Middle East Technical University Department of Computer Engineering

CENG 334

Introduction to Operating Systems

Spring 2016-2017

Homework 2 - Marine Transportation Simulation

Due date: 28 04 2017, Friday, 23:59

1 Objective

This assignment aims to familiarize you with the basics of multi-threaded applications and synchronization using C. Your task is to implement a simulator for marine transportation. You will implement necessary structures and simulate each entity through individual threads. You will use mutexes, semaphores and condition variables to synchronize these threads.

Keywords: *Marine Transportation Simulation, Thread, Semaphore, Mutex, Condition Variable*

2 Problem Definition

We want to simulate the transportation of cargoes among docks by a group of ships (threads). There are a N_d docks each of which can store an unlimited number of cargoes. Each cargo is labeled with its destination dock's id. Initially each dock is pre-loaded with cargoes.

You are asked to implement ship threads that are created at the beginning of the simulation. Initially the ships will be empty and be given a pre-determined route of docks and will be at open sea. During their visit, they will load and unload cargoes from each dock based based on their pre-determined route.

The simulation will be subject to the following constraints:

1. Each dock has a limited capacity for ships, and an unlimited capacity for cargoes.
2. Each dock is initialized with a set of cargoes, each labeled a destination dock.
3. The ships will start at open sea with no cargoes onboard.
4. Each ship has a pre-determined capacity for carrying cargoes.
5. Each ship should visit docks in its route in sequence.
6. The time it takes for each leg of of the route is given as part of the route information.
7. For each dock on its route, a ship should
   (a) travel to the dock,
   (b) enter the dock,

(c) unload the cargoes destined for that dock,

(d) load the cargoes that are destined for the subsequent docks on its route unless the dock is the final on the route, and

(e) leave the dock unless the dock is the final on the route.

8. A ship has to wait to enter, if a dock's capacity is full with ships. Ships should be allowed on a first-come, first-served basis.

9. A ship cannot unload its cargo to a dock if there are other ships loading cargoes. The ship wait indefinitely until all loading ships finished loading. The ship should be allowed to unload its cargo immediately after last loading ship finished the loading.

10. A ship cannot load cargo from a dock if there are other ships unloading their cargoes. The ship has to wait for limited time to load cargo due to unloading ships if there are cargoes destined for the subsequent docks on ship' route. If there is no such cargo, then the ship should be notified immediately. If other ships don't finish unloading their cargo on time, then the ship leaves the dock without loading new cargo. Otherwise, the ship should be allowed to load cargo immediately after last unloading ship finished the unloading.

11. Multiple ships can unload their cargoes simultaneously at a dock, however single ship cannot unload multiple cargoes simultaneously.

12. Multiple ships can load their cargoes simultaneously at a dock, however single ship cannot load multiple cargoes simultaneously.

13. Once the ship arrives at its final dock, it is destroyed after unloading its cargoes.

You may find pseudo-code ship threads at below,

Algorithm 1 Ship thread main routine

```
function ShipRoutine(Route, Capacity, TravelTime, ArrivalTime, ShipId)
    Storage ⇐ {}
    travelTime ⇐ ArrivalTime
    WriteOutput(ShipId, 0, 0, CREATE_SHIP)
    for each dock with id DockId on Route do
        Travel(ShipId, DockId, travelTime)
        travelTime ⇐ TravelTime
        Enter(ShipId, DockId)
        if ∃ cargo with destination DockId on Storage then
            Unload(ShipId, DockId, Storage)
        end if
        if the dock is not the last entry on the Route then
            Load(ShipId, DockId, Route, Storage, Capacity)
            Exit(ShipId, DockId)
        end if
    end for
    WriteOutput(ShipId, 0, 0, DESTROY_SHIP)
end function
```

