Technologies for Big Data Analytics

Εργασία 1 - Threads and Processes - Υποερώτημα 2

Antonis Prodromou

2025-11-21

Πίνακας περιεχομένων

1 Περιγραφή λύσης - Υποερώτημα 2

Για την επίλυση του προβλήματος, ακολουθούμε την αρχή της Μοναδικής Αρμοδιότητας, δηλαδή όλες οι κλάσεις και οι λογισμικές μονάδες (modules) να έχουν μόνο μια, ξεκάθαρα ορισμένη αρομοδιότητα. Επομένως δημιουργούμε τις παρακάτω 6 κλάσεις, σε δύο επίπεδα (ανώτερου επιπέδου και ένθετες):

1.1 Κλάση ανώτερου επιπέδου

ProducerConsumerMain

Αυτή είναι η κύρια κλάση από την οποία παίρνει το όνομά του το αρχείο (ProducerConsumerMain.java). Είναι public, δηλαδή μπορεί να χρησιμοποιηθεί από άλλα αντικείμενα και είναι ορατή από παντού. Περιέχει τη μέθοδο `main()` - το σημείο εισόδου του προγράμματός μας.

1.2 Εσωτερικές κλάσεις

- **Disease** - τοπική εσωτερική κλάση (ορίζεται εντός της μεθόδου `main()`). Υλοποιεί το `Runnable`, αναπαριστώντας το νήμα παραγωγού (*producer*).
- **Hospital** - τοπική εσωτερική κλάση (επίσης εντός της `main()`). Υλοποιεί το `Runnable`, αναπαριστώντας το νήμα καταναλωτή (*consumer*).
- **Registry** - στατική ένθετη κλάση (ορίζεται σε επίπεδο κλάσης). Διατηρεί αρχείο καταγραφής των δεδομένων του νοσοκομείου και της κοινόχρηστης κατάστασης (*shared state*).
- **DiseaseGeneration** - στατική ένθετη κλάση. Διαχειρίζεται τη λογική παραγωγής (δημιουργία νέων κρουσμάτων).
- **DiseaseHealing** - στατική ένθετη κλάση. Διαχειρίζεται τη λογική κατανάλωσης (θεραπεία και εξιτήρια κρουσμάτων).

Υπάρχει επίσης μια εξωτερική εξάρτηση (dependency) με όνομα `IterationLog`, ορισμένη σε ξεχωριστό αρχείο, η οποία χρησιμοποιείται για την καταγραφή των αποτελεσμάτων μας.

1.3 Modules

Οι λογισμικές μονάδες διακρίνονται στις εξής:

Ενότητα	Σκοπός
DiseaseGeneration	Ενσωματώνει όλη τη λογική παραγωγής (δημιουργία νέων κρουσμάτων).
DiseaseHealing	Ενσωματώνει όλη τη λογική θεραπείας (εξιτήρια ασθενών).
Registry	Κεντρική ενότητα διαχείρισης κατάστασης. Κοινή μεταξύ των νημάτων.
IterationLog (εξωτερική)	Ενότητα καταγραφής δεδομένων / *logging*.

Αυτές είναι επαναχρησιμοποιήσιμες, λογικά απομονωμένες και θα μπορούσαν να τοποθετηθούν σε δικά τους αρχεία χωρίς να χαθεί η λειτουργικότητα.

1.4 Παράμετροι

Οι παράμετροι του προγράμματος ορίζονται - σύμφωνα και με την εκφώνηση της άσκησης - με static final, γιατί είναι παράμετροι του μοντέλου και όχι καταγραφής, άρα δεν ενημερώνονται κατά την εκτέλεσή του.

