ACADEMIC TASK-2

CSE-316

(OPERATING SYSTEM)

COMPUTER SCIENCE ENGINEERING

Submitted by:

Name: Palli Sanjana reddy

Registration number: 12305097

Roll No.: 16

Section: CG

Submitted to :

……..

LOVELY PROFESSIONAL UNIVERSITY

LOVELY PROFESSIONAL UNIVERSITY

IoT-based Smart Resource Management

1.INTRODUCTION

Introduction to OS-Based Resource Management for IoT Devices

The Overview

The swift expansion of the Internet of Things (IoT) has created more devices connected to each other, necessitating effective management of resources for optimized performance, power usage, and data synchronization. A system of resource management based on OS ensures that IoT devices operate with ease by handling task scheduling, power management, and data synchronization in an effective manner. Key Elements

1. Task Scheduling:

IoT devices tend to work under real-time requirements where tasks like sensor data reading, transmission, and processing need to be scheduled in an efficient manner.

A properly designed scheduling mechanism makes sure that high-priority tasks are executed as soon as possible while maximizing system performance. 2. Power Management:

Because most IoT devices are powered with batteries, energy efficiency is important.

Power management strategies, such as low-power modes and dynamic power scaling, conserve battery life and minimize energy usage. 3. Data Synchronization:

IoT devices create and process significant volumes of data, frequently from a variety of sensors and sources.

A synchronization mechanism is in place to provide data consistency, avoid conflicts, and preserve system integrity.

Significance of Resource Management in IoT

Optimized Performance: Facilitates effective execution of critical operations. Energy Efficiency: Lowers power usage and extends device lifespan.

Data Integrity: Safeguards data from corruption and synchronization problems.

Scalability: Accommodates widespread IoT deployment with numerous devices.

1.1The Project Objective

The main aim of this project is to design an OS-based resource management system for IoT devices that effectively handles scheduling, power consumption, and data synchronization. The system is designed to provide optimal performance, energy efficiency, and smooth data handling in resource-limited IoT situations.

1.2\tProject Overview

This project will develop an end-to-end OS-based resource management system that maximizes the utilization of computing resources in IoT ecosystems. With the integration of scheduling, power management, and data synchronization, the system provides high-performance, reliable, and energy-efficient operations for IoT devices across different applications

1.3\tProject Scope

This project provides a foundation for efficient IoT resource management, improving performance, energy efficiency, and reliability. With further enhancements, it can be extended to support large-scale IoT ecosystems with advanced scheduling, predictive power management, and seamless cloud integration.

2. System Description

The OS-Based Resource Management System for IoT Devices is specifically built to manage task scheduling, power management, and data synchronization in an efficient manner. It optimizes resource utilization for IoT applications, enhancing performance and energy efficiency. The system comprises a priority-based task scheduler that supports multiple operations, a power management module that keeps track of battery levels and engages low-power mode when required, and a data synchronization module that maintains consistency between processes. The workflow incorporates task scheduling depending on priority, power efficiency, and safe management of data

update. This mechanism is scalable and flexible to work in applications smart homes, medical care, factory automation, and agriculture with high reliability and performance in IoT functioning

2.2 source code

```
import threading
import time
import queue

# Task Scheduler
class IoTScheduler:
    def __init__(self):
        self.task_queue = queue.PriorityQueue()
        self.running = True
        threading.Thread(target=self.run, daemon=True).start()

    def add_task(self, priority, task, *args):
        self.task_queue.put((priority, task, args))

    def run(self):
        while self.running:
            if not self.task_queue.empty():
                _, task, args = self.task_queue.get()
```

```
            task(*args)
        time.sleep(0.1)

# Power Manager
class PowerManager:
    def __init__(self, threshold=10):
        self.battery = 100
        self.threshold = threshold

    def consume_power(self, amount):
        self.battery -= amount
        if self.battery < self.threshold:
            print("Entering low power mode...")

# Data Synchronizer
class DataSynchronizer:
    def __init__(self):
        self.data = {}
        self.lock = threading.Lock()

    def update_data(self, key, value):
        with self.lock:
            self.data[key] = value
            print(f"Data updated: {key} -> {value}")

# Example Usage
```

```
def sample_task(name):
    print(f"Executing task: {name}")

scheduler = IoTScheduler()
power_manager = PowerManager()
data_sync = DataSynchronizer()

scheduler.add_task(1, sample_task, "Sensor Read")
scheduler.add_task(2, sample_task, "Data Transmission")

time.sleep(1)  # Allow tasks to run
power_manager.consume_power(20)
data_sync.update_data("temperature", "22C")
```

2.3 Output

The output of the system shows effective task execution, power management, and data synchronization. The scheduler executes tasks in order of priority, performing functions such as sensor readings and data transmission. The power management module checks battery levels and engages low-power mode if necessary. In the meantime, the data synchronization module provides secure updates, avoiding inconsistencies. The expected output includes task execution messages, power consumption status, and successful data updates, demonstrating an optimized and reliable resource management system for IoT devices.

