<u>CS F372 - OPERATING SYSTEMS, SEM II 2023-2024</u>

<u>ASSIGNMENT 2: AIR TRAFFIC CONTROL SYSTEM</u>

<u>MAXIMUM MARKS: 45</u>

<u>HARD DEADLINE: 12:30 AM, 22nd APRIL 2024</u>

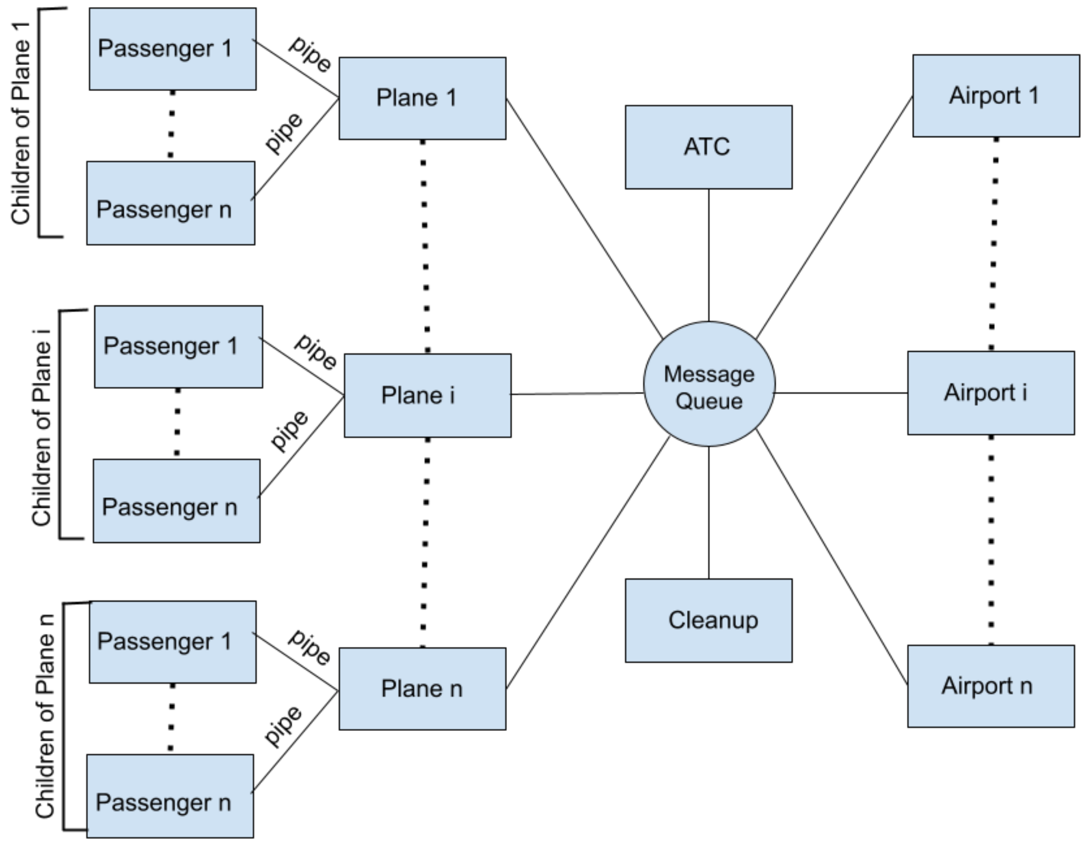
Note: This document contains the problem statement, assignment constraints, submission guidelines, plagiarism policy and demo guidelines. Please go through the entire document and not just the problem statement.

Problem Statement:

In this assignment, we are going to use the different OS concepts learnt so far to create an **Air Traffic Control System**. This system consists of the following entities (simulated as processes):

- Plane (Each plane is a single-threaded process). Two types of planes are considered.
    - Passenger plane: includes passengers, 5 cabin crew members and 2 pilots
    - Cargo plane: includes only 2 pilots
- Airport (Each airport is a multi-threaded process)
- Air Traffic Controller (ATC) (This is a single-threaded process)
- Passenger traveling on a specific passenger plane (Each passenger process is a child of the corresponding plane process and each passenger process is single-threaded)
- Cleanup (This is a single-threaded process)

The overall block diagram of the Air Traffic Control System is depicted in the following figure. Note that the diagram only depicts passenger plane processes. If the plane is of type cargo, there will be no passengers onboard. In the figure, ATC implies Air Traffic Controller.



