# Software Requirements Specification (SRS)

## Project Title:

**Multi-Process Real-Time IoT Smart Home Server**

## Platform:

Linux (C Language with IPC mechanisms)

# Introduction

## Purpose

This document outlines the software requirements for a multi-process, real-time IoT smart home server. The system is designed to manage and monitor a large number of IoT devices (e.g., sensors, smart appliances) concurrently, enabling centralized control and visualization through a responsive dashboard.

## Scope

The system supports real-time data acquisition, processing, control, and visualization from hundreds of IoT devices. It uses Linux-based IPC mechanisms and multithreading within each process to ensure concurrency, scalability, and responsiveness. The system is composed of three main processes:

* Device Communication Process (DCP)
* Data Processing & Control Process (DPCP)
* UI & Visualization Process (UIVP)

## Definitions, Acronyms, and Abbreviations

* + - **IPC: Inter-Process Communication**
    - **DCP: Device Communication Process**
    - **DPCP: Data Processing & Control Process**
    - **UIVP: UI & Visualization Process**
    - **TCP: Transmission Control Protocol**
    - **SIGINT/SIGTERM/SIGUSR1: Unix signals for process control**

# Functional Requirements

|  |  |  |
| --- | --- | --- |
| **ID** | **Function** | **Description** |
| FR1 | Device Connection Handling | Accept TCP connections from IoT devices and spawn threads for each. |
| FR2 | Sensor Data Acquisition | Read sensor data from devices and store in shared memory. |
| FR3 | Event Detection | Detect events like motion, fire, and intrusion using sensor data. |
| FR4 | Automated Control | Trigger actions like locking doors or turning on lights. |
| FR5 | UI Rendering | Display real-time data, alerts, and device status on dashboard. |
| FR6 | Real-Time Updates | Push updates to UI via sockets or shared memory. |
| FR7 | Signal Handling | Respond to system signals for shutdown, alerts, and recovery. |

1. **Non-Functional Requirements**
   * + **Concurrency**: Each process supports multithreading for parallel task execution.
     + **Scalability**: Capable of handling hundreds of devices simultaneously.
     + **Reliability**: Includes watchdog threads and signal handling for fault tolerance.
     + **Efficiency**: Optimized for low CPU and memory usage.
     + **Responsiveness**: UI updates in real-time with minimal latency.
     + **Maintainability**: Modular design for easy updates and debugging.

# Software and Hardware Requirements

## Software Requirements

* + - **OS**: Ubuntu Linux
    - **Compiler**: GCC
    - **Tools**: ipcs, ipcrm, shmget, semget, msgget, socket, pthread
    - **Language**: C

## Hardware Requirements

* Simulated IoT devices (software emulated sensors and actuators)
* Standard Linux machine with network capabilities

# System Overview (Process-Based)

**Process Flow:**

* **Device Communication Process (DCP)**
* Accepts TCP connections
* Thread per device for communication
* Shares data via shared memory or message queues
* **Data Processing & Control Process (DPCP)**
* Threads for validation, event detection, and control logic
* Uses mutexes/semaphores for synchronization
* Communicates with DCP and UIVP via IPC
* **UI & Visualization Process (UIVP)**
* Threads for rendering, data fetching, and socket handling
* Displays sensor data, alerts, and device status
* Synchronizes updates using semaphores or condition variables

# Constraints

* + - Manual launching of each process in separate terminals
    - Simulated devices used for testing
    - IPC resources must be released on exit using cleanup routines
    - Signal handling must be implemented for graceful shutdown and emergency alerts

# Appendices

## Assumptions

* + Devices are simulated and connect via TCP
  + Users will manually start each process

## Shared memory and message queues are used for IPC

## B.Glossary

* **Sensor Data**: Information collected from IoT devices
* **Event Detection**: Identifying significant occurrences like motion or fire
* **Control Action**: Automated response triggered by detected events
* **Dashboard**: UI interface displaying system status and alerts

A diagram of a computer system

AI-generated content may be incorrect.