Kernel Internals

Santosh Sam Koshy, JD
C-DAC, Hyderabad
santoshsk@cdac.in

AGENDA

- Recapitulation of previous lecture
- Learning Outcomes for this topic
- Introduction
- ioctl implementation

RECAPITULATION

- In previous lecture, we learnt how to build a simple char device driver
- How to read from and write to the device
- Manipulating different kernel data structures

INTRODUCTION

- Only reading to and writing from the device is not enough
- Drivers need some hardware control mechnism also
- The ***ioctl*** method provides a mechanism to perform control actions on the device through the user space as well as in kernel space

Prototype of *ioctl*

- In user space, *ioctl* system call has following prototype:

  int ioctl (int fd, unsigned long cmd, …)

  - *fd* is file descriptor of the device file
  - *cmd* is used to select the behavior of the device
  - Third argument is an optional argument with no type check during compilation
- In kernel space, prototype for *ioctl* is:

  long (*ioctl) (struct file *filp, unsigned int cmd, unsigned long arp)

  - *inode* and *filp* pointers are the values corresponding to the file descriptor *fd*
  - *cmd* argument is passed from the user unchanged
  - Optional argument is passed in the form of an *unsigned long*
  - If invoking program is not passing third argument, the *arg* value received will be undefined
- In the file_operations structure, specify – 'unlocked_ioctl' to implement functionality

ioctl commands

- ***ioctl* commands** are represented using some numbers
- These numbers should be unique to prevent an unwanted operation on a device mistakenly:
  - To find a unique number see Documentation/userspace-api/ioctl/ioctl-number.rst
  - This text lists the magic numbers which is used throughout the kernel
- This number is 32 bits wide which includes 4 bitfields:
  - **type(magic number):** choose one number only and use through out your program. 8 bits
  - **Ordinal number:** sequential number, 8bits wide
  - **Direction:** direction of data transfer,if particular command involves data transfer. 2 bits wide
    - The possible values are: _IOC_NONE, _IOC_READ, _IOC_WRITE AND _IOC_READ|_IOC_WRITE
  - **Size:** the size of user data involved. It's 14 bits wide
- These are shared between the user application and the kernel driver.

Magic Numbers -Internally

Bits	Meaning
31-30 : 00 10 01 11	 no parameters: uses _IO macro read: _IOR write: _IOW read/write: _IOWR
29-16	size of arguments
15-8	ascii character supposedly unique to each driver
7-0	Ordinal Number/Unique Command Differentiation

Creating *ioctl* Commands

- IOCTL commands can be defined in a header file using special functions that aggregate the bit fields into unique commands
  - **_IO(type, nr):** command that has no argument
  - **_IOR(type, nr, datatype):** for reading data from driver
  - **_IOW(type, nr, datatype):** for writing data
  - **_IOWR(type, nr, datatype:** for bidirectional transfers
- This header file is required by the application and the kernel driver that uses the command
- Types and ordinal number is passed as argument and size of the datatype can be passed directly
  - DO NOT PASS – **sizeof(type) – the 3rd argument is already replaced by sizeof() type**

```
/* Using 'k' as magic number */
/* Refer Documentation/ioctl/ioctl-numbers.txt for free magic number */
#define serial_magic    'k'

/* Defining the ioctl commands */
#define serial_reset     _IO(serial_magic, 0)
#define serial_baud_set  _IOW(serial_magic, 1, int)
#define serial_status    _IOR(serial_magic, 2, char)
#define serial_control   _IOWR(serial_magic, 3, char)

#define serial_maxnr    4
~
```

ioctl – Kernel Level Implementation

```
long (*ioctl) (struct file *filp, unsigned int cmd, unsigned long arp)
{ //Implementation of ioctl at kernel level is usually a switch  statement based on command number
    Switch (cmd) {
        Case BAUDRATE:
            //Do something here to set the baud rate….
            Break;
        Case PARITY:
            //Do something here to set the parity ….
            Break;
        Case STOP BITS:
            //Do something here to set the stop bits….
            Break;
        Case All:
            //Do something here to set the all the data using a user defined structure..….
            Break;
        Default: The default returns -EINVAL(“Invalid argument”)
            //Error – No such command
            Break;
    }
}
```

