Middle East Technical University Department of Computer Engineering

CENG 334

Introduction to Operating Systems

Spring 2020-2021

Homework 2 - Drone Transport Simulation

Due date: 11 05 2021, Tuesday, 23:59

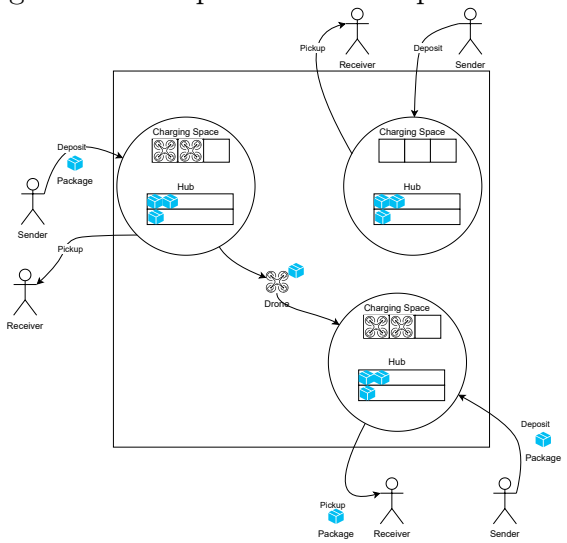
1 Objective

This assignment aims to familiarize you with the development of multi-threaded applications and synchronization using C or C++. Your task is to implement a simulator for drone transport by simulating different agents within the scenario using different threads. Towards this end, you will be using mutexes, semaphores and condition variables for synchronization between them.

Keywords— Thread, Semaphore, Mutex, Condition Variable

2 Problem Definition

Figure 1: Example Drone Transport Schema



We want to simulate the transportation of packages between senders and receivers using drone hubs and drones (see Figure 1). Specifically, senders will deposit their packages to drone hubs, drones will transport the package to the destination hubs and receivers will pick up their packages from the destination hubs. Drone hubs have limited capacity for incoming

and outgoing packages. Outgoing packages are stored at the outgoing storage and incoming packages are stored at the incoming storage in the hub. Drones have limited range and need to charge at hubs to fill up after transporting a package. Similar to limited package space, drone hubs have limited charging (parking) space for drones. Drones needs to park in one of these spaces in order to drop and load packages. In a busy scenario, drone brings a package to a hub, parks in one of the spaces, drop package to incoming storage of the hub, load another package from outgoing storage of the hub, and as long as it is in the parking space, it is charged.

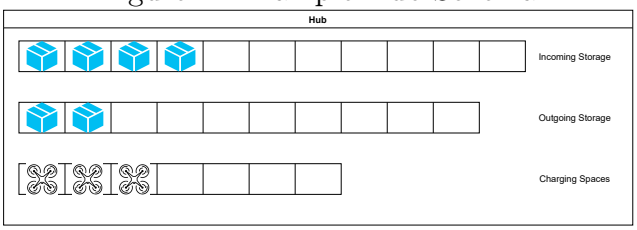
The processing is handled by four types of agents whose functions are described as below:

- **Senders** simulate people sending packages. They each access to a different hub and send packages to receivers. Each package is addressed to a specific hub and a receiver. Senders need to wait when the outgoing storage gets full before sending a packet. There exists a maximum number of packages a sender can deposit. Upon reaching that number, it quits.

- **Receivers** simulate people picking up the packages that they have been sent to. Each receiver is assigned to a different hub. They will pick up their packages from the incoming storage of the hub. They will continue until the hub they have been assigned to, stop its operation.

- **Drones** transport packages from one hub to another. Each drone will reserve package and charging space before departing to the destination. They may need to wait before departing from the hub until they have enough energy for the trip or a space is available at the destination. They can also be called from nearby hubs and will travel to them to without a package if and when that happens. This time they only reserve charging space before departing. They will continue working until all of the hubs have stopped operating.

Figure 2: Example Hub Schema



- **Hubs**(see Figure 2) are at the center of this transportation simulation. They hold all of the incoming and outgoing packages, charge the drones and even call drones when necessary. Each hub has a dedicated sender and a receiver.
  Upon receiving a package from its sender, the hub instructs the drone with the highest current range to deliver the package to the destination. If there are no drones available, it will call drones from nearby hubs in distance order until one is found. If there are no available drones, the hubs will wait either a specific duration or the arrival of a drone. If no drones arrive, it will call the nearby hubs again until a drone is found or a drone arrives. Hubs can store fixed number of packages for incoming and outgoing transfers.
  Hubs operate until all of the senders stop sending packages and they have no package left in their incoming and outgoing storage. They do not have to wait for the drones to finish charging.

