Τμήμα Μηχανικών Η/Υ και Πληροφορικής Πανεπιστημίου Πατρών
CEID_NE4117: *Κατανεμημένα Συστήματα*
Ακαδημαϊκό Έτος 2023-24
Διδάσκων: Σπύρος Κοντογιάννης

Προγραμματιστική Άσκηση για το Σπίτι
Μοντέλα Επικοινωνίας & Συντονισμού για Κατανεμημένα Συστήματα

Ανακοίνωση: Παρασκευή, 26 Ιανουαρίου 2024 **Παράδοση:** Παρασκευή, 01 Μαρτίου 2024
Τελευταία Ενημέρωση: Παρασκευή, 26 Ιανουαρίου 2024

1. Σύνοψη Εργασίας

Στόχος της παρούσας εργασίας είναι η εξοικείωση με μοτίβα επικοινωνίας για αλληλεπίδραση διεργασιών σε ένα κατανεμημένο σύστημα, προκειμένου να αξιοποιηθούν για την επίτευξη στοιχειωδών κατανεμημένων υπολογισμών. Βασική πλατφόρμα που θα χρησιμοποιηθεί είναι ο ανοιχτού κώδικα διαμεσολαβητής ανταλλαγής μηνυμάτων RabbitMQ (https://www.rabbitmq.com/) που συζητήθηκε και στο πλαίσιο του μαθήματος.

Αναμενόμενα αποτελέσματα της εργασίας είναι τα εξής:

- Εξοικείωση με μηχανισμούς αλληλεπίδρασης μεταξύ διεργασιών ενός κατανεμημένου συστήματος, μέσω ανταλλαγής μηνυμάτων.
- Υλοποίηση διαφορετικών μοντέλων επικοινωνίας.
- Αξιοποίηση συγκεκριμένων μοντέλων επικοινωνίας για την επίτευξη ενός κατανεμημένου υπολογισμού.

2. Βασικά Βήματα Υλοποίησης

2.1. Εξοικείωση με τον ανοιχτού κώδικα διαμεσολαβητή μηνυμάτων RabbitMQ

Ως πρώτο βήμα, θα πρέπει να γίνει εγκατάσταση του ανοιχτού-κώδικα διαμεσολαβητή μηνυμάτων **RabbitMQ**. Δείτε λεπτομέρειες εγκατάστασης στο https://www.rabbitmq.com/download.html.

Θα πρέπει να γίνει εγκατάσταση στον υπολογιστή σας, ανάλογα με το λειτουργικό σύστημα που χρησιμοποιείτε (πχ, MS Windows, Linux, MacOS, UNIX), κι όχι μέσω dockers, όχι μόνο του RabbitMQ, αλλά (πρώτα) και του **Erlang/OTP** για υποστήριξη του RabbitMQ (διαβάστε πολύ προσεκτικά τις οδηγίες εγκατάστασης, ανάλογα με το λειτουργικό σύστημα που χρησιμοποιείτε). Επίσης, θα αξιοποιηθεί η υλοποίηση του AMQP πρωτοκόλλου που παρέχεται από τη βιβλιοθήκη **Pika** της γλώσσας Python που θα χρησιμοποιήσετε, στο πλαίσιο της παρούσας εργασίας (δείτε λεπτομέρειες στον εξής σύνδεσμο: https://pika.readthedocs.io/en/stable/).

Στη συνέχεια, προτείνεται ισχυρά να πειραματιστείτε με τα 6 πρώτα πρότυπα μοντέλα επικοινωνίας μέσω του RabbitMQ διαμεσολαβητή, που είναι ήδη υλοποιημένα (και) σε Python ως παραδείγματα, στο tutorial που θα βρείτε στον ακόλουθο σύνδεσμο: https://www.rabbitmq.com/getstarted.html.

2.2. Υλοποίηση συγκεκριμένων μοτίβων επικοινωνίας

Στα ακόλουθα μοτίβα επικοινωνίας, θεωρήστε για κάθε αποστολέα ή παραλήπτη κάποιου μηνύματος ότι έχει μοναδικό προσδιοριστικό (ID), που θα το ορίζετε επιλέγοντας τυχαία μια 4-ψήφια συμβολοσειρά στο δεκαδικό σύστημα, στο διάστημα [1000,9999]. Κάθε αποστολέας μηνύματος θα περιλαμβάνει στο μήνυμα προς αποστολή, εκτός από το σώμα του μηνύματος, το δικό του προσδιοριστικό και τον χρόνο αποστολής (σύμφωνα με το τοπικό

του ρολόι). Κάθε παραλήπτης μηνύματος θα τυπώνει τον χρόνο παραλαβής του μηνύματος και (φυσικά) το πλήρες μήνυμα όπως ακριβώς υποβλήθηκε από τον αποστολέα (δλδ, μαζί με το ID του αποστολέα και τον χρόνο αποστολής, εκτός από το σώμα του μηνύματος).

