Tema1
Managementul Depozitelor cu Roiuri Robotice Autonome

Dan Novischi

23 martie 2020

1. Introducere

În urma pandemiei cauzate de COVID-19 o companie de distribuție a produselor medicale dorește să își eficentizeze operațiunile de management a unui depozit de marfă utilizand un roi robotic autonom (en. autonomous robotic swarm). Compania dispune de inventarul produselor din depozit, itinerariul tirurilor de transport si roiul robotic.

Scopul temei este să implementăm sistemul de management a depozitului prin care se urmăreste dezvoltarea fucționalităților asociate comportamentelor roiului robotic de colectare a pachetelor de marfă din depozit si încărcarea tirurilor într-un mod eficient.

2. Detaliile Problemei

Tema va fi implementată în limbajul C, folosind mulțimi și liste înlănțuite și este organizata după cum urmează:

- `app.c` – aplicația principala de management.
- `test.c` – teste asociate checker-ului.
- `WearhouseManager.h` – definițiile structurilor de date și prototipurile funcțiilor.
- `WearhouseManager.c` – definițiile funcțiilor de lucru asupra structuriilor de date.
- `Makefile` – makefile-ul pentru compilarea proiectului
- `check.sh` – checker-ul pentru validarea implementarilor asociate functiilor din `WearhouseManager.c`.
- `test/ref` – directorul ce contine rezultatele de referinta.
- `wearhouse` – fisier text care contine inventarul pachetelor din depozit unde:
  - prima line contine numarul pachetelor din depozit (ex: `500`)

  - ficare line succesiva contine detaliile fiecarui pachet, date de prioritatea de livrare si destinatia acestuia (ex: `8,Alba-Iulia`).

- `parckinglot` – fisier text care contine detaliile despre itinerariul tirurilor si robotii care intra in componenta roiului robotic, unde fiecare linie este asociata ori unui tir ori unui robot. Astfel:

  - liniile care încep cu litera `"T"` se referă la tiruri si au in componeta următorii parametrii separți prin virgulă:
    `<destinatia>`
    `<capacitatea tirului>`
    `<timpul de tranzit dus-intors>`
    `<ora de plecare>`
    `<0 = în tranzit sau 1 = parcat la depozit>`

    spre exemplu: "`T,Timisoara,9,21,19,1`" sau "`T,Bistrita,7,11,7,0`".
  - liniile care încep cu litera `"R"` se referă la roboții care alcătuiesc roiul si au un singur parametru dat de capacitatea de stocare a pachetelor. Spre exemplu: "`R,4`" sau "`R,3`".

Constrangerile generale

Constangerile generale asociate sistemului de management al depozitului sunt urmatoarele:

- Robotii care nu efectueaza in mod curent o sarcina de incarcare sau descare se vor mentine intr-o lista de standby.
- Roboții nu pot scoate din depozit mai multe pachete decat capacitatea lor.
- Pentru a depune pachetele robotii vor forma randuri (liste) de descarcare aferente unor tiruri.
- Robotii nu pot descărca alte pachete decat cele aferente destinatiei tirului.
- Robotii se redistribuie odata ce au descarcat pachetele asociate unui tir aferent.
- Tirurile pleaca la ora din itinerariu indiferent ca au fost incarcate sau nu.
- Robotii care stateau la randul asociate unui tir care a plecat, vor fi redistribuiti.
- Tirurile nu pot fi incarcate mai mult decat capacitatea lor de transport.

O vizualizare genrica a sistemului in timpul functionarii este prezentata in Figura 1.

