Chapter 9
Concurrency Control and Heartbeat-Driven Recovery for Autonomous Medical Robots

John Christopher Flores Quiambao
The Hong Kong Polytechnic University, Hong Kong

Xiao Yi Wang
The Hong Kong Polytechnic University, Hong Kong

Zhengyang Li
http://orcid.org/0009-0007-2367-8817
The Hong Kong Polytechnic University, Hong Kong

Dai Cheng Han
http://orcid.org/0009-0001-2244-0619
The Hong Kong Polytechnic University, Hong Kong

Ting Xuan Wu
The Hong Kong Polytechnic University, Hong Kong

Mohammad Al Khaldy
http://orcid.org/0009-0009-7502-4668
University of Petra, Jordan

ABSTRACT

Medicine is a field that has historically had a high demand for workers with little supply, this high demand is not helped by the fact that medicine is one of the volatile fields of service work in the world with constantly changing situations that require quick responses where mistakes can potentially cost the lives of patients. The use of robots can help alleviate some of the burden of the workers, especially nurses. Simple tasks like the delivery of medicines and crucial tools which require nothing more than moving items from one place to another can be delegated to robots. The issue comes in creating a system that can seamlessly coordinate these robots such that they do not become more of a burden than the issues they originally tried to

DOI: 10.4018/979-8-2600-1101-0.ch009

fix. This paper aims to provide an implementation of an OS core that can handle concurrency and synchronization of multiple working robots within a hospital, allowing the simulation of real-life situations. We also aim to provide an effective framework through which future researchers are able to build upon for application to real-life systems.

1. INTRODUCTION

1.1 Background

The increasing use of IoT devices in factories and environmental monitoring has brought a real engineering challenge: how can the host system reliably collect, process and display high-frequency data streams from various types of sensors without losing packets, disrupting state sharing, or slowing down user queries? Even if the scale is small, to solve this problem, process isolation, inter-process communication, thread concurrency, synchronized shared memory, and persistent file input and output must be implemented.

To address this challenge, this chapter placed it in a controlled academic environment and implemented a complete end-to-end sensor data pipeline. A three-axis force sensor, two thermometers, and a three-axis accelerometer all run as independent operating system processes, sending readings to a parent gateway process via anonymous Unix pipes. The gateway uses a parallel thread pool to aggregate readings within a one-second time window. Then we persist the results to a JSON file. After that, we provided the data through a real-time network dashboard to record the performance and result. Therefore, a system is built in this way. For each major component, it directly applies a set of different operating system concepts, making a clear and measurable relationship between software design and operating system-level behavior in the real-world cases-based.

1.2 System Architecture

The platform is built with a four-tiered pipeline structure in a single Cargo workspace. We will mention it one by one:

Layer 1 – Sensor Simulation (Component 1, sensor_sim crate): Each sensor type is implemented by a Rust struct. ltt runs a configurable generation thread. In other words,the thermometer generates 10 samples per second, the accelerometer generates 20 samples per second, and the force sensor generates 15 samples per second. The readings are first stored in a custom, lock-free circular