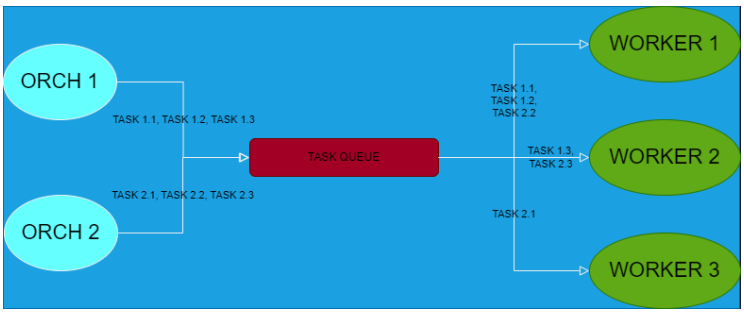
Σε κάθε περίπτωση, η υποστηριζόμενη επικοινωνία θα μπορεί να γίνεται με **ασύγχρονο** τρόπο, δίχως να υπάρχει κίνδυνος απώλειας μηνυμάτων (πχ, λόγω αποτυχίας σύνδεσης στην ουρά του καναλιού επικοινωνίας, ή αδυναμίας παραλαβής από κάποιον παραλήπτη) για τις περιπτώσεις που κάτι τέτοιο είναι εφικτό.

Εξηγήστε, για κάθε μοτίβο επικοινωνίας που υλοποιείτε, ποια θα ήταν τα δυνατά σημεία και ποια τα αδύνατα σημεία ως υποδομή επικοινωνίας για μιας διεργασίας-ενορχηστρωτή ανάθεσης καθηκόντων σε διεργασίες-εργάτες. Συγκεκριμένα, σχολιάστε στην αναφορά σας τα εξής σημεία:

- Είναι βέβαιο ότι θα ανατεθούν (κάποια στιγμή) σε εργάτες όλα τα καθήκοντα?
- Υπάρχει τρόπος επιβεβαίωσης της ανάληψης καθήκοντος από κάποιον εργάτη?
- Υπάρχει τρόπος επιβεβαίωσης της εκτέλεσης ενός καθήκοντος από κάποιον εργάτη?
- Πόσο επιβαρύνεται το δίκτυο (ως προς το πλήθος των μηνυμάτων που ανταλλάσσονται μεταξύ της ουράς και των διεργασιών) για την εξυπηρέτηση των καθηκόντων?
- Είναι εφικτή κάποιου είδους εξισορρόπηση του φόρτου εργασίας μεταξύ των ενεργών εργατών?

2.2.1. Μοντέλο Αποστολής-Παράδοσης Μηνυμάτων μέσω Κοινής Ουράς (4 μονάδες)

Αξιοποιώντας το μοτίβο επικοινωνίας «ουρών εργασίας» που παρουσιάζεται αναλυτικά στο παράδειγμα https://www.rabbitmq.com/tutorials/tutorial-two-python.html, δημιουργήστε μια απλή υπηρεσία **ασύγχρονης** επικοινωνίας, όπου διάφοροι παραλήπτες (receivers) συνδέονται και παραλαμβάνουν υποβληθέντα καθήκοντα σε μια ουρά μηνυμάτων TASKS QUEUE, τα οποία υποβάλλονται (ασύγχρονα) από αυθαίρετους αποστολείς (senders). Καθένα από τα μηνύματα που παραλαμβάνονται από την ουρά, παραδίνονται μόνο σε έναν παραλήπτη. Αν υπάρχουν περισσότεροι του ενός ενεργοί παραλήπτες (τη στιγμή υποβολής, ή με την πρώτη ευκαιρία που θα συνδεθούν νέοι παραλήπτες στην ουρά), τότε ακολουθείται από τον message broker πολιτική round-robin για την παράδοσή τους στους παραλήπτες. Κάθε υποβολέας θα καταγράφει το νέο καθήκον που δημιουργεί με τη μορφή ενός μηνύματος που αποστέλλει στην ουρά. Για παράδειγμα:



SENDER #1 SENDER #2

```
PS C:\Users\Spyros KONTOGIANNIS\Qsync\SPYROS-PERSONAL-SPACE @ NAS\my
> python .\rabbitMQ_one-one_SENDER.py 0001 TASK 1.1
------------------------------------------------------------
[SENDER 0001] Just sent the following message to the HELLO QUEUE:
        [SENDER 0001]    SND TIME: Fri Jan  5 17:34:14 2024
                         MSG BODY: TASK 1.1
============================================================

PS C:\Users\Spyros KONTOGIANNIS\Qsync\SPYROS-PERSONAL-SPACE @ NAS\my
> python .\rabbitMQ_one-one_SENDER.py 0001 TASK 1.2
------------------------------------------------------------
[SENDER 0001] Just sent the following message to the HELLO QUEUE:
        [SENDER 0001]    SND TIME: Fri Jan  5 17:34:40 2024
                         MSG BODY: TASK 1.2
============================================================

PS C:\Users\Spyros KONTOGIANNIS\Qsync\SPYROS-PERSONAL-SPACE @ NAS\my
> python .\rabbitMQ_one-one_SENDER.py 0001 TASK 1.3
------------------------------------------------------------
[SENDER 0001] Just sent the following message to the HELLO QUEUE:
        [SENDER 0001]    SND TIME: Fri Jan  5 17:34:43 2024
                         MSG BODY: TASK 1.3
============================================================
```

```
PS C:\Users\Spyros KONTOGIANNIS\Qsync\SPYROS-PERSONAL-SPACE @ NAS\my
> python .\rabbitMQ_one-one_SENDER.py 0002 TASK 2.1
------------------------------------------------------------
[SENDER 0002] Just sent the following message to the HELLO QUEUE:
        [SENDER 0002]    SND TIME: Fri Jan  5 17:35:18 2024
                         MSG BODY: TASK 2.1
============================================================

PS C:\Users\Spyros KONTOGIANNIS\Qsync\SPYROS-PERSONAL-SPACE @ NAS\my
> python .\rabbitMQ_one-one_SENDER.py 0002 TASK 2.2
------------------------------------------------------------
[SENDER 0002] Just sent the following message to the HELLO QUEUE:
        [SENDER 0002]    SND TIME: Fri Jan  5 17:35:20 2024
                         MSG BODY: TASK 2.2
============================================================

PS C:\Users\Spyros KONTOGIANNIS\Qsync\SPYROS-PERSONAL-SPACE @ NAS\my
> python .\rabbitMQ_one-one_SENDER.py 0002 TASK 2.3
------------------------------------------------------------
[SENDER 0002] Just sent the following message to the HELLO QUEUE:
        [SENDER 0002]    SND TIME: Fri Jan  5 17:35:23 2024
                         MSG BODY: TASK 2.3
============================================================
```