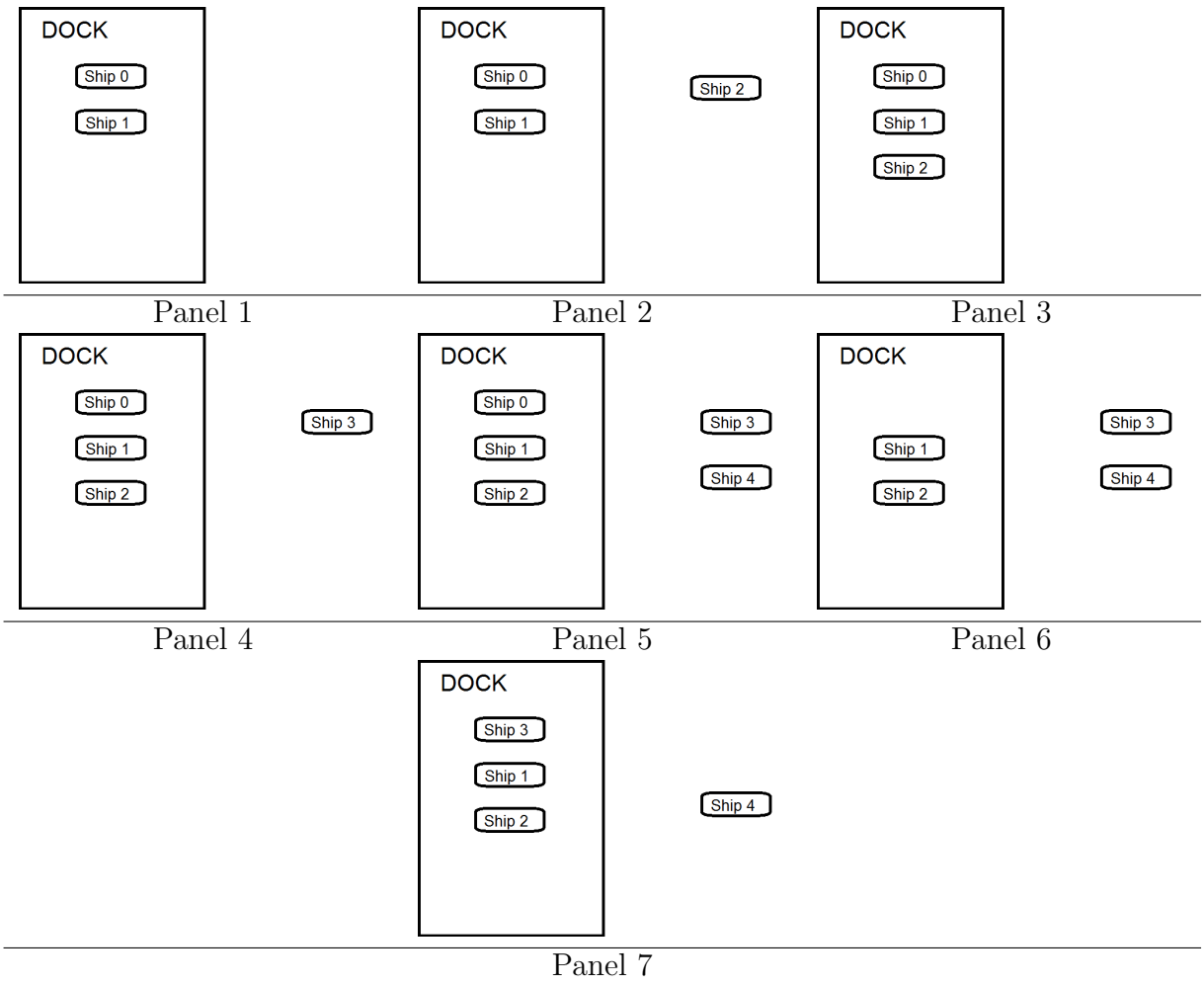
Algorithm 2 Ship thread sub-routines

```
function TRAVEL(ShipId, DockId, travelTime)
    Sleep travelTime milliseconds
end function

function ENTER(ShipId, DockId)
    WriteOutput(ShipId, DockId, 0, REQUEST_ENTRY)
    Query the dock for entrance
    Wait for the permission, busy wait is forbidden
    WriteOutput(ShipId, DockId, 0, ENTER_DOCK)
    Enter the dock
end function

function UNLOAD(ShipId, DockId, Storage)
    WriteOutput(ShipId, DockId, 0, REQUEST_UNLOAD)
    Query the dock for unloading cargo
    Wait for the permission, busy wait is forbidden
    for each cargo with id CargoId in Storage do
        if destination of the cargo is the dock with id DockId then
            WriteOutput(ShipId, DockId, CargoId, UNLOAD_CARGO)
            Sleep 2 milliseconds
            Storage ⇐ Storage \ {the cargo}
        end if
    end for
end function

function LOAD(ShipId, DockId, Route, Storage, Capacity)
    if |Storage| ≠ Capacity then
        WriteOutput(ShipId, DockId, 0, REQUEST_LOAD)
        Query the dock for new cargo
        Wait for the permission for 20 milliseconds, busy wait is forbidden
        if Timeout or no available cargo then
            Return
        end if
        for each cargo with id CargoId stored at the dock do
            if destination of the cargo included in the remaining part of Route then
                WriteOutput(ShipId, DockId, CargoId, LOAD_CARGO)
                Sleep for 3 milliseconds
                Storage ⇐ Storage ∪ {the cargo}
                if |Storage| = Capacity then
                    Return
                end if
            end if
        end for
    end if
end function

function EXIT(ShipId, DockId)
    WriteOutput(ShipId, DockId, 0, LEAVE_DOCK)
end function
```