The problem statement of the assignment consists of the following parts.

1. Write a POSIX-compliant C program called **plane.c**:

   a. On execution, each instance of this program creates a separate plane process, i.e., if the executable file corresponding to plane.c is plane.out, then each time plane.out is executed on a separate terminal, a separate plane process is created indicating a new plane along with the passengers (passenger process creation is mentioned later) who are traveling on that plane. For every instance of plane.out executed on the terminal, the same source code should be used. Multiple source files should not be created for different instances of plane processes. Concurrent execution support should be provided for the plane processes. Note that not all plane processes will be necessarily deployed at the same time at the beginning of the system execution.

   b. When a plane process is run, first, it will ask the user to enter a positive integer as its unique plane ID. The prompt message should be as follows:

      ```
      Enter Plane ID:
      ```

      Out of the total plane processes being concurrently executed, it is guaranteed that the plane ID will be sequentially assigned by the user, will be monotonically increasing, will be a positive integer lying in the range 1 to 10 (1 and 10 inclusive) and will be unique for each plane. You can assume that the user always gives a valid input for the plane ID in the proper increasing order.

   c. Next, the plane process requests to know the type of plane from the user using the below prompt message:

      ```
      Enter Type of Plane:
      ```

      If the plane is a passenger plane, the input will be the integer value 1. If the plane is a cargo plane, the input will be the integer value 0. In response to this prompt, the user will only enter the integer value 1 or 0. You do not have to handle errors where the user enters any other integer value. Note that out of all the plane processes deployed, some will be designated as passenger planes and the rest as cargo planes. The number of each type of plane is dynamic and will vary from one execution to the next. So, do not make any assumptions regarding these numbers.

   d. If the plane is of passenger type, the plane process requests to know the number of occupied seats for the plane by asking the user to enter a positive integer. The prompt message should be as follows:

      ```
      Enter Number of Occupied Seats:
      ```

      For each passenger plane process being concurrently executed, this value will be input by the user and will be within the bounds from 1 to 10, both 1 and 10 inclusive. The number of occupied seats is always equal to the number of passengers on the plane. Note that the maximum number of passengers for any passenger plane is 10. Each passenger process will be a child of that plane process. Thus, for each passenger in the plane, the plane process will create a child process and an ordinary pipe for IPC between the plane process and the corresponding passenger. It is to be noted that a separate pipe is used for communication between a plane process and each one of its passenger processes. Thus, if there are four passengers in the plane, the plane process will create four separate pipes. Note that you will be creating POSIX compliant ordinary pipes here.

   e. Each passenger process will take input from the user regarding the weight of his/her luggage using the prompt message below:

      ```
      Enter Weight of Your Luggage:
      ```

      The user will enter the corresponding integer value for the luggage of each passenger process. The maximum weight of a luggage is 25 kgs. Each passenger will have at most 1 piece of luggage. If the passenger is traveling without luggage, the weight is entered as 0. Hence, some input has to be entered for the luggage weight of each passenger. All the passenger processes will communicate the luggage weight to the corresponding plane process using the pipes mentioned in 1.(d). It is guaranteed that luggage weight will be either 0 or a positive integer value less than or equal to 25.

f. Each passenger process will take input from the user regarding his/her body weight using the prompt message below:

```
Enter Your Body Weight:
```

The user will enter the corresponding integer value for each passenger process. The maximum weight of a passenger is 100 kgs. All the passenger processes will communicate their weights to the corresponding plane process using the pipes mentioned in 1.(d). Each individual passenger weight is guaranteed to be a positive integer between 10 and 100, both inclusive.

g. For a passenger plane, the total weight of the plane is calculated as the sum of all weights of the luggage items, the sum of all the weights of the passengers and the sum of all the weights of the crew members. The average weight of the crew members is 75 kgs and there are 7 crew members on a passenger plane, 2 pilots and 5 flight attendants.

h. If the plane is of cargo type, the plane process requires the user to input the number of cargo items the plane is carrying using the prompt message below:

```
Enter Number of Cargo Items:
```

For each cargo plane process being concurrently executed, this integer value will be input by the user and will be within the bounds from 1 to 100, both 1 and 100 inclusive.