Implementation of *ioctl* Commands at kernel level

- Simple switch case statements are used to implement at kernel level

```
unsigned int BAUD_VALUE;          /* for storing the baud value */
unsigned char STATUS, CONTROL;    /* used as a temporary buffer */
int retval;                       /* for storing return value   */

/* arg is the data sent from the user space */
switch(cmd)
{
        case serial_baud: // setting the baud rate of serial device
                retval = __get_user(BAUD_VALUE, (int __user *)arg);  /* *arg -> BAUD_VALUE */
                outw(BAUD_VALUE, address_baud_reg); /* baud value is written to device baud register */
                break;

        case status:    // getting the status of serial device
                STATUS = inb(address_status_reg);         /* status is read from the device status register */
                retval = __put_user(STATUS, (char __user *)arg); /* STATUS -> *arg */
                break;

        case serial_control:    // getting the status result
                retval = __get_user(CONTROL, (char __user *)arg); /* *arg -> CONTROL*/
                outb(CONTROL, address_control_reg); /* control bit writing to the device */
                if(retval == 0)         /* if writing to the device is successful */
                {
                        CONTROL = inb(address_control_reg); /* returning the current control value */
                        retval = __put_user(CONTROL, (char __user *)arg); /* CONTROL -> *arg */
                }
                break;
}
```

Optimised Data Transfer

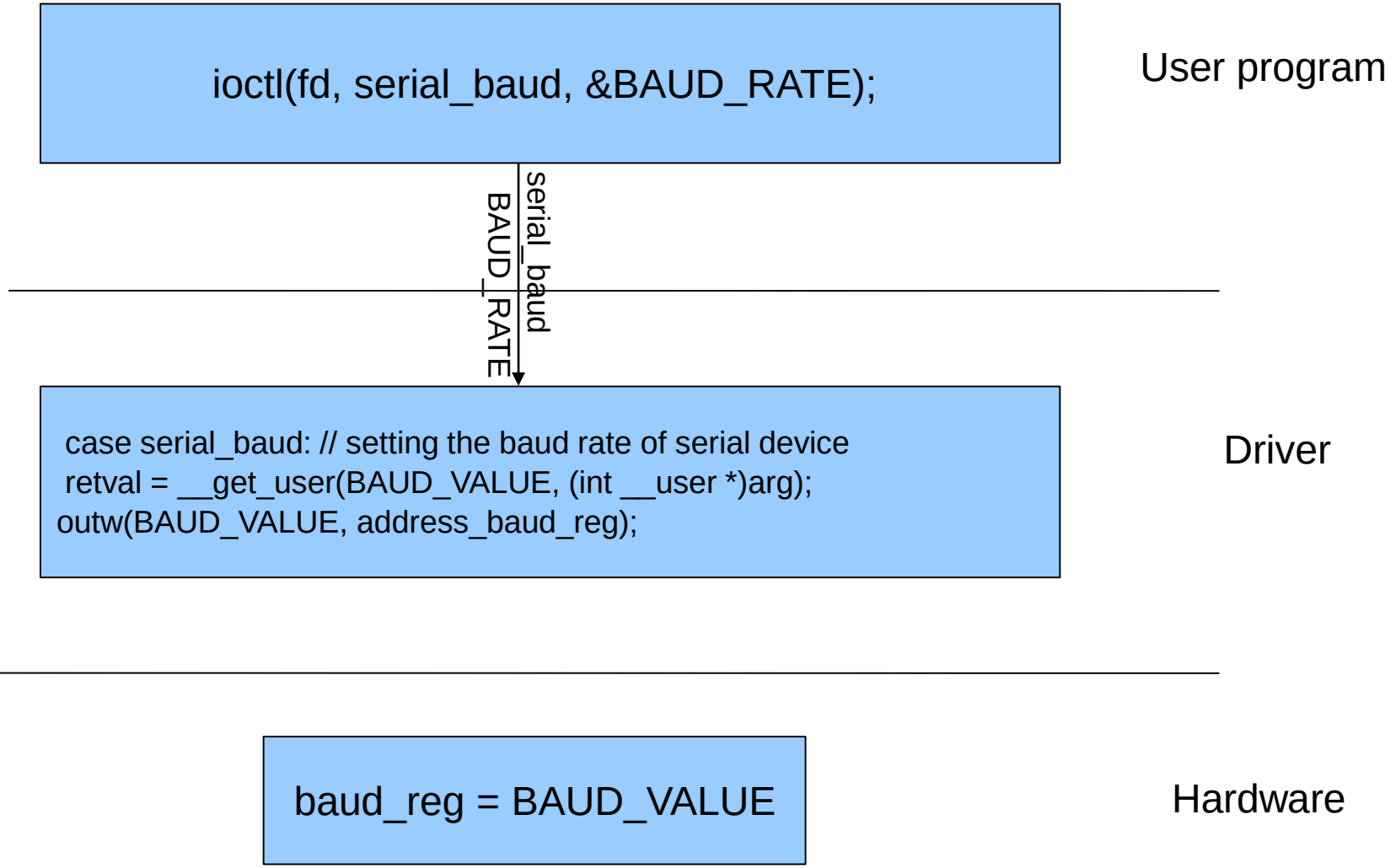
- Since in *ioctl*, data transfer is rarely more than 8 bytes; in such case following functions works well:
- put_user(data, ptr) and __put_user(data, ptr): this macro write data to user space
  - Relatively faster than *copy_to_user()*
  - Size of data transfer depends on type of *ptr* argument
  - *__put_user()* performs less checking
- Similarly, get_user(data, ptr) and __get_user(data, ptr)