You shall implement the sender, receiver, drone and hub as threads and synchronize their activities. The number of each thread, their attributes and starting information will be provided to you (for details see Input Specifications). The threads will be created at the beginning of the simulation. Initially, each one of the senders and receivers is assigned to a given hub and the drone also start parked at given hub. Incoming and outgoing package storage of every hub will be empty and drones will be fully charged. After creation, your main thread should wait for all the agents to finish before stopping. The pseudo-code of simulation agent threads are given in the following pages:

Algorithm 1: Sender thread main routine

Data: ID, Speed, TotalPackages, AssignedHub
CurrentHub $\leftarrow$ *AssignedHub*
RemainingPackages $\leftarrow$ *TotalPackages*
`FillSenderInfo`(*SenderInfo, ID,* CurrentHub*,* RemainingPackages*, NULL*)
`WriteOutput`(*SenderInfo, NULL, NULL, NULL,* **SENDER_CREATED**)
while *there are remaining packages* **do**
    Select a random hub and its receiver to send a package to
    `WaitCanDeposit` ()
    `SenderDeposit` (Package)
    `FillPacketInfo`(*PacketInfo, ID,* CurrentHub*, ReceiverID, ReceivingHub*)
    `FillSenderInfo`(*SenderInfo, ID,* CurrentHub*,* RemainingPackages*, PackageInfo*)
    `WriteOutput`(*SenderInfo, NULL, NULL, NULL,* **SENDER_DEPOSITED**)
    RemainingPackages $\leftarrow$ RemainingPackages $- 1$
    Sleep for the duration given for the sender (see Input Specifications)
end
`SenderStopped` ()
`FillSenderInfo`(*SenderInfo, ID,* CurrentHub*,* RemainingPackages*, NULL*)
`WriteOutput`(*SenderInfo, NULL, NULL, NULL,* **SENDER_STOPPED**)

The functions are explained below:

- `WaitCanDeposit`: Wait until there is outgoing storage space available on the hub.
- `SenderDeposit`: Informs the hub that the package has been deposited to the outgoing storage.
- `SenderStopped`: Signal all the hubs that this sender has stopped sending packages. If this is the last sender, hubs can quit after their incoming and outgoing package storage are emptied.

Algorithm 2: Receiver thread main routine

```
Data: ID, Speed, AssignedHub
CurrentHub ← AssignedHub
FillReceiverInfo(ReceiverInfo, ID, CurrentHub, NULL)
WriteOutput(NULL, ReceiverInfo, NULL, NULL, RECEIVER_CREATED)
while there are active hubs do
    Check if there any (incoming) packages for the receiver on the hub
    for each of the packages do
        ReceiverPickUp (Package)
        FillPacketInfo(PackageInfo, SenderID, SendingHub, ID, CurrentHub)
        FillReceiverInfo(ReceiverInfo, ID, CurrentHub, PackageInfo)
        WriteOutput(NULL, ReceiverInfo, NULL, NULL,
         RECEIVER_PICKUP)
        Sleep for duration given for the receiver (see Input Specifications)
    end
end
FillReceiverInfo(ReceiverInfo, ID, CurrentHub, NULL)
WriteOutput(NULL, ReceiverInfo, NULL, NULL, RECEIVER_STOPPED)
```

The functions are explained below:

- `ReceiverPickUp`: Informs the hub that the package has been picked up from the incoming package storage. This means that if there are drones waiting to deliver to this hub, one of them can deliver at that point.