i. If the plane is of cargo type, next, the plane process requires the user to input the average weight of the cargo items the plane is carrying using the prompt message below:

```
Enter Average Weight of Cargo Items:
```

For each cargo plane process being concurrently executed, this integer value will be input by the user and will be within the bounds from 1 to 100, both 1 and 100 inclusive.

j. For a cargo plane, the total weight of the plane is calculated as the sum of all weights of the cargo items and the sum of all the weights of the crew members. The average weight of the crew members is 75 kgs and there can be a maximum of 2 crew members on a cargo plane, 2 pilots only.

k. Next, the plane process (both passenger and cargo) requests to know the airport departing from using the prompt message as follows (information about airport number is given later):

```
Enter Airport Number for Departure:
```

The user will enter the airport number corresponding to the airport for departure. The maximum number entered as the airport number is 10 and the minimum number is 1. It is guaranteed that the airport with the corresponding airport number exists and the airport number is an integer value.

l. Next, the plane process (both passenger and cargo) requests to know the airport for arrival using the prompt message as follows:

```
Enter Airport Number for Arrival:
```

The user will enter the airport number corresponding to the airport for arrival. The maximum number entered as the airport number is 10 and the minimum number is 1. It is guaranteed that the airport with the corresponding airport number exists and the airport number is an integer value. In the rest of the document, the term plane/plane process will refer to both passenger and cargo planes. It is assumed that a plane will not depart from and arrive at the same airport.

m. The details of airport arrival, airport departure, plane ID, total weight of the plane, type of plane and number of passengers (this does not include crew members and is relevant only for passenger planes) need to be stored appropriately using some data structure (but definitely not using files, marks will be deducted if you use files).

n. Once the plane is ready for departure, the plane process will send a message to the air traffic controller containing the plane details (as mentioned in 1.(m)) and the air traffic controller in turn sends a message containing the plane details to the departure airport to begin the boarding/loading and departure process via the single message queue described in 2.(c) later. For a cargo plane, boarding implies loading of the cargo.

o. Once the plane arrives at the arrival airport and the deboarding/unloading process is completed, the air traffic controller process (after receiving a confirmation from the arrival airport) informs the plane process that the deboarding/unloading is completed via the single message queue of 2.(c). For a cargo plane, deboarding implies unloading the cargo. Upon receiving this intimation, the plane process displays the following message before terminating itself.

```
Plane <Plane ID> has successfully traveled from Airport <Airport Number
of Departure Airport> to Airport <Airport Number of Arrival Airport>!
```

The boarding/loading and deboarding/unloading processes and the duration of the journey are simulated using `sleep()` . Each boarding/loading and deboarding/unloading process will take 3 seconds and each plane journey duration will be 30 seconds. The plane process should only terminate upon receiving the intimation from the air traffic controller.

2. Write a POSIX-compliant C program called **airtrafficcontroller.c**:

a. Only one instance of this process will be executed in the entire system, and this instance will keep running until it is asked to terminate by the cleanup process. This process is used to direct planes to airports and from one airport to another.

b. When the air traffic controller process is run, it will first ask for the number of airports that the air traffic controller will handle/manage. The prompt message should be as follows:

```
Enter the number of airports to be handled/managed:
```

It is guaranteed that the number of airports will be a positive integer lying in the range 2 to 10 (2 and 10 inclusive). You can assume that the user always gives a valid input for the number of airports.

c. Once the number of airports has been input, the air traffic controller process will set up a single message queue, to receive messages from and send messages to airport and plane processes as needed. It should send messages to the appropriate airports and planes as required (refer to 1.(n) and 1.(o)). Here, you need to figure out the logic for the one-to-one mapping between a sender and a receiver via a single message queue. Note that the communication between the air traffic controller process and the airport and plane processes should involve only a single message queue. Your implementation should not use multiple message queues.

d. When the air traffic controller receives the plane details from a plane process, the air traffic controller informs the departure airport to begin the boarding/loading and the takeoff process via the single message queue. Once the plane has taken off, the departure airport informs the air traffic controller via the single message queue that the takeoff process is complete via the single message queue. The air traffic controller also informs the arrival airport regarding the arrival of the plane. Once a plane lands at the arrival airport and the deboarding/unloading process is complete, the arrival airport informs the air traffic controller that the landing and the deboarding/unloading process is complete via the single message queue. The air traffic controller then informs the same to the plane process and the plane process terminates.

