kernel. Profs exports process specific information, Debugfs is used to exporting the debug information by the developer. |  |  |
|  |  |  |  |  |  |  |
|  |  | * 1. When we do DMA in kernel where memory is allocated (virtual or pysical memoy ) ?   <https://stackoverflow.com/questions/40890861/why-is-kmalloc-more-efficient-than-vmalloc>   * 1. How the malloc internally work?  Kmalloc and Vmalloc ? <https://stackoverflow.com/questions/40890861/why-is-kmalloc-more-efficient-than-vmalloc> | | | |  |
|  |  |  |  |  |  |  |
|  |  |  | 1. What is Difference b/w kernel and Driver ? kernel- It is core of operating system, its manages Resource, Scheduling, Process Management, File Management, N/w Management…etc  Driver: It is a piece of code reside below kernel and interact/server to hardware devices.   Probably all h/w devices are associated with some driver. This interaction is done by means of communication bus.       1. What are types of drivers and explain them? Char : it is h/w file that read and write the data in char by char fashion at a time. Keyboard, Mouse….   Block: it is a h/w file hat read/write the data in block size at a time. All disk like HDD, USB …etc  N/w: it process the data in packet format. Ex: Ethernet Card and all     1. Write Makefile?   obj-m += hello.o    All:  Make -C /lib/modules/'uname -r'/build M=$(shell PWD) modules  Clean:  Make -C /lib/modules/'uname -r'/build M=$(shell PWD) clean | | | |
|  |  |  |  |  |  |  |
|  |  |  |  | spi1 {    spi1\_default: spi1\_default {  /\* active state \*/  mux {  /\* MOSI, MISO, CLK \*/  pins = "gpio4", "gpio5", "gpio7";  function = "blsp\_spi2";  };    config {  pins = "gpio4", "gpio5", "gpio7";  drive-strength = <12>; /\* 12 MA \*/  bias-disable = <0>; /\* No PULL \*/  };  };          i2c\_4 {  i2c\_4\_active: i2c\_4\_active {  /\* active state \*/  mux {  pins = "gpio18", "gpio19";  function = "blsp\_i2c4";  };  config {  pins = "gpio18", "gpio19";  drive-strength = <2>;  bias-disable;  };  }; |  |  |
|  |  |  |  |  |  |  |