HUSNIDDIN RAMAZANOV / 200104004802

CSE-344 Homework #3 Report

Objective

This report describes the implementation of a satellite ground station that coordinates communications between Earth and multiple satellites. The system has three engineers available to assist satellites with updates, while satellites of varying priority levels arrive randomly requesting assistance. Each satellite has a limited connection window to establish communication and complete updates. The system prioritizes satellites based on their priority levels, with higher-priority satellites being served first.

Implementation Overview

The solution implements a multi-threaded system with two types of threads:

- satellite() threads representing satellites requesting connections
- engineer() threads representing engineers serving satellite requests

The synchronization between these threads is achieved using semaphores and mutexes, with engineers working based on a priority queue system.

Shared Resources

1. availableEngineers: Counter variable tracking available engineers
2. requestQueue: Priority queue managing incoming satellite requests
3. engineerMutex: Mutex protecting access to shared resources

Semaphores

1. newRequest: Signaled by satellites when a new request arrives
2. requestHandled: Signaled by engineers once they pick up a satellite request

Implementation Details

Priority Queue Implementation

The priority queue implementation ensures higher-priority satellites are served first:

```
void enqueue(SatelliteQueue* q, Satellite s) {
    int i = q->size++;

    // Sort by priority in descending order (higher priority first)
    while (i > 0 && q->data[i-1].priority < s.priority) {
        q->data[i] = q->data[i-1];
        i--;
    }
    q->data[i] = s;
}
```

```
Satellite dequeue(SatelliteQueue* q) {
    // Return the highest priority element (at index 0)
    Satellite sat = q->data[0];

    // Shift remaining elements
    for (int i = 0; i < q->size - 1; i++) {
        q->data[i] = q->data[i + 1];
    }

    q->size--;
    return sat;
}
```

Satellite Thread Implementation

Each satellite thread:

1. Generates a random priority level
2. Establishes a random connection timeout window
3. Attempts to request engineer assistance
4. Waits for an engineer with a timeout
5. Aborts if the timeout expires before connecting

```
void* satellite(void* arg) {
    int id = *(int*)arg;
    free(arg);

    // Generate random priority between 1-5
    srand(time(NULL) + id);
    int priority = rand() % 5 + 1;

    // Random wait time between 1-6 seconds
    int waitTime = rand() % 6 + 1;

    Satellite sat = { id, priority };

    printf("[SATELLITE] Satellite %d requesting (priority %d)\n", id, priority);

    pthread_mutex_lock(&engineerMutex);
    enqueue(&requestQueue, sat);
    pthread_mutex_unlock(&engineerMutex);

    sem_post(&newRequest);

    struct timespec ts;
    clock_gettime(CLOCK_REALTIME, &ts);
    ts.tv_sec += waitTime;
```

```
    if (sem_timedwait(&requestHandled, &ts) == -1) {
        if (errno == ETIMEDOUT) {
            pthread_mutex_lock(&engineerMutex);
            // Check if we're still in the queue before removing
            int found = 0;
            for (int i = 0; i < requestQueue.size; ++i) {
                if (requestQueue.data[i].id == id) {
                    found = 1;
                    removeSatellite(&requestQueue, id);
                    break;
                }
            }
            pthread_mutex_unlock(&engineerMutex);

            if (found) {
                printf("[TIMEOUT] Satellite %d timeout %d second.\n", id, waitTime);
            }
        }
    }

    return NULL;
}
```

Engineer Thread Implementation

Each engineer thread:

1. Waits for a satellite request
2. Always selects the highest-priority satellite from the queue
3. Processes the satellite request
4. Signals when the request has been handled
5. Exits when all satellites have been processed

```
void* engineer(void* arg) {
    int id = *(int*)arg;
    free(arg);

    while (1) {
        sem_wait(&newRequest);

        pthread_mutex_lock(&engineerMutex);
        if (isEmpty(&requestQueue)) {
            pthread_mutex_unlock(&engineerMutex);
            continue;
        }
        Satellite sat = dequeue(&requestQueue);
        availableEngineers--;
        pthread_mutex_unlock(&engineerMutex);

        printf("[ENGINEER %d] Handling Satellite %d (Priority %d)\n", id, sat.id, sat.priority);

        // Simulate processing time (2 seconds)
        sleep(2);

        printf("[ENGINEER %d] Finished Satellite %d\n", id, sat.id);

        pthread_mutex_lock(&engineerMutex);
        availableEngineers++;
        pthread_mutex_unlock(&engineerMutex);

        sem_post(&requestHandled);

        // If all satellites are handled, exit
        pthread_mutex_lock(&engineerMutex);
        if (isEmpty(&requestQueue)) {
            pthread_mutex_unlock(&engineerMutex);
            printf("[ENGINEER %d] Exiting...\n", id);
            break;
        }
        pthread_mutex_unlock(&engineerMutex);
    }
    return NULL;
}
```

Main Function

The main function:

1. Initializes synchronization primitives
2. Creates engineer threads
3. Creates satellite threads with delays between them
4. Waits for all threads to complete
5. Cleans up resources

Execution Results

When executed, the program demonstrates the following behaviors:

1. Satellites request connections with random priorities
2. Engineers handle satellites in priority order (highest first)
3. If a satellite's timeout expires before an engineer becomes available, it aborts
4. Once all satellites have been handled or have timed out, engineers exit