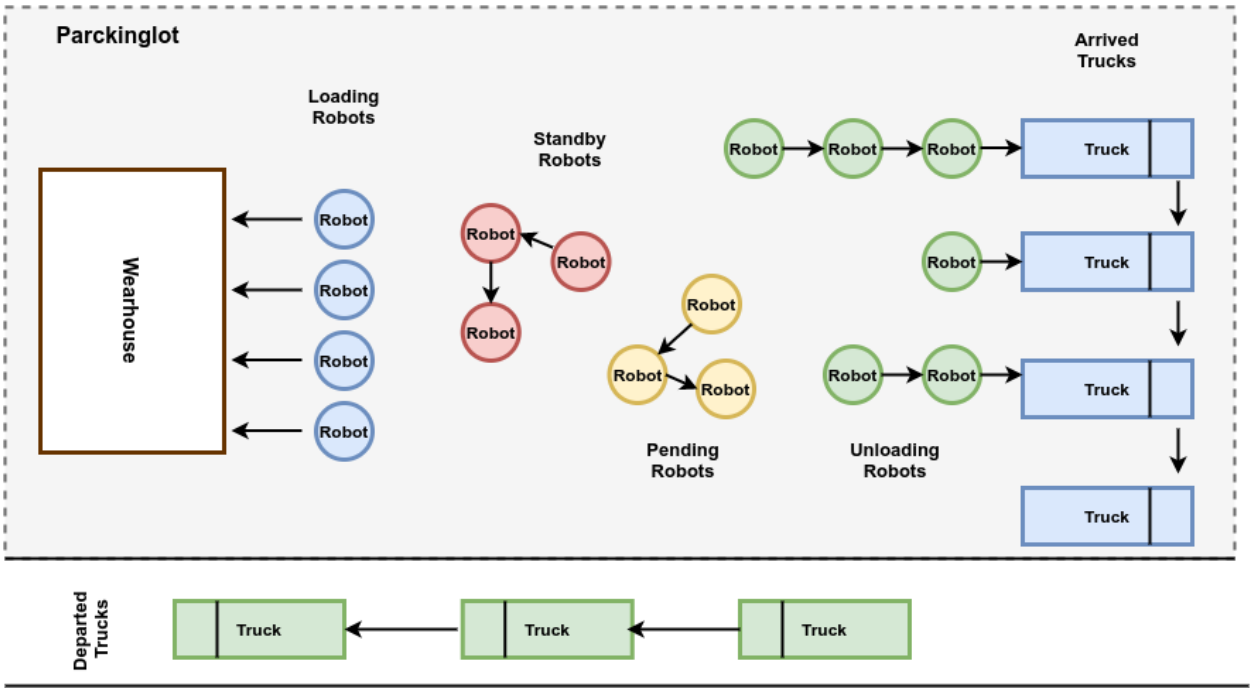

Figura 1: Vedere generica a sistemului in funcțiune.

3. Cerințe

3.1. Packages and Manifests

Pachetele (en. packages) sunt descriese de urmatoarea structură:

```
typedef struct Package{
  long priority;
  char* destination;
}Package;
```

unde `priority` este prioritatea pachetului, iar `destination` este destinatia acestuia.

Pentru a facilita lucrul cu pachete diversele invetare (en. manifest) sunt reprezentate de o listă dublu înlănlțuită cu urmatoarea definitie pentru un nod:

```
typedef struct Manifest{
  Package* package;
  struct Manifest* next;
  struct Manifest* prev;
}Manifest;
```

unde `package` reprezinta pachetul dintr-un nod asociat inventarului.
Avand aceste definiții la dispozitie implementati urmatoarele funcții în fișierul `WearhouseManager.c`:

- `create_package` – creeaza un pachet cu prioritate și destinația date. Destinația se va duplica în cazul în care este diferita de `NULL`, altfel va fi nula.
- `destroy_package` – distruge un pachet impreuna cu destinatia acestuia.
- `create_manifest_node` – creeaza un nod de itinerariu initializand campurile la `NULL`.
- `destroy_manifest_node` – distruge un nod primit ca parametru si pachetul asociat acestuia.

