**Δεληγιάννη Μυρτώ 1067389 – Τμήμα Μηχανικών Η/Υ και Πληροφορικής**

**Φεβρουάριος 2024**

**Κατανεμημένα Συστήματα – Project**

**2.2.1 Μοντέλο Αποστολής – Παράδοσης Μηνυμάτων μέσω Κοινής Ουράς**

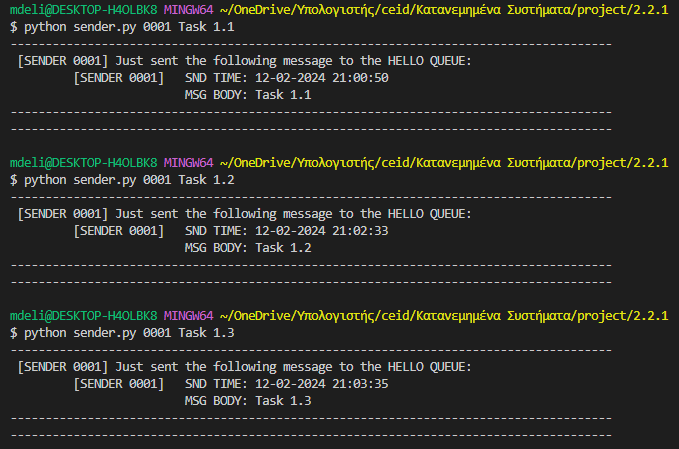
Αξιοποιώντας τον σύνδεσμο <https://www.rabbitmq.com/tutorials/tutorial-two-python.html>, προσθέτοντας τις απαραίτητες παραμέτρους (αναγνωριστικά αποστολέα, παραλήπτη και μηνύματος/διεργασίας, χρόνοι ενεργοποίησης, αποστολής και παραλαβής των μηνυμάτων/διεργασιών), υλοποιήθηκε μια απλή υπηρεσία στην οποία, διάφοροι παραλήπτες λαμβάνουν ασύγχρονα μηνύματα από αυθαίρετους αποστολείς και από τον message broker ακολουθείται πολιτική Round Robin.

Δυο παραδείγματα εκτέλεσης:

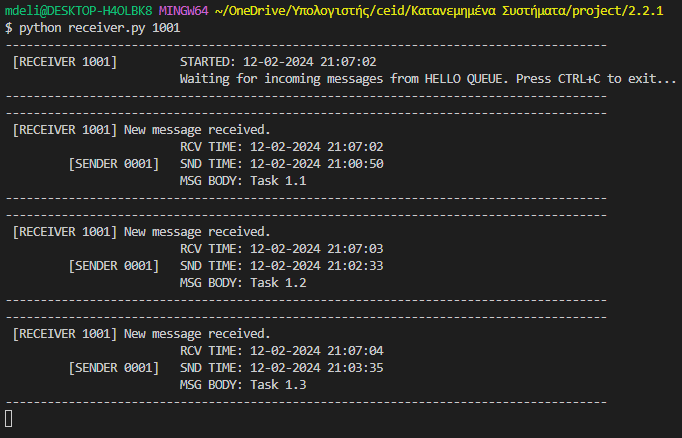
1. Ένας αποστολέας ενεργοποιείται και στέλνει μηνύματα. Αργότερα, μόλις

ενεργοποιηθεί ο παραλήπτης, ξεκινάει να εξυπηρετεί με τη σειρά τα μηνύματα.

SENDER:

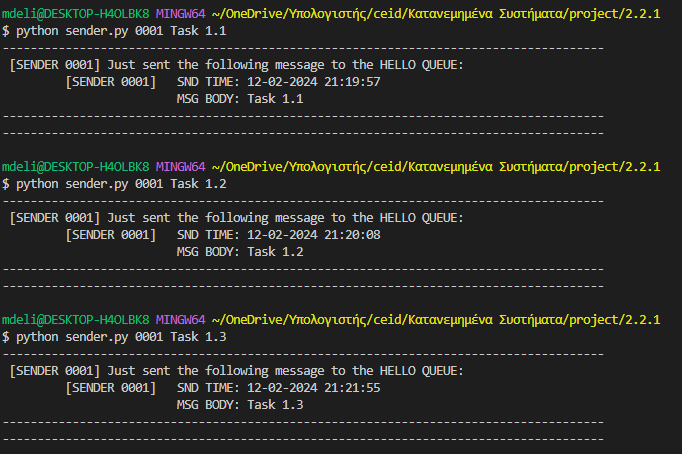


RECEIVER:

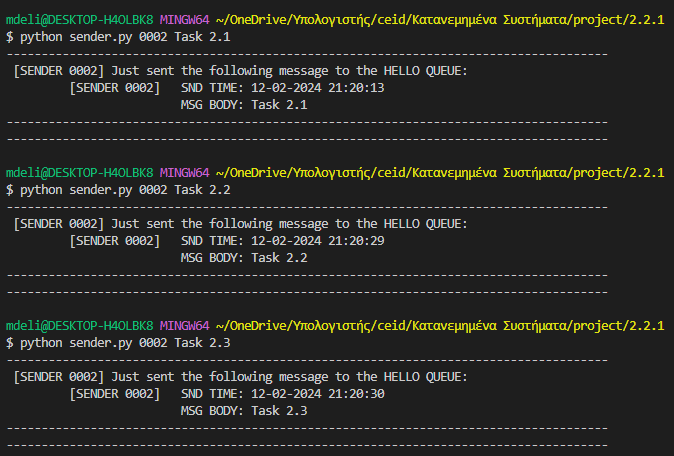


1. Πρώτα ενεργοποιούνται 3 παραλήπτες και περιμένουν, ύστερα ενεργοποιούνται 2 αποστολείς και στέλνουν από 3 μηνύματα ο καθένας

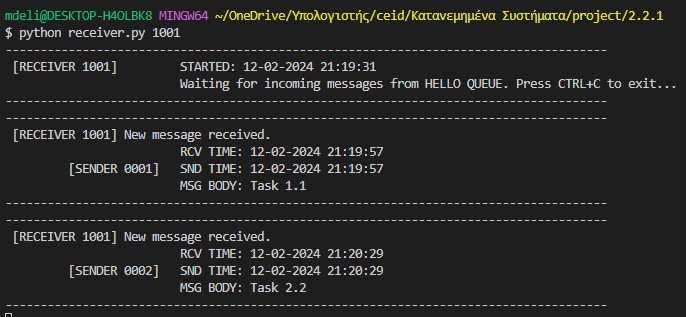
SENDER 1:



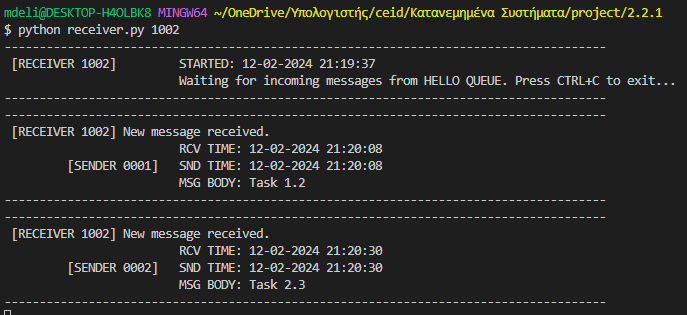
SENDER 2:



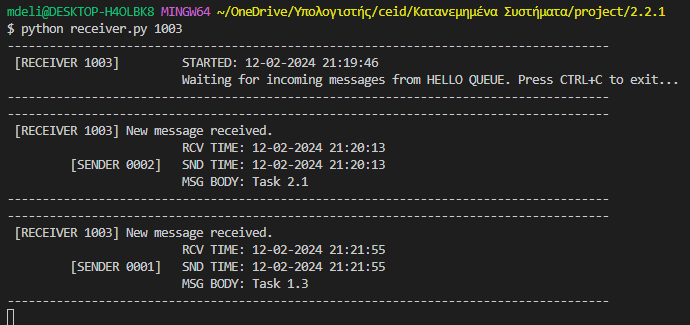
RECEIVER 1:



RECEIVER 2:



RECEIVER 3:

