Onurhan Talan

210104004065

CSE 344

Homework 3

**1-Introduction**

In the engineerSatellite program, three **engineer threads** wait for **satellite threads** to request an update. The engineer threads remain idle until a satellite thread sends a request.

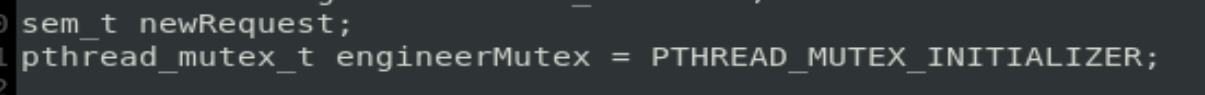
When a request is received, one of the available engineer threads accepts it and starts processing. If multiple requests arrive at the same time, the engineer selects the one with the highest priority. If no request is available, all engineers wait. Eventually, if no new requests are generated and no pending requests exist, the engineer threads terminate.

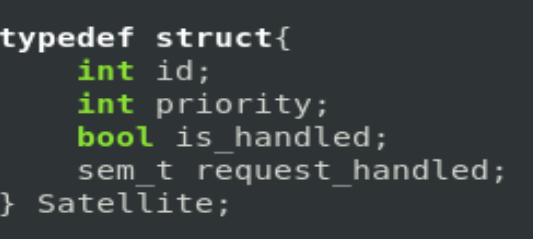
If a satellite thread makes a request while all engineers are busy, and no engineer becomes available before the satellite's limited connection window expires, the request is aborted.

**2-Code Explanation**

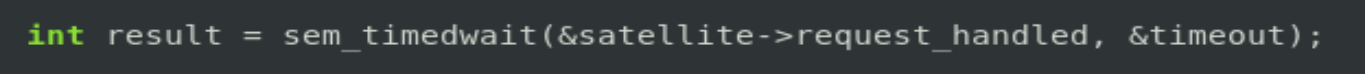
This program implements a classic **producer-consumer problem**. Engineer threads may attempt to access shared resources simultaneously, leading to race conditions. To prevent this, a mutex named **engineerMutex** is used to ensure only one engineer enters its critical section at a time.

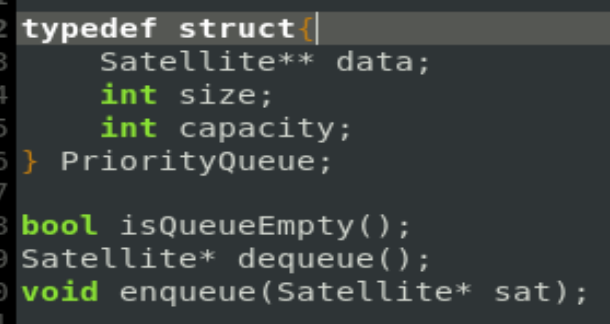
If there are no requests, engineer threads should remain idle. Although busy waiting is one solution, it wastes CPU resources. Instead, a semaphore named **newRequest** is used to block engineers until a request is available. When a satellite thread submits a request, it calls **sem\_post()** on this semaphore to notify the waiting engineers.



