I2C 16x2 LCD and TCP Communication Project

Team-5

July 13, 2025

1 Project Overview

This project integrates a 16x2 LCD display with an I2C interface to a Raspberry Pi, controlled by a custom Linux kernel module, and uses TCP communication to display messages sent from an x86 machine. The Raspberry Pi acts as a TCP server, receiving messages from a TCP client running on the x86 machine. These messages are passed to the kernel module via IOCTL calls and displayed on the LCD. A device tree configures the I2C hardware during boot. This document details the hardware setup, software components, working mechanism, device tree configuration, and execution steps.

1.1 Objectives

- Display messages sent from an x86 machine (TCP client) on a 16x2 LCD connected to a Raspberry Pi (TCP server) via I2C.
- Use a Linux kernel module to control the LCD and handle user-space communication via IOCTL.
- Configure the I2C hardware using a device tree for seamless integration with the Linux kernel.

1.2 Components

- **Hardware**:
  - Raspberry Pi (e.g., Raspberry Pi 4) running Raspberry Pi OS.
  - 16x2 LCD with PCF8574 I/O expander (I2C address: 0x27).
  - x86 machine running the TCP client.
  - Network (Wi-Fi or Ethernet) connecting the Raspberry Pi and x86 machine.
- **Software**:
  - Linux kernel module (`my_driver.c`): Controls the LCD via I2C and provides a character device interface.
  - TCP server (`tcp_server.c`): Runs on the Raspberry Pi, receives messages, and sends them to the kernel module.

- TCP client (`tcp_client.c`): Runs on the x86 machine, sends messages to the Raspberry Pi.
- Device tree overlay: Configures the I2C bus and LCD hardware.

2 Hardware Setup

2.1 Raspberry Pi and LCD Connection

- **LCD**: A 16x2 character LCD based on the HD44780 controller, interfaced via a PCF8574 I/O expander for I2C communication.
- **Pin Connections**:
  - VCC: Connected to 5V (Pin 2 or 4 on Raspberry Pi).
  - GND: Connected to GND (Pin 6 or 14).
  - SDA: Connected to GPIO 2 (Pin 3, I2C1 SDA).
  - SCL: Connected to GPIO 3 (Pin 5, I2C1 SCL).
- **I2C Address**: 0x27 (configurable via A0–A2 jumpers on the PCF8574).
- **Contrast Adjustment**: A potentiometer on the I2C module adjusts contrast until two rows of rectangles appear.
- **Backlight**: Enabled by default via a jumper on the I2C module.

2.2 Network Setup

- The Raspberry Pi is connected to a network via Ethernet or Wi-Fi.
- IP address: 192.168.0.140 (as specified in `tcp_client.c`).
- The x86 machine is on the same network, running the TCP client.

3 Software Components

3.1 Device Tree Configuration

- **Purpose**: Configures the I2C1 bus and registers the LCD as an I2C device at address 0x27.
- **Device Tree Overlay** (`i2c-lcd-overlay.dts`):

```
/dts-v1/;
/plugin/;

/ {
    compatible = "brcm,bcm2835";

    fragment@0 {
        target = <&i2c1>;
        __overlay__ {
```

```
            #address-cells = <1>;
            #size-cells = <0>;
            status = "okay";

            lcd@27 {
                compatible = "team-5,rg1602a-lcd";
                reg = <0x27>;
                display-height-chars = <2>;
                display-width-chars = <16>;
                i2c-expander = "pcf8574";
            };
        };
    };
};
```

- **Compilation and Loading**:
  1. Compile: `dtc -@ -I dts -O dtb -o i2c-lcd-overlay.dtb i2c-lcd-overlay.dts`
  2. Copy to `/boot/overlays/`: `sudo cp i2c-lcd-overlay.dtb /boot/overlays/`
  3. Enable in `/boot/config.txt`: Add `dtoverlay=i2c-lcd`
  4. Reboot: `sudo reboot`
- **Boot Process**: The bootloader loads the kernel image and `.dtb` file, enabling the I2C1 bus and registering the LCD with the compatible string `"team-5,rg1602a-lcd"`.