3.2. Wearhouse, Robot and Loading

Depozitul este reprezentat de o mulțime neordonată de pachete și are următoarea definiție:

```
typedef struct Wearhouse{
  Package ** packages;
  long size;
  long capacity;
}Wearhouse;
```

unde `packages` este un array de pachete allocate dinamic, `size` reprezintă numărul de pachete stocate în depozit și `capacity` reprezintă capacitatea totală a depozitului. În timp ce, roboții sunt reprezentați de următoare structură:

```
typedef struct Robot{
  Manifest *manifest;
  long size;
  long capacity;
  struct Robot *next;
}Robot;
```

unde `manifest` reprezită pointerul către începutul listei de pachete prelevate de robot, `size` reprezintă numărul de pachete stocate de robot, `capacity` reprezintă capacitatea totală de stocare a robotului și campul `next` indică către umatorul robot dintr-o listă.
Avand aceste definiții la dispozitie implementați următoarele funcții în fișierul `WearhouseManager.c`:

- `create_wearhouse` – creează depozitul inițializand corespunzător campurile aferente structurii.
- `wearhouse_is_empty` – Întoarce `1` dacă depozitul este gol și `0` altfel.
- `wearhouse_is_full` – Întoarce `1` dacă depozitul este plin și `0` altfel.
- `wearhouse_max_package_priority` – determină prioritatea cea mai mare a pachetelor stocate în depozit.
- `wearhouse_min_package_priority` – determină prioritatea cea mai mică a pachetelor stocate în depozit.

- `destroy_wearhouse` – distruge depozitul, inclusiv pachetele stocate în acesta.
- `create_robot` – crează un robot, inițializand corespunzător campurile structurii.
- `robot_is_full` – Întoarce `1` dacă robotul este încarcat complet și `0` altfel.
- `robot_is_empty` – Întoarce `1` dacă robotul este gol și `0` altfel.
- `robot_get_wearhouse_priority_package` – Întoarce primul pachet cu prioritatea primită ca parametru din depozit.
- `robot_remove_wearhouse_package` – Elimina un anumit pachet din depozit.
- `robot_load_one_package` – Încarcă în robot un pachet, unde pachetul va fi stocat în itinerariul acestuia în ordine sortată după prioritate și destinație. În cazul în care două sau mai multe pachete au aceeași prioritea, acestea se vor sorta aditionala după ordinea alfabetică a destinațiilor.
- `robot_load_packages` – Încarcă numarul maxim de pachete în robot din depozit.

  Hint: va puteti folosi de funcțiile implementate anterior: `wearhouse_max_package_priority`, `wearhouse_min_package_priority`, `robot_get_wearhouse_priority_package`, `robot_load_one_package` și `robot_remove_wearhouse_package`.
- `robot_get_destination_highest_priority_package` – Întoarce pachetul cu prioritatea cea mai mare aferent unei destinații particulare din itinerariul robotului.
- `destroy_robot` – distruge un robot, inclusiv itinerariul de pachete asociat acestuia.

3.3. Unloading Robots and Trucks

În descrierea unui tir, pe langa date aferente capcitații sau a destinației, au fost înglobate mai multe aspecte, precum: planificarea orarului de plecări și transit, itinerariul pachetelor încarcate și lista de roboți care asteaptă încarcarea unor pachete. Astfel, din punct de vedere aplicației un tir are umatoarea descriere:

```
typedef struct Truck{
  Manifest *manifest;
  Robot *unloading_robots;
  char* destination;
  long size;
  long capacity;
  long in_transit_time;
  long transit_end_time;
  long departure_time;
  struct Truck* next;
}Truck;
```

unde `manifest` reprezintă un pointer catre inceputul itinerariului de pachete încarcate, `unloading_robots` reprezintă un pointer către începutul listei de roboți care asteaptă să încarce pachete, `destination` este detinația tirului, `size` și `capacity` reprezintă numarul de pachete încărcate, respectiv capacitatea totală a tirului, `in_transit_time` reprezintă timpul de tranzit scurs, `transit_end_time` arată timpul de tranzit total, `departure_time` indică ora plecării unui tir și campul `next` este un link către urmatorul tir dintr-o listă.

