## Software Requirements Specification (SRS)

### Project Title:

Distributed Medical Device Monitoring & Donor Management System

### Platform:

Linux (C Language with IPC mechanisms and Socket Programming)

## Introduction

### Purpose

This document outlines the approach taken to implement a distributed medical device monitoring and donor management system. The system is designed to simulate real-time patient monitoring, donor matching, lab test processing, and medication alerting using IPC and socket communication in a Linux environment.

### Scope

The system allows users to register as either a patient or a donor. Patient vitals are monitored in real-time and analyzed for critical conditions. If a patient is found to be in a critical state, alerts are sent to the donor module to allocate a suitable donor. The system uses shared memory, signals, and sockets to facilitate communication between modules.

### Definitions, Acronyms, and Abbreviations

* **IPC**: Inter-Process Communication
* **TUI**: Terminal User Interface
* **SIGUSR1**: Signal used to input donor details
* **SIGUSR2**: Signal used to trigger donor allocation
* **TCP**: Transmission Control Protocol

## Functional Requirements

|  |  |  |
| --- | --- | --- |
| ID | Function | Description |
| 1 | User Registration | Main module collects user name, age, blood group, and role (patient/donor). |
| 2 | Vital Monitoring | Patient module simulates HR, BP, O2 using threads and updates shared memory. |
| 3 | Lab Test Analysis | Lab test module receives vitals and checks for critical conditions. |
| 4 | Alert Triggering | Medical alert module sends SIGUSR2 to donor module if patient is critical. |
| 5 | Donor Allocation | Donor module receives alert and matches a suitable donor from donor.db. |
| 6 | Donor Registration | Donor details are added via SIGUSR1 signal to donor module. |
| 7 | Socket Communication | All modules communicate via TCP sockets for distributed simulation. |

## Non-Functional Requirements

* **Concurrency**: Patient module uses multiple threads for vitals.
* **Responsiveness**: Signals ensure immediate alert handling.
* **Scalability**: Socket-based design allows modules to run on separate machines.
* **Maintainability**: Modular design simplifies debugging and updates.

## Software and Hardware Requirements

### Software Requirements

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

### Hardware Requirements

* Simulated patient vitals and donor data
* No physical medical devices required

## System Overview (Process-Based)

**Process Flow:**

* **Main Module**: Collects user input and routes to patient or donor module. (Madhuri V)
* **Patient Module**: Simulates vitals using threads and updates shared memory. (Madhuri V)
* **Lab Test Module**: Receives vitals via shared memory and checks for critical status. (Harsha Vardhan)
* **Medical Alert Module**: Sends SIGUSR2 to donor module if patient is critical. (Adithya)
* **Donor Module**: Handles SIGUSR1 for donor input and SIGUSR2 for donor allocation. (Harsha Vardhan)
* **ICU Vital Tracking:** Display multiple patient vitals in real-time. (Adithya)
* **Documentation** (Adithya)

## Constraints

* Shared memory must be protected using mutexes.
* Signals must be handled gracefully to avoid race conditions.
* Manual launching of modules in separate terminals required.

## Appendices

### Assumptions

* Users will run each module manually in separate terminals.
* Simulated data is acceptable for testing.

### Glossary

* **Vitals**: Patient health metrics like HR, BP, O2.
* **Donor**: A person whose details are stored and matched to patients.
* **Signal**: A Unix mechanism for asynchronous event handling.

This document captures the current implementation strategy and system design for the distributed medical monitoring and donor management system. It reflects the modular, real-time, and IPC-driven nature of the solution.

A screenshot of a computer flowchart

AI-generated content may be incorrect.