**IOCTL – Linux Device Driver Input Output Control**

This article has been taken from [Embetronix’s Tutorial on Linux Device Drivers](https://embetronicx.com/tutorials/linux/device-drivers/ioctl-tutorial-in-linux/). The chapter referenced is its [Tutorial Part 8 – I/O Control in Linux IOCTL()](https://embetronicx.com/tutorials/linux/device-drivers/ioctl-tutorial-in-linux/).

**Introduction**

Operating system segregates virtual memory into kernel space and user space.  Kernel space is strictly reserved for running the kernel, kernel extensions, and most device drivers. In contrast, user space is the memory area where all user mode applications work and this memory can be swapped out when necessary.

There are many ways to Communicate between the User space and Kernel Space, they are:

* **IOCTL**
* Procfs
* Sysfs
* Configfs
* Debugfs
* Sysctl
* UDP Sockets
* Netlink Sockets

In this tutorial we will look at the IOCTL.

**IOCTL**

IOCTL is referred as Input and Output Control, which is used to talking to device drivers. This system call is available in most driver categories.  The major use of this is in case of handling some specific operations of a device for which the kernel does not have a system call by default. Or the user can use existing ioctl’s to communicate with devices.

Some real time applications of ioctl are ejecting the media from a “cd” drive, to change the Baud Rate of Serial port, Adjust the Volume, Reading or Writing device registers, etc.

**Steps involved in IOCTL**

There are some steps involved to use IOCTL.

* Create or find the IOCTL command in driver
* Create IOCTL command in User space application
* Use IOCTL system call in User space

**Create IOCTL command in User space application**

You may have to define the ioctl command like how it is defined in driver. Let us say our driver has two ioctl’s that we are interested in – WR\_VALUE and RD\_VALUE. One of them writes a value of type int32 and the other reads it. The source code for this driver can be found in [Embetronix’s Tutorial Part 8](https://embetronicx.com/tutorials/linux/device-drivers/ioctl-tutorial-in-linux/).

**Example:**



|  |  |
| --- | --- |
| 1  2 | #define WR\_VALUE \_IOW('a','a',int32\_t\*)  #define RD\_VALUE \_IOR('a','b',int32\_t\*) |

**Use IOCTL system call in User space**

Include the header file <sys/ioctl.h>.

We need to call the ioctl command from a user application as follows:

long ioctl( "file descriptor","ioctl command", "Arguments");

<file descriptor>: This the open file on which the ioctl command needs to be executed, which would generally be device files.  
<ioctl command>: ioctl command which is implemented to achieve the desired functionality  
<arguments>: The arguments that needs to be passed to or from the ioctl command.

**Example:**



|  |  |
| --- | --- |
| 1  2  3 | ioctl(fd, WR\_VALUE, (int32\_t\*) &number);    ioctl(fd, RD\_VALUE, (int32\_t\*) &value); |

**Application Source Code**

This application is used to write the value to the driver. Then read the value again.



|  |  |
| --- | --- |
|  | #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <fcntl.h>  #include <unistd.h>  #include<sys/ioctl.h>    #define WR\_VALUE \_IOW('a','a',int32\_t\*)  #define RD\_VALUE \_IOR('a','b',int32\_t\*)    int main()  {          int fd;          int32\_t value, number;          printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");          printf("\*\*\*\*\*\*\*[WWW.EmbeTronicX.com\*\*\*\*\*\*\*\n](http://WWW.EmbeTronicX.com*******/n)");          printf("\nOpening Driver\n");  **fd = open("/dev/etx\_device", O\_RDWR);**          if(fd < 0) {                  printf("Cannot open device file...\n");                  return 0;          }            printf("Enter the Value to send\n");          scanf("%d",&number);          printf("Writing Value to Driver\n");  **ioctl(fd, WR\_VALUE, (int32\_t\*) &number);**            printf("Reading Value from Driver\n");  **ioctl(fd, RD\_VALUE, (int32\_t\*) &value);**          printf("Value is %d\n", value);            printf("Closing Driver\n");  **close(fd);**  } |

**Building Driver and Application**

* Build the driver by using Makefile (*sudo make*)
* Use below line in terminal to compile the user space application.

gcc -o test\_app test\_app.c

**Execution (Output)**

As of now, we have driver.ko and test\_app. Now we will see the output.

* Load the driver using sudo [insmod](https://linux.die.net/man/8/insmod) driver.ko
* Run the application (sudo ./test\_app)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
\*\*\*\*\*\*\*[WWW.EmbeTronicX.com\*\*\*\*\*\*\*](http://WWW.EmbeTronicX.com*******)

Opening Driver  
Enter the Value to send

* Enter the value to pass

25  
Writing Value to Driver  
Reading Value from Driver  
Value is 625  
Closing Driver

* Now check the value using dmesg

Device File Opened...!!!  
Value = 25  
Device File Closed...!!!

* Our value 25 was passed to the kernel, it was squared, and stored in the kernel.

This is the simple example using ioctl in driver. If you want to send multiple arguments, put those variables into a structure and pass the structure.