3.2 Kernel Module (`my_driver.c`)

- **Purpose**: Initializes the LCD, registers a character device (`/dev/lcd_ioctl`), and handles IOCTL calls to display messages.
- **Key Functions**:
  - `lcd_init_display`: Initializes the LCD in 4-bit mode with commands:
    - 0x33: Initializes in 8-bit mode.
    - 0x32: Switches to 4-bit mode.
    - 0x28: Sets 2 lines, 5x7 matrix.
    - 0x0C: Display on, cursor off.
    - 0x06: Cursor increment mode.
    - 0x01: Clear display.
  - `lcd_write`: Sends data in 4-bit mode, splitting bytes into high and low nibbles.
  - `lcd_strobe`: Toggles the ENABLE bit to latch data.
  - `lcd_message`: Writes a string to a specified line (0x80 for line 1, 0xC0 for line 2).

  - `lcd_ioctl`: Handles IOCTL commands (`LCD_IOCTL_LINE1`, `LCD_IOCTL_LINE2`) to display user-space strings.
  - `lcd_i2c_probe`: Initializes the LCD, registers the character device, and displays default messages ("GOOD AFTERNOON!" and "HELLO MOHAN").
  - `lcd_i2c_remove`: Clears the LCD and cleans up resources.
- **Note**: The module incorrectly uses `platform_driver_register`; it should use `i2c_add_driver` for I2C drivers.

```
#include <linux/module.h>
#include <linux/i2c.h>
#include <linux/delay.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/cdev.h>
#include <linux/device.h>

// Device and class names
#define DEVICE_NAME "lcd_ioctl"
#define CLASS_NAME  "lcd"

// IOCTL commands
#define LCD_IOCTL_MAGIC  'M'
#define LCD_IOCTL_LINE1  _IOW(LCD_IOCTL_MAGIC, 1, char *)
#define LCD_IOCTL_LINE2  _IOW(LCD_IOCTL_MAGIC, 2, char *)

// ... (rest of the code as provided)
static int __init lcd_driver_init(void)
{
    pr_info("lcd_driver: registering I2C driver\n");
    return i2c_add_driver(&lcd_driver); // Corrected from platform_driver_register
}

// ... (rest of the code)
```

3.3 TCP Server (`tcp_server.c`)

- **Purpose**: Runs on the Raspberry Pi, listens for TCP connections on port 8080, receives two strings, and passes them to the kernel module via IOCTL.
- **Operation**:
  - Opens `/dev/lcd_ioctl`.
  - Creates a TCP socket, binds to `INADDR_ANY` on port 8080, and listens for connections.
  - Accepts a client connection and receives two messages (up to 16 characters each).

- Sends messages to the kernel module using `LCD_IOCTL_LINE1` and `LCD_IOCTL_LINE2`.
- Sends an acknowledgment to the client.

```c
#include <stdio.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define DEVICE "/dev/lcd_ioctl"
#define LCD_IOCTL_MAGIC 'M'
#define LCD_IOCTL_LINE1 _IOW(LCD_IOCTL_MAGIC, 1, char *)
#define LCD_IOCTL_LINE2 _IOW(LCD_IOCTL_MAGIC, 2, char *)
#define PORT 8080
#define BUFFER_SIZE 17

// ... (rest of the code as provided)
```

3.4 TCP Client (`tcp_client.c`)

- **Purpose**: Runs on the x86 machine, sends two user-input strings to the Raspberry Pi for display on the LCD.
- **Operation**:
  - Creates a TCP socket and connects to the Raspberry Pi (IP: 192.168.0.140, port: 8080).
  - Prompts the user for two strings (max 16 characters each).
  - Sends the strings to the server and receives an acknowledgment.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define SERVER_IP "192.168.0.140"
#define PORT 8080
#define BUFFER_SIZE 17

