📚 Technical Concepts Documentation:

Automotive UI Enhancements

Project: Real-Time Automotive Control, Monitoring, and Visualization System

Date: October 25, 2024

Summary: This document outlines the core concepts and technologies implemented to modernize the ncurses-based Automotive Control System User Interface (UI). The focus of this development iteration was on introducing **Unicode support** and **color-coded symbolic indicators** to significantly enhance the dashboard's visual clarity and immediate status recognition.

1. Core Concept: Wide Character Support

The primary technical shift involved adopting wide characters to allow the UI to display rich Unicode symbols instead of standard ASCII text.

1.1. Ncurses Wide-Character Integration

Header Inclusion: The standard ncurses header was replaced with the wide-character specific version to ensure all necessary function prototypes were correctly declared:

*#include <ncursesw/ncurses.h>*

Wide Functions: All status printing functions were updated to their wide-character counterparts:

*mvaddwstr();*

Wide Character Literals: All Unicode symbols were defined using the `L"` prefix (e.g., `L"\u26A0"` for the warning sign).

1.2. Locale Configuration

To ensure the terminal can correctly interpret and render the multi-byte UTF-8 encoded symbols, the program's locale was configured dynamically:

setlocale(LC\_ALL, "");

The empty string tells the program to use the environment's default locale (typically UTF-8).

2. Enhanced Data Visualization and Status Feedback

The sensor data dashboard (`engine\_on\_screen`) was restructured to use color and symbols for instantaneous status updates.

2.1. Standardized Color Coding

New color pairs were defined to establish a visual hierarchy for status severity.

2.2. Symbolic Representation of Status

Key sensor readings and control statuses were mapped to specific Unicode symbols for immediate visual communication, eliminating the need to read descriptive text first.

|  |  |  |  |
| --- | --- | --- | --- |
| Data Indicator | Logic (Condition) | Unicode Symbol | Symbol Meaning |
| Low Fuel | fuel\_status == -1 | ⚠️ | Alert/Warning |
| Fuel Full | fuel\_status == 0 | ⛽ | Fuel Pump |
| High Speed | speed > 100 RPM | ↓ | Decrease speed |
| Reverse Camera | Gear pos == 6 | ⮪ | Reverse Direction |
| Object Detected | Gear pos == 6 | ⚠️ | Hazard Detected |
| Airbag Status | Crash Happen | ✔ or ✖ | Airbag Deploy or not |

3. Centralized Data Structure Management (`sensor.h`)

To maintain code consistency, reduce redundancy, and manage the complexity of Inter-Process Communication (IPC), a single header file, `**sensor.h**`, was created to define all core data structures.

**Single Source of Truth**: `sensor.h` is included in all necessary source files( sensor.c, subsystem.c, ui.c and server.c) guaranteeing a consistent definition across the entire system.

**Structure Definitions**: The file defines the entire system architecture:

**1.** ecu**\_**sensor: Defines all vehicle inputs/readings\*\* (e.g., `engine\_temp`, `fuel\_level`).

2. ecu\_control: Defines all system outputs/commands (e.g., `fan\_status`, `airbag`).

3. ECU(Shared Structure): Encapsulates the `sensor` and `control` structures along with the mandatory **pthread\_mutex\_t lock**; for synchronization.

4. Global Pointer: The declaration **ECU\* shm\_ecu**; is placed in `sensor.h`, providing a system-wide pointer to the shared memory segment.

4. Inter-Process Communication (IPC) and Synchronization

The system uses efficient mechanisms for real-time data exchange.

Shared Memory: Standard POSIX calls (`shmget`, `shmat`) are used to map the centralized ECU structure into the memory space of both the sensor and UI processes.

Mutex Synchronization: A Pthread Mutex embedded within the ECU structure manages access. The UI process explicitly locks the mutex before reading data and unlocks it immediately, preventing data corruption and ensuring real-time integrity.

5. Compilation Dependency

The adoption of the wide-character library requires a specific flag during the build process.

Required Linker Command:

gcc UI.c -o UI -lncursesw -lpthread

The flag `-**lncursesw**` is mandatory for the linker to find the function definitions for all wide-character ncurses calls.