Avand acesta definiție la dispozitie implementați următoarele funcții în fișierul `WearhouseManager.c`:

- `create_truck` – Întoarce un tir nou inițializand campurile aferente parametrilor de intrare. Fucția duplică string-ul de destniatie sau îl seteaza la NULL în functie de parametrul de intrare. Alte campuri vor fi inițializate la NULL sau `0` după caz.
- `truck_is_full` – Întorace `1` sau `0` dacă tirul este plin.
- `truck_is_empty` – Întorace `1` sau `0` dacă tirul este gol.
- `truck_destination_robots_unloading_size` – calculează numărul total de pachete ce urmează a fi descarcate pentru o anumită destinație de către roboții din lista de `unloading_robots`.
- `destroy_truck` dealocă un tir, inclusiv listele asociate itinerariului de pachete și roboților care asteaptă descărcarea.
- `robot_unload_packages` transferă toate pachetele aferente destinatiei tirului din itinerariul unui robot în itinerariul tirului în cauză.
-

3.4. Parkinglot, Robots and Trucks

Pentru a facilita interactiunea dintre roboți și tiruri, avem urmatoarea descriere pentru parcarea unde rezidă amble entități:

```
1  typedef struct Parkinglot{
2    Truck* arrived_trucks;
3    Truck* departed_trucks;
4    Robot* pending_robots;
5    Robot* standby_robots;
6  }Parkinglot;
```

unde `arrived_trucks` reprezintă santinela de început a unei liste circulare ce conține tirurile sosite (prezente) în parcarea depozitului, `departed_trucks` reprezintă santinela de început a unei liste circulare ce conține tirurile care se află în tranzit, `pending_robots` reprezintă o santinela de început a unei liste circulare ce conține roboții care nu au apucat să descarce pachetele păna în momentul plecarii tirului și `standby_robots` reprezintă santinela de început a unei liste circulare ce conține roboții fară nici o sarcină (încarcare/descărcare) asocitată.

- `create_parkinglot` – creează parcarea aferentă depozitului, unde toate campurile sunt inițializate respectand descrierea de mai sus.
- `parkinglot_add_robot` – adaugă un robot din roiul robotic în parcare în funcție de starea de încarcare a robotului în cauză. Dacă robotul este gol, acesta va fi introdus în ordine sortată a capacităților în lista de `standby_robots`. Altfel, va fi introdus în ordine sortată a numărului de pachete în lista de `pending_robots`.
- `parkinglot_remove_robot` – elimină un robot din lista corespunzătoare starii de încarcare a robotului (`standby_robots` sau `pending_robots`).
- `parckinglot_are_robots_peding` – verifică dacă există roboți în lista de `pending_robots`. Funcția întoarce `1` sau `0`.
- `parkinglot_are_arrived_trucks_empty` – verifică dacă toate tirurile sosite (prezente) în parcare sunt goale. Funcția întoarce `1` sau `0`.
- `parkinglot_are_trucks_in_transit` – verifcă dacă sunt tiruri care se află în tranzit.
- `destroy_parkinglot` – distruge parcarea, inclusiv listele asociate.

3.5. Trucks Schedules and Robot Transfers

Avand structurile descrise la punctele precedente, implementati următoarele functionalități:

- `truck_departed` – transferă tirul furnizat ca parametru în lista `departed_trucks`. Tirul va fi introdus în lista mentionată în ordinea sortată a timpului de plecare `departure_time`. Totodată, acesta va fi eliminat din lista de `arrived_trucks` dacă există.
- `truck_arrived` – transferă tirul furnizat ca parametru în lista `arrived_trucks`. Tirul va fi introdus în lista mentionată în ordinea sortată alfabetic a destinației (`destination`) și în ordine crescătoare a timpilor de plecare `departure_time`.
- `truck_update_transit_times` – actualizează timpi de tranzit (`in_transit_time`) al tirurilor plecate. La expirarea țimpului de tranzit, un tir trebuie trecut în lista `arrived_trucks` conform descrierii de la punctul anterior.
- `truck_transfer_unloading_robots` – transferă roboții, aferenți listei de descărcare asociate tirului primit ca parametru, în lista de `pending_robots` sau de `pending_robots` în funcție de stare de încarcare a acestora.
- `truck_update_depatures` – actualizează lista de plecări aferentă orei primită ca parametru.

