INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE CENTRATOM TRIESTE

The United Nations University

SMR/774 - 17

THIRD COLLEGE ON MICROPROCESSOR-BASED REAL-TIME CONTROL - PRINCIPLES AND APPLICATIONS IN PHYSICS
26 September - 21 October 1994

ADDITIONAL MATERIAL (II) TO LECTURES PRESENTED BY

Ulrich RAICH
C.E.R.N.-European Organization
For Nuclear Research
P.S. Division
CH- 1211 Geneva
SWITZERLAND

These are preliminary lecture notes, intended only for distribution to participants.

MAIN BUILDING STRADA COSTIERA, 11 TEL. 22401 TELEFAX 224163 TELEX 460392 ADRIATICO GUEST HOUSE VIA GRIGNANO, 9 TEL. 224241 TELEFAX 224531 TELEX 460449
MICROPROCESSOR LAB. VIA BEIRUT, 31 TEL. 224471 TELEFAX 224600 TELEX 460392 GALILEO GUEST HOUSE VIA BEIRUT, 7 TEL. 22401 TELEFAX 2240310 TELEX 460392

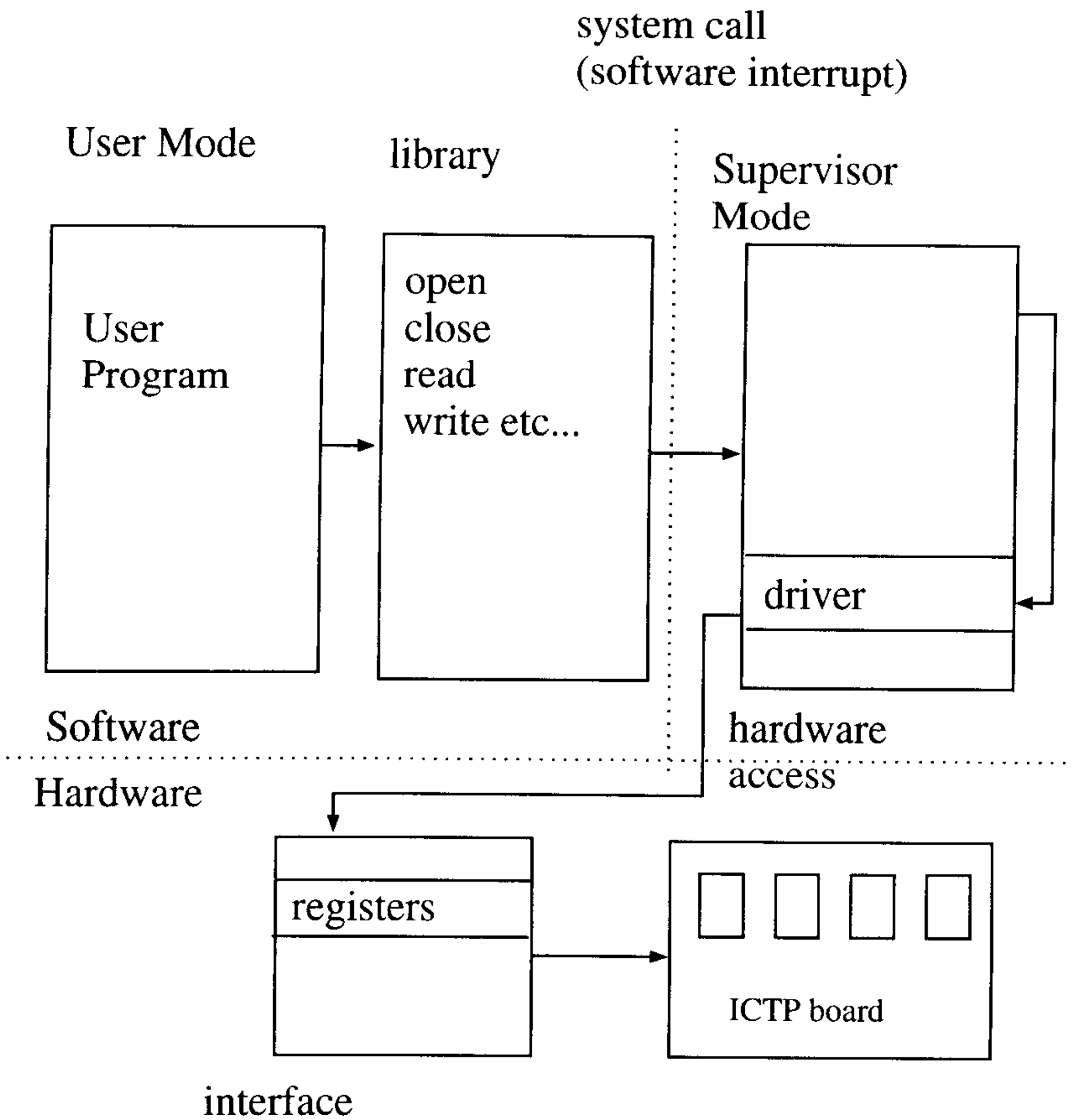
Problems when writing device drivers

- Device drivers are an integral part of the system kernel
- they execute in supervisor mode (see later)
- device driver writers do not have access to the standard libraries
- debugging problems (normal debugger does not work, printf does not work)
- there are timing constraints
- often the hardware (and software) documentation is missing

Advantages of Device Drivers

- Resource protection (access by multiple processes)
- Accessibility by any user
- Isolation of hardware intricacies into the driver (these problems are not seen by the application programmer)
- Possibility to treat interrupts

Device Driver access seen from the Application Program



Device Drivers seen from the Application Program

The device driver is accessed like a normal file:

- fd = **open**(/dev/ictp,O_RDONLY)
  or O_WRONLY or O_RDWR
  this opens the driver and returns a file descriptor
- ret_code = **read**(fd,buffer,nchars)
  allows to read nchar byte from the device into buffer
- ret_code = **write**(fd,buffer,nchars)
  writes out nchars bytes from the buffer to the device
- **ioctl**(fd,request,argp)
  int fd,request; char *argp;
  controls the device driver
- pos = **lseek**(fd,offset,whence)
  sets the file pointer
- **close**(fd) closes the driver

All drivers are collected in the /dev directory

ls -l /dev/ictp will give:

crw--w--w-1 1 root 31 8 Sep 94 /dev/ictp

indicates a character device driver

A typical application program acessing a driver

