Answer 3

Locks, Mutexes and Semaphores

```
/* Semaphores */
sem_t *tShirtGuys; // club coordinators semaphore

/* Mutex */
pthread_mutex_t stageLock = PTHREAD_MUTEX_INITIALIZER; // access to Stage

/* CV */
pthread_cond_t noAStage // no Acoustic Stage Empty
pthread_cond_t noEStage // no Electric Stage Empty
pthread_cond_t anyStage // no Stage Available
pthread_cond_t anySinger // some singer available to join a performance
pthread_cond_t stageOrPerf // no Stage or performance going on with no
singer
```

Why not use more semaphores?

- Semaphores while a good theoritical tool are much less used than CVs in practical applications
- Using Semaphores with mutex is redundant when CV already automatically locks CV
- Sem_timedwait is not in POSIX, it is not available in all linux distros or even MACOSX (which i use) (Pthread_cond_wait is in POSIX)

Threads

- All the musicians are different threads, they exhibit all their states in this thread

Global Resources

```
int tMin, tMax;
int waitTime;
int aStage, eStage;
int performanceNoSigner = 0;
char singersQueue[QUEUE_SIZE][NAME_SIZE];
int qReadPos, qWritePos;
int stageNoA[QUEUE_SIZE], stageNoE[QUEUE_SIZE];
int numAcoustic, numElectric;
```

- All the stages are global stages (eStage, aStage)
- Number of performances without a Singer (performanceNoSigner)
- singersQueue (names of singers who have sent a signal to join, qRead and qWrite help interface with queue)
- Performance time for each singer (tMin, tMax)
- StageNoA, StageNoE stores the number of the stages available or not (for bonus)

Working

main

1. takes input
2. initializes semaphores
3. creates all the threads with correct arguments and gives stages according to instruments
4. waits for all threads to rejoin and destroys the semaphores

Musicians and Singers

1. parses args to `musicianData` struct called `data`
2. sleeps till time of arrival
3. when it arrives it gets the maximum time it will wait and stores it in `timeLimit`
4. then according to its stage it calls `performAcoustic`, `performElectric`, `performAny`. The three functions are very similar. Their job is to wait appropriate conditional variable (like aStage for acoustic stage) and acquire the stageLock. The singers case is handled separately in `singersCase`
5. In Singers case with the stage it also checks for the available performances to join after which it decides to join a type of stage in `goOnStage` or `joinAMusician` depending on available resources and randomization. If it joins the stage singer is treated like a normal musician from now on.
6. In `joinAMusician` the singers adds itself to `singersQueue` and the thread exits
7. For other musicians at after acquiring the stageLock and waiting on the conditional variable if the time is above limit they leave. If not they call `performanceHandler`
8. `performanceHandler` takes stage and musician info as arguments. It takes care of printing the output regarding the performances and decreasing the amount of stages, and deciding the length of performance. This step also gets the stage number for the bonus requirement. Then `performanceHandler` calls `perform` which is the actual performance part.
9. `perform` is a conditional timed wait on `anySinger` CV. If the thread gets a signal on this CV before performance duration then this means a singer has joined the performance. The name of singer is read from the singer queue and the the code sleeps for additional 2 seconds after sleeping for the leftover time of the normal duration is over. This function returns true if a singer joined else false. Also the name of singer is passed back to the function through an argument
10. After `perform` function is over the control jumps back to`performanceHandler` .It prints required output and increments the available stages while signaling the respective CVs.
11. Now the thread waits on `collectTShirts` which uses semaphore to limit the threads entering this section to number of coordinators. This is also called for the singer if the `perform` function returns true. When the musician gets a tShirt then the thread exits.