RECEIVER #1

```
PS C:\Users\Spyros KONTOGIANNIS\Qsync\SPYROS-PERSONAL-SPACE @ NAS\mywork\mycourses\CEID\CEID_
> python .\rabbitMQ_one-one_RECEIVER.py 1001
============================================================
[RECEIVER 1001]          STARTED: 2024-01-05, Fri Jan  5 17:34:23 2024
                         Waiting for incoming messages from HELLO QUEUE. Press CTRL+C to exit
------------------------------------------------------------
[RECEIVER 1001] New message received.
                         RCV TIME: Fri Jan  5 17:34:23 2024
        [SENDER 0001]    SND TIME: Fri Jan  5 17:34:14 2024
                         MSG BODY: TASK 1.1
------------------------------------------------------------
[RECEIVER 1001] New message received.
                         RCV TIME: Fri Jan  5 17:34:40 2024
        [SENDER 0001]    SND TIME: Fri Jan  5 17:34:40 2024
                         MSG BODY: TASK 1.2
------------------------------------------------------------
[RECEIVER 1001] New message received.
                         RCV TIME: Fri Jan  5 17:35:20 2024
        [SENDER 0002]    SND TIME: Fri Jan  5 17:35:20 2024
                         MSG BODY: TASK 2.2
------------------------------------------------------------
[RECEIVER 1001]          STOPPED BY END-USER: 2024-01-05, Fri Jan  5 17:41:03 2024
============================================================
```

RECEIVER #2

```
PS C:\Users\Spyros KONTOGIANNIS\Qsync\SPYROS-PERSONAL-SPACE @ NAS\mywork\mycourses\CEID\CEID_
> python .\rabbitMQ_one-one_RECEIVER.py 1002
============================================================
[RECEIVER 1002]          STARTED: 2024-01-05, Fri Jan  5 17:34:28 2024
                         Waiting for incoming messages from HELLO QUEUE. Press CTRL+C to exit
------------------------------------------------------------
[RECEIVER 1002] New message received.
                         RCV TIME: Fri Jan  5 17:34:43 2024
        [SENDER 0001]    SND TIME: Fri Jan  5 17:34:43 2024
                         MSG BODY: TASK 1.3
------------------------------------------------------------
[RECEIVER 1002] New message received.
                         RCV TIME: Fri Jan  5 17:35:23 2024
        [SENDER 0002]    SND TIME: Fri Jan  5 17:35:23 2024
                         MSG BODY: TASK 2.3
------------------------------------------------------------
[RECEIVER 1002]          STOPPED BY END-USER: 2024-01-05, Fri Jan  5 17:42:28 2024
============================================================
```

RECEIVER #3

```
PS C:\Users\Spyros KONTOGIANNIS\Qsync\SPYROS-PERSONAL-SPACE @ NAS\mywork\mycourses\CEID\CEID_
> python .\rabbitMQ_one-one_RECEIVER.py 1003
============================================================
[RECEIVER 1003]          STARTED: 2024-01-05, Fri Jan  5 17:34:31 2024
                         Waiting for incoming messages from HELLO QUEUE. Press CTRL+C to exit
------------------------------------------------------------
[RECEIVER 1003] New message received.
                         RCV TIME: Fri Jan  5 17:35:18 2024
        [SENDER 0002]    SND TIME: Fri Jan  5 17:35:18 2024
                         MSG BODY: TASK 2.1
------------------------------------------------------------
[RECEIVER 1003]          STOPPED BY END-USER: 2024-01-05, Fri Jan  5 17:41:11 2024
============================================================
```

2.2.2. Στοιχειώδες Μοντέλο Ενορχήστρωσης Ανάθεσης Καθηκόντων (6 μονάδες)