Algorithm 3: Drone thread main routine

```
Data: ID, Speed, StartingHub, MaximumRange
CurrentHub ← StartingHub
CurrentRange ← MaximumRange FillDroneInfo(DroneInfo, ID, CurrentHub,
  CurrentRange, NULL, 0)
WriteOutput(NULL, NULL, DroneInfo, NULL, DRONE_CREATED)
while there are active hubs do
    WaitSignalFromHub ()
    if The current hub assigns a package to be delivered to another then
        WaitAndReserveDestinationSpace (DestinationHub)
        WaitForRange ()
        CurrentRange ← min(CurrentRange + ChargedRange, MaximumRange)
        TakePackageFromHub (Package)
        FillPacketInfo(PackageInfo, SenderID, SendingHub, ReceiverID,
          ReceivingHub)
        FillDroneInfo(DroneInfo, ID, CurrentHub, CurrentRange, PackageInfo, 0)
        WriteOutput(NULL, NULL, DroneInfo, NULL, DRONE_PICKUP)
        Sleep the duration of travel (duration calculation will be given later)
        CurrentRange ← CurrentRange − (Distance/Speed)
        CurrentHub ← DestinationHub
        DropPackageToHub (Package)
        FillPacketInfo(PackageInfo, SenderID, SendingHub, ReceiverID,
          ReceivingHub)
        FillDroneInfo(DroneInfo, ID, CurrentHub, CurrentRange, PackageInfo, 0)
        WriteOutput(NULL, NULL, DroneInfo, NULL, DRONE_DEPOSITED)
    end
    else if A nearby hub is requesting a drone for package delivery then
        WaitAndReserveChargingSpace (DestinationHub)
        WaitForRange ()
        CurrentRange ← min(CurrentRange + ChargedRange, MaximumRange)
        FillDroneInfo(DroneInfo, ID, CurrentHub, CurrentRange, NULL,
          DestinationHub)
        WriteOutput(NULL, NULL, DroneInfo, NULL, DRONE_GOING)
        Sleep the duration of travel (duration calculation will be given later)
        CurrentRange ← CurrentRange − (Distance/Speed)
        CurrentHub ← DestinationHub
        FillDroneInfo(DroneInfo, ID, CurrentHub, CurrentRange, NULL, 0)
        WriteOutput(NULL, NULL, DroneInfo, NULL, DRONE_ARRIVED)
    end
end
FillDroneInfo(DroneInfo, ID, CurrentHub, CurrentRange, NULL, 0)
WriteOutput(NULL, NULL, DroneInfo, NULL, DRONE_STOPPED)
```

The functions are explained below:

- `WaitSignalFromHub:` Waits signal from the hub indicating whether a package has been deposited or a nearby hub is requesting a drone.
- `WaitDestinationSpace:` Waits until an incoming package and charging space is available at the destination hub and reserves them.
- `WaitChargingSpace:` Waits until a charging space is available at the destination hub and reserves it.
- `WaitForRange:` Waits until the current range of the drone is sufficient for the travel between hubs.
- `TakePackageFromHub:` Signal to the hub that an outgoing package has been picked up.
- `DropPackageToHub:` Signal to the destination hub that an incoming package has arrived.

Algorithm 4: Hub thread main routine

```
Data: ID, IncomingSize, OutgoingSize, ChargingSize
FillHubInfo(HubInfo, ID)
WriteOutput(NULL, NULL, NULL, HubInfo, HUB_CREATED)
while there are active senders or packages in either storage do
    WaitUntilPackageDeposited ()
    start:
    if There are drones in the hub then
        Select the drone with the highest current range
        AssignAndNotifyDrone (Package, Drone)
    else
        CallDroneFromHubs ()
        if No drone is found in other hubs then
            WaitTimeoutOrDrone ()
            goto start
        end
    end
end
HubStopped ()
FillHubInfo(HubInfo, ID)
WriteOutput(NULL, NULL, NULL, HubInfo, HUB_STOPPED)
```

The functions are explained below:

- `WaitUntilPackageDeposited`: Waits until a package is deposited from a sender.
- `AssignAndNotifyDrone`: Assign the package to the drone to be delivered to the destination. After assigning the package, notify the drone for delivery.
- `CallDroneFromHubs`: Starting from the closest hub to the farthest, call a drone until one is found. Only one drone should be called at the end if there is any available in the other hubs.
- `WaitTimeoutOrDrone`: Wait a specific duration until a drone arrives. The specific duration is 1 units of time. Time units will be explained later.
- `HubStopped`: Notify all other simulation objects that this hub is not active anymore.

The simulation will be subjected to these constraints.

1. Initially, hubs have no packages in their incoming or outgoing storage. All drones start fully charged. Senders, receivers assigned to and drones start at a given hub.

2. Senders should be able to deposit packages while a drone is picking up a package. Similarly, receivers actions should not block senders and vice versa.

3. Receivers should be able to pick up packages while a drone is depositing a package.

2.1 Distance and Range Specifications

All distance measurement are given in a time unit format. Time unit is measured in milliseconds. The details is given in the following list:

- The distance between two hubs is given in a time format. This means that a drone can travel between two hubs in $distance/speed$ time units. For example if the distance is 10 units between Hub 1 and Hub 2. A drone with a speed of 2 can travel between them in 5 units of time.