3 Sample Scenarios

3.1 Entry

Lets consider entry synchronization for single dock. We will assume that dock has enough capacity for 3 ships, and 2 ships are docked at moment. 3 more ship will arrive before these 2 ships leave. For the simplicity, we will ignore any cargo related details.



Explanation

Panel 1: There are two ships already docked on the dock, and there is no ship waiting to enter.

Panel 2: Ship 2 arrives, and requests entry.

Panel 3: Ship 2 immediately given permission to enter, and docks as there is room for one more ship.

Panel 4: Ship 3 arrives, and requests entry.

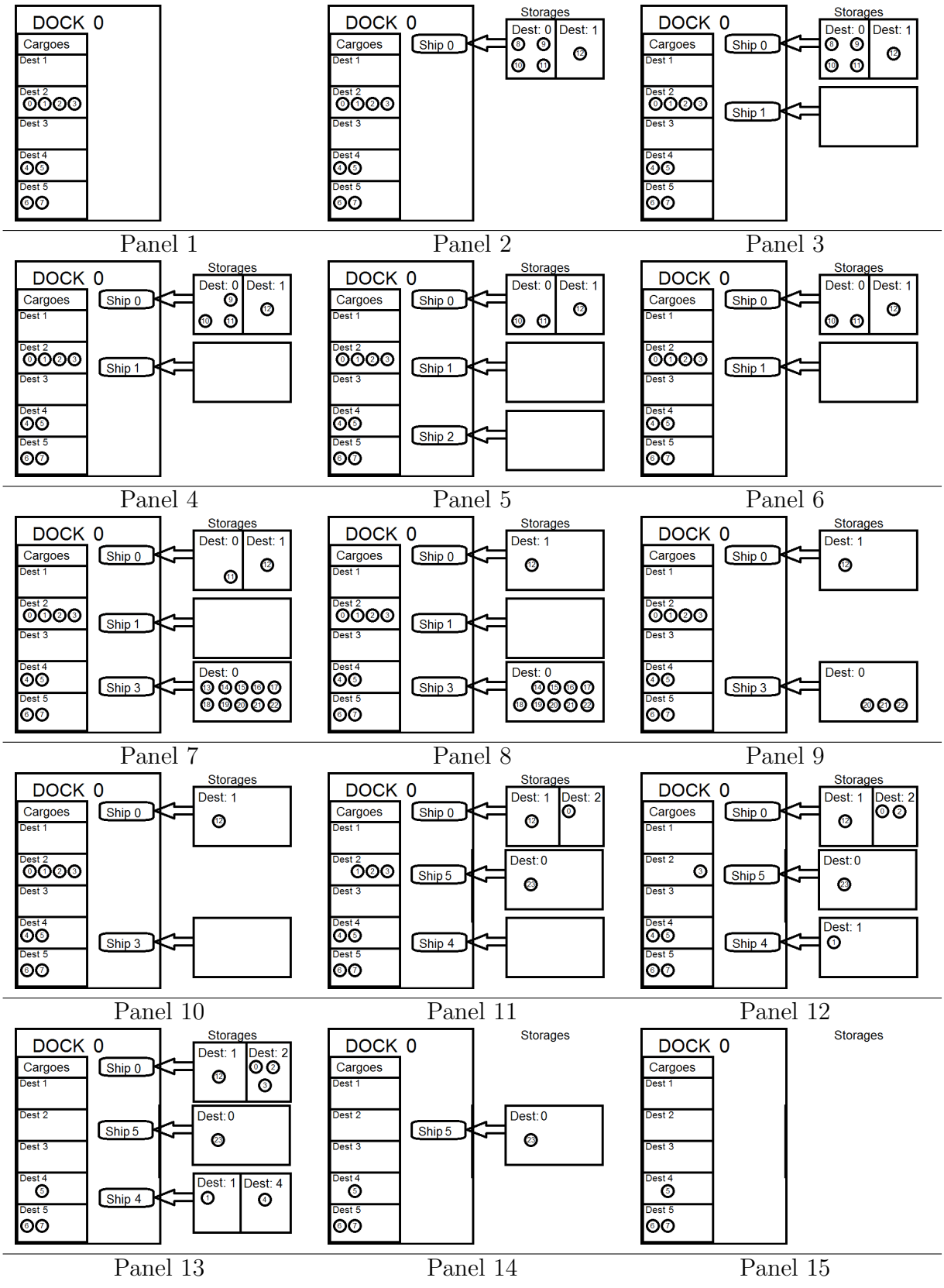
Panel 5: Ship 3 waits as there is no remaining space on the dock. Ship 4 also arrives, and requests entry.

