# Report on exercise #1

Matteo Corain S256654

Laboratory #13 – System and device programming – A.Y. 2018-19

The proposed solution for the exercise makes use of four types of data structures:

- `TRAIN`: it is used to store information about a train, including:
  - An identifier (`DWORD id`);
  - The number of passengers (`DWORD passengers`);
  - The current station in which the train is (`DWORD currentStation`);
  - The direction in which the train is travelling (`DWORD direction`);
  - The line on which the train is travelling (`DWORD line`);
  - A boolean flag, set when the train has to be suppressed (`BOOL suppress`);
  - The handle of the relative train thread (`HANDLE trainThread`);
  - A pointer to the following train structure (`struct train* next`, the trains data structure is managed as a list).
- `STATION`: it is used to store information about a station, including:
  - An identifier (`DWORD id`);
  - The number of commuters in the two directions (`DWORD commuters[2]`);
  - The count of the number of times the number of commuters in each direction is higher than 75 (`DWORD countHigh[2]`) and lower than 30 (`DWORD countLow[2]`);
  - The event to be signaled when the commuters thread updates the number of commuter (`HANDLE eventNewCommuters`);
  - The events representing the two binaries inside the station (`HANDLE binaries[2]`);
  - The handle of the relative station thread (`HANDLE stationThread`);
  - The list of trains originated by the station (`LPTRAIN trains`);
  - A critical section to protect the access to the `commuters` variable (can be accessed both by the commuters and the train threads, `CRITICAL_SECTION csCommuters`).
- `SHARED_VARS`: it is used to hold data that is common to all threads, including:
  - The three time parameters: `DWORD time1`, `time2` and `time3`;
  - The number of created trains (`DWORD trainCount`, declared as `volatile` since it will be incremented using an interlocked function);
  - The start time of the application, used for printing messages with some relative time information (`DWORDLONG startTime`);
  - The handle of the commuters thread (`HANDLE commutersThread`);
  - The semaphore on which the commuters thread will wait for the stations to produce new commuters (`HANDLE semNextCommuters`);
  - The array of stations (`STATION stations[NUM_STAT]`).
- `THREAD_DATA`: it is used to pass the correct indices to the station and train threads, including the ID of the station, the ID of the train and a pointer to the shared variables.

The main thread, after having checked the number of parameters, parses the time parameters from the command line, initializes the stations structures and the relative synchronization variables and creates the stations (passing a `THREAD_DATA` structure opportunely initialized to each of them) and commuters (passing the `SHARED_VARS` structure) threads. Then, it starts waiting for the commuters thread to complete, operation which effectively blocks the thread until the application is closed.

The commuters thread, after initializing the random seed with its thread identifier, performs the following actions in an infinite loop:

- It waits for all the stations to perform a signal on `semNextCommuters`;
- For each station, it updates, protected in the `csCommuters` critical section, the number of commuters in each direction by summing a random number in 0-100;
- It signals the `eventNewCommuter` event to unlock each station thread;
- It sleeps `time1` seconds before restarting its operations.

The station threads, after having retrieved the corresponding station structure from the shared variables, also runs an infinite loop, in which:

- It waits for new commuters to be produced, by using the `eventNewCommuter` event;
- For each direction, it checks the number of commuters (protected in the `csCommuters` critical section), appropriately incrementing or resetting the counters `countHigh` and `countLow`;
- In case `countHigh` gets equal or higher to 3, a new train is created as follows:
    - It allocates and populates a `TRAIN` structure; in order to prevent concurrent updates to the `trainCount` shared variable (e.g. stations are concurrently allocating new trains), the increment is performed with the corresponding interlocked function;
    - The line on which the train will travel is chosen based on the station, or randomly in case the train departs from the central station;
    - The number of passengers for the train is decided as the minimum value between the number of commuters on the platform and the maximum capacity of the train; this number of passengers is also removed from the number of waiting commuters, possibly causing the reset of the `countHigh` counter (it may fall below 75);
    - It allocates and populates a `THREAD_DATA` structure to be passed to the train thread, which is created shortly after (it will be deallocated at the end of the train thread);
    - It attaches the allocated train at the end of the trains list of the station.
- In case `countLow` gets equal or higher to 3 and the station has allocated at least a train (the train list does not begin with a NULL value), the first train is suppressed as follows:
    - The first train structure is retrieved by accessing the head of the trains list;
    - The list is updated by setting as the first element the next one;
    - The suppression flag is set in the train structure, which causes the termination of the relative train thread;
    - The train thread is joined and the respective handle closed;
    - The train structure is freed.
- It releases the `semNextCommuters` semaphore to unlock the commuters thread.

Finally, the train threads, after having retrieved the corresponding train structure from the shared variables by traversing the list of allocated trains by the initial station, runs a loop until the `suppress` flag is not set, in which it performs the following actions:

- It selects the next station to proceed to, based on the current station and on the line on which the train it is travelling;
- It sleeps `time2` seconds while travelling to the next station;
- It waits to enter the station on the correct binary, represented by the two `binaries` events of the station it is trying to access;
- It acquires the `csCommuters` critical section of the newly entered station (it will update the number of commuters);

- It selects a random number of commuters leaving the train (up to the number of current passengers), which is removed from the total number of passengers of the train;
- It selects a random number of commuters leaving the train (up to the minimum between the current remaining capacity of the train and the current number of commuters at the platform in the train's direction), which is removed from the total number of commuters at the platform and added to the total number of train's passengers;
- It releases the critical section;
- It waits for time3 seconds waiting for passengers to get in or out;
- It releases the binary of the station, starting its travel to the next station.

When a train thread terminates, it frees its associated `THREAD_DATA` structure (the `TRAIN` structure is freed in the station thread which terminates the train).