```
/***************************************************/
/* Try to run the colombo board from the parallel   */
/* interface using the ictp driver                  */
/* U. Raich 14.9.94                                 */
/***************************************************/
#include "/usr/include/stdio.h"
#include "/usr/include/fcntl.h"
#include <sys/ioctl.h>
#include "ictp.h"

void main()
{
  int fd,i,ret_code;
  unsigned long mode;
  unsigned char buffer[12];

/*
  open the device driver for writing
*/
  fd = open("/dev/ictp0",O_WRONLY);
  if (fd <  0) {
    perror ("Could not open ictp port:");
    exit(-1);
  }
  else
   printf("ictp port successfully opened for writing!\n");

/*
  in raw mode
  we must code data and chip select signals ourselves
*/
  mode = ICTP_MODE_RAW ;
  printf("setting mode : %d\n",mode);
  ret_code = ioctl(fd,ICTP_SET_MODE,mode);
  if (ret_code < 0)
    perror("ioctl");

  buffer[0]=0x1f;
  buffer[1]=0x17;
  buffer[2]=0x1f;

  buffer[3]=0x2f;
  buffer[4]=0x2b;
  buffer[5]=0x2f;
```

```
  buffer[6]=0x3f;
  buffer[7]=0x3d;
  buffer[8]=0x3f;

  buffer[9]=0x4f;
  buffer[10]=0x4e;
  buffer[11]=0x4f;

  if (write(fd,buffer,12) != 12)
    perror("after write ");

  close(fd);

}
```

Types of device Drivers

- **Block Device Drivers**
  Fixed size buffers.
  interaction via a system supplied buffer cache
  used for disk systems in conjunction with the file system
- **Character Device Drivers**
  We will look exclusivly at these
  Have a direct interaction with the hardware
  transfer data on a byte by byte basis
- **terminal drivers**
  character device drivers with special support for terminals
- **STREAM drivers**
  used for high speed serial communication
  e.g. networks

Major and Minor Device

Often an interface has several channels however a driver has only a single read and write routine

Usually the I/O card (type of I/O) is assigned a **major** number the channel a **minor** number

Drivers are accessed the same way as ordinary files (redirect, append, pipe ... also works on devices)

The "**special files**" are located in /dev

Naming convention: name of driver followed by minor number e.g ictp0

In order to create a special file:

mknod special file [b] [c] major minor e.g.
mknod /dev/ictp0 c 31 0

In our driver we need 3 different "reads".