Επεκτείνετε το μοτίβο επικοινωνίας της ενότητας 2.2.1 ώστε να δημιουργήσετε μια ουρά-ενορχηστρωτή δουλειών με ποικίλους φόρτους υπολογισμού (πχ, αποτυπώνονται ως εκτιμήσεις υποτιθέμενων χρόνων εκτέλεσης σε seconds), οι οποίες υποβάλλονται ασύγχρονα στην ουρά από διάφορους αποστολείς καθηκόντων (οι φόρτοι θα δίνονται ως μια ακολουθία παραμέτρων εισόδου, από τον χρήστη που καλεί τον εκάστοτε αποστολέα σας) σε εργάτες (οι παραλήπτες) που είναι διαθέσιμοι (δηλαδή, εκτελούνται) τη στιγμή της ανάθεσης. Κάθε εργάτης, αλλά και ο ενορχηστρωτής, θα διαθέτουν το δικό τους (τυχαία επιλεγμένο, ή δίνεται από τον χρήστη) 4-ψήφιο αναγνωριστικό. Κάθε εργάτης υπολογίζει και αποτυπώνει στην οθόνη τον συνολικό φόρτο (άθροισμα εκτιμήσεων φόρτων εργασίας) που του έχει ανατεθεί μέχρι στιγμή, τόσο ως απόλυτη τιμή αλλά και με τη μορφή μιας μπάρας προόδου. Η ανάθεση των εργασιών (ανάκλησή τους από την ουρά) γίνεται από τους συνδεδεμένους εργάτες σε round-robin λογική (αυτή είναι η de facto λειτουργία της ουράς). Κάθε εργάτης, όταν λαμβάνει ένα καθήκον για εξυπηρέτηση, επιβεβαιώνει τη λήψη και τίθεται σε καθεστώς ύπνωσης (sleep mode) για χρόνο ίσο με τον εκτιμώμενο χρόνο εκτέλεσής του. Κάθε νέα κλήση της ρουτίνας εργάτη θα δημιουργεί έναν καινούργιο εργάτη που τίθεται σε καθεστώς αναμονής για ανάληψη καθηκόντων προς εξυπηρέτηση από την ουρά εργασιών. Κάθε εκτέλεση της ρουτίνας ενορχηστρωτή δημιουργεί έναν νέο ενορχηστρωτή καθηκόντων, με συγκεκριμένους εκτιμώμενους χρόνους εκτέλεσης (πχ, σε δευτερόλεπτα) που δίνονται ως ακολουθία φυσικών (μικρών) θετικών ακεραίων αριθμών (πχ, από 1 έως 5 δευτερόλεπτα), που υποβάλλονται στην ουρά εργασιών για ανάθεση. Η εκτέλεση της ρουτίνας ενορχηστρωτή θα πρέπει να γίνεται ως εξής:

ORCHESTRATOR #1

```
> python .\rabbitMQ_task-allocation_ORCHESTRATOR.py 0001 1 2 1 3 1 4 1 5
============================================================
[ORCHESTRATOR 0001] Process Name = .\rabbitMQ_task-allocation_ORCHESTRATOR.py
         TASKS SUBMISSION TIME: Fri Jan  5 21:29:24 2024
------------------------------------------------------------

ORCHESTRATOR-ID: 0001; Task-ID: 1; Duration: 1 (seconds)
------------------------------------------------------------

ORCHESTRATOR-ID: 0001; Task-ID: 2; Duration: 2 (seconds)
------------------------------------------------------------

ORCHESTRATOR-ID: 0001; Task-ID: 3; Duration: 1 (seconds)
------------------------------------------------------------

ORCHESTRATOR-ID: 0001; Task-ID: 4; Duration: 3 (seconds)
------------------------------------------------------------

ORCHESTRATOR-ID: 0001; Task-ID: 5; Duration: 1 (seconds)
------------------------------------------------------------

ORCHESTRATOR-ID: 0001; Task-ID: 6; Duration: 4 (seconds)
------------------------------------------------------------

ORCHESTRATOR-ID: 0001; Task-ID: 7; Duration: 1 (seconds)
------------------------------------------------------------

ORCHESTRATOR-ID: 0001; Task-ID: 8; Duration: 5 (seconds)
------------------------------------------------------------

[ORCHESTRATOR 0001] Just enqueued 8 tasks to WORK QUEUE, to be serviced by active workers
============================================================
```

ORCHESTRATOR #2

```
> python .\rabbitMQ_task-allocation_ORCHESTRATOR.py 0002 5 1 4 1 3 1 2 1
============================================================
[ORCHESTRATOR 0002] Process Name = .\rabbitMQ_task-allocation_ORCHESTRATOR.py
         TASKS SUBMISSION TIME: Fri Jan  5 21:29:32 2024
------------------------------------------------------------

ORCHESTRATOR-ID: 0002; Task-ID: 1; Duration: 5 (seconds)
------------------------------------------------------------

ORCHESTRATOR-ID: 0002; Task-ID: 2; Duration: 1 (seconds)
------------------------------------------------------------

ORCHESTRATOR-ID: 0002; Task-ID: 3; Duration: 4 (seconds)
------------------------------------------------------------

ORCHESTRATOR-ID: 0002; Task-ID: 4; Duration: 1 (seconds)
------------------------------------------------------------

ORCHESTRATOR-ID: 0002; Task-ID: 5; Duration: 3 (seconds)
------------------------------------------------------------

ORCHESTRATOR-ID: 0002; Task-ID: 6; Duration: 1 (seconds)
------------------------------------------------------------

ORCHESTRATOR-ID: 0002; Task-ID: 7; Duration: 2 (seconds)
------------------------------------------------------------

ORCHESTRATOR-ID: 0002; Task-ID: 8; Duration: 1 (seconds)
------------------------------------------------------------

[ORCHESTRATOR 0002] Just enqueued 8 tasks to WORK QUEUE, to be serviced by active workers
============================================================
```