Panel 6: Ship 0 leaves the dock. Ship 3 and 4 still waiting for permission.

Panel 7: Ship 3 given permission immediately after ship 0 leaves. Ship 3 given permission instead of ship 4, because it came first.

3.2 Load and Unload

Lets consider synchronization of load and unload operations for single dock. We will assume that dock has enough capacity for 6 ships. For the simplicity, we will ignore any entrance/exit related details.



Explanation

Panel 1: There are no ships on the dock 0. Currently there are 4 cargoes destined to dock 1, 2 cargoes destined to dock 4 and 5.

Panel 2: Ship 0 arrives, and requests permission to unload its cargo. Its capacity is 5 and its route is 0-1-3-2.

Panel 3: Ship 0 given permission to unload. Ship 1 arrived, and requested permission to load cargo. Its capacity is 3 and its route is 0-4-3.

Panel 4: Ship 0 unloads cargo with id 8, and sleeps for 2 milliseconds. Ship 1 is waiting.

Panel 5: Ship 0 unloads cargo with id 9, and sleeps for 2 milliseconds. Ship 1 is still waiting. Ship 2 arrives, and requests permission to load cargo. Its capacity is 6 and its route is 0-3-1.

Panel 6: Ship 2 leaves immediately from the dock, as there is no available cargo.

Panel 7: Ship 0 unloads cargo with id 10, and sleeps for 2 milliseconds. Ship 1 is still waiting. Ship 3 arrives, and requests permission to unload cargo. Its capacity is 10 and its route is 1-3-0.

Panel 8: Ship 0 unloads cargo with id 11, sleeps for 2 milliseconds, and requests permission to load cargo. Ship 1 is still waiting. Ship 3 receives permission immediately, unloads cargo with id 13 and sleeps for 2 milliseconds.

Panel 9: Ship 3 unloads cargoes with id 14, 15, 16, 17, 18, and 19, and sleeps 12 milliseconds. Ship 0 is still waiting. Ship 1 leaves the dock due to timeout.

Panel 10: Ship 3 unloads cargoes with id 20, 21, and 22, and sleeps 6 milliseconds. Afterwards, ship 3 removed from the simulation and ship 0 is given permission to load cargo.

Panel 11: Ship 0 loads cargo with id 0 and sleeps 3 milliseconds. Ship 4 arrives and requests permission to load cargo. Permission given immediately. Its capacity is 2 and its route is 3-0-2-4. Ship 5 arrives and requests permission to unload cargo. Its capacity is 10 and its route 0-3-1.

Panel 12: Ship 0 loads cargo with id 2 and sleeps 3 milliseconds. Ship 4 loads cargo with id 3 and sleeps 3 milliseconds. Ship 5 is waiting.

Panel 13: Ship 0 loads cargo with id 3 and sleeps 3 milliseconds. Ship 4 loads cargo with id 4 and sleeps 3 milliseconds. Ship 5 is still waiting.

Panel 14: Ship 0 leaves the dock as there is no available cargo destined to some dock in its route at the dock 0. Ship 4 leaves the dock as it has 2 cargoes on board which is its capacity. Ship 5 given permission to unload cargo.

Panel 15: Ship 5 unloads cargo with id 23 and sleeps 2 milliseconds. Afterwards, it requests permission to load cargo and leaves the dock as there is no available cargo.45238

4 Implementation Specifications

1. Each ship should be implemented as separate threads. When a ship thread created, following function call should be made

   `WriteOutput`(ShipId, 0, 0, **CREATE_SHIP**)

2. You should call `InitWriteOutput` function before creating ship threads.

3. Ship threads should be created in detached mode, and they should be released when ship reaches at its final destination. Thread should make following function call immediately before it releases its resources.