1.5 Multi-threading

Εφόσον έχω δύο threads που εκτελούν τις διεργασίες μου, ορίζω τις συναρτήσεις παραγωγής και κατανάλωσης με τον τύπο synchronized. Αυτή η μέθοδος διασφαλίζει πως μόνο ένα thread μπορεί να εκτελεί ανά οποιαδήποτε στιγμή αλλαγές πάνω σε ένα στιγμιότυπο (instance) ενός αντικειμένου, αποφεύγοντας έτσι ένα race condition. Εσωτερικά, η Java το επιτυγχάνει αυτό χρησιμοποιώντας ένα monitor lock, κλειδώνοντας κάθε φορά το registry:

```
        public synchronized void produce() throws InterruptedException {
        ...
            synchronized (registry) {
            ...
                }
```

1.6 Καταγραφές

Για την καταγραφή της λειτουργίας του προγράμματος και των αποτελεσμάτων του προγράμματος (monitoring), χρησιμοποιούμε μια σειρά από μεταβλητές, οι οποίες:

- Ορίζονται στην κλάση Registry, που αποτελεί κατά μια έννοια τον πυρήνα αποθήκευσής τους.
- Καταγράφουν τα αποτελέσματα ανά επανάληψη (iteration) μέσα στις μεθόδους `DiseaseGeneration.produce()` και `DiseaseHealing.heal()`.
- Αποθηκεύονται στην κλάση IterationLog (εξωτερική κλάση).

Οι μεταβλητές αυτές δίνονται στον παρακάτω πίνακα:

Μεταβλητή	Σκοπός	Ενημερώνεται από
incomingPatients	Αριθμός νέων ασθενών που δημιουργήθηκαν σε αυτήν την επανάληψη.	Producer (`produce()`)
patientsBeingTreated	Αριθμός ασθενών που νοσηλεύονται αυτήν τη στιγμή.	Producer και Consumer
patientsRejected	Ασθενείς που απορρίφθηκαν επειδή το νοσοκομείο ήταν πλήρες.	Producer
patientsAdmitted	Ασθενείς που έγιναν δεκτοί και εισήχθησαν για θεραπεία.	Producer
totalSickPatients	Συνολικό άθροισμα όλων των περιστατικών που έχουν δημιουργηθεί ποτέ.	Producer
healedPatients	Ασθενείς που θεραπεύτηκαν στον τρέχοντα κύκλο θεραπείας.	Consumer
totalPatientsHealed	Συνολικός αριθμός ασθενών που έχουν θεραπευτεί μέχρι στιγμής.	Consumer
totalPatientsRejected	Συνολικός αριθμός ασθενών που έχουν απορριφθεί μέχρι στιγμής.	Producer
generationIterations	Πόσες φορές έχει εκτελεστεί ο βρόχος του παραγωγού (producer).	Producer
healingIterations	Πόσες φορές έχει εκτελεστεί ο βρόχος του καταναλωτή (consumer).	Consumer
totalsLog	Μια λίστα που αποθηκεύει λεπτομερή αρχεία καταγραφής για κάθε επανάληψη (δημιουργία και θεραπεία).	Producer και Consumer

1.7 Τερματισμός λειτουργίας

Στη λειτουργία των thread επιλέχθηκε όταν ολοκληρώνει τις επαναλήψεις του το ένα thread, η μεταβλητή raceFinished να αλλάζει από False σε True, ειδοποιώντας το δεύτερο thread να σταματήσει την λειτουργία του. Χρησιμοποιήσαμε για τον σκοπό αυτό μεταβλητή ατομικού τύπου `AtomicBoolean`, η οποία διασφαλίζει πως η μεταβλητή raceFinished μπορεί να ενημερωθεί μόνο από ένα thread ανά φορά, δηλαδή με τρόπο ατομικό (atomic), που αποτρέπει ένα race condition.

1.8 Μετατροπή για λύση με τρία νοσοκομεία

Για να προγραμματίσουμε μια λύση με τρία νοσοκομεία, καταρχήν θα δημιουργούσαμε τρία threads, ένα για το καθένα. Τα threads αυτά θα είχαν μια κεντρική διαχείριση μέσω του `Registry`, άρα θα παρέμενε ίδιο το `synchronized (registry)`, για να μετράμε ασθενείς σε αναμονή, θεραπευμένους κλπ.