Οι ενεργοί εργάτες (τη στιγμή της υποβολής νέων καθηκόντων από κάποιον ενορχηστρωτή) θα αναλαμβάνουν τα καινούρια καθήκοντα σε μια round-robin λογική (είναι η de facto συμπεριφορά της ουράς). Για κάθε νέο καθήκον που τους ανατίθεται, τίθενται σε καθεστώς ύπνωσης (sleep) για τον αντίστοιχο χρόνο (σε δευτερόλεπτα), πριν αναλάβουν το επόμενο αίτημα, ενώ θα αποτυπώνεται με τη μορφή μπάρας προόδου και ο συνολικός υπολογιστικός φόρτος που τους έχει ανατεθεί μέχρι τότε. Για παράδειγμα, στο σενάριο που υπάρχουν (πριν την εκτέλεση των ενορχηστρωτών) τρεις ενεργοί εργάτες:

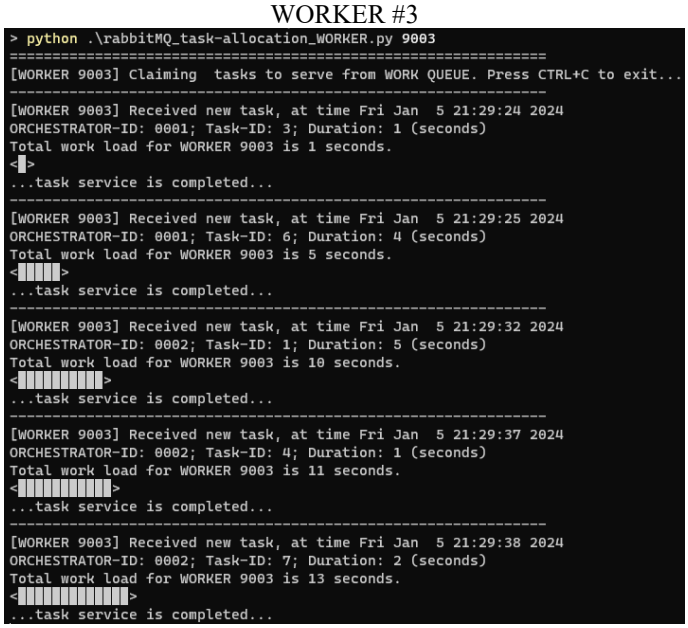
WORKER #1

```
> python .\rabbitMQ_task-allocation_WORKER.py 9001
============================================================
[WORKER 9001] Claiming  tasks to serve from WORK QUEUE. Press CTRL+C to exit...
------------------------------------------------------------
[WORKER 9001] Received new task, at time Fri Jan  5 21:29:24 2024
ORCHESTRATOR-ID: 0001; Task-ID: 1; Duration: 1 (seconds)
Total work load for WORKER 9001 is 1 seconds.
<▉>
...task service is completed...
------------------------------------------------------------
[WORKER 9001] Received new task, at time Fri Jan  5 21:29:25 2024
ORCHESTRATOR-ID: 0001; Task-ID: 4; Duration: 3 (seconds)
Total work load for WORKER 9001 is 4 seconds.
<▉▉▉▉>
...task service is completed...
------------------------------------------------------------
[WORKER 9001] Received new task, at time Fri Jan  5 21:29:28 2024
ORCHESTRATOR-ID: 0001; Task-ID: 7; Duration: 1 (seconds)
Total work load for WORKER 9001 is 5 seconds.
<▉▉▉▉▉>
...task service is completed...
------------------------------------------------------------
[WORKER 9001] Received new task, at time Fri Jan  5 21:29:32 2024
ORCHESTRATOR-ID: 0002; Task-ID: 2; Duration: 1 (seconds)
Total work load for WORKER 9001 is 6 seconds.
<▉▉▉▉▉▉>
...task service is completed...
------------------------------------------------------------
[WORKER 9001] Received new task, at time Fri Jan  5 21:29:33 2024
ORCHESTRATOR-ID: 0002; Task-ID: 5; Duration: 3 (seconds)
Total work load for WORKER 9001 is 9 seconds.
<▉▉▉▉▉▉▉▉>
...task service is completed...
------------------------------------------------------------
[WORKER 9001] Received new task, at time Fri Jan  5 21:29:36 2024
ORCHESTRATOR-ID: 0002; Task-ID: 8; Duration: 1 (seconds)
Total work load for WORKER 9001 is 10 seconds.
<▉▉▉▉▉▉▉▉▉▉>
...task service is completed...
```