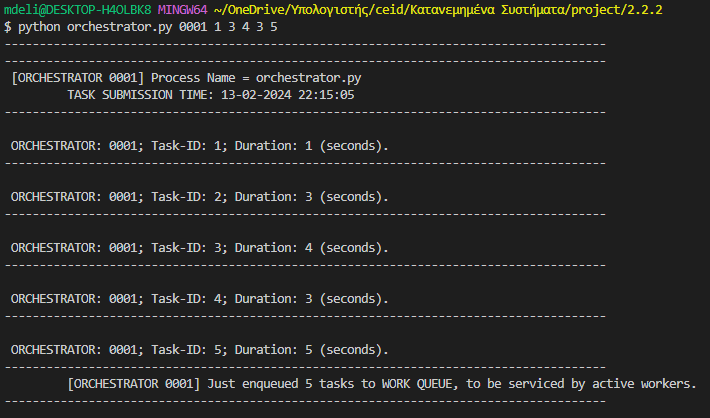


* + 1. **Στοιχειώδες Μοντέλο Ενορχήστρωσης Ανάθεσης Καθηκόντων**

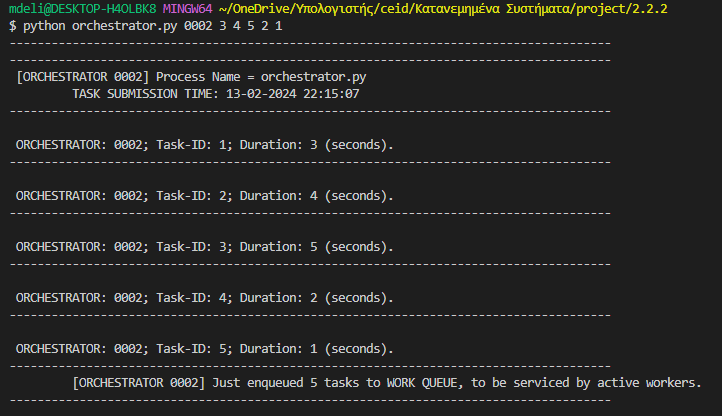
Τροποποιώντας τον κώδικα του προηγούμενου ερωτήματος ώστε ο ενορχηστρωτής (πρώην-αποστολέας) να αποστέλλει στους εργάτες μια ουρά διεργασιών αντί για μια μοναδική εργασία τη φορά, επιτυγχάνεται το Στοιχειώδες Μοντέλο Ενορχήστρωσης Ανάθεσης Καθηκόντων. Κάθε ξεχωριστή διεργασία στην ουρά, έχει τον δικό της εκτιμώμενο χρόνο εκτέλεσης, τον οποίο λαμβάνει υπόψιν ο εκάστοτε εργάτης που θα την εκτελέσει. Αν κάποιος εργάτης ξαφνικά σταματήσει να είναι ενεργός, η διεργασία δεν χάνεται. Θα μεταφερθεί στην ουρά «νεκρών» μηνυμάτων (dead – letter queue) και αργότερα θα επιστρέψει πίσω στην αρχική ουρά για να αποσταλλεί ξανά σε κάποιον από τους εναπομείναντες ενεργούς εργάτες.

Τα παραπάνω φαίνονται στο επόμενο παράδειγμα. Αρχικά δημιουργούμε 3 εργάτες που περιμένουν να εξυπηρετήσουν. Έπειτα δημιουργούμε 2 ενορχηστρωτές, οι οποίοι αποστέλλουν από 5 διεργασίες διαφορετικής χρονικής διάρκειας ο καθένας. Κάποια στιγμή παρατηρούμε ότι η λειτουργία του τελευταίου εργάτη (με ID: 9003) διακόπτεται κατά την εκτέλεση της διεργασίας με ID: 4 που εστάλη από τον ενορχηστρωτή με ID: 0001. Ωστόσο, μπορούμε να δούμε ότι η συγκεκριμένη εργασία δεν χάθηκε, αφού αργότερα εκτελέστηκε επιτυχώς από τον εργάτη με ID: 9001.

