Software Requirements Specification (SRS)

Project Title:

Real-Time Automotive Control and Monitoring System

Platform:

Linux (C Language using IPC, Multithreading, Signal Handling)

1. Introduction

1.1 Purpose

This document outlines the software requirements for a real-time automotive control and monitoring system. The system simulates various vehicle subsystems such as engine, gear, fuel, and safety sensors, and enables concurrent monitoring and control using multithreading, shared memory, and signal handling.

1.2 ScopeThe system is designed to:

- Simulate ECU sensor data (engine temperature, speed, gear, fuel level, etc.)
- Control vehicle subsystems (ignition, brake, fan, airbag, etc.)
- Use shared memory for inter-process communication between sensor and control modules
- Handle asynchronous events using signals (e.g., emergency stop)
- Provide real-time updates and fault tolerance

Definitions, Acronyms, and Abbreviations

- **ECU: Electronic Control Unit**
- **IPC: Inter-Process Communication**
- **RTOS: Real-Time Operating System**
- **POSIX: Portable Operating System Interface**
- **UI: User Interface**

2. Functional Requirements

ID	Function	Description
FR1	Sensor Data Acquisition	Acquire engine temperature, speed, gear, fuel level, etc., using threads.
FR2	Shared Memory Communication	Share sensor and control data between processes using shared memory.
FR3	Signal Handling	Handle signals like SIGUSR1 for ignition off and emergency stop.
FR4	Control Logic Execution	Update control parameters based on sensor values (e.g., airbag, fan, brake).
FR5	Real-Time Monitoring	Continuously display sensor and control data in the subsystem process.
FR6	Fault Tolerance	Ensure safe shutdown and cleanup of shared memory on termination.

3. Non-Functional Requirements

- **Concurrency: Use POSIX threads for parallel sensor simulation.**
- **Synchronization: Use mutex locks to avoid race conditions.**
- **Reliability: Handle signal interruptions and ensure memory cleanup.**
- **Efficiency: Maintain low CPU usage and minimal latency.**
- **Scalability: Easily extendable to include more sensors or control logic.**

4. Software and Hardware Requirements

4.1 Software Requirements

- **OS: Ubuntu Linux**
- **Compiler: GCC**
- **Tools: shmget, shmat, signal, pthread, ipcs, ipcrm**
- **Language: C**

4.2 Hardware Requirements

- Simulated environment (no physical sensors required)
- Terminal-based execution

5. System Overview (Pocess-Based)

Process Flow:

1.) Main ECU Process:

- **Initializes shared memory**
- **Starts engine sensor thread**
- **Handles signal for ignition off**

2.) Subsystem Process:

- **Reads shared memory**
- **Displays sensor and control data**
- **Terminates on ignition off**

3.) IPC Mechanism:

- **Shared Memory:**

- **ecu_sensor structure for sensor data**
- **ecu_control structure for control data**

- **Signal Handling:**

  SIGUSR1: Used to turn off ignition and stop threads safely

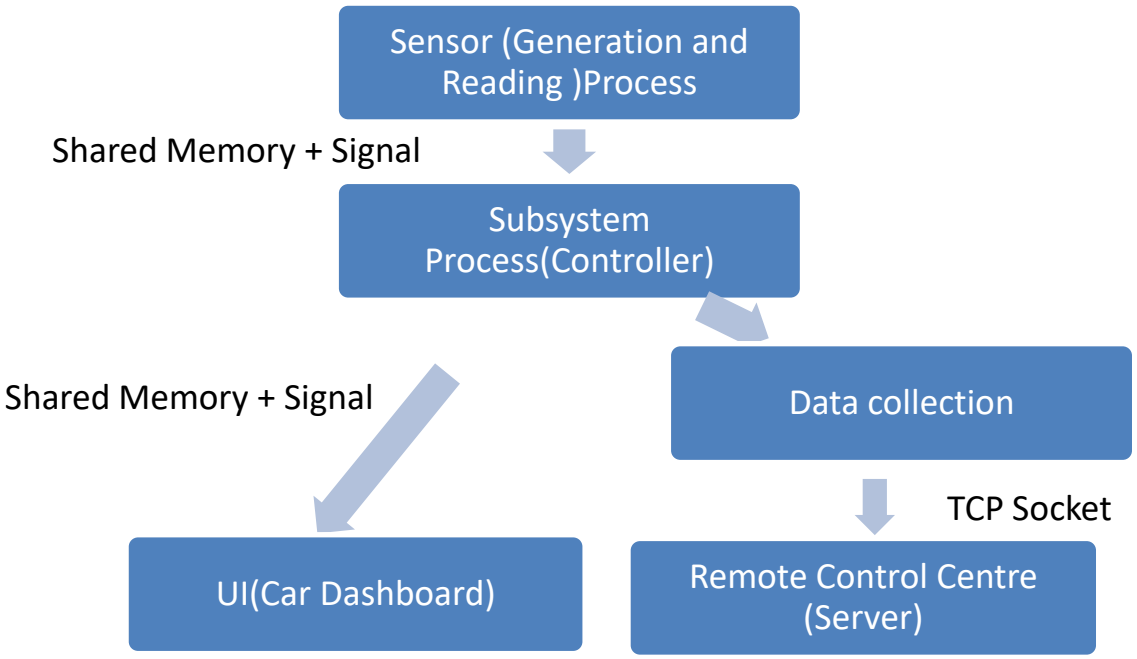
6. Constraints

- Manual launching of ECU and subsystem processes
- Shared memory must be cleaned up on exit
- Only simulated data used (no real sensors)

7. Appendices

A. Assumptions

- Users will run both ECU and subsystem processes manually
- Sensor values are randomly generated for simulation
- Signal handling is limited to basic control (ignition off)


Sensor (Generation and Reading)Process
Shared Memory + Signal
Subsystem Process(Controller)
Shared Memory + Signal
Data collection
TCP Socket
UI(Car Dashboard)
Remote Control Centre (Server)