WORKER #2

```
> python .\rabbitMQ_task-allocation_WORKER.py 9002
============================================================
[WORKER 9002] Claiming  tasks to serve from WORK QUEUE. Press CTRL+C to exit...
------------------------------------------------------------
[WORKER 9002] Received new task, at time Fri Jan  5 21:29:24 2024
ORCHESTRATOR-ID: 0001; Task-ID: 2; Duration: 2 (seconds)
Total work load for WORKER 9002 is 2 seconds.
<▉▉>
...task service is completed...
------------------------------------------------------------
[WORKER 9002] Received new task, at time Fri Jan  5 21:29:26 2024
ORCHESTRATOR-ID: 0001; Task-ID: 5; Duration: 1 (seconds)
Total work load for WORKER 9002 is 3 seconds.
<▉▉▉>
...task service is completed...
------------------------------------------------------------
[WORKER 9002] Received new task, at time Fri Jan  5 21:29:27 2024
ORCHESTRATOR-ID: 0001; Task-ID: 8; Duration: 5 (seconds)
Total work load for WORKER 9002 is 8 seconds.
<▉▉▉▉▉▉▉▉>
...task service is completed...
------------------------------------------------------------
[WORKER 9002] Received new task, at time Fri Jan  5 21:29:32 2024
ORCHESTRATOR-ID: 0002; Task-ID: 3; Duration: 4 (seconds)
Total work load for WORKER 9002 is 12 seconds.
<▉▉▉▉▉▉▉▉▉▉▉▉>
...task service is completed...
------------------------------------------------------------
[WORKER 9002] Received new task, at time Fri Jan  5 21:29:36 2024
ORCHESTRATOR-ID: 0002; Task-ID: 6; Duration: 1 (seconds)
Total work load for WORKER 9002 is 13 seconds.
<▉▉▉▉▉▉▉▉▉▉▉▉▉>
...task service is completed...
```

WORKER #3



Ας δούμε όμως τι θα γίνει (για τα ίδια καθήκοντα που δημιουργούν οι δύο προαναφερθέντες ενορχηστρωτές) , αν κάποιος από τους εργάτες (παρότι αρχικά διαθέσιμος) ξαφνικά τίθεται εκτός λειτουργίας (πχ, λόγω τερματισμού):

WORKER #1

WORKER #2

WORKER #3

```
> python .\rabbitMQ_task-allocation_WORKER.py 9003
============================================================
[WORKER 9003] Claiming  tasks to serve from WORK QUEUE. Press CTRL+C to exit...
------------------------------------------------------------
[WORKER 9003] Received new task, at time Fri Jan  5 21:39:25 2024
ORCHESTRATOR-ID: 0001; Task-ID: 3; Duration: 1 (seconds)
Total work load for WORKER 9003 is 1 seconds.
<.>
...task service is completed...
------------------------------------------------------------
[WORKER 9003] Received new task, at time Fri Jan  5 21:39:26 2024
ORCHESTRATOR-ID: 0001; Task-ID: 6; Duration: 4 (seconds)
Total work load for WORKER 9003 is 5 seconds.
<.....>
Interrupted
```

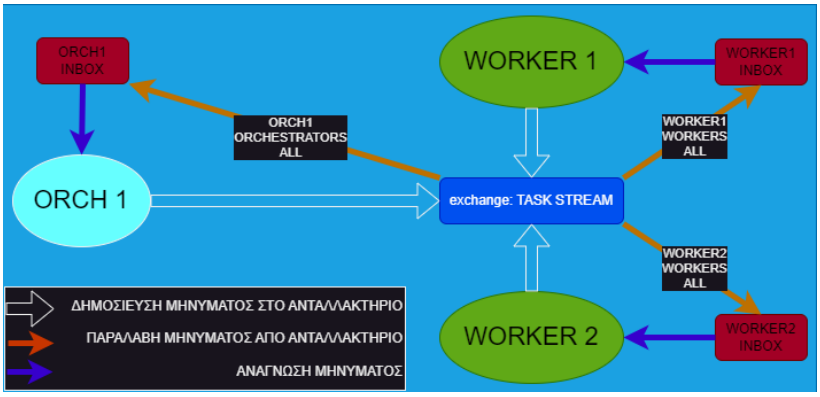
Θα πρέπει να φροντίσετε, με τις κατάλληλες επιλογές για την ουρά εργασιών (δείτε το παράδειγμα στο RabbitMQ), ώστε τελικά να ανατεθούν όλα τα καθήκοντα μεταξύ των εργατών που παραμένουν ενεργοί, παρά τις πιθανές καταρρεύσεις ενεργών εργατών. Μια τέτοια περίπτωση είναι και το παραπάνω παράδειγμα, όπου ο 3ος εργάτης πρόλαβε να επιβεβαιώσει τη διεκπεραίωση του καθήκοντος 1.3, αλλά πριν προλάβει να επιβεβαιώσει τη διεκπεραίωση του καθήκοντος 1.6 (δλδ, κατά τη διάρκεια της ρουτίνας CALLBACK που αναλαμβάνει την εξυπηρέτησή του) διακόπηκε από τον τελικό χρήστη, με αποτέλεσμα να μη στείλει ACK στην ουρά εργασιών. Ο ίδιος ο διαμεσολαβητής της ουράς επαναδρομολόγησε (πάλι με round-robin λογική) τα καθήκοντα που είχαν ανατεθεί στον 3ο εργάτη και παραμένουν ακόμη στην ουρά (συμπεριλαμβανομένου του 1.6) στους άλλους δύο εργάτες που παρέμειναν ενεργοί.

2.2.3. Μοντέλο Φιλτραρίσματος Μηνυμάτων (10 μονάδες)

Σε αυτή τη φάση καλείστε να αξιοποιήσετε την έννοια του ανταλλακτηρίου (exchange) προκειμένου να δημιουργήσετε ένα κανάλι επικοινωνίας στο οποίο μεταδίδουν ταυτόχρονα πολλοί δημοσιευτές (publishers) και ταυτόχρονα μέσα από το οι συνδρομητές (subscribers) μπορούν να παρακολουθούν επιλεγμένες θεματικές ενότητες. Προς αυτή την κατεύθυνση θα σας βοηθήσουν τα ακόλουθα παραδείγματα μοτίβων επικοινωνίας από το tutorial του RabbitMQ.

