Satellite Request Handling System Documentation

Overview

This program implements a priority-based multithreaded request handling system between satellites and engineers using POSIX threads and semaphores.

Key Components

Data Structures

- Satellite: Represents a satellite thread with unique ID, priority, semaphore signals, and timeout status.
- Engineer: Represents an engineer thread with ID and associated pthread.
- PriorityQueue: A max-heap-based queue for handling satellite requests in priority order.

Process Flow

1. Initialization:
   - Satellite and Engineer structures are initialized.
   - Semaphores and mutexes are set up.
   - Priority queue is initialized for request handling.

2. Satellite Thread:
   - Posts request with priority.
   - Waits on sem_timedwait() for a response within 5 seconds.
   - Marks itself as timed out if not handled in time.

3. Engineer Thread:
   - Waits for available satellite requests.
   - Handles the request from the highest priority satellite.
   - Signals completion via the requestHandled semaphore of that satellite.

4. Queue Handling:
   - Implemented as a max-heap.
   - Ensures high-priority requests are serviced first.

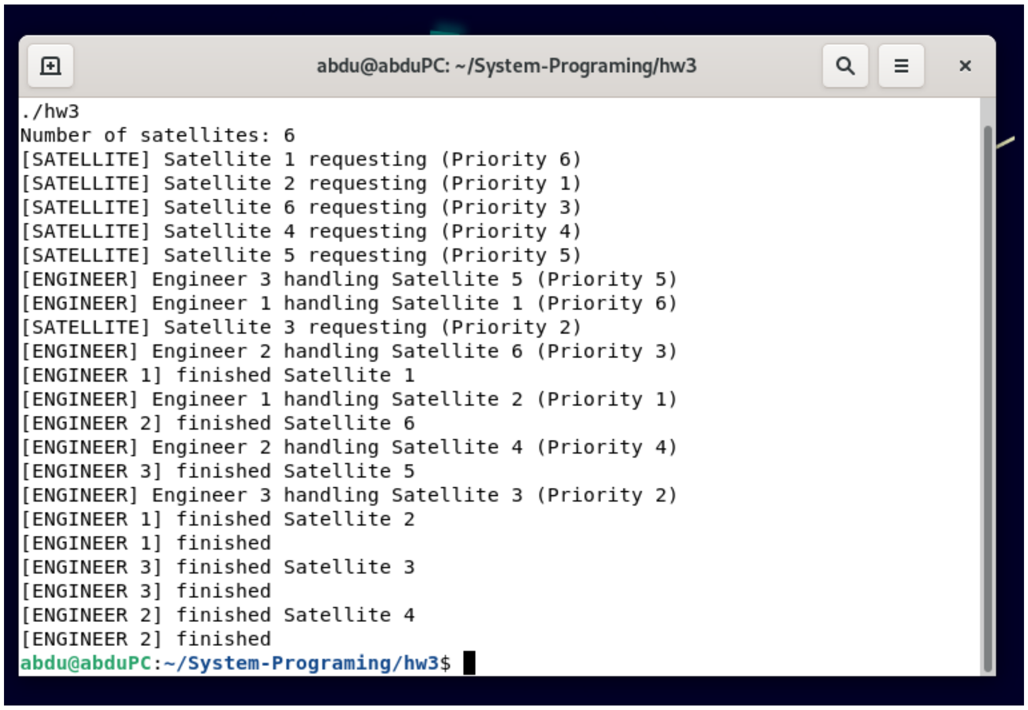
Functions

Main Functions
- main(): Entry point, initializes satellites, engineers, and threads.
- satellite(void*): Behavior of satellite threads including semaphore wait.
- engineer(void*): Behavior of engineer threads including queue polling.

Queue Functions
- initQueue(): Initializes the priority queue.
- insert(): Adds a satellite request to the queue based on priority.
- extractMax(): Retrieves and removes the satellite with the highest priority.
- swap(): Helper for maintaining the heap property.

Usage
Compile with:
gcc -pthread -o satellite_system main.c

Run with:
./satellite_system



```
./hw3
Number of satellites: 6
[SATELLITE] Satellite 1 requesting (Priority 6)
[SATELLITE] Satellite 2 requesting (Priority 1)
[SATELLITE] Satellite 6 requesting (Priority 3)
[SATELLITE] Satellite 4 requesting (Priority 4)
[SATELLITE] Satellite 5 requesting (Priority 5)
[ENGINEER] Engineer 3 handling Satellite 5 (Priority 5)
[ENGINEER] Engineer 1 handling Satellite 1 (Priority 6)
[SATELLITE] Satellite 3 requesting (Priority 2)
[ENGINEER] Engineer 2 handling Satellite 6 (Priority 3)
[ENGINEER 1] finished Satellite 1
[ENGINEER] Engineer 1 handling Satellite 2 (Priority 1)
[ENGINEER 2] finished Satellite 6
[ENGINEER] Engineer 2 handling Satellite 4 (Priority 4)
[ENGINEER 3] finished Satellite 5
[ENGINEER] Engineer 3 handling Satellite 3 (Priority 2)
[ENGINEER 1] finished Satellite 2
[ENGINEER 1] finished
[ENGINEER 3] finished Satellite 3
[ENGINEER 3] finished
[ENGINEER 2] finished Satellite 4
[ENGINEER 2] finished
abdu@abduPC:~/System-Programing/hw3$
```

Error Handling
- Handles overfilled queues with a warning.
- Checks return values from semaphore and mutex operations.
- Uses sem_timedwait for timeout handling to avoid indefinite blocking.

Security and Resource Management
- All semaphore and thread resources are initialized and destroyed properly.
- Shared data access is protected with mutexes.

Limitations
- Fixed-size queue (MAX_SATELLITE).
- Timeout is hardcoded to 5 seconds per satellite.
- No dynamic thread scaling or recovery after timeout.