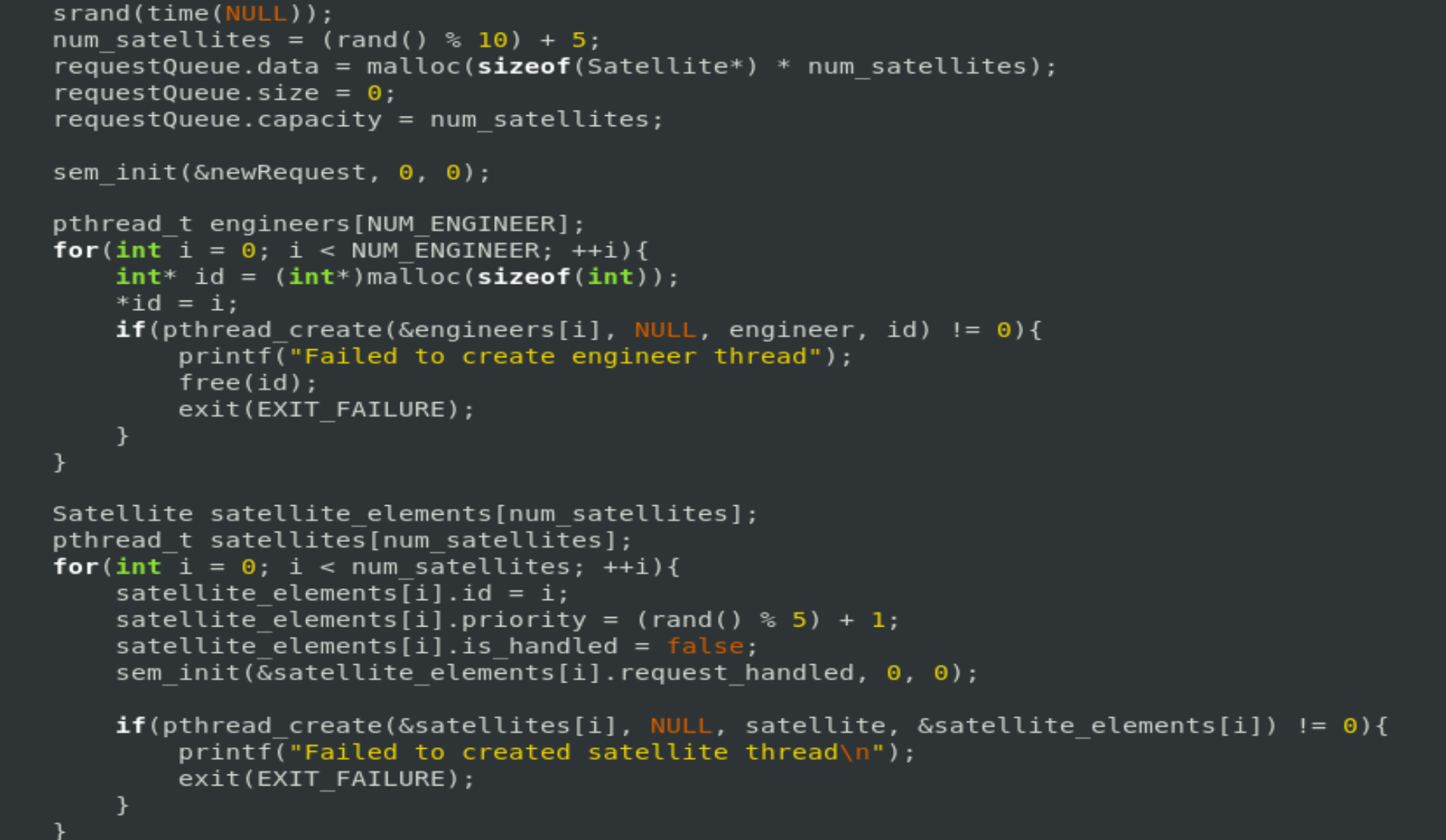
 Each satellite’s **ID**, **priority**, and **handled state** need to be stored. A structure named Satellite is defined for this purpose.

Because each satellite has a limited connection window, a semaphore named **request\_handled** is created for each satellite. The satellite thread uses **sem\_timedwait()** to wait for a limited period. If an engineer picks up the request, it calls **sem\_post()** to notify the satellite that the request is being handled.

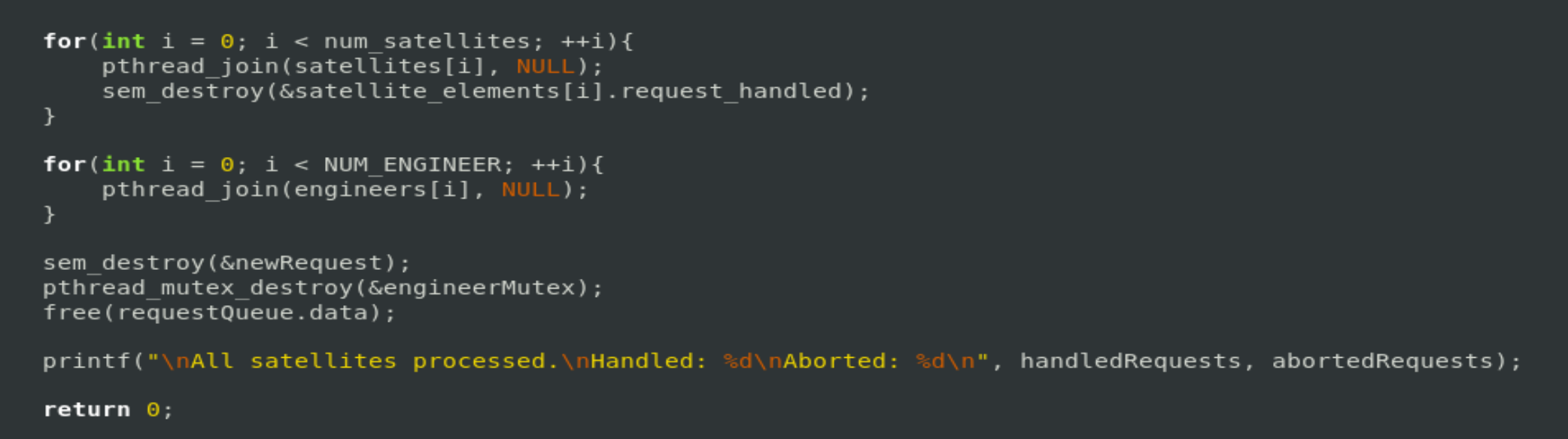


 An array-based priority queue and the necessary methods to do operations on the queue are defined. The requests are stored in descending order based on their priority.