- https://www.rabbitmq.com/tutorials/tutorial-three-python.html
- https://www.rabbitmq.com/tutorials/tutorial-four-python.html
- https://www.rabbitmq.com/tutorials/tutorial-five-python.html

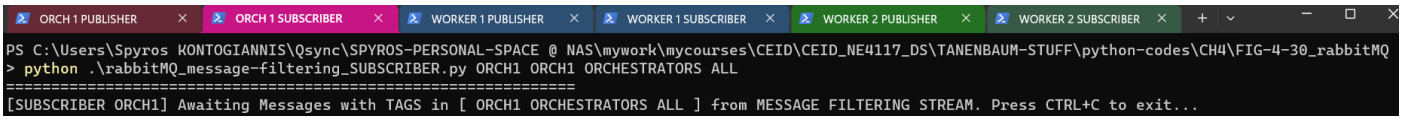
Το μοτίβο επικοινωνίας που θα υιοθετήσετε θα έχει την εξής μορφή:

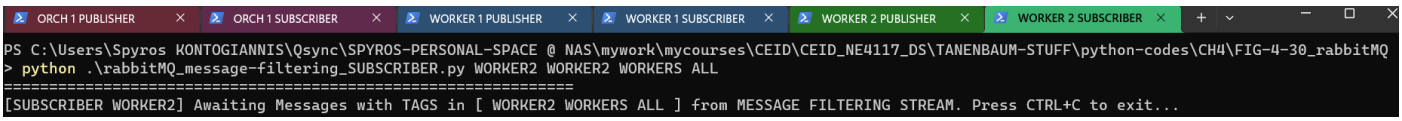


Κάθε διεργασία έχει συγκεκριμένο ρόλο (ενορχηστρωτής ή εργάτης) και μοναδικό αναγνωριστικό. Ο ρόλος ενός ενορχηστρωτή είναι να υποβάλλει καθήκοντα προς εκτέλεση σε ένα κοινό ανταλλακτήριο (έστω το exchange= 'TASK STREAM'). Ο ρόλος ενός εργάτη είναι να αναλαμβάνει (με μοναδικό τρόπο) καθήκοντα προς εξυπηρέτηση. Όλη η επικοινωνία γίνεται μέσω του ανταλλακτηρίου TASK STREAM. Προκειμένου να γίνει η ανάθεση (ενός ή περισσότερων) νέων καθηκόντων που θέλει να υποβάλλει ένας ενορχηστρωτής, ακολουθείται το εξής πρωτόκολλο:

(1) Όλοι οι ενεργοί εργάτες/ενορχηστρωτές εγγράφονται στο ανταλλακτήριο FILTERING STREAM, παρακολουθώντας τα TAGS των δικών τους αναγνωριστικών (αφορά μηνύματα αποκλειστικά προς αυτούς) και το

TAG ALL (αφορά μηνύματα προς όλους). Κάθε εργάτης παρακολουθεί επίσης μηνύματα με το TAG WORKERS (προς όλους τους εργάτες), ενώ κάθε ενορχηστρωτής παρακολουθεί επίσης μηνύματα με το TAG ORCHESTRATORS (προς όλους τους ενορχηστρωτές).

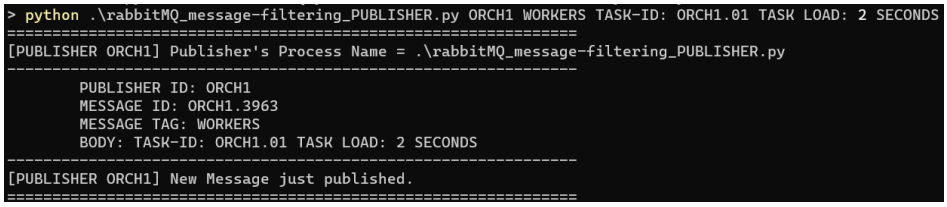




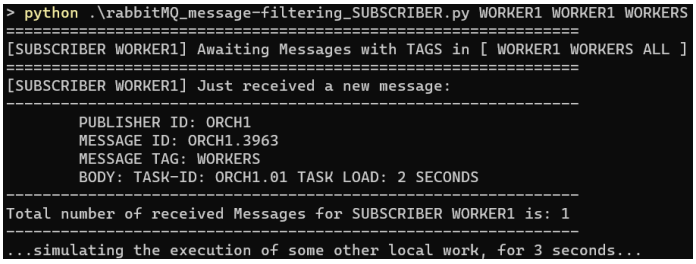
(2) Με την εμφάνιση ενός νέου καθήκοντος, ο ενορχηστρωτή αποστέλλει μήνυμα με προς όλους τους εργάτες με μορφότυπο, που εκλαμβάνεται από αυτούς ως αίτημα δήλωσης διαθεσιμότητας.