e. When a plane travels from one airport to another, the air traffic controller keeps a track of this in the **AirTrafficController.txt** file, by appending the following message at the end of the file for each plane journey:

```
Plane <Plane ID> has departed from Airport <Airport Number of Departure
Airport> and will land at Airport <Airport Number of Arrival Airport>.
```

This information should be appended to the file after a plane departs from the departing airport and before it lands at the arrival airport, i.e., while the plane is making its journey. Assume the AirTrafficController.txt file to be present in the same directory as the other source files.

f. When the air traffic controller process receives the termination request for the entire system from the cleanup process (described later), it stops accepting any future messages from planes, and sends termination messages to all airports via the single message queue that are connected to it only when all planes have landed and completed their deboarding process, and no planes are currently present at any airport. Once all airport processes have been terminated and confirmation messages about airport termination have been received, the air traffic controller process will perform relevant cleanup activities and terminate.

3. Write a POSIX-compliant C program called **cleanup.c**:

a. This cleanup process will keep running along with the other processes and only one instance of this program is executed. This process will keep displaying a message as:

```
Do you want the Air Traffic Control System to terminate?(Y for Yes and
N for No)
```

b. If N is given as input, the cleanup process keeps running as usual and will not communicate with any other process, displaying the above message again. If Y is given as input, the cleanup process will inform the Air Traffic Controller process to terminate using the single message queue. After passing the termination information to the Air Traffic Controller, the cleanup process will terminate. When the Air Traffic Controller receives the termination information from the cleanup process, it begins the termination task only when all planes have landed and completed their deboarding process, and no planes are currently present at any airport. The termination task involves cleaning up all the IPC constructs, sending termination information to all the airport processes and other cleanup tasks as required.

4. Write a POSIX-compliant C program called **airport.c:**

a. On execution, each instance of this program creates a separate airport process, i.e., if the executable file corresponding to airport.c is airport.out, then each time airport.out is executed on a separate terminal, a separate airport process is created. For every instance of airport.out executed on the terminal, the same source code should be used. Multiple source files should not be created for different instances of airport processes. Concurrent execution support should be provided for the airport processes. Note that the number of airport processes deployed is equal to the number of airports given as user input to the air traffic controller process. Assume that all airport processes are deployed together at the beginning.

b. Each airport process displays a prompt message as below:

```
Enter Airport Number:
```

Airport numbers will start from 1 and keep on monotonically increasing. This number is guaranteed to be a positive integer lying in the range 1 to 10, both inclusive. Hence, we are assuming that the maximum number of airports is 10.

c. Next, another prompt message is displayed to the user as shown below:

```
Enter number of Runways:
```

The number of runways for each airport will be an even number lying in the range 1 to 10, both inclusive.

d. Each runway has one attribute - loadCapacity, i.e., a runway having a loadCapacity of X can handle the arrival/departure of a plane having total weight lesser than or equal to X. Possible values of loadCapacity are integers lying in the range 1,000 and 12,000 kgs, both values inclusive. For taking the loadCapacity input for each runway of an airport, a prompt message is displayed as follows:

```
Enter loadCapacity of Runways (give as a space separated list in a single line):
```

Apart from the runways specified by the user, we will always assume that an additional backup runway is present at every airport having loadCapacity of 15,000 kgs. Hence, if the user has given 4 as the input for number of runways for an airport, there will be a 5th runway having a loadCapacity of 15,000 kgs. This runway will only be used for arrival or departure of planes having total weight greater than the loadCapacity of any other runway of that airport. The backup runway is never used if there exists at least one runway having loadCapacity greater than or equal to the plane's total weight, even if that involves some amount of delay in the arrival or departure in case the runway is being used by another plane.

e. An airport process receives a message from the air traffic controller regarding the arrival or departure of a plane via the single message queue. Upon receiving such a message, the airport process creates a new thread. Note that an airport can be handling multiple arrivals and departures. Now, two cases can arise: Departure and Arrival.

f. *Departure:*

i. The airport process receives the plane details (that will arrive at or depart from that airport) from the air traffic controller via the single message queue.

