Jason Louie
V00804645

CSC 360 Design Document

1. How many threads are you going to use? Specify the task that you intend each thread to perform.

There will be as many threads as indicated in the first integer of the input file. Each thread will call thrFunction() which prints the statements of arrival, start and finish time as well as go through a waiting queue when there is already a flow in use.

2. Do the threads work independently? Or, is there an overall “controller” thread?

The threads work independently and will be signalled out of the wait when the following conditions are met:

- condition variable is signalled
- No flow is on transmission, maintained by the ‘available’ variable
- The thread/flow that is first in line (top) of the queue

3. How many mutexes are you going to use? Specify the operation that each mutex will guard.

I will be using one mutex variable to ensure the critical section is atomic (all of the critical section must be performed, or none if the transmission is already in use)

The mutex operation in requestPipe() function guards all actions when the very first flow enters transmission as well as other flows until wait() is called (causing mutex to temporarily unlock until signalled). After the threads under wait() becomes signalled, the mutex protects the maintenance of the queueList.

In releasePipe(), the mutex protects the operations of releasing the flow from transmission as well as broadcast.

4. Will the main thread be idle? If not, what will it be doing?

Yes, the main thread will be waiting/idle until all the threads are finished by means of pthread_join().

5. How are you going to represent flows? what type of data structure will you use?

I will be using the typedef struct to hold the values of a flow’s id, arrival time, transmission time and priority as provided in the template.

6. How are you going to ensure that data structures in your program will not be modified concurrently?

The data structure will never be modified after the array ‘flowList’ parses all of the flows in the input file. The only modification lies in the queueList which is only a tracker to maintain the positions of each flow up next.

7. How many convars are you going to use? For each convar:
(a) Describe the condition that the convar will represent.
I will be using one convar.

(b) Which mutex is associated with the convar? Why?

The convar named ‘trans_cvar’ will be associated with the mutex ‘trans_mtx’

(c) What operation should be performed once pthread cond wait() has been unblocked and re-acquired the mutex?

After wait() has been signalled and unblocked, the function ‘pthread_mutex_unlock(&trans_mutex)’ needs to be called at the end of the critical section; otherwise no other threads will gain access to the mutex and it will cause a spin loop.

8. In 25 lines or less, briefly sketch the overall algorithm you will use. You may use sentences such as: If a flow finishes transmission, release trans mutex.

The mfs.c program reads a text file followed by a format and stores each flow in an array holding its respective values. Then for each flow, a thread is created, calling upon thrFunction(). The first flow enters and prints each operation it undergoes (Flow X arrives/starts/finishes at time Y) since initially there are no flows on transmission. Although all other threads undergo the same procedure, they must enter a queue that is sorted after each insert to wait for the first flow to finish. When the first flow that entered transmission finishes, it signals a broadcast to all other threads telling that there are no flows on transmission. The thread that is first in line will get priority according to the rules stated in the requirements; the cycle continues until all threads have had a chance on transmission and finishes.