COMP 4735: Operating Systems Concepts

Lesson 14 - part 2 - Par Driver Example

Rob Neilson

rneilson@bcit.ca

Parallel Port Driver

par driver

We will modify mem so that it interacts with the parallel port

Recall:

- parallel port lives at base address `0x378`
- it has 3 registers:
  - data: `BASE (0x378)`
  - status: `BASE+1`
  - control: `BASE+2`

- we will start by ignoring status and control … we just want to get some signals going out of the port

par driver -2

Summary of the changes we need to make:

- don't need our byte of RAM (memory_buffer)
  - we will just send the byte out to 0x378
- need to reserve the parallel port address for use by our driver
  - to avoid conflicts with other driver
- need to replace memory_buffer accesses with BASE+0 reads and writes

par – reserving the port - 1

- IO port address are fixed address
- the hardware will likely have jumpers that indicate it's address in IO space
- if we want to write to the parallel port we have to reserve it's port address

A typical */proc/ioports* file:

```
0000-001f : dma1
0020-003f : pic1
0040-005f : timer
0060-006f : keyboard
0080-008f : dma page reg
00a0-00bf : pic2
00c0-00df : dma2
00f0-00ff : fpu
0170-0177 : ide1
01f0-01f7 : ide0
02f8-02ff : serial(set)
0300-031f : NE2000
0376-0376 : ide1
03c0-03df : vga+
```

par – reserving the port - 2

- use check_region to see if any other drivers are using the port addresses that we want
- use request_region to reserve the ports for our driver

```
int check_region(unsigned long start, unsigned long len);

struct resource *request_region(unsigned long start,
                                unsigned long len,
                                char *name);

void release_region(unsigned long start, unsigned long len);
```

par – reserving the port - 3

we will modify the mem_init function to include the following:

```
/* Registering port */
port = check_region(0x378, 1);
if (port) {
   printk("<1>par: cannot reserve 0x378\n");
   result = port;
   goto fail;
}
request_region(0x378, 1, "par");
```

par – removing the module

- we also need to modify our release function to free the port when our driver is unloaded

```
/* Make port free! */
if (!port) {
   release_region(0x378,1);
}
```

par – reading from the device

- read from a port with the inb() kernel function
- `par_buffer` is just a local var, because we are simply going to copy it to user space

```
/* Reading port */
char par_buffer;

par_buffer = inb(0x378);
```

par – writing to the device

- write to a port with the outb() kernel function
- `par_buffer` is, again, just a local var to contain the output byte, which we will copy from user

```
/* Writing to the port */

outb(par_buffer,0x378);
```

par driver - revised include's and declarations - 1

```
/* Necessary includes for device drivers */

/* ...all includes the same as mem, except... */

#include <asm/io.h>      /* inb, outb */

/* Declaration of par.c functions */

int par_open      (struct inode *inode,
                   struct file *filp);
int par_release   (struct inode *inode,
                   struct file *filp);
ssize_t par_read  (struct file *filp, char *buf,
                   size_t count, loff_t *f_pos);
ssize_t par_write (struct file *filp, char *buf,
                   size_t count, loff_t *f_pos);

int par_major = 61;

/* Control variable for memory reservation */
/* of the parallel port */

int port;
```

par.c init function

```
int par_init(void) {
   int result;

   result = register_chrdev(par_major,
                            “par", &par_fops);
   if (result < 0) {
      printk(
      "<1>par: cannot get major %d\n", par_major);
      return result;
   }

   /* Registering port */
   port = check_region(0x378, 1);
   if (port) {
      printk("<1>par: cannot reserve 0x378\n");
      result = port;
      goto fail;
   }
   request_region(0x378, 1, "par");

   printk("<1>Inserting module: par\n");
   return 0;

fail:
   par_exit();
   return result;
}
```

par.c exit function

```
void par_exit(void) {

   /* Freeing the major number */
   unregister_chrdev(par_major, “par");

   /* Make port free! */
   if (!port) {
      release_region(0x378,1);
   }
   printk("<1>Removing module: par\n");
}
```

par.c open/close functions

```
int par_open (struct inode *inode,
              struct file *filp)   {
   /* Success */
   return 0;
}


int par_release (struct inode *inode,
                 struct file *filp)   {
   /* Success */
   return 0;
}
```

par.c read function - 1

```
ssize_t par_read (struct file *filp, char *buf,
                  size_t count, loff_t *f_pos)  {

   /* space for the byte coming in from device */
   char par_buffer;

   /* read the device */
   par_buffer = inb(0x378);

   /* Transfer data to user space */
   copy_to_user(buf, &par_buffer, 1);

   /* Changing reading position as needed */
   if (*f_pos == 0) {
      *f_pos+=1;
      return 1;
   } else {
      return 0;
   }
}
```

par.c write function

```
ssize_t par_write (struct file *filp, char *buf,
                   size_t count, loff_t *f_pos) {
   char *tmp;

   /* space for the byte we read from the dev */
   char par_buffer;

   tmp=buf+count-1;
   copy_from_user(&par_buffer, tmp, 1);

   /* Write to the port */
   outb(par_buffer,0x378);

   return 1;
}
```

a user app to test our driver

```
#include <stdio.h>
#include <unistd.h>

int main() {
   unsigned char byte,dummy;
   FILE * PARPORT;

   /* Opening the device par */
   PARPORT=fopen("/dev/par","w");

   /* Turn off buffering of file i/o, which is */
   /* default operation for char stream devices */
   setvbuf(PARPORT,&dummy,_IONBF,1);

   led=1;

   while (1) {
      /* Writing to the parallel port */
      /* to turn on a LED */
      printf("Led value is %d\n",led);
      fwrite(&led,1,1,PARPORT);
      sleep(1);

      /* Updating the led value */
      led<<=1;
      if (led == 0)
          led = 1;
    }

 fclose(PARPORT);
 }
```

End of slides