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

The number of threads we are going to use is the same as the number of flows.

These threads are created to execute the flows which are imported from the input file.

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

The threads are working independently. There is no controller thread.

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

I'm going to use 2 mutexes: trans_mtx and queue_mtx.

trans_mtx is used to lock and unlock the transmission pipe to make sure only one thread is on transmission every time. The operations are :

```
pthread_mutex_t trans_mtx = PTHREAD_MUTEX_INITIALIZER ;
pthread_mutex_lock(&trans_mtx);
pthread_mutex_unlock(&trans_mtx);
pthread_mutex_destroy(&trans_mtx);
```

queue_mutx is used to lock and unlock the queue to make sure only one flow could be added into queue each time. The operations are:

```
pthread_mutex_t queue_mtx = PTHREAD_MUTEX_INITIALIZER ;
pthread_mutex_lock(&queue_mtx);
pthread_mutex_unlock(&queue_mtx);
pthread_mutex_destroy(&queue_mtx);
```

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

No, it won't be idle. It will executes the functions and methods in main method.

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

I'm going to create a struct named flow to represent object flow.

As following shows:

```
typedef struct
{
        float arrivalTime ;
        float transTime ;
        int priority ;
        int id ;
} flow;
```

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

In my program, all the flows will be put into a list and there will also have a thread list where each thread executes the corresponding flow which is at the same index as it. There is a mutex called queue_mtx. Once the queue is going to be modified, this queue will be locked such that each time, only one thread can access this queue list.

7. How many convars are you going to use? For each convar:

I'm going to use one convar: trans_cvar.

(a) Describe the condition that the convar will represent.

when a thread finishes its transmission, I will use a function pthread_cond_broadcast() to send signal to all the threads which are locked by this condition variable.

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

trans_mtx is associated with the convar. Because all the waiting threads will be locked by trans_mtx which is related to trans_cvar. Once a thread finishes transmission, it will send a signal to all the waiting threads and then the flow with highest priority will start transmission once it accepts the signal.

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

Once pthread_cond_wait() has been unblocked, use:

```
pthread_mutex_lock(&queue_mtx);
trans_id=queueList[count_queue-1]->id;
pthread_mutex_unlock(&queue_mtx);
```

If the trans_id(id of flow with highest priority) equals the id of tested flow, this flow will stop waiting and start transmission. If not, this flow will continue waiting.

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.

```
read flows from input file
create threads for each flow
For each thread, after created:
    wait until arrived
    send request to pipe:
        lock trans mutex
        if pipe is available&&queue is empty,start transmission and unlock trans mutex
        if not, put it int0 the queue and sort the queue
        wait until the pipe is available again and has highest priority;
        otherwise, continue waiting
    lock the queue mutex, update the queue, and unlock queue mutex
    wait the transmission time of executing flow
    once the transmission is finished,unlock the trans mutix, send signal to each thread
                        in  waiting queue and then release the pipe
wait until each thread are finished
destroy mutexes and condition variable
the program ends
```