Implementation of *ioctl* at user-space

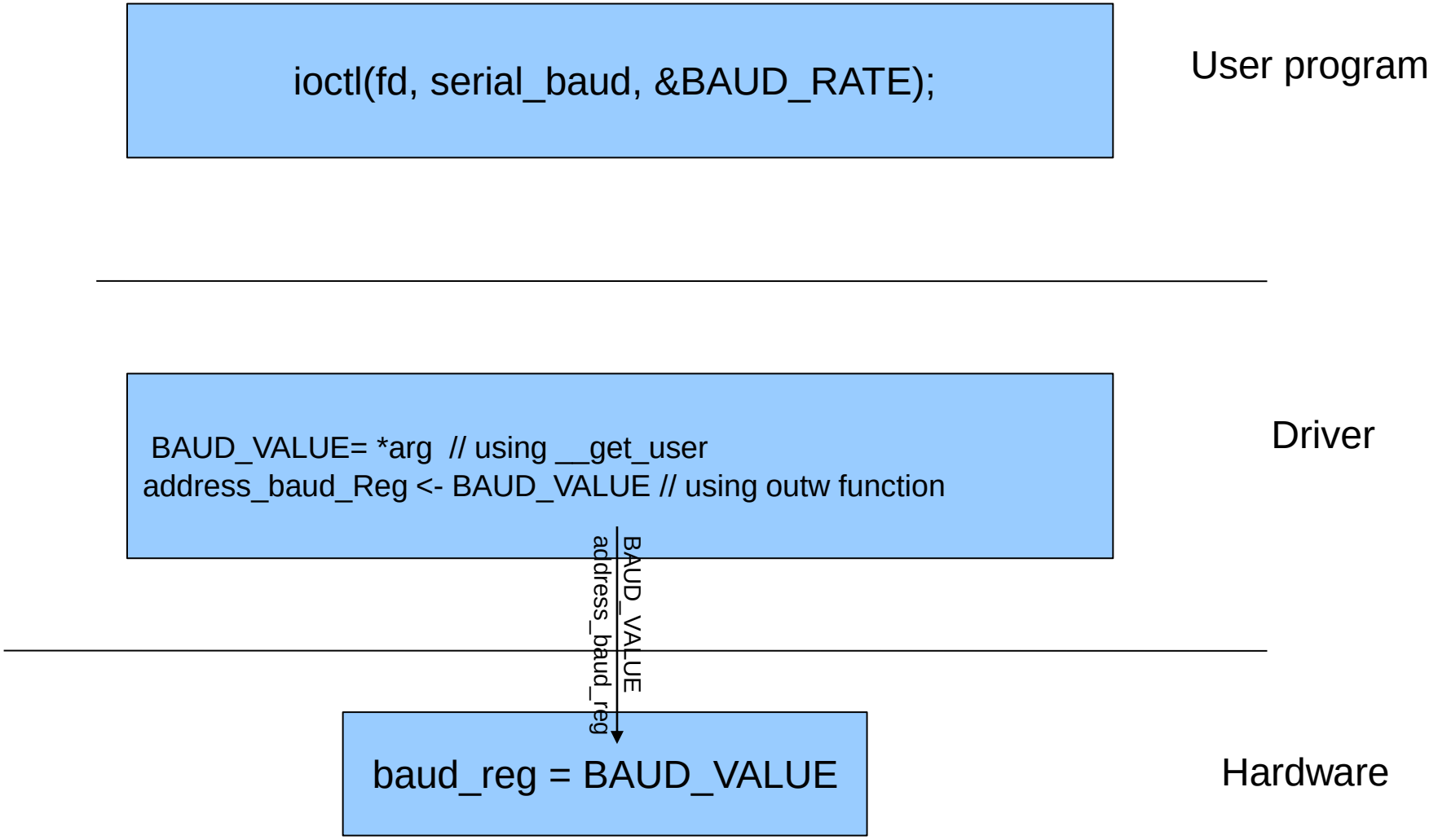
- It's just calling the function with appropriate cmd arguments

```
ioctl(fd, serial_baud, &BAUD_RATE);              /* sending the baud rate*/
retval = ioctl(fd, serial_status, &status);      /* getting the status of the serial device*/
retval = ioctl(fd, serial_control, &control);    /* sending the control bits and recieving
                                                  * the control status */
```

ioctl quick view



ioctl quick view



Thank you

Centre for Development of Advanced Computing,
Hyderabad