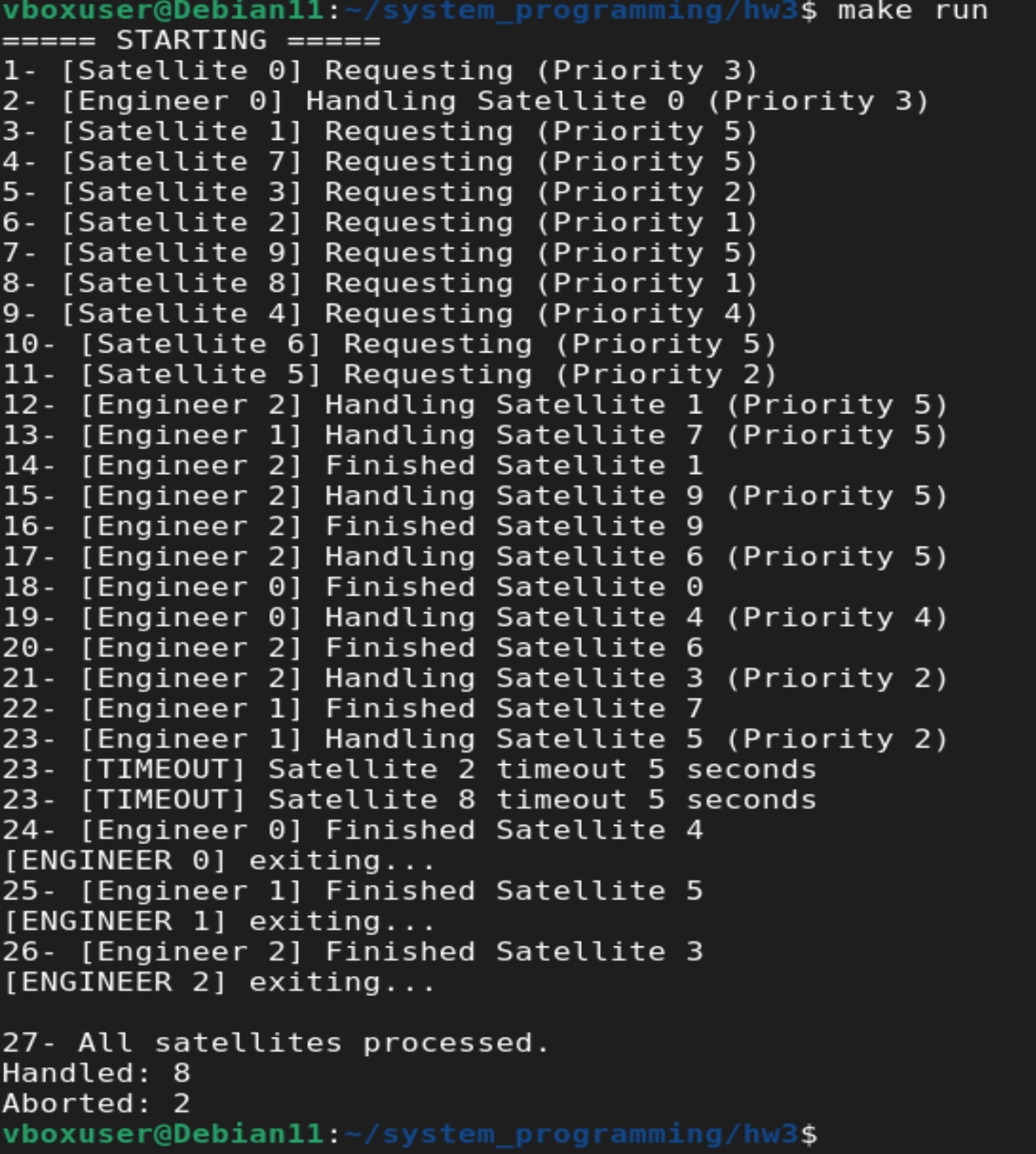
Initialization of the queue, engineer threads, satellites, and satellite threads.



Destruction of dynamically allocated sources, mutex, and semaphores.



**Example Output**

****

**-Line 19** clearly shows that engineers process the request with the highest priority, even if lower-priority requests arrived earlier.

**-Line 23,24** demonstrate that if no engineer processes a request within the allowed time, the request is aborted.