- Read the switches
- Read the number of interrupts on IRQ5 and IRQ7
  => use 3 minor numbers

Installing the Device Driver into the System

2 possibilities:

- Link the device driver into the system at SysGen (system generation)

  —modify mem.c (a kernel routine calling each driver initialization routine) zu initialize the chips and register the driver with the system
  insert: mem_start = ictp_init(mem_start)

  —the driver must contain the ictp_init routine

  —recompile the kernel and install the new system

- Use **modules** package

The kernel contains hooks to dynamically install a device driver into the system. The modules package provides utility programs to perform the installation and removal.

- **insmod** ictp.o installs the ictp driver into the system
- **lsmod** lists all modules installed by insmod
- **rmmod** ictp removes the ictp driver from the system
- **ksyms** lists the exported kernel symbols

The driver must contain 2 routines:

- int **init_module**(void) (equivalent to ictp_init)
  initializes the chip and registers the driver
- void **cleanup_module**(void)
  unregisters the driver

Driver Layout

The driver consists of 2 files:

- **include file** (ictp.h) containing all definitions
  —register addresses
  —symbolic names for initialization bits
  —symbolic names for ioctl functions
  —symbolic names for ioctl arguments

The **driver code**

- installation and cleanup routine
- driver jump table:

```
static struct file_operations ictp_fops = {
        NULL,               /* lseek, not used in ictp driver */
        ictp_read,          /* read routine */
        ictp_write,         /* write routine */
        NULL,               /* readdir not used in ictp driver */
        NULL,               /* select function */
        ictp_ioctl,         /* ioctl driver control function */
        NULL,               /* mmap */
        ictp_open,          /* open function */
        ictp_close,         /* release function */
        NULL                /* fsync */
}
```

- code for each of the non NULL entries in the above table

Typical example of an include file

```
/*****************************************************/
/* Definitions of 8255 addresses and control bits   */
/* U. Raich 31.8.94                                  */
/*****************************************************/

#include <sys/ioctl.h>

#define ICTP_MAJOR            31
#define ICTP_NO                3

/*
 * defines for 8255 ports
 */

#define ICTP_A 0x300
#define ICTP_B 0x301
#define ICTP_C 0x302
#define ICTP_S 0x303

/*
 * defines ICTP status and control register bits
 */

#define ICTP_MODE_SELECT    0x80
#define ICTP_A_MODE_0       0x00
#define ICTP_A_MODE_1       0x20
#define ICTP_A_MODE_2       0x40

#define ICTP_B_MODE_0       0x00
#define ICTP_B_MODE_1       0x04

#define ICTP_INPUT_A        0x10
#define ICTP_OUTPUT_A       0x00
#define ICTP_INPUT_B        0x02
#define ICTP_OUTPUT_B       0x00
#define ICTP_INPUT_C_LOW    0x01
#define ICTP_OUTPUT_C_LOW   0x00
#define ICTP_INPUT_C_HIGH   0x04
#define ICTP_OUTPUT_C_HIGH  0x00

#define ICTP_AVAILABLE      1
#define ICTP_NOT_AVAILABLE  0

#define ICTP_SILENCE        0x09
#define ICTP_NOISE          0x08
#define ICTP_BUZZER_BIT     0x10
#define ICTP_BUZZER_ON         1
```

```
#define ICTP_BUZZER_OFF         0

#define ICTP_MODE_RAW           0
#define ICTP_MODE_SINGLE_DIGIT  1
#define ICTP_MODE_FULL_NUMBER   2

#define ICTP_BUSY               1
#define ICTP_FREE               0

#define ICTP_READ_SWITCHES      0
#define ICTP_READ_IRQ7_COUNT    1
#define ICTP_READ_IRQ5_COUNT    2

/*
  the ioctl codes:
*/
#define ICTP_SET_MODE           IOC_IN  | 0x0001
#define ICTP_GET_MODE           IOC_OUT | 0x0001
#define ICTP_SET_BUZZER         IOC_IN  | 0x0002
#define ICTP_GET_BUZZER         IOC_OUT | 0x0002
```