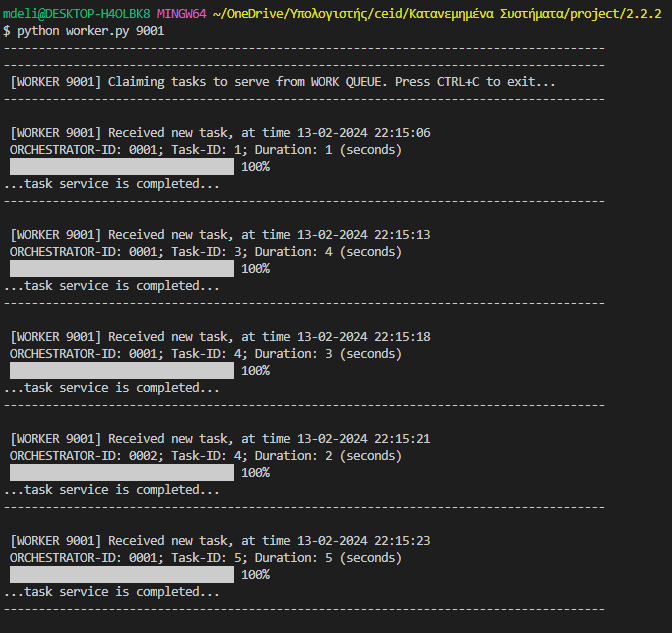
ORCHESTRATOR 0001:



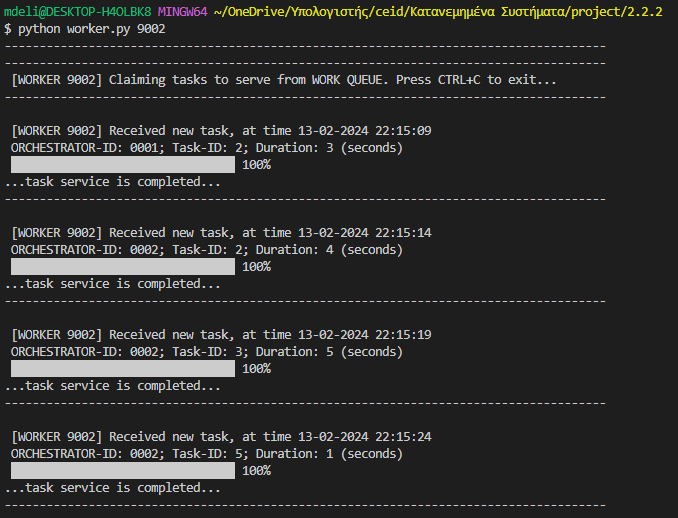
ORCHESTRATOR 0002:



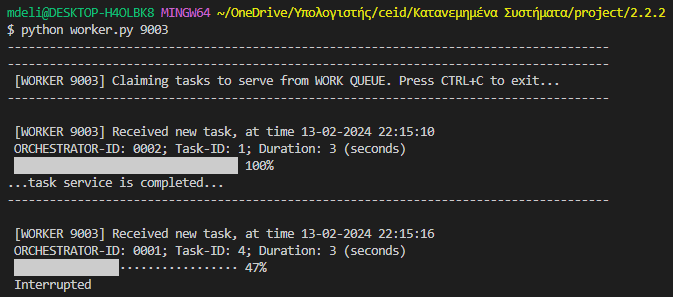
WORKER 9001:



WORKER 9002:



WORKER 9003:



**2.2.3 Μοντέλο Φιλτραρίσματος Μηνυμάτων**

Σε αυτή τη φάση, αξιοποιώντας την έννοια του ανταλλακτηρίου (exchange), οι προηγούμενοι κώδικες τροποιποιήθηκαν με βάση τα παραδείγματα από το rabbitmq.com ώστε να υιοθετηθεί το ζητούμενο μοτίβο επικοινωνίας. Κάθε publisher, είτε είναι ενορχηστρωτής, είτε εργάτης, αποστέλλει ένα μήνυμα με ένα tag, μια ετικέτα, που δηλώνει σε ποιά ομάδα ή σε ποιόν subscriber συγκεκριμένα απευθύνεται το μήνυμα αυτό. Αυτό εγγράφεται στο ανταλλακτήριο task\_stream και περιμένει να εξυπηρετηθεί. Κάθε subscriber διαθέτει μια λίστα ετικετών που δηλώνουν ποιά μηνύματα μπορεί να εξυπηρετήσει. Αν σε κάποιο μήνυμα αναγνωρίσει κάποια από τις ετικέτες της λίστας, σημαίνει ότι μπορεί να το αναλάβει, αλλιώς το απορρίπτει και το μήνυμα συνεχίζει να περιμένει στην ουρά.