Αυτό που θα άλλαζε όμως είναι πως το `notify()` θα μετατρέπονταν `notifyAll()` και στη συνέχεια τα νήματα θα ανταγωνίζονταν για το ποιό θα προλάβει το monitor lock. Ο λόγος είναι πως αν διατηρούσαμε το notify()αυτό δεν θα ήξερε αν πρέπει να πάει σε κάποιο από τα νοσοκομεία (και σε ποιό) ή στον παραγωγό. Εμείς θέλουμε να ξυπνήσουν όλα τα υπόλοιπα νήματα.

Για να σηματοδοτήσουμε το τέλος των interations (και να μην αφήσουμε π.χ. τον παραγωγό σε αναμονή επ' αόριστο) θα χρησιμοποιούσαμε έναν AtomicInteger, που όταν έφτανε τον αριθμό 3 (για τα 3 νοσοκομεία) θα άλλαζε έναν AtomicBoolean από False σε True, για να σταματήσει και ο παραγωγός. Η αντίστροφη πορεία θα γινόταν εάν τερμάτιζε πρώτος ο παραγωγός.

---

2 Αποτελέσματα

Από τα αποτελέσματα που δίνονται στον παρακάτω πίνακα παρατηρούμε πως:

- Οι επαναλήψεις του thread generation αυξάνονται ανά δευτερόλεπτο, εκτός από τις περιπτώσεις όπου έχουμε φτάσει το system capacity, οπότε το thread μπαίνει σε sleep mode (`wait()`), έως ότου το ξυπνήσει μέσω του `notify()` το healing thread, π.χ. στις επαναλήψεις του Generation με αριθμό 9, 11, και 15. Για αυτόν τον λόγο, παρότι η

παραγωγή και η κατανάλωση έχουν περίοδους 1 και 5 δευτερόλεπτα αντίστοιχα, οι τελικές επαναλήψεις είναι 30 και 16 (στήλες 1 και 2) και όχι με αναλογία 5:1.

- Καταγράφονται σωστά οι συνολικοί ασθενείς (106), 49 εκ των οποίων θεραπεύονται, 37 δεν βρίσκουν θέση και 20 νοσηλεύονται όταν σταματάνε οι επαναλήψεις.

