April 14, 2025

Homework #3:

Objective:

A satellite ground station coordinates communications between Earth and multiple satellites. There are **three engineers** available to assist satellites with their updates. Satellites (with varying priority levels) arrive randomly and request engineer support.

Each satellite has a **limited connection window** to establish communication and complete updates (for instance, a maximum of 5 seconds). If no engineer becomes available within the allowed connection window, the satellite will **abort the update** and leave. The satellites have different **priority levels** (e.g., military, weather, commercial), and **higher-priority satellites are served first**.

To implement this scenario, two types of threads must be created: `satellite()` and `engineer()`.

The `satellite()` thread represents the satellite requesting a connection. The `engineer()` thread represents an engineer ready to serve a satellite.

The synchronization between these threads must be achieved using **semaphores** and **mutexes**. The engineers work based on a **priority queue** system, always selecting the satellite with the highest priority that is waiting.

To manage shared resources:

- A counter variable named `availableEngineers` must be maintained.
- A priority queue (`requestQueue`) must be used to manage incoming satellite requests.
- These shared resources must be protected using a **mutex** (`engineerMutex`).

Semaphores should be used as follows:

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

Each satellite will attempt to post its request and wait for an engineer. If no engineer becomes available **before the timeout period expires**, the satellite will leave without being updated.

--

Test Scenario (Expected Results):

1. Scenario Overview:

- There are three engineers ready to serve satellite requests.
- A random number is generated to simulate satellites of varying priority.
- Only one satellite can initiate a connection attempt at a time (mutex protected).

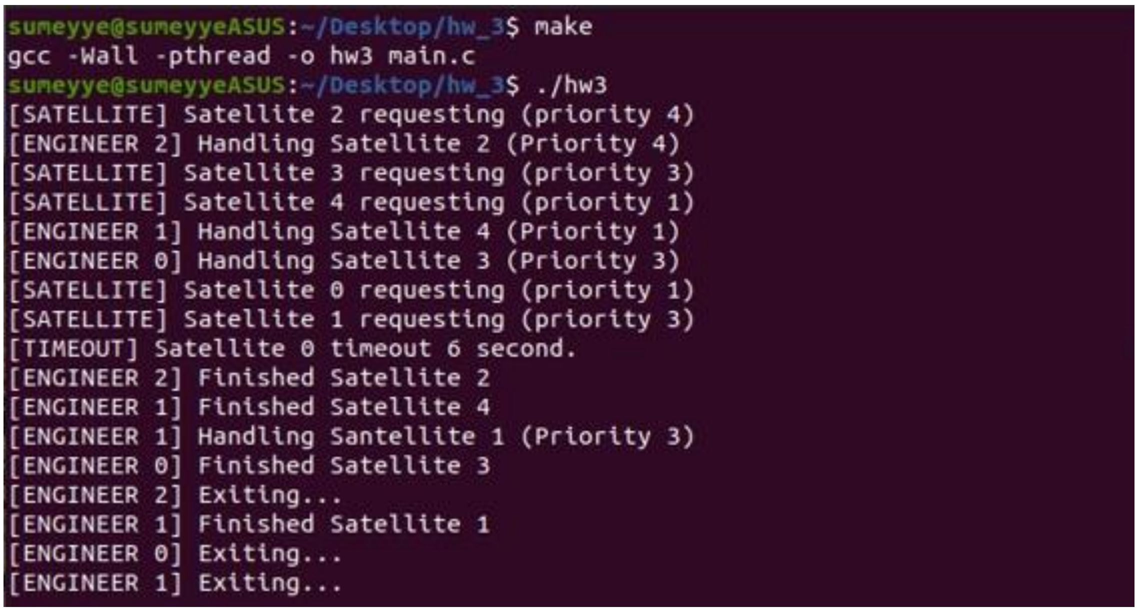
2. Workflow:

- `satellite()` represents the satellite initiating a request.
- `engineer()` represents the engineer ready to handle requests.
- Threads are synchronized using **semaphores** and **mutex**.
- **Timeout** is applied: if a satellite cannot get an engineer within its window, it leaves.

3. Example Scenario

- 5 satellites and 3 engineers are defined.
- Satellites are assigned random priority values.
- Satellite 2, with priority 4, requests an update.
- One of the three engineers immediately picks up and starts processing this request.
- Then, Satellites 3 (priority 3) and 4 (priority 1) request a connection one after another. Based on the priority values, Satellite 4 is processed first.
- Satellites 0 and 1 also request connections. However, since all 3 engineers are busy, these satellites are placed in the waiting queue.
- Satellite 0's timeout expires and it drops out.

- Before Satellite 1's timeout expires, one of the engineers becomes available and processes Satellite 1's request.
- Finally, all engineers complete their tasks and exit.



```
sumeyye@sumeyyeASUS:~/Desktop/hw_3$ make
gcc -Wall -pthread -o hw3 main.c
sumeyye@sumeyyeASUS:~/Desktop/hw_3$ ./hw3
[SATELLITE] Satellite 2 requesting (priority 4)
[ENGINEER 2] Handling Satellite 2 (Priority 4)
[SATELLITE] Satellite 3 requesting (priority 3)
[SATELLITE] Satellite 4 requesting (priority 1)
[ENGINEER 1] Handling Satellite 4 (Priority 1)
[ENGINEER 0] Handling Satellite 3 (Priority 3)
[SATELLITE] Satellite 0 requesting (priority 1)
[SATELLITE] Satellite 1 requesting (priority 3)
[TIMEOUT] Satellite 0 timeout 6 second.
[ENGINEER 2] Finished Satellite 2
[ENGINEER 1] Finished Satellite 4
[ENGINEER 1] Handling Santellite 1 (Priority 3)
[ENGINEER 0] Finished Satellite 3
[ENGINEER 2] Exiting...
[ENGINEER 1] Finished Satellite 1
[ENGINEER 0] Exiting...
[ENGINEER 1] Exiting...
```

4. Details:

- If engineers are fully occupied, the satellite waits (up to a timeout) and leaves if necessary.
- Shared data (like `availableEngineers` and `requestQueue`) are properly protected with mutex.
- Satellites are served based on their **priority** (priority queue structure).
- Engineers constantly listen for new requests via the `newRequest` semaphore.
- Semaphores and mutexes are initialized and used correctly.

Grading Criteria:

Criteria	Penalty

No compilation	-100 points
Missing Makefile or Makefile without "make clean"	-30 points
Each memory leak encountered	-20 points
No report was submitted (you must test and demonstrate every step)	-100 points
Failure in one of the tasks	-10 points

Late submissions will not be accepted.

Deadline: April 24, 2025, at 23.59 (Do not request an extension for the deadline.)

If you have any questions about Homework 3, you can send an e-mail to sunsur@gtu.edu.tr.

Good Luck