// ... (rest of the code as provided)
```

4 Working Mechanism

4.1 Data Flow

1. **Boot and Initialization**:
   - The Raspberry Pi boots, loading the kernel and device tree blob (`.dtb`).
   - The device tree enables the I2C1 bus and registers the LCD at address 0x27.
   - The kernel module is loaded (`insmod my_driver.ko`), matching the device tree's `compatible` string (`"team-5,rg1602a-lcd"`).
   - The `lcd_i2c_probe` function initializes the LCD and creates `/dev/lcd_ioctl`, displaying "GOOD AFTERNOON!" and "HELLO MOHAN".
2. **TCP Communication**:
   - The TCP server (`tcp_server`) runs on the Raspberry Pi, listening on port 8080.
   - The TCP client (`tcp_client`) runs on the x86 machine, sending two strings to the Raspberry Pi.
3. **LCD Display**:
   - The server receives the strings and uses IOCTL calls to pass them to the kernel module.
   - The kernel module writes the strings to the LCD via I2C, updating lines 1 and 2.
   - The server sends an acknowledgment to the client.

4.2 Execution Steps

1. **Prepare Raspberry Pi**:
   - Enable I2C: `sudo raspi-config` Interfacing Options I2C Enable.
   - Install kernel headers: `sudo apt-get install raspberrypi-kernel-headers`.
   - Compile and load the device tree overlay.
2. **Compile and Load Kernel Module**:
   - Compile: `make -C /lib/modules/$(uname -r)/build M=$(pwd) modules`.
   - Load: `sudo insmod my_driver.ko`.
   - Verify: `ls /dev/lcd_ioctl`, `i2cdetect -y 1`.
3. **Compile and Run TCP Programs**:
   - Compile: `gcc tcp_server.c -o tcp_server`, `gcc tcp_client.c -o tcp_client`.
   - Run server on Raspberry Pi (via SSH): `./tcp_server`.
   - Run client on x86 machine: `./tcp_client`.

4. **Test**:
   - Enter two strings in the client (e.g., "Welcome to RPi" and "LCD Display Test").
   - Verify the strings appear on the LCD.
   - Check the client terminal for the server's acknowledgment.

5 Example Output

- **On Boot (LCD)**:

```
GOOD AFTERNOON!
HELLO MOHAN
```

- **Client Input (x86 Machine)**:

```
Enter Line1 for LCD (max 16 chars): Welcome to RPi
Enter Line2 for LCD (max 16 chars): LCD Display Test
Server: Messages received and displayed
```

- **LCD Output**:

```
Welcome to RPi
LCD Display Test
```

- **Server Terminal (RPi)**:

```
Server listening on port 8080...
Client connected
Received Line1: Welcome to RPi
Received Line2: LCD Display Test
```

6 Debugging and Troubleshooting

- **I2C Issues**:
  - Verify I2C address: `i2cdetect -y 1`.
  - Check wiring and contrast adjustment.
  - Ensure 5V or 3.3V logic levels match.
- **Kernel Module Errors**:
  - Check `dmesg` for errors: `dmesg | tail`.
  - Correct `platform_driver_register` to `i2c_add_driver`.
  - Verify `compatible` string in device tree.

- **TCP Issues**:
  - Verify IP address: `ping 192.168.0.140`.
  - Check port 8080: `sudo netstat -tuln | grep 8080`.
  - Allow port: `sudo ufw allow 8080`.
- **IOCTL Errors**:
  - Ensure `/dev/lcd_ioctl` exists: `ls /dev/lcd_ioctl`.
  - Set permissions: `sudo chmod 666 /dev/lcd_ioctl`.

7 Future Enhancements

- Add support for custom LCD characters using CGRAM.
- Handle multiple TCP clients using `fork` or threads.
- Implement retry mechanisms and timeouts for TCP communication.
- Enable dynamic IP discovery using mDNS.
- Add IOCTL command for backlight control.

8 Conclusion

This project demonstrates the integration of embedded systems, Linux kernel programming, and networking. The Raspberry Pi uses a device tree to configure the I2C LCD, a kernel module to control it via IOCTL, and a TCP server-client architecture to enable remote messaging. The system is robust, scalable, and suitable for applications like remote displays or IoT status panels.