CSE 344

SYSTEMS PROGRAMMING

FALL 2025

HW #3

Emre Güleç

200104004045

INTRODUCTION

This program simulates a satellite-maintenance control centre in which five **satellite** threads request help from three **engineer** threads.

Main goals

- **Priority service** – every request carries an integer priority; engineers always pick the highest-priority job waiting.
- **Bounded wait** – a satellite abandons its request if no engineer begins within TIMEOUT seconds.
- **Resource accounting** – a global counter availableEngineers always shows how many engineers are free.
- **Graceful shutdown** – once every satellite has either been served or has timed out, special "poison-pill" requests shut the engineers down cleanly.

Implementation techniques

- A manually managed linked-list **priority queue** protected by its own mutex.
- A counting semaphore newRequest to wake sleeping engineers when new work arrives.
- One **private semaphore per satellite** so an engineer can wake exactly the right requester.
- Extra mutexes for very short critical sections (requestMutex and engineerMutex).

CODE EXPLANATION

Below each major item of the program is explained in turn.

Global constants

- TIMEOUT – seconds a satellite will wait for an engineer before giving up.
- WORK_TIME – seconds an engineer spends "repairing" a satellite.
- NUM_SATELLITES, NUM_ENGINEERS – how many threads of each kind to create.

Data structures

- **struct Satellite** – one node per request in the priority queue.
  - id satellite identifier (negative for shutdown pills)
  - priority higher numbers mean higher priority

- sem_t *reply pointer to the satellite's private semaphore (NULL for pills)
- next link to the next node

- **PriorityQueue queue** – the queue head pointer plus a mutex (lock).

Global synchronisation objects

- sem_t newRequest – counts how many requests are waiting.
- pthread_mutex_t requestMutex – briefly locks enqueue + semaphore-post as one atomic step.
- pthread_mutex_t engineerMutex – protects the availableEngineers counter.

FUNCTIONS:

remove_request_by_id(int id)

Searches the queue (mutex already held by caller), unlinks and frees the node with the matching id.
Used by satellites that time out so their stale request does not clog the queue.

enqueue_with_sem(int id, int priority, sem_t *reply)

Inserts a new node in priority order (highest first).
Does **not** post newRequest; callers must do that themselves after returning.

dequeue()

Takes the first node (highest priority) from the queue and returns it.
Assumes an engineer has already performed sem_wait(&newRequest), so the queue is not empty.

satellite(void *arg)

1. Extract its id and random priority from the heap-allocated argument array.
2. Initialise a private semaphore satSem to 0.
3. Lock requestMutex, call enqueue_with_sem, post newRequest, unlock.
4. Wait on satSem with a timeout of TIMEOUT seconds.
    - If it wakes normally, an engineer has accepted the job.
    - If it times out (ETIMEDOUT), it logs a message and removes its own request with remove_request_by_id.
5. Destroy satSem and exit.

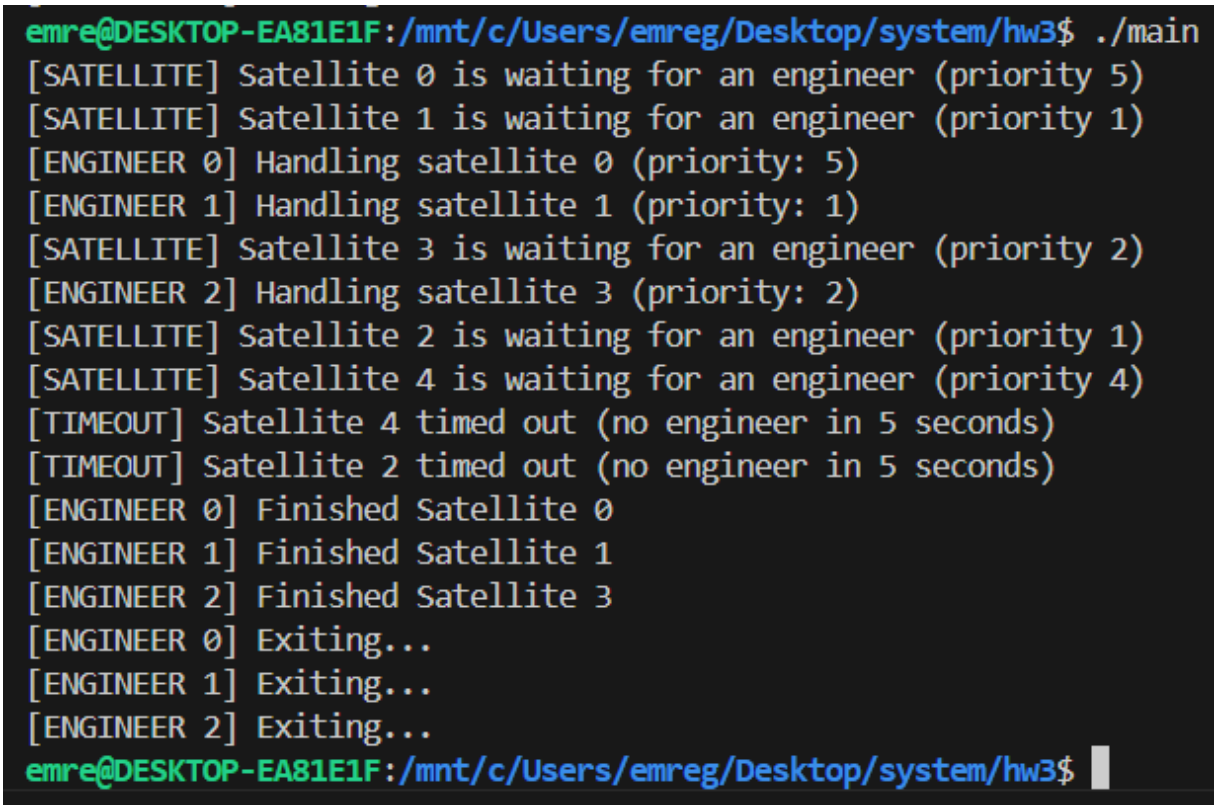
engineer(void *arg)

1. Enable POSIX thread cancellation (deferred).
2. Infinite service loop:
   a. sem_wait(&newRequest) – block until at least one queued request exists.
   b. dequeue() – take the highest-priority node.
   c. If sat->id < 0 the node is a **poison pill**: free it and break the loop.
   d. Otherwise:
      - Decrement availableEngineers under engineerMutex.
      - Print a log line, then sem_post(sat->reply) to wake the correct satellite.
      - sleep(WORK_TIME) to simulate repair.
      - Increment availableEngineers, log completion, free the node.
3. After the loop, sleep one second to simulate shutdown activities and print an exit message.

main()

1. Seed the random generator and initialise newRequest.
2. Create NUM_ENGINEERS engineer threads.
3. Create NUM_SATELLITES satellite threads, each with a random priority 1-5.
4. pthread_join every satellite thread (they either finish service or time out).
5. Send one **shutdown pill** per engineer: enqueue a node with id == -1, priority == INT_MIN, reply == NULL, then post newRequest.
6. Join the engineer threads.
7. Destroy all semaphores and mutexes and return 0.

SCREENSHOTS



Here, work time for engineers is 6 seconds and timeout is 5 seconds. So, both satellite 4 and 2 timed out. Satellite/engineer count or time constants can easily be changed to display different scenarios.