ii. After receiving the plane details, the airport process creates a new thread. This thread finds a runway such that the loadCapacity of the runway is greater than or equal to the total weight of the plane. Note that the thread should do a best-fit selection here. For eg. The total weight of the plane is 2500 kgs and there are two available runways having loadCapacity values of 2557 kgs and 3000 kgs, then the thread selects the runway having loadCapacity 2557 kgs. If however, none of the runways have a loadCapacity greater than or equal to the plane's total weight, the backup runway is used. Once the thread has assigned a runway to a plane, it takes care of the boarding/loading process of 3 seconds and then a message is sent by this thread to the air traffic controller specifying that a plane with plane ID has successfully completed takeoff from the airport. Additionally, the thread displays a message on the console as follows:

```
Plane <Plane ID> has completed boarding/loading and taken off from Runway No. X of Airport No. Y.
```

Note that the same thread handles the boarding/loading process and the takeoff.

iii. The thread is terminated after the above message is displayed. Takeoff is simulated using a sleep of 2 seconds before sending the message to the air traffic controller.

iv. Only one plane can be present on a runway at a given time, and one plane can only takeoff from one runway. You need to ensure appropriate synchronization here because of the best-fit selection criterion. In case a plane needs to wait since the most appropriate runway is unavailable, you need to simulate it using proper synchronization construct(s). Only mutex/semaphore should be used for this. If you use `sleep()` instead, marks will be deducted.

g. *Arrival:*

i. The air traffic controller informs the airport about the arrival of a plane (along with the plane details) via the single message queue.

ii. Upon receiving this information, the airport process creates a new thread for handling the plane's arrival. Each arrival will be handled by a new thread created by the airport process. The thread created by the airport process selects a runway from the available runways for landing of the plane using the best-fit logic explained in Departure. However, if the loadCapacity of none of the airport runways is greater than or equal to the total weight of the arriving plane, then the additional backup runway of 15,000 kgs will be used. Once a runway has been assigned for the plane's landing, landing is simulated with a sleep of 2 seconds before performing deboarding/unloading of the plane (deboarding/unloading is of 3 seconds).

iii. Deboarding/unloading is handled by the same thread that is handling the arrival of a plane.

iv. Once the plane has deboarded/unloaded, the thread sends a message to the air traffic controller process indicating that it has reached its destination and deboarded/unloaded successfully. Then, the thread prints the following message on the console:

```
Plane <Plane ID> has landed on Runway No. X of Airport No. Y and
has completed deboarding/unloading.
```

After displaying the above message, the thread terminates.

v. Synchronization is required for handling simultaneous arrivals (this has been explained in Departure). Only mutex/semaphore should be used for synchronization. If you use `sleep()` instead, marks will be deducted.

Upon receiving the termination intimation from the air traffic controller via the single message queue, the airport process terminates after performing any cleanup activity as applicable.

5. For the sake of clarity, the entire workflow is reiterated here. You should not simply refer to the sub-points of 5 to complete the entire implementation.

a. Plane process is created, it takes user inputs and sends a message to the air traffic controller regarding its departure along with the details.

b. The air traffic controller first checks if the termination request was received earlier from the cleanup process. If yes, then it rejects the plane's departure request, informing the plane that no further departures will happen, else performs the step mentioned in 5.(c).

c. The air traffic controller receives the plane's request and informs both the departure and arrival airports. This is because, when the plane is currently taking off from a given airport, the arrival airport must know that a plane will be arriving soon from another airport, and must wait for that plane (before terminating in case a termination intimation was received). Otherwise, let us assume that the arrival airport does not know about the existence of the departing plane. Then, if the arrival airport had received a termination request earlier, it may end up terminating if there are no other requests at that airport, and the plane that departed will not be able to contact the arrival airport.

d. The departure airport receives the message from the air traffic controller, assigns a runway and completes the boarding/loading and takeoff processes. The departure airport informs the air traffic controller and displays the relevant message on the screen.

e. Once the takeoff completes, the air traffic controller receives a message from the departure airport and makes an appending entry into the .txt file.

f. The arrival airport, on receiving the intimation from the air traffic controller, assigns a runway for the plane's landing and waits for the plane's arrival (the plane needs to complete its 30 seconds journey). The arrival airport handles the landing and the deboarding/unloading processes, sends a message to the air traffic controller that landing and deboarding/unloading completed successfully and displays the relevant message on the screen.

g. The air traffic controller receives a message from the arrival airport that landing was successful, and informs the plane that the flight was successful.

h. The plane process displays the relevant message on the screen and terminates.