Στο επόμενο παράδειγμα εκτελούνται οι εντολές της εκφώνησης για να επιβεβαιωθεί η παραπάνω λειτουργία.

ORCH1-subscriber:

A screenshot of a computer

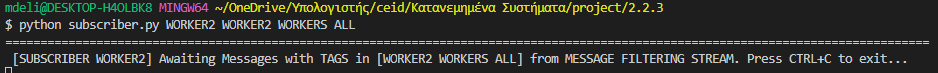
Description automatically generated

WORKER1-subscriber:

A screenshot of a computer

Description automatically generated

WORKER2-subscriber:



ORCH1-publisher:

A screenshot of a computer

Description automatically generated

WORKER1-publisher:

A screenshot of a computer

Description automatically generated

WORKER2-publisher:

A screenshot of a computer

Description automatically generated

**2.3 Υπολογισμός Ελάχιστων / Μέγιστων Ενδείξεων Θερμοκρασίας**

**2.3.1 Δημιουργία δικτύου Επικάλυψης για Επικοινωνία Διεργασιών με χρήση Ανταλλακτηρίου Μηνυμάτων**

Αρχικά, στο αρχείο process.py δημιουργήθηκε η κλάση Process, η οποία δημιουργεί αντικείμενα που διαθέτουν το δικό τους ξεχωριστό αναγνωριστικό και μια λίστα αναγνωριστικών των γειτόνων τους. Στην κλάση επίσης περιλαμβάνεται η μέθοδος get\_process\_by\_pid(pid, processes), η οποία επιστρέφει ένα αντικείμενο τύπου process με βάση το id που του δίνεται στο πεδίο pid.

Το αρχείο κώδικα publisher.py έχει περίπου την ίδια λειτουργία με εκείνο του προηγούμενου ερωτήματος. Καθορίζει τον τρόπο με τον οποίο συμπεριφέρεται μια διεργασία όταν αποστέλλει ένα μήνυμα στους γείτονές της. Αρχικά, δημιουργούνται οι διεργασίες της εκφώνησης 𝑃 = {01,04,09,11,14,18,20,21,28}, με τους αντίστοιχους γείτονές τους, όπως φαίνεται στο σχήμα, δηλαδή αναπαρίσταται το δίκτυο επικάλυψης.

A screen shot of a computer program

Description automatically generated A diagram of a network

Description automatically generated

Το πρόγραμμα λαμβάνει από τον χρήστη, μέσω του τερματικού το αναγνωριστικό της διεργασίας που θέλει να αποστείλει το μήνυμα (sender\_id), μια λίστα αναγνωριστικών των επιθυμητών παραληπτών του μηνύματος (recipient\_ids) και τέλος το ίδιο το μήνυμα. Ελέγχει αν οι επιθυμητοί παραλήπτες είναι γείτονες και αναλόγως αγνοεί το μήνυμα ή το αποστέλλει στους επιθυμητούς παραλήπτες. Καλείται ως εξής:

python publisher.py <sender\_id> <recipient\_ids> <message>

Προσοχή: τα στοιχεία της λίστας recipient\_ids κατά την κλήση της publisher.py πρέπει να χωρίζονται μόνο με κόμμα (‘,’) και χωρίς κενά.

Παράδειγμα κλήσης: python publisher.py 01 04,09,18 Hello World!

