Project Description

The course project on *Synchronization* is a research and programming activity that will enable the students to write solutions to synchronization problems using semaphores, locks and monitors (and condition variables).

CalTrainII Automation

CalTrainII has decided to improve its efficiency by automating not just its trains but also its passengers. (See Figure 1 for CalTrainII system, consisting of 8 train stations and maximum of 16 trains. The number of trains being dispatched depends on the passenger demand.)

From now on, passengers will be robots. Each robot and each train is controlled by a thread. You have been hired to write synchronization functions that will guarantee orderly loading of trains. You must define a structure ***struct station***, plus several functions described below.

When a train arrives in the station and has opened its doors, it invokes the function
station_load_train(struct station *station, int count)

where count indicates how many seats are available on the train. The function must not return until the train is satisfactorily loaded (all passengers are in their seats, and either the train is full or all waiting passengers have boarded). ***Note: the number of seats may vary among trains and should be treated as an input parameter***

When a passenger robot arrives in a station, it first invokes the function
station_wait_for_train(struct station *station)

This function must not return until a train is in the station (i.e., a call to station_load_train is in progress) and there are enough free seats on the train for this passenger to sit down. Once this function returns, the passenger robot will move the passenger on board the train and into a seat (you do not need to worry about how this mechanism works). Once the passenger is seated, it will call the function
station_on_board(struct station *station)
to let the train know that it's on board.

Create a file caltrain.c (or use JAVA, C++, C#) that contains a declaration for **struct station** and defines the three functions above, plus the function **station_init**, which will be invoked to initialize the station object when CalTrainII boots. In addition:

1. Research on the use of locks and monitors (condition variables)in process/thread synchronization.
2. Write a solution (program) using functions and locks for condition variables (monitors).

   Use only the following functions (e.g., no semaphores or other synchronization primitives).

   lock_init (struct lock *lock)
   lock_acquire(struct lock *lock)
   lock_release(struct lock *lock)
   cond_init(struct condition *cond)
   cond_wait(struct condition *cond, struct lock *lock)
   cond_signal(struct condition *cond, struct lock *lock)
   cond_broadcast(struct condition *cond, struct lock *lock)

You may assume that there is never more than one train in the station at once, and that any passenger can get-on and get-off in any station.

Your code must allow multiple passengers to board simultaneously (it must be possible for several passengers to have called station_wait_for_train, and for that function to have returned for each of the passengers, before any of the passengers calls station_on_board).

Your code must not result in busy-waiting.

3. Write another solution (program) using semaphores.
4. Compare the two solutions (experiment and test using different case scenarios).
5. Prepare the final Report using the outline below:

   I. Introduction. Give a brief description of the project.

   II. Process Synchronization . Discuss each process synchronization method/technique/solution used in relation to the problem (CalTrain)

   III. Results and Analysis. Compare the behavior/performance of the two solutions for each case scenario you have identified. Use tables and figures appropriately to show your comparative analysis.

   IV. Conclusion.

   V. References.

   Criteria for grading the solution and documentation

   - 30% Correctness and Appropriateness of the solutions to conduct the synchronization activity
   - 30% Complexity (degree of difficulty) of the synchronization solution
   - 20% Visualization & Graphics
   - 10% Clarity and Thoroughness of the comparative analysis (monitors vs semaphores)
   - 10% Overall document presentation, e.g., format (title page, page numbers, sections, tables and figures), references, and language (spelling, choice of words and grammar)

6. Make an oral presentation of your project output on March 24 (Friday) @ 9 am

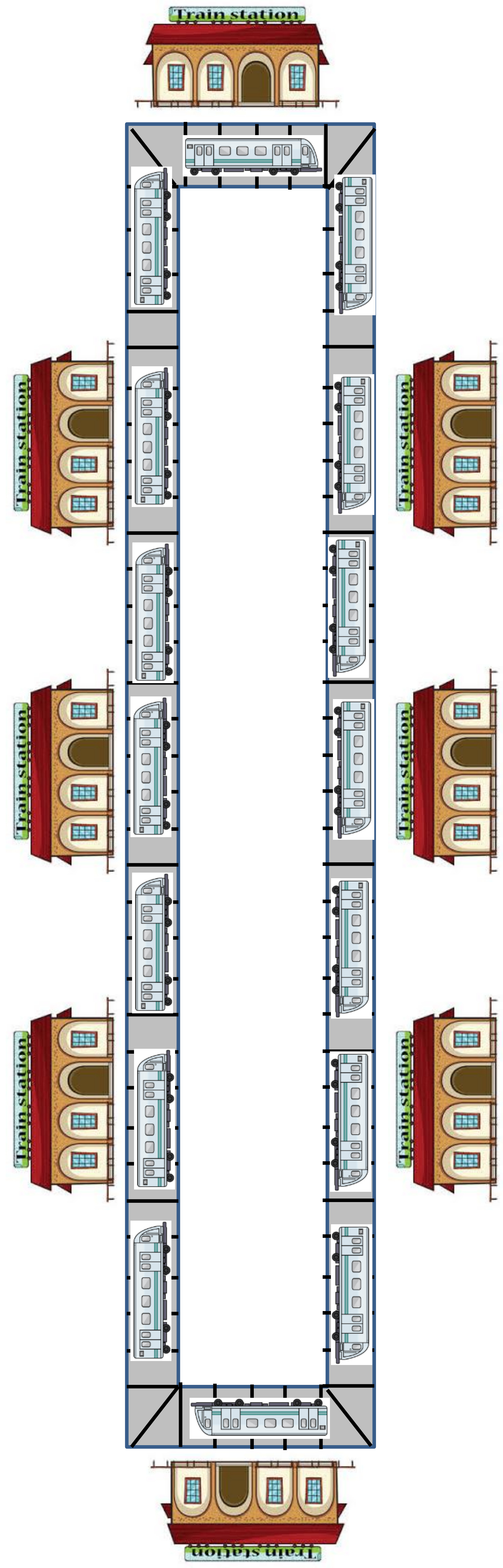
Final Deliverables:

Technical Report following ACM publication format (www.acm.org/sigs/publications/pubform.doc)

- Softcopy (pdf) of the Technical Report must be uploaded using dropbox on July 23, 2017 @ 23:59 PM. Follow the file naming convention: **MCO2 <section>_<lastnames of members>.<ext>**
- Printed copy of the Technical Report must be submitted during the first 15 minutes of the class on **July24/25, Monday/Tuesday**.
- Late submissions will receive 10 points deduction per day. No late submissions will be accepted after July 26.
- Demo schedule TBA
- **Plagiarized works will automatically be given a grade of 0.0 for the course.**

References:

http://web.stanford.edu/~ouster/cgi-bin/cs140-winter16/problemSet0.php


Train station
Train station
Train station
Train station
Train station
Train station
Train station
Train station