2.1 Παρουσίαση αποτελεσμάτων

Για την παρουσίαση των αποτελεσμάτων, εξάγω τη λίστα IterationLog σε csv. Στη συνέχεια το εισάγω σε Quarto notebook και οπτικοποιώ τα αποτελέσματα χρησιμοποιώντας τις βιβλιοθήκες pandas και matplotlib της python.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('producer_consumer_results_30_1000.csv')
df
# df.style.hide(axis="index")
```

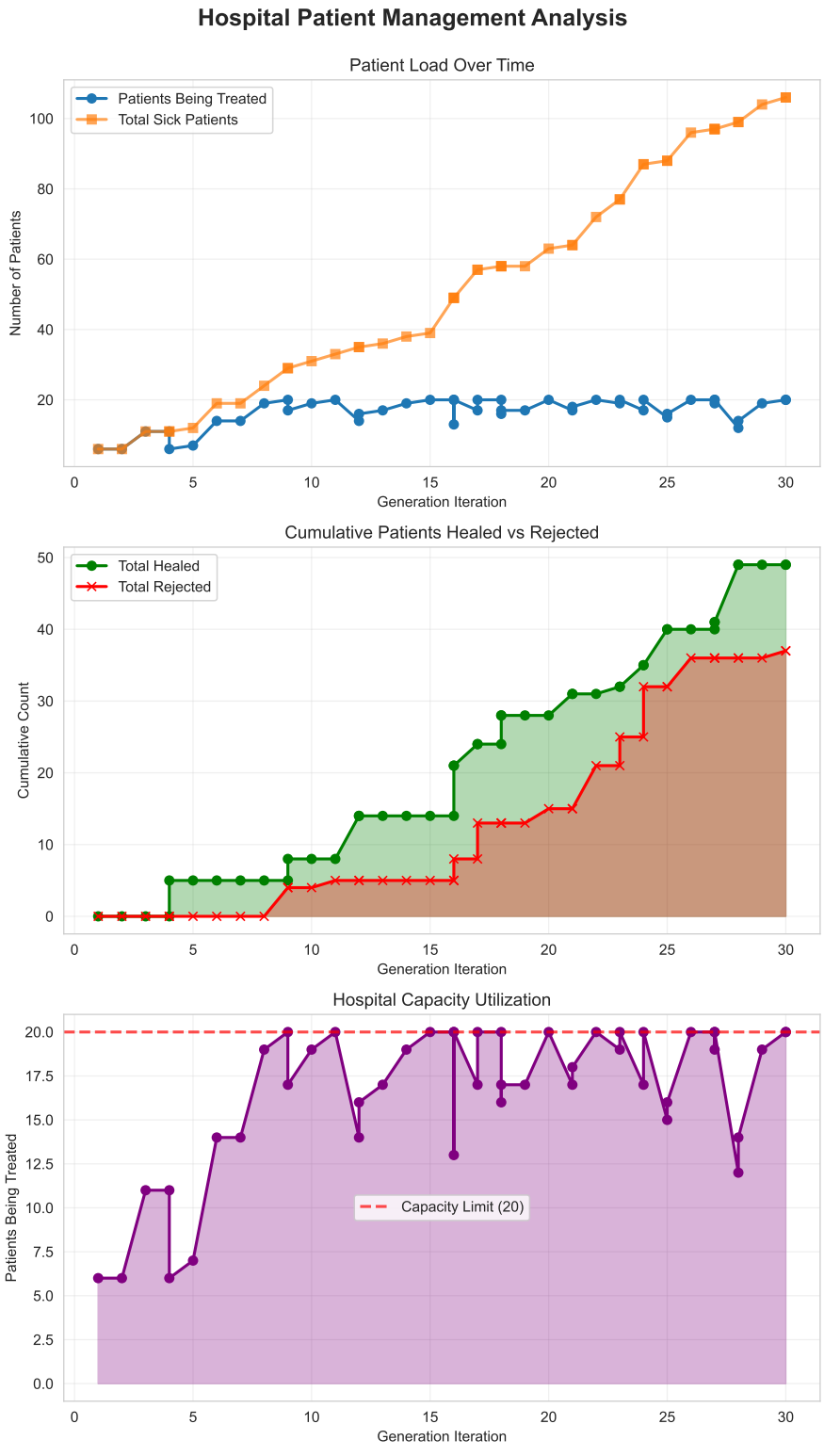
	GenerationIter	HealingIter	IncomingPatients	PatientsBeingTreated	PatientsRejected	PatientsAd
0	1	0	6	6	0	6
1	2	0	0	6	0	0
2	3	0	5	11	0	5
3	4	0	0	11	0	0
4	4	1	0	6	0	0
5	5	1	1	7	0	1
6	6	1	7	14	0	7
7	7	1	0	14	0	0
8	8	1	5	19	0	5
9	9	1	5	20	4	1
10	9	2	5	17	4	1
11	10	2	2	19	4	2
12	11	2	2	20	1	1
13	12	3	2	14	1	1
14	12	3	2	16	1	2
15	13	3	1	17	1	1
16	14	3	2	19	1	2

	GenerationIter	HealingIter	IncomingPatients	PatientsBeingTreated	PatientsRejected	PatientsAd
17	15	3	1	20	1	1
18	16	4	10	20	1	1
19	16	5	10	13	1	1
20	16	5	10	20	3	7
21	17	6	8	17	3	7
22	17	6	8	20	5	3
23	18	7	1	20	5	3
24	18	8	1	16	5	3
25	18	8	1	17	5	1
26	19	8	0	17	5	0
27	20	8	5	20	2	3
28	21	9	1	17	2	3
29	21	9	1	18	2	1
30	22	9	8	20	6	2
31	23	10	5	19	6	2
32	23	10	5	20	4	1
33	24	11	10	17	4	1
34	24	11	10	20	7	3
35	25	12	1	15	7	3
36	25	12	1	16	7	1
37	26	12	8	20	4	4
38	27	13	1	20	4	4
39	27	14	1	19	4	4
40	27	14	1	20	4	1
41	28	15	2	12	4	1
42	28	15	2	14	4	2
43	29	15	5	19	4	5
44	30	15	2	20	1	1
45	30	16	2	20	1	1