Example output:

```
hussi@HUSNIDDIN:/mnt/c/Users/Husniddin/OneDrive/Desktop/systemhw/hw3$ make
make: Nothing to be done for 'all'.
hussi@HUSNIDDIN:/mnt/c/Users/Husniddin/OneDrive/Desktop/systemhw/hw3$ ./main
[SATELLITE] Satellite 0 requesting (priority 2)
[ENGINEER 0] Handling Satellite 0 (Priority 2)
[SATELLITE] Satellite 1 requesting (priority 1)
[ENGINEER 1] Handling Satellite 1 (Priority 1)
[ENGINEER 0] Finished Satellite 0
[ENGINEER 0] Exiting...
[SATELLITE] Satellite 2 requesting (priority 4)
[ENGINEER 2] Handling Satellite 2 (Priority 4)
[ENGINEER 1] Finished Satellite 1
[ENGINEER 1] Exiting...
[SATELLITE] Satellite 3 requesting (priority 4)
[ENGINEER 2] Finished Satellite 2
[ENGINEER 2] Handling Satellite 3 (Priority 4)
[SATELLITE] Satellite 4 requesting (priority 3)
[ENGINEER 2] Finished Satellite 3
[ENGINEER 2] Exiting...
[TIMEOUT] Satellite 4 timeout 2 second.
hussi@HUSNIDDIN:/mnt/c/Users/Husniddin/OneDrive/Desktop/systemhw/hw3$
```

Challenges and Solutions

1. **Priority Queue Management**:
    - Challenge: Ensuring that satellites are always handled in order of their priority

- Solution: Implemented a priority-based insertion algorithm in the enqueue function

2. **Timeout Handling**:
   - Challenge: Correctly handling the situation where a satellite times out
   - Solution: Used sem_timedwait with proper error checking to detect timeouts
3. **Race Conditions**:
   - Challenge: Avoiding race conditions when multiple threads access shared resources
   - Solution: Used mutex locks to protect all access to shared resources
4. **Memory Management**:
   - Challenge: Preventing memory leaks in thread creation and destruction
   - Solution: Properly freed allocated memory after thread creation

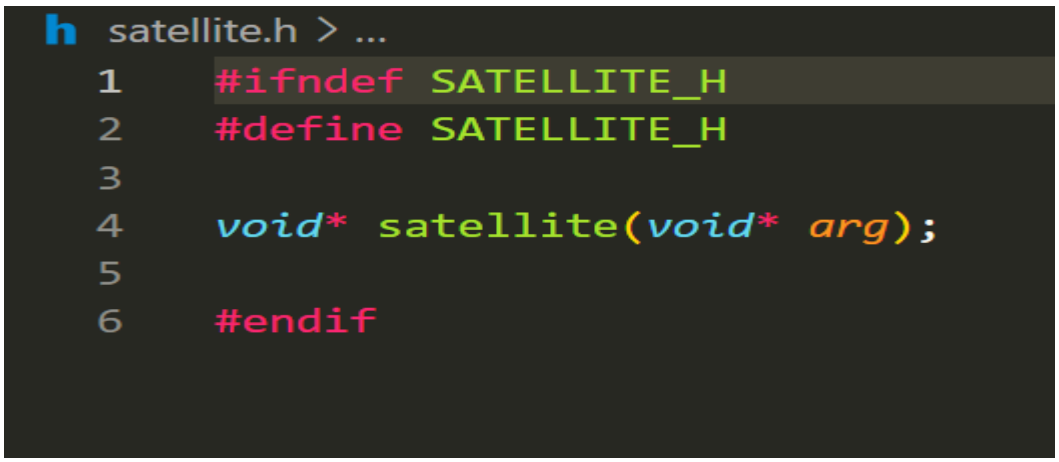
Conclusion

The implemented solution successfully demonstrates a satellite ground station system that coordinates communications between multiple satellites and engineers. The system correctly prioritizes higher-priority satellites and handles timeout situations appropriately. The solution follows the requirements specified in the assignment and produces the expected behaviors in the test scenario.

All code is properly synchronized using mutexes and semaphores, ensuring thread safety while allowing maximum concurrency when possible. The priority queue implementation ensures that engineers always service the highest priority satellites first, and the timeout mechanism ensures that satellites don't wait indefinitely for service.

Code Listing

satellite.h



engineer.h

```h
#ifndef ENGINEER_H
#define ENGINEER_H

void* engineer(void* arg);

#endif
```

queue.h

```h
#ifndef QUEUE_H
#define QUEUE_H

#define MAX_QUEUE 100

typedef struct {
    int id;
    int priority;
} Satellite;

typedef struct {
    Satellite data[MAX_QUEUE];
    int size;
} SatelliteQueue;

void initQueue(SatelliteQueue* q);
void enqueue(SatelliteQueue* q, Satellite s);
Satellite dequeue(SatelliteQueue* q);
int isEmpty(SatelliteQueue* q);
void destroyQueue(SatelliteQueue* q);
void removeSatellite(SatelliteQueue* q, int id);

#endif
```

Makefile

makefile

```
all: main

main: main.c satellite.c engineer.c queue.c
	gcc -o main main.c satellite.c engineer.c queue.c -lpthread

clean:
	rm -f main

```