4. Compilare, Testare si Notare

4.1. Compilare

Pentru compilarea tuturor aplicațiilor folosiți comanda "`make`". Aceasta are urmatorul output pentru un program fără erori de sintaxă sau warning-uri:

```
$ make
gcc -O0 -std=c9x -g -Wall  -o WearhouseManager WearhouseManager.c app.c WearhouseManager.h  -D_POSIX_C_SOURCE=200809L
gcc -O0 -std=c9x -g -Wall  -o Test WearhouseManager.c test.c WearhouseManager.h  -D_POSIX_C_SOURCE=200809L
```

4.2. Testare

Testarea se va realiza utilizand checker-ul pus la dispoziție folosind comanda `$ ./check.sh`. O evaluare corecta va avea urmatorul output:

```
$ ./check.sh
************* Building targets *************
gcc -O0 -std=c9x -g -Wall  -o WearhouseManager WearhouseManager.c app.c WearhouseManager.h  -D_POSIX_C_SOURCE=200809L
gcc -O0 -std=c9x -g -Wall  -o Test WearhouseManager.c test.c WearhouseManager.h  -D_POSIX_C_SOURCE=200809L
************* Testing *************
Test  0 ..................................................... PASS
Score: 2/100

Test  1 ..................................................... PASS
Score: 4/100

Test  2 ..................................................... PASS
Score: 6/100

Test  3 ..................................................... PASS
Score: 8/100

Test  4 ..................................................... PASS
Score: 10/100

Test  5 ..................................................... PASS
Score: 12/100

Test  6 ..................................................... PASS
Score: 17/100

Test  7 ..................................................... PASS
Score: 19/100

Test  8 ..................................................... PASS
Score: 21/100

Test  9 ..................................................... PASS
Score: 23/100

Test 10 ..................................................... PASS
Score: 25/100

Test 11 ..................................................... PASS
Score: 30/100

Test 12 ..................................................... PASS
Score: 35/100

Test 13 ..................................................... PASS
Score: 40/100
```

```
Test 14 ..................................................... PASS
Score: 42/100

Test 15 ..................................................... PASS
Score: 44/100

Test 16 ..................................................... PASS
Score: 47/100

Test 17 ..................................................... PASS
Score: 49/100

Test 18 ..................................................... PASS
Score: 52/100

Test 19 ..................................................... PASS
Score: 54/100

Test 20 ..................................................... PASS
Score: 58/100

Test 21 ..................................................... PASS
Score: 61/100

Test 22 ..................................................... PASS
Score: 65/100

Test 23 ..................................................... PASS
Score: 69/100

Test 24 ..................................................... PASS
Score: 71/100

Test 25 ..................................................... PASS
Score: 73/100

Test 26 ..................................................... PASS
Score: 75/100

Test 27 ..................................................... PASS
Score: 77/100

Test 28 ..................................................... PASS
Score: 82/100

Test 29 ..................................................... PASS
Score: 87/100

Test 30 ..................................................... PASS
Score: 90/100

Test 31 ..................................................... PASS
Score: 95/100

Test 32 ..................................................... PASS
Score: 100/100

Congrats! All tests passed.
```

4.3. Notare

Evaluarea temei este data de rezultatul obținut la testarea pe vmchecker cu urmatoarele constrangeri:

- Punctajul obtinut este dat de checker.
- Memory leak-urile se vor depuncta cu 10p din 100p.
- Temele copiate vor fi penalizate cu -100p si la sursă si la destinație.