```
# Set style for better-looking plots
sns.set_style("whitegrid")

# Load the data
df = pd.read_csv('producer_consumer_results_30_1000.csv')

# Create a figure with multiple subplots
fig, axes = plt.subplots(3, 1, figsize=(8, 14))
fig.suptitle('Hospital Patient Management Analysis\n', fontsize=16, fontweight='bol
```

```
# 1. Patients Over Time
ax1 = axes[0]
ax1.plot(df['GenerationIter'], df['PatientsBeingTreated'],
         marker='o', label='Patients Being Treated', linewidth=2)
ax1.plot(df['GenerationIter'], df['TotalSickPatients'],
         marker='s', label='Total Sick Patients', linewidth=2, alpha=0.7)
ax1.set_xlabel('Generation Iteration')
ax1.set_ylabel('Number of Patients')
ax1.set_title('Patient Load Over Time')
ax1.legend()
ax1.grid(True, alpha=0.3)

# 3. Cumulative Healing Progress
ax3 = axes[1]
ax3.plot(df['GenerationIter'], df['TotalPatientsHealed'],
         marker='o', color='green', linewidth=2, label='Total Healed')
ax3.plot(df['GenerationIter'], df['TotalPatientsRejected'],
         marker='x', color='red', linewidth=2, label='Total Rejected')
ax3.fill_between(df['GenerationIter'], df['TotalPatientsHealed'],
                 alpha=0.3, color='green')
ax3.fill_between(df['GenerationIter'], df['TotalPatientsRejected'],
                 alpha=0.3, color='red')
ax3.set_xlabel('Generation Iteration')
ax3.set_ylabel('Cumulative Count')
ax3.set_title('Cumulative Patients Healed vs Rejected')
ax3.legend()
ax3.grid(True, alpha=0.3)

# 6. Hospital Capacity Utilization
ax6 = axes[2]
ax6.plot(df['GenerationIter'], df['PatientsBeingTreated'],
         marker='o', linewidth=2, color='purple')
ax6.axhline(y=20, color='red', linestyle='--', linewidth=2,
           label='Capacity Limit (20)', alpha=0.7)
ax6.fill_between(df['GenerationIter'], 0, df['PatientsBeingTreated'],
                 alpha=0.3, color='purple')
ax6.set_xlabel('Generation Iteration')
ax6.set_ylabel('Patients Being Treated')
ax6.set_title('Hospital Capacity Utilization')
ax6.legend()
ax6.grid(True, alpha=0.3)
```

```
plt.tight_layout()
plt.show()
```


Hospital Patient Management Analysis
Patient Load Over Time
Patients Being Treated
Total Sick Patients
Number of Patients
Generation Iteration
Cumulative Patients Healed vs Rejected
Total Healed
Total Rejected
Cumulative Count
Generation Iteration
Hospital Capacity Utilization
Capacity Limit (20)
Patients Being Treated
Generation Iteration