Jose Gordillo
V00773366
CSC 360
Assignment 2
Design Document

Thread Overview

The design for this assignment is meant to be as simple as possible. It will use n + 1 threads where n represents the number of threads specified on the file. The extra thread is the main thread which oversees starting the other n threads. The other n threads are the ones specified by the assignment; they will be the ones "transmitting". So, they will sleep the amount of time specified (arrival time) and then, when it's their turn, will sleep again (transmission time). As mentioned before, the main thread will start the other threads. After starting all of them, it will sleep until all the threads have finished executing. All the threads will work mostly independently of each other. They will only share resources and signal each other.

Mutex Overview

The design I'm using will only use a single mutex. This mutex (called trans_mutex) oversees protecting the request and release pipe operations. The request pipe operation includes changing the value of the using-pipe flag, adding itself to the waiting queue, and transmitting. The release pipe operation consists of changing the value of the using-pipe flag and then broadcasting to all the threads waiting conditionally. Since the mutex protects the request pipe function, the waiting queue will be protected and it will not be modified concurrently. This guarantees the integrity of the queue.

Data Structures Overview

The design used needs only 3 data structures: 2 arrays and 1 linked list. The first array saves the ids of the threads that are created by the main thread. The second array is used by the main thread to store the read info about the other threads. This comes directly from the file the user specifies. Each element in this array is a struct that contains id, arrival time, transmission time, priority, and original position in file. This data structure is just used to know what information to pass to the created threads and to know how many threads to create. The final data structure is the actual waiting queue. This queue is a linked list of structs. The structs hold the relevant info for each thread. From all these data structures, only the waiting queue is protected by a mutex. The other 2 are not at risk of concurrent reading/writing.

The idea is to use the waiting queue such that only the first element in the queue can transmit. Since the queue is protected by the mutex, it will always have the right order of flows; only one node at a time can be added. After a flow stops transmitting, it is removed from the queue. Also, the queue is resorted every time a new node is added.

Convar Overview

The design uses a single convar that is used to stop all the waiting threads that are not the next one in queue from transmitting out of turn. Therefore, this convar is associated with the trans_mutex mentioned earlier. The idea is that if a thread that's not supposed to transmit gets into the critical section, then a while loop will catch it and it will be forced to wait conditionally until the convar is broadcasted. If it is not its time (again), then it will be forced to wait conditionally again and again until it is its turn. Once the convar is signaled and it gets out of the loop, the thread can proceed to remove itself from the queue and then transmit.

<u>Algorithm Overview</u>

```
Main:
      Read file and start respective threads (each thread calls thd_function)
      Wait for all the created threads to finish


Thd_function:
      Sleep the arrival_time amount of seconds
      Print “Thread arrived”
      If pipe is available and the queue is empty:
            Lock pipe
            Print “started transmitting”
            Transmit
            Print “finished transmitting”
            Release pipe
      Else:
            Add itself to queue and sort queue
            Wait until being first in line and pipe is free
            Remove itself from queue
            Lock pipe
            Print “started transmitting”
            Transmit
            Print “finished transmitting”
            Release pipe
```