Birla Institute of Technology and Sciences, Pilani

January, 2025

Report By:
Group 10

Morse Code Decoding Device Driver

Shourya Kothari 2021B3AA1526G (33%)
Satyam Shukla 2022A3PS1158H (33%)
Prachi Raj 2021B5AA2188G (34%)

Morse Code Device Driver Analysis Report

Overview

This report analyzes a Linux device driver implementation for decoding Morse code input. The system consists of three main components:

1. A kernel module (`morse_driver.c`) that registers a character device to handle Morse code input
2. A header file (`morse_driver.h`) that defines IOCTL commands and timing constants
3. A user-space application that reads Morse code from a serial port and forwards it to the kernel module(`morse_send.c`)

The workflow creates a complete system for receiving Morse code signals from a serial device, decoding them, and writing the decoded output to a file.

Morse Code Mapping

The system uses the following Morse code mapping for the English alphabet:

A: .-	J: .---	S: ...
B: -...	K: -.-	T: -
C: -.-.	L: .-..	U: ..-
D: -..	M: --	V: ...-
E: .	N: -.	W: .--
F: ..-.	O: ---	X: -..-
G: --.	P: .--.	Y: -.--
H:	Q: --.-	Z: --..
I: ..	R: .-.	

….. : space

—-- : new line

Kernel Module Analysis (`morse_driver.c`)

Key Components

1. **Character Device Registration**: Registers a character device with dynamic major number allocation.
2. **Data Structures**:
   - Morse code mapping array
   - Input buffer for storing Morse code
   - Mutex for thread safety
   - Timer for periodic processing
   - Work queue for deferred processing
3. **Core Functionality**:
   - IOCTL interface for user-space communication
   - Morse code decoding logic
   - File I/O for storing decoded results

Workflow

1. **Initialization** (`morse_driver_init`):
   - Registers a character device
   - Initializes work queue and timer
   - Sets up periodic timer callback
2. **IOCTL Handling** (`morse_ioctl`):
   - Receives Morse code input from user space
   - Stores input in kernel buffer
   - Schedules work for processing
3. **Decoding Process** (`morse_work_handler`):
   - Parses Morse code symbols separated by spaces
   - Maps each Morse sequence to its corresponding letter
   - Concatenates decoded letters
   - Writes result to a file
4. **Cleanup** (`morse_driver_exit`):
   - Cancels pending timer and work
   - Frees allocated memory
   - Unregisters the character device

Header File Analysis (`morse_driver.h`)

The header file defines:

1. **IOCTL Commands**:
   - `MORSE_IOC_RESET`: Command to reset the Morse input buffer
   - `MORSE_IOC_SEND`: Command to send Morse code to the driver
2. **Timing Constants**:
   - `DOT_DURATION`: 200ms (short press)
   - `DASH_DURATION`: 600ms (long press)

User-Space Application Analysis

The user application serves as a bridge between the serial device and the kernel module:

Workflow

1. **Initialization**:
   - Opens the Morse driver device (`/dev/morse_driver`)
   - Opens and configures the serial port (`/dev/ttyUSB0`)
2. **Main Loop**:
   - Continuously reads data from serial port
   - Cleans and processes the received data
   - Forwards valid Morse code to the kernel module via IOCTL
   - Provides user feedback via console messages

Design Considerations

Strengths

1. **Separation of Concerns**:
   - Kernel module handles decoding logic
   - User application handles I/O and communication
2. **Thread Safety**:
   - Uses mutex for synchronization

    - Defers processing to work queue
3. **Error Handling**:
    - Includes checks for memory allocation
    - Handles file I/O errors

Areas for Improvement

1. **Error Recovery**:
    - Limited handling of invalid Morse code (returns '?')
    - Could benefit from more robust recovery mechanisms
2. **Configuration**:
    - Hardcoded file path for output
    - Could be made configurable via IOCTL
3. **Concurrency Model**:
    - Uses a timer that fires every 200ms regardless of input
    - Could adopt an event-driven approach

Technical Implementation Details

Morse Code Decoding Logic

The decoding algorithm:

1. Parses input string character by character
2. Builds Morse symbols (dots and dashes) until a space is encountered
3. Looks up the completed symbol in the Morse code array
4. Appends the corresponding letter to the output
5. Writes the complete decoded string to a file

Thread Synchronization

The module uses:

- A mutex (`morse_mutex`) to protect the shared input buffer
- A work queue to defer processing to a safe context
- A flag (`decoding_done`) to prevent redundant processing

File I/O

The decoded output is written to `/home/prachi/Downloads/os_project_copy/morse_driver.txt` using kernel file operations:

- `filp_open` to open the file
- `kernel_write` to write data
- `filp_close` to close the file

Conclusion

The Morse code device driver demonstrates an effective integration of kernel and user-space components to create a functional input system. The implementation showcases important Linux kernel programming concepts including character devices, IOCTL communication, work queues, and timer management.

The modular design allows for potential extensions, such as supporting additional character sets or implementing more advanced error correction mechanisms.

REFERENCES

Oleg Kutkov - simple linux character device driver-
https://olegkutkov.me/2018/03/14/simple-linux-character-device-driver/#:~:text=A%20character%20device%20is%20one%20of%20the%20simplest,any%20data%2C%20byte%20by%20byte%2C%20like%20a%20stream.

Writing our first linux driver - youtube
https://www.youtube.com/watch?v=hMsA1bA1Upk