## 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.

24-10-2025

**Cross‑device Communication via Sockets**

* 1. We implemented TCP sockets to enable reliable communication between two processes running on different devices (or the same device). The main program acts as a client, sending structured data (e.g., name, age, blood group) to the donor server. The donor server receives:
  2. **Op = 1** (ADD\_DONOR): Enqueues the donor in a System V message queue (KEY\_2) and appends to a local file (donor.bin) for persistent logging.
  3. **Op = 2** (CHECK\_DONOR): Accepts a blood group and replies with either FOUND … (when a matching donor is available in the queue) or NOT\_FOUND.

**Multi‑process Synchronization with Semaphores & Shared Memory**

1. We used POSIX named semaphores and System V shared memory to coordinate multiple independent processes on the patient side:
2. **Patients SHM** : Holds up to 5 patient slots (bed management), patient identity, and live vitals (HR, BP, O₂) produced by patient.c.
3. **Status SHM** : lab.c writes evaluated status (critical/not) and formatted messages; message.c reads and acts on them.

**Semaphores**

1. **/sem\_new\_patient** — Signaled by main.c when a new patient is admitted; patient.c spawns 3 threads per patient (HR, BP, O₂) and begins continuous data generation.
2. /**sem\_vitals\_update** — Posted by patient.c whenever new vitals are written; lab.c continuously consumes these updates and applies rules.
3. /**sem\_lab\_to\_msg** — Posted by lab.c after each evaluation; message.c displays the message and, if critical, performs a donor check via sockets. When a donor is found, message.c discharges the patient (sets active=0, occupied=0) and triggers cleanup of the patient’s threads.

**UI Creation (ncurses) — In Progress**

We are building a terminal UI using ncurses in the main program (first.c) to provide an operator-friendly dashboard:

**Admission Menu**

1. **Add Patient**: Collects name / age / blood group. If a bed is available (≤ 5), writes to Patients SHM and signals /sem\_new\_patient. Otherwise, it shows “Beds are FULL”.
2. **Add Donor**: Collects donor info and sends it via sockets (Op=1) to the donor server for queuing and logging.

**Live Dashboard**

1. Displays bed occupancy (up to 5), patient identity, and the latest vitals (HR, O₂, BP) updated by patient.c.
2. Shows lab evaluation status (critical/not) and donor responses (FOUND/NOT\_FOUND) consumed by message.c.