Το αρχείο κώδικα consumer.py έχει περίπου την ίδια λειτουργία με το subscriber.py του προηγούμενιυ ερωτήματος. Καθορίζει την συμπεριφορά μιας διεργασίας που λαμβάνει ένα μήνυμα από μια άλλη. Όπως και για το publisher.py, γίνεται πάλι η αρχικοποίηση των διεργασιών σύμφωνα με το σχήμα της εκφώνησης, αφού δεν γνωρίζουμε ποιο από τα δύο αρχεία θα εκτελεστεί πρώτο από τον χρήστη. Μια διεργασία – παραλήπτης, αφού πρώτα ελέγξει αν βρίσκεται στην λίστα των πιθανών παραληπτών του μηνύματος, βλέπει το μήνυμα και το εξυπηρετεί. Καλείται ως εξής:

python consumer.py <recipient\_id>

Παράδειγμα κλήσης: python consumer.py 04

Παράδειγμα χρήσης και screenshots:

Process 01 – publisher:

A screenshot of a computer program

Description automatically generated

Process 04 – publisher:

A screenshot of a computer program

Description automatically generated

Process 11 – publisher:

A screenshot of a computer program

Description automatically generated

Process 04 – consumer:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

Process 18 – consumer:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

Process 20 – consumer:

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

**2.3.2 Δημιουργία Παλμών Χρονισμού και Δειγματοληψίας Τιμών Αισθητήρων Θερμοκρασίας**

Αρχικά, στο αρχείο classes.py, κατασκευάστηκε η κλάση HeartbitAndTemperatureGenerator, με τις εξής μεθόδους:

* \_\_init\_\_(self, sampling\_interval): Καλείται με την δημιουργία κάθε αντικειμένου τύπου HeartbitAndTemperatureGenerator που παίρνει σαν όρισμα τον χρόνο που θα πρέπει να μεσολαβεί μεταξύ διαδοχικών παλμών, μια αυθαίρετη τιμή, πχ 30 sec.
* generate\_temperature\_samples(self): Με την κλήση της [round(random.uniform(0, 40), 1) for \_ in range(32)], δημιουργούνται και επιστρέφονται 32 δείγματα (όσοι είναι και οι αισθητήρες) θερμοκρασίας (τυχαίων τιμών από το 0 ως το 40).
* send\_samples\_to\_processes(self): Η μέθοδος αυτή παράγει τα δείγματα και την χρονοσφραγίδα της στιγμής που εκείνη εκτελείται, της στιγμής δηλαδή που ξεκινάει η δειγματοληψία. Επιστρέφει τον συνδυασμό αυτών των 2 μεταβλητών, ο οποίος αποτελεί και το μήνυμα που θα αποστέλλεται στις διεργασίες.

Έπειτα, ο κώδικας των διεργασιών, δηλαδή τα αρχεία consumer.py και publisher.py τροποποιούνται ώστε να στέλνεται το μήνυμα που παράγει η μέθοδος send\_samples\_to\_processes(). Αυτό επιτυγχάνεται με τη δημιουργία ενός αντικειμένου τύπου HearbitAndTemperatureGenerator (hbtg = HeartbitAndTemperatureGenerator(30)) για το οποίο θα τρέχει η μέθοδος send\_samples\_to\_processes() και θα γίνεται μέρος του συνολικού μηνύματος (message\_body = hbtg.send\_samples\_to\_processes()). Οι υπόλοιπες λειτουργίες θα παραμείνουν ίδιες.

Η κλήση του consumer.py θα γίνεται όπως προηγουμένως, δηλαδή:

python consumer.py <recipient\_id>.

Η κλήση του publisher.py αλλάζει, εφόσον το μήνυμα δεν δίνεται πλέον από τον χρήστη, αλλά από το ίδιο το πρόγραμμα, άρα θα είναι:

python publisher.py <sender\_id> <recipient\_ids>

Προσοχή: όπως και προηγουμένως, τα στοιχεία της λίστας recipient\_ids κατά την κλήση της publisher.py πρέπει να χωρίζονται μόνο με κόμμα (‘,’) και χωρίς κενά.

Παράδειγμα: Η ανταλλαγή των μηνυμάτων γίνεται με τον ίδιο τρόπο όπως παραπάνω, ώστε να φαίνονται οι διαφορές μεταξύ των ερωτημάτων.

Process 01 – publisher:

A black background with many colored lines

Description automatically generated

Process 04 – publisher:

A screen shot of a computer

Description automatically generated

Process 11 – publisher:

A screen shot of a computer

Description automatically generated

Process 04 – consumer:

A black screen with many lines

Description automatically generated

Process 18 – consumer:

A screen shot of a computer

Description automatically generated

Process 20 – consumer:

A screen shot of a computer

Description automatically generated