Ziya İçöz

ID: 69022

Sarp Göl

ID: 72368

Spacecraft Control with POSIX Threads

Part 1:

In this part, we were asked to implement a simulation using POSIX threads simulating rocket launching, landing assembling, and emergency jobs. To simulate this, we created our jobs with given probabilities in our main function. Since each job had different properties and requirements, we made different functions for different jobs and made threads run those functions specifically. Every time a job occurs due to a probability in the simulation, we created a new thread. To coordinate and synchronize these threads we created another thread, and this thread runs the control tower function. Control tower thread is a singleton and synchronizes the jobs accordingly. Control tower thread and main function use real-time and run until the input time exceed the simulation time.

Race condition and main Implementation:

To prevent race conditions and data corruption we used mutexes and queues. Each job and pads have a different mutex. When a job arrives, job runs its function and locks its mutex. Enqueues itself to its queue and then waits for the signal of the control tower. Whenever the conditions are satisfied, control tower sends a signal to the thread indicating that a pad is empty and ready for a job. After the signal thread locks the pad mutex and does its job (sleeps for a given amount of time). After completing the job enqueues itself from its queue and unlocks the pad mutex. We used mutexes and queues also for pads to avoid any race conditions between the jobs.

Giving Priority to Landing jobs:

In our implementation landing jobs have a higher priority due to the lack of technology. We implemented this behavior by checking the landing job queue size first and signaling landing threads before other job types.

Pad Mutexes, Enqueue and Dequeue:

To prevent collisions in pads we used mutexes and queues for both pads. When a job decides to use the pad after getting signaled by the control tower, it locks the mutex of the corresponding pad, dequeues itself from its queue, and then enqueues itself to the pad queue. After the job is completed, thread dequeues itself from the pad queue and unlocks pad mutex simulating completion.

Part 2:

Starvation solution to ground jobs:

In this part, we added maximum-wait time and counter for the land jobs. To implement this, we added a peek function to the queue.c file and a start and end time to the job struct. Peek function returns the first element in the queue without removing the job from the queue. We check the waiting time and count for ground jobs and if one of them satisfies the conditions we give priority to that job type however this causes starvation for landing jobs. To solve this starvation problem, we added waiting time and count for all the job types. This simulates aging and priority of a job increases as the time passes since its first entry time to the queue. Enter time of a job is initialized when a job is first enqueued to its own queue.

Part 3:

Emergency landing:

To simulate emergency landing of rockets we created 2 emergency landing threads. These threads also run their emergency landing function and follow the same procedure for enqueuing and dequeuing to their job queue. Emergency jobs have the highest priority. We do not allow any new jobs after an emergency occurs and control tower immediately signals emergency threads whenever a pad is empty. If both pads are empty emergency jobs can use both pads at the same time. The using procedure of the pads is same with other job types but they are handled differently by the control tower.

Part 4:

Log Files:

To calculate turnaround time, we added entry and end time in job struct. When a job enqueues itself to its queue or dequeues itself from its queue, we initialized start or end time and wrote the data into a log file using fprintf and fputs functions.

Important Note:

Part 1 and Part 2 Control Towers:

We have two control tower functions for part 1 and part 2. Control Tower function is for part 1 and it is not starvation free. Starvation free one which implements part 2 is Controltower2.