Sequence of steps to be taken in order to implement the device driver

- Implement the init_module and cleanup_module routines only and check if the driver can be installed into the system

How do we know if the install worked?

1.) put a **printk** (print on console) statement into the init_module code

2.) check with lsmod

```
/*
 * Implements the ICTP character device driver.
 * Create the device with:
 *
 * mknod /dev/ictp c 31 0
 *
 * - U. Raich
 * 13.3.94 : First version working with PC parallel printer
port
 *
 * Modifications:
 * 30.8.94 : U.R. complete rewrite for Manuel's board
 */


/* Kernel includes */

#include <linux/errno.h>
#include <linux/fs.h>
#include <linux/major.h>
#include <asm/segment.h>
#include <linux/kernel.h>
#include <linux/signal.h>
#include <linux/module.h>
#include <linux/sched.h>
#include <ictp.h>

#include <asm/io.h>
#include <asm/segment.h>
#include <asm/system.h>
```

```
/*
 * NB. we must include the kernel idenfication string in to
install the module.
 * See the Makefile for release.h
 */
#include "release.h"


extern int printk( const char* fmt, ...);

int init_module( void) {
        int i;
        unsigned char testvalue = 0;

        printk( "ictp:  init_module called\n");

/*
  register the device driver with the system
*/
        if (register_chrdev(HW_MAJOR, "ictp", &ictp_fops)) {
             printk("register_chrdev  failed:  goodbye  world
:-(\n");
          return -EIO;
        } else
          printk( "ictp: driver registered!\n");

        return 0;
}

void
cleanup_module( void) {
  int i,busy = 0;
  printk( "ictp:  cleanup_module called\n");

  for (i=0;i<ICTP_NO;i++)
    if (ictp_busy[i] == ICTP_BUSY)
       busy = 1;
  if (busy)
    printk("ictp: device busy, remove delayed\n");

  if (unregister_chrdev(HW_MAJOR, "ictp") != 0) {
    printk("cleanup_module failed\n");
  } else {
    printk("cleanup_module succeeded\n");
  }
}
```


12

Result: lsmod finds the module,

but the printk message did not appear anywhere

Questions: Who is right (lsmod or printk) ?

Where should the message come out ?

Does printk only work when the driver is compiled into the system but not with insmod?

Is there a problem with X-Windows?

Experiment: Try to compile the driver into the system and save new system on floppy. Boot from floppy.

Result: insmod message arrive on boot (on non X console because X is only started after initializing the drivers)

but cleanup_module message is still not visible!

but ... => we are actually installing the driver ok!!!

So... where do the messages go?

Advice from a Guru: The messages go to the console device (/dev/console) so try: date > /dev/console

This comes out on the "console window". However driver messages don't.

If nothing else helps, read the manual!

This gives the idea to poke around the /usr/adm area (see System Administrators Guide)

In /usr/adm/syslog I find my printk messages!!!

Changing the syslog configuration file (/etc/syslog.conf) allows me to **redirect kernel messages** from syslog to /dev/console!

Now printk works as expected

Situation after 2 lectures

- Driver include file is (at least partially) written
- Driver can be installed and uninstalled. We also know how to compile it into the kernel
- Driver jumptable exists but contains only NULLs.
- Hardware for reading switches and writing displays is understood
- Read routine (for switches, not interrupts) exists but only in non driver form
- A simple write display routine (non driver form) exists
- printk works, some debugging is therefore possible

Next Steps: Write the displays with driver

- Implement the open and close routines
- Implement read and write routines
- Put the entry points of above routines into the jumptable
- Put debugging information into all routines
- Design decision: Only a single process may access the driver at a time => return "busy" error if a second open is attempted