- The range of a drone is given in flight of time format. This means that after traveling between hubs a drone will have $range - distance/speed$ range left. Considering the previous example, if the drone had a range of 10, then it would have 5 left after traveling the distance.

- The drones charge based on how much time is spent stationed at a hub. Naturally, it is capped at its maximum range. Again considering the same example, If the drone spent 5 units of time at a hub after taking of again, it will recover 5 of its range back. It will have 10 units of range again.

- The time sender and receiver waits between packets is given in units of time. For example, if the unit time is 100 ms and a sender needs to wait for 2 units of time. It should wait minimum of 200 ms between two packets.

3 Implementation Specifications

1. Each agent should be implemented as separate thread. When a agent thread created, following function call should be made for every agent:

   - **Sender:**
     `WriteOutput`(SenderInfo, NULL, NULL, NULL, **SENDER_CREATED**)
   - **Receiver:**
     `WriteOutput`(NULL, ReceiverInfo, NULL, NULL, **RECEIVER_CREATED**)
   - **Drone:**
     `WriteOutput`(NULL, NULL, DroneInfo, NULL, **DRONE_CREATED**)
   - **Hub:**
     `WriteOutput`(NULL, NULL, NULL, HubInfo, **HUB_CREATED**)

2. You should call InitWriteOutput function before creating threads.

3. Main thread should wait for every thread to finish before exiting. Each agent should make the following call before exiting:

   - **Sender:**
     `WriteOutput`(SenderInfo, NULL, NULL, NULL, **SENDER_STOPPED**)
   - **Receiver:**
     `WriteOutput`(NULL, ReceiverInfo, NULL, NULL, **RECEIVER_STOPPED**)
   - **Drone:**
     `WriteOutput`(NULL, NULL, DroneInfo, NULL, **DRONE_STOPPED**)
   - **Hub:**
     `WriteOutput`(NULL, NULL, NULL, HubInfo, **HUB_STOPPED**)

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

5. You can use `helper.h` and `helper.c` files for time and distance related functions for convenience. Time unit measurement `UNIT_TIME` is given to you in the `helper.h` file as well. It can be subject to change during grading.

4 Input Specifications

Information related to simulation agents will be given through standard input.

First line contains the number of hubs (N_T) in the simulation. Following N_T lines contain the properties of the hubs with i^{th} ID in the following format:
- I_H O_H C_H D_1 D_2 ... D_N

- I_H is an integer representing the size of the incoming package storage.
- O_H is an integer representing the size of the outgoing package storage.
- C_H is an integer representing the number of the charging spaces for drones.
- The values D_1 to D_N represents the distances between current hub to the other hubs. Naturally, the distance to itself will be zero and the values will be symmetrical between hubs.

Following N_T lines contain the properties of the sender with i^{th} ID (All IDs start from 1) in the following format:
- S_S H_S T_S where

- S_S is an integer representing the time sender waits between two packets.
- H_S is an integer representing the assigned hub id of the sender.
- T_S is an integer representing the total number packages that this sender will send.

Following N_T lines contain the properties of the receivers with i^{th} ID in the following format:
- S_R H_R

- S_R is an integer representing the time receiver waits between two packets.
- H_R is an integer representing the assigned hub id of the receiver.

Next line contains the number of drones (N_D) in the simulation. Following N_D lines contain the properties of the drones with i^{th} ID in the following format:
- S_D H_D R_D

- S_D is an integer representing travel speed of the drone.
- H_D is an integer representing the starting hub id of the drone.
- R_D is an integer representing the maximum range of the drone.

Example:

```
3
10 10 3   0 4 8
5  5  2   4 0 2
10 5  2   8 2 0
1  1  10
2  2  5
1  3  15
2  2
1  3
4  1
2
2  1  5
4  3  3
```

NOTE: Normally there will only be a single space separation between values. This example is given in this format for visual clarity.

5 Specifications

- Your code must be written in C or 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.h`, `writeOutput.h`, `helper.h` and `helper.c` files as your submission for these files will be overwritten.
- There will be penalty for bad solutions. Non terminating (deadlocked or otherwise) 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 Cow and ODTUClass for updates and clarifications.
- Please ask your questions related to homework through Cow instead of emailing directly to teaching assistants if your question does not contain solution or code.

6 Submission

Submission will be done via ODTUClass. 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 be able to compile and your executable should run using this command sequence.

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

If there is a mistake in any of the 3 steps mentioned above, you will lose 10 points.