Assignment #2: MFS Design Document
CSC 360: Operating Systems

Shreyas Devalapurkar
V00827994

Introduction
In this assignment, I implemented a multi-flow task scheduler that allows you to properly schedule flows (tasks) based on priority, arrival time, and transmission time. In order to do this, I used threads, mutexes, and condition variables from the POSIX pthread library in C.

Specific Questions
These were some of the particular questions that were kept in mind before implementation:

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

I will most probably use a thread for each flow. The task that each thread will have will be to print the status updates and output all state changes to the console.

2) Do the threads work independently? Or is there an overall "controller" thread?

The threads will probably work independently. Each thread will perform its desired action and using pthread_join(), I should be able to suspend execution of other threads until the target thread has completed its transmission (terminated) which I will know based on the condvar.

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

I will use a single mutex. It will guard the fact that multiple threads/flows cannot be in transmission simultaneously; only one flow is on transmission at any given time. Also, to avoid concurrent modification of my list.

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

Well each thread will correspond to one flow, so while the program is in execution, there should be one thread being executed at all times. It will be outputting its state changes to the console, and when it has terminated/finished transmission, the next most important flow can begin its transmission. This will continue until all flows have been handled.

5) How are you going to represent flows? What type of data structure will you use?

Flows will be a struct my program where they have attributes such as the priority, flow number, arrival time, and transmission time, numbuffer (the line they correspond to) and also a value that can tell me whether or not a certain flow is currently transmitting or is idle. This way, I will be able to access those necessary fields whenever needed in my program. In order to represent flows, I will use a linked list as my data structure. I decided on this data structure because it is

easier to access and remove certain elements using linked lists, compared to an array implementation. I had used an array in assignment 1 and realized that it will be more effective to use a linked list this time.

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

By using a mutex and locking certain areas to synchronize what is happening, and not allow any concurrent modifications to occur.

7) How many convars are you going to use? For each convar:
   a) Describe the condition that the convar will represent.
   b) Which mutex is associated with the convar? Why?
   c) What operation should be performed once pthread_cond_wait() has been unblocked and re-acquired the mutex?

I will be using a single condition variable cond. The condition that the condition variable will represent is waiting for a flow that has begun transmission to end. Once the flow has finished transmission, then other waiting flows can be unblocked and the most important flow will get next transmission. I have only a single mutex, so it will be the one associated with the condition variable. Once, pthread_cond_wait() has been unblocked and re-acquired the mutex, then I will most probably use a broadcast call to awaken all waiting flows so the next (most important) to-be transmitted flow can begin its transmission.

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.*

- Create a thread for as many number of flows that exist (one thread per flow)
- When a flow arrives, set a mutex lock
- If there are no other flows being transmitted currently, then add the flow to a queue and unlock the mutex and begin its transmission
- If there are other flows being transmitted, then add the flow to a queue, unlock the mutex, but give a notification that the current flow is waiting
- Keep waiting until the transmitting flow has completed its transmission. Use pthread_cond_wait() for this. Then the next most important flow can begin its transmission
- Use pthread_join() to suspend execution of other threads until the completion of the currently transmitting thread based on the condition variable
- When a flow finishes transmission, set a mutex lock and move to the next flow in the queue
- Use a broadcast signal to awaken all threads, select the one which needs to be handled next and unlock the mutex
- Every time a flow is added to the queue, it is done so based on importance rules (priority, arrival time, transmission time)
- Once all threads/flows are done transmitting, exit and destroy the mutex and condition variables