Homework 3

Anhelina Bondarenko

Std: 220104004928

**Report**

**1. Introduction**

This report analyzes a C-based multithreaded system simulating a satellite support service managed by engineers. The program handles concurrency using POSIX threads (pthread), semaphores (sem\_t), and mutexes (pthread\_mutex\_t). It models the scenario where multiple satellite requests (with varying priorities) are processed by a limited number of engineers.

**2. System Overview**

The system consists of:

* A **shared priority queue** for satellite requests.
* A **fixed pool of engineers** (NUM\_ENGINEERS = 3), implemented as threads.
* Each **satellite** is also represented as a thread and is associated with a priority and timeout mechanism.
* The system uses synchronization mechanisms to ensure thread-safe access to shared data.

**3. Purpose and Role of Each Synchronization Primitive**

**3.1 Mutexes**

**engineerMutex**

* **Purpose**: Synchronizes access to the availableEngineers variable and the shutdownFlag.
* **Usage**:
  + Prevents race conditions when engineers check or update availableEngineers.
  + Ensures correct evaluation of the shutdown state and queue status.

**queueMutex**

* **Purpose**: Protects the **priority request queue** from concurrent modifications.
* **Usage**:
  + Locks access when a satellite enqueues its request or when an engineer dequeues a request.
  + Ensures thread-safe modifications to the linked list structure.

**3.2 Semaphores**

**sem\_t newRequest**

* **Purpose**: Signals engineers that a new satellite request has arrived.
* **Usage**:
  + Posted (sem\_post) by the satellite thread after it enqueues its request.
  + Waited on (sem\_wait) by engineers to detect new incoming requests.
  + Also used during shutdown to unblock any waiting engineer threads.

**sat[i].handled\_sem (Per-satellite semaphore)**

* **Purpose**: Used by engineers to signal that a specific satellite request is being handled.
* **Usage**:
  + The satellite thread waits (sem\_timedwait) on this semaphore for up to TIMEOUT seconds.
  + If no engineer handles the request within the timeout, the satellite times out and removes itself from the queue.
  + Engineers post (sem\_post) to this semaphore when they start processing the satellite's request.

**4. Function Descriptions and Their Purpose**

**4.1 main()**

* Initializes mutexes and semaphores.
* Creates threads for engineers and satellites.
* Signals system shutdown after all satellites are processed.
* Joins threads and performs cleanup.

**4.2 satellite(void\* arg)**

* Simulates a satellite request:
  + Assigns a random priority.
  + Enqueues itself in the priority queue.
  + Waits for an engineer to handle it using a timed semaphore wait.
  + If not handled in time, removes itself from the queue and logs a timeout message.

**4.3 engineer(void\* arg)**

* Represents an engineer continuously processing satellite requests:
  + Waits on newRequest semaphore.
  + Acquires engineerMutex to check for shutdown and queue state.
  + Dequeues a satellite if available, signals the satellite’s semaphore, and simulates work with sleep.
  + Increments and decrements the count of availableEngineers.

**4.4 enqueue(PriorityQueue\* pq, Satellite\* satellite)**

* Adds a satellite to the priority queue in descending priority order.
* Uses linked list logic to maintain the queue order.

**4.5 dequeue(PriorityQueue\* pq)**

* Removes and returns the highest-priority satellite from the queue.

**4.6 removeFromQueue(PriorityQueue\* pq, int id)**

* Removes a satellite from the queue based on its ID.
* Called if the satellite times out before being serviced.

**4.7 engineer\_cleanup(void\* arg)**

* Called on thread cancellation/cleanup.
* Frees the dynamically allocated engineer ID pointer.

**5. Test scenarios**

**Greater number –> higher priority**

*Timeout 2 sec*

**Case 1**

* 5 satellites and 3 engineers are defined.
* Satellites are assigned random priority values.
* Satellite 0 (priority 3), satellite 1 (priority 4), satellite 2 (priority 2), satellite 3 (priority 4), satellite 4 (priority 2), request a connection one after another. Based on the priority values they are put in the queue.
* Satellites 1,3,0 are picked by engineers as highest prioritized
* None of engineers were able to finish their job within 2 seconds timeout
* So, Satellite 2’s and 4`s timeouts expire and they drop out.
* Finally, all engineers complete their tasks and exit.

Зображення, що містить текст, знімок екрана, Шрифт, дизайн

Вміст, створений ШІ, може бути неправильним.

**Case 2**

* 5 satellites and 3 engineers are defined.
* Satellites are assigned random priority values.
* Satellite 2, with priority 1, requests an update.
* One of the three engineers immediately picks up and starts processing this request.
* Satellite 3, with priority 2, requests an update.
* One of the two available engineers immediately picks up and starts processing this request.
* Satellite 4 (priority 1), satellite 1 (priority 2), request a connection one after another. Based on the priority values they are put in the queue.
* Satellite 1 is picked as a higher priority
* Satellite 0, with priority 2, requests an update and is placed in a queue
* Engineer 0 finishes his work and pick satellite 0
* Satellite 4’s timeout expires and it drops out
* Finally, all engineers complete their tasks and exit.

Зображення, що містить текст, знімок екрана, Шрифт

Вміст, створений ШІ, може бути неправильним.

*Timeout 5 sec*

**Case 3**

* 5 satellites and 3 engineers are defined.
* Satellites are assigned random priority values.
* Satellite 0, with priority 2, requests an update.
* One of the three engineers immediately picks up and starts processing this request.
* Satellite 1 (priority 2), satellite 2 (priority 2), satellite 3 (priority 1), satellite 4 (priority 3), request a connection one after another. Based on the priority values they are put in the queue.
* Satellite 4 and 1 were picked sequentially according to priorities and FIFO principle
* Engineer 0 finishes his work and pick satellite 2
* Engineer 1 finishes his work and pick satellite 1
* One by one engineers finish their work and exit

Зображення, що містить текст, знімок екрана, Шрифт, дизайн

Вміст, створений ШІ, може бути неправильним.

**Case 4**

Purpose of this case is to show that principle of FIFO works for equal priority (2) satellites

Зображення, що містить текст, знімок екрана, Шрифт

Вміст, створений ШІ, може бути неправильним.

**6. Key Concepts Demonstrated**

* **Producer-consumer synchronization** using semaphores.
* **Priority-based scheduling** using a custom priority queue.
* **Thread-safe shared resource access** using mutexes.
* **Timeout handling** with per-thread semaphores and sem\_timedwait.
* **Thread cleanup and memory management** for dynamically allocated thread arguments.

**7. Conclusion**

This program effectively demonstrates a real-world simulation of priority-based task handling with concurrency control. The use of mutexes ensures shared data consistency, while semaphores provide efficient inter-thread communication. The satellite timeout mechanism adds robustness by preventing indefinite waiting, and the orderly shutdown sequence prevents deadlocks or resource leaks.