   `WriteOutput`(ShipId, 0, 0, **DESTROY_SHIP**)

4. Simulator should use `WriteOutput` function to output information, and no other information should be printed.

5 Input Specifications

Information related docks, ships and cargoes will be given through stdin (standard input). First line will contain number of docks (N_D), number of ships (N_S), and number of cargoes (N_C). Second line will contain N_d many non-zero integers. i^{th} integer in this line will correspond to capacity of the dock with id i. Following N_S lines will have following format,

- T C A L d_0 d_1 ... d_{L-1}

$L + 4$ many space separated unsigned integers, where

- T represents the travel time of the ship. The ship will sleep T milliseconds to travel between two docks.
- C represents the number of cargo that can be carried by the ship.
- A represents the arrival time of the ship to first dock in its route. The ship should arrive at that dock A milliseconds after the start of the simulation.
- L is the length of the ship' route.
- d_i is the id of i^{th} dock in the ship' route.

i^{th} line contains information regarding the ship with id i.
Next N_C lines will have following format.

- A D

2 space separated unsigned integers, where

- A is the id of the dock which the cargo is initially stored at.
- D is the id of the dock which is the destination of the cargo.

i^{th} line contains information regarding the cargo with id i.

6 Homework Specifications

- Your code must be written in C on Linux. No other platforms and languages will be accepted.
- You are allowed to use `pthread.h` and `semaphore.h` libraries for the threads, semaphores, condition variables and mutexes. Your solution should not employ busy wait. Your Makefile should not contain any library flag other than `-lpthread`. It will be separately controlled.
- Submissions will be evaluated with black box technique with different inputs. Consequently, output order and format is important. Please make sure that calls to `WriteOutput` function done in the correct thread and correct step. Also, please do not modify "`writeOutput.c`" and "`writeOutput.h`" files as your submission for these files will be overwritten,

- There will be penalty for bad solutions. Non terminating simulations will get zero from the corresponding input.
- Your submission will be evaluated on lab computers (ineks).
- Everything you submit should be your own work. Usage of binary source files and codes found on internet is strictly forbidden.
- Please follow the course page on piazza for updates and clarifications.
- Please ask your questions related to homework through piazza instead of emailing directly to teaching assistants.
- Partial grades will be given according to following scheme,
    1. Admission of ships into docks satisfies capacity constraints → 20 points
    2. Admission of ships into docks in the correct order → 15 points
    3. Mutual exclusion of cargo load and unload → 25 points
    4. Cargo load constraints satisfied → 35 points
    5. Timing constraints and step order → 5 points

7 Submission

Submissions should be done via COW. Create a tar.gz file named hw2.tar.gz that contains all your source code files together with your Makefile. Your tar file should not contain any folders. Your code should compile and the resulting executable should run using following command sequence.

```
$ tar -xf hw2.tar.gz
$ make all
$ ./simulator
```

The name of your executable is important. **If there is a mistake in any of the 3 steps mentioned above, you will lose 10 points.**

8 Cheating

We have zero tolerance policy for cheating. People involved in cheating will be punished according to the university regulations. Cheating Policy: Students may discuss the concepts among themselves or with the instructor or the assistants. However, when it comes to doing the actual work, it must be done by the student alone. As soon as you start to write your solution or type it, you should work alone. In other words, if you are copying text directly from someone else - whether copying less or typing from someone else's notes or typing while they dictate - then you are cheating (committing plagiarism, to be more exact). This is true regardless of whether the source is a classmate, a former student, a website, a program listing found in the trash, or whatever. Furthermore, plagiarism even on a small part of the program is cheating. Also, starting out with code that you did not write, and modifying it to look like your own is cheating. Aiding someone else's cheating also constitutes cheating. Leaving your program in plain sight or leaving a computer without logging out, thereby leaving your programs open to copying, may constitute cheating depending upon the circumstances. Consequently, you should always take care to prevent others from copying your programs, as it certainly leaves you open to accusations of cheating. We have automated tools to determine cheating. Both parties involved in cheating will be subject to disciplinary action. [Adapted from `http://www.seas.upenn.edu/cis330/main.html`]