6. Only passenger planes and the corresponding passengers bear parent-child relationships.
7. Use `wait()` and `pthread_join()` appropriately as required. Perform all relevant error handling for all system calls properly without which marks will be deducted.
8. You can assume that all the C files as well as the .txt file are present in the same directory.
9. Only the specific IPC mechanisms (pipes and a single message queue) mentioned in the problem statement should be used and only for the purpose as mentioned. No other IPC mechanism should be used in the entire implementation.
10. You should not use file(s) instead of message queue. Only a single message queue should be used in the entire implementation.
11. You should not use any additional `sleep()` apart from the ones mentioned in the problem statement.
12. Synchronization for runways should be implemented using mutex/semaphore only.
13. You are allowed to use additional data structure(s).
14. You are not allowed to use shared memory.

15. You are not allowed to take any additional inputs other than those mentioned in the problem statement. If you do, marks will be deducted.

16. There should not be any need to terminate any process using Ctrl + C.

17. You need to close unused pipe ends, wherever applicable.

18. Marks will be deducted if you violate any constraint mentioned in the problem statement.

19. YOU SHOULD NOT USE ANY UNSTRUCTURED PROGRAMMING CONSTRUCTS, ESPECIALLY goto STATEMENTS.

SUBMISSION GUIDELINES:

- All programs should be POSIX-compliant C programs.
- All codes should run on Ubuntu 22.04 systems.
- Submissions are to be done through a Google Form the details of which will be shared later.
- There should be only submission per group.
- Each group should submit a zipped file containing all the relevant C programs and a text file containing the correct names and IDs of all the group members. The names and IDs should be written in uppercase letters only. The zipped file should be named as **GroupX_A2** (X stands for the group number). You will be notified of your group number after a few days.
- If there are multiple submissions from a single group, any one of the submissions will be considered randomly.
- Your group composition for Assignment 2 has been frozen now. You cannot add or drop any group member now.
- No extensions will be granted beyond the given deadline.
- Do not attempt any last-minute submissions. You need to be mindful of situations such as laptops not working at the last moment, websites becoming unresponsive, etc.
- There will be a penalty for late submissions.

PLAGIARISM POLICY:

- All submissions will be checked for plagiarism.
- Lifting code/code snippets from the Internet is plagiarism. Taking and submitting the code (partially or entirely) of another group(s) is also plagiarism. However, plagiarism does not imply discussions and exchange of thoughts and ideas (not code).
- All cases of plagiarism will result in awarding a hefty penalty. Groups found guilty of plagiarism may be summarily awarded zero also.
- All groups found involved in plagiarism, directly or indirectly, will be penalized.
- The entire group will be penalized irrespective of the number of group members involved in code exchange and consequently plagiarism. So, each member should ensure proper group coordination.
- The course team will not be responsible for any kind of intellectual property theft. So, if anyone is lifting your code from your laptop, that is completely your responsibility. Please remember that it is not the duty of the course team to investigate cases of plagiarism and figure out who is guilty and who is innocent.

- Please be careful about sharing code among your group members via any online code repositories. Be careful about the permission levels (like public or private). Intellectual property theft may also happen via publicly shared repositories.

DEMO GUIDELINES:

- The assignment also consists of a demo component to evaluate each student's effort and level of understanding of the implementation and the associated concepts.
- The demos will be conducted in either the I-block labs or D-block labs. Therefore, the codes submitted via the Google Form will be tested on the lab machines.
- No group will be allowed to give the demo on their own laptop.
- The codes should run on Ubuntu 22.04.
- All group members should be present during the demo.
- Any absent group member will be awarded zero.
- The demos will be conducted in person.
- The demos will not be rescheduled.
- Though this is a group assignment, each group member should have full knowledge of the complete implementation. During the demo, questions may be asked from any aspect of the assignment.
- Demo slots will be made available in due time. You need to book your demo slots as per the availability of your entire group.
- Only the code submitted through the Google Form will be used for the demo.
- Be mindful of the submission that you make. No re-submissions will be allowed.
- Each group member will be evaluated based on the overall understanding and effort. A group assignment does not imply that each and every member of a group will be awarded the same marks.
- Any form of heckling and/or bargaining for marks with the evaluators will not be tolerated during the demo.