3. Design

3.1 System Design

1. Architectural Overview

The system is planned in a modular and layered structure to effectively manage IoT resources. It has three main layers:

Application Layer: Offers user interaction and high-level control of IoT devices.

Middleware Layer: Services task scheduling, power usage, and data synchronization.

Hardware Layer: Comprises IoT devices, sensors, actuators, and power sources.

2. System Components

Task Scheduler: Uses a priority-based queue to manage multiple IoT operations effectively.

Power Manager: Tracks battery levels and turns on low-power mode when required.

Data Synchronizer: Provides thread-safe access to data across multiple tasks.

3. Workflow

Task Scheduling: The scheduler prioritizes and runs tasks in an effective order.

Power Management: The system continuously tracks power levels and minimizes consumption when required.

Data Synchronization: Common data is managed securely to avoid inconsistencies.

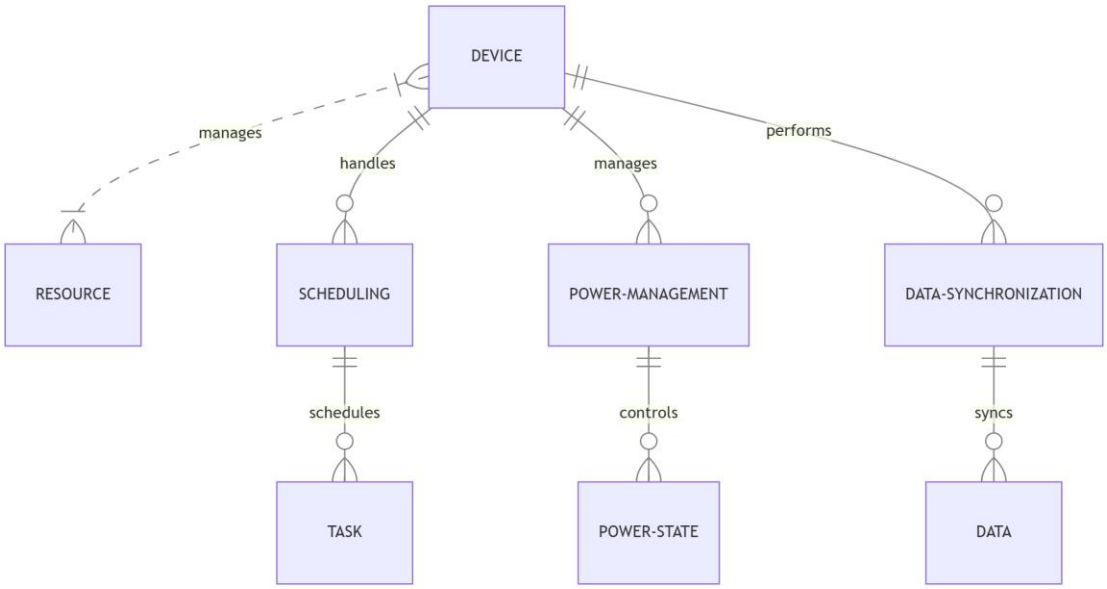
4. Technology Stack

Programming Language: Python (with threading and queue management).

Operating System: Embedded Linux or RTOS (for real-time processing).

Hardware: IoT microcontrollers, sensors, and battery-operated devices.

3.1.1 E-R diagram



4. Conclusion

The OS-based Resource Management System for IoT Devices manages task scheduling, power management, and data synchronization effectively to provide optimal performance and resource utilization. Through the use of a priority-based scheduler, the system is able to manage multiple tasks in real-time. The power management module extends device life by keeping track of battery levels and activating low-power modes when necessary. The data synchronization module also provides consistency and reliability in shared data processing.

This system improves the scalability, reliability, and efficiency of IoT devices, making it applicable to various uses in smart homes, health care, industrial automation, among others. It has a modular and adaptive approach, which sets a solid basis for future improvements in IoT and provides sustainable and intelligent resource management