<orchestrator id> WORKERS TASK ID: <new task id> TASK LOAD: <new task's computational load> SECONDS

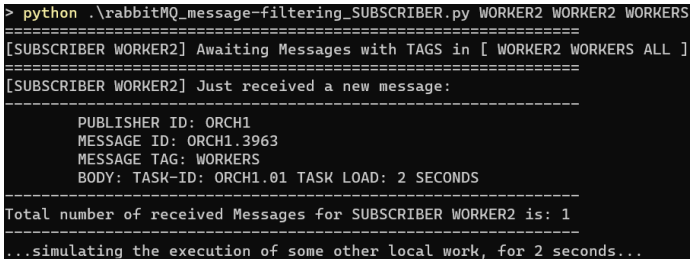
ORCH1



WORKER1



WORKER2

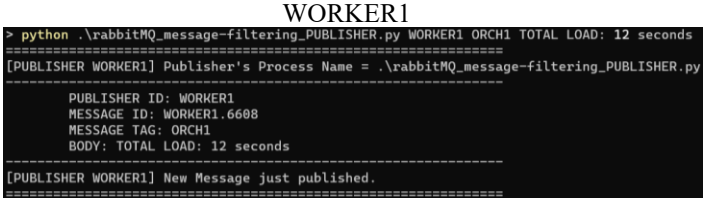


(3) Κάθε ενεργός εργάτης απαντά στο αίτημα δήλωσης διαθεσιμότητας συγκεκριμένου ενορχηστρωτή, με μήνυμα προς αυτόν με τον εξής μορφότυπο:

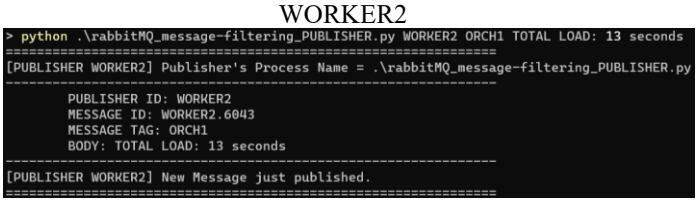
<worker id> <orchestrator tag> TOTAL LOAD: <worker's current work load (eg, in seconds)>

για επιβεβαίωση διαθεσιμότητας και αναφορά του τρέχοντος συνολικού φόρτου (από τη στιγμή που ενεργοποιήθηκε μέχρι τη στιγμή του αιτήματος για δήλωση διαθεσιμότητας). Για παράδειγμα:

WORKER1



WORKER2



ORCH1

```
> python .\rabbitMQ_message-filtering_SUBSCRIBER.py ORCH1 ORCH1 ORCHESTRATORS ALL
============================================================
[SUBSCRIBER ORCH1] Awaiting Messages with TAGS in [ ORCH1 ORCHESTRATORS ALL ] fro
============================================================
[SUBSCRIBER ORCH1] Just received a new message:
------------------------------------------------------------
        PUBLISHER ID: WORKER1
        MESSAGE ID: WORKER1.6608
        MESSAGE TAG: ORCH1
        BODY: TOTAL LOAD: 12 seconds
------------------------------------------------------------
Total number of received Messages for SUBSCRIBER ORCH1 is: 1
------------------------------------------------------------
...simulating the execution of some other local work, for 4 seconds...
============================================================
[SUBSCRIBER ORCH1] Just received a new message:
------------------------------------------------------------
        PUBLISHER ID: WORKER2
        MESSAGE ID: WORKER2.6043
        MESSAGE TAG: ORCH1
        BODY: TOTAL LOAD: 13 seconds
------------------------------------------------------------
Total number of received Messages for SUBSCRIBER ORCH1 is: 2
------------------------------------------------------------
...simulating the execution of some other local work, for 4 seconds...
```

(4) Ο ενορχηστρωτής συλλέγει τις απαντήσεις που έλαβε και στη συνέχεια αποφασίζει για τον τρόπο ανάθεσης των νέων καθηκόντων που δημιουργεί, σύμφωνα με κάποια συγκεκριμένη **πολιτική ανάθεσης**. Στο πλαίσιο της παρούσας εργασίας, θεωρήστε ότι ο ενορχηστρωτής οργανώνει τα νέα (ένα, ή περισσότερα) καθήκοντά του σε φθίνουσα σειρά χρόνων εκτέλεσης και στη συνέχεια αποφασίζει να τα αναθέσει ένα προς ένα σε κάποιον από τους διαθέσιμους εργάτες με άπληστο τρόπο, ώστε να επιτυγχάνεται η ελαχιστοποίηση του μέγιστου συνολικού φόρτου μεταξύ των εργατών μετά την ανάθεση των νέων καθηκόντων. Τέλος, αποστέλλει σε καθέναν από τους ενεργούς εργάτες ξεχωριστό μήνυμα, με μορφότυπο:

<orchestrator id> <worker tag> ASSIGNED TASK-ID: <new task id> TASK LOAD: <new task's work load>

για τα καθήκοντα που τους αναθέτει. Για παράδειγμα:

ORCH1

```
> python .\rabbitMQ_message-filtering_PUBLISHER.py ORCH1 WORKERS TASK-ID: ORCH1.01 TASK LOAD: 2 SECONDS
============================================================
[PUBLISHER ORCH1] Publisher's Process Name = .\rabbitMQ_message-filtering_PUBLISHER.py
------------------------------------------------------------
        PUBLISHER ID: ORCH1
        MESSAGE ID: ORCH1.3963
        MESSAGE TAG: WORKERS
        BODY: TASK-ID: ORCH1.01 TASK LOAD: 2 SECONDS
------------------------------------------------------------
[PUBLISHER ORCH1] New Message just published.
============================================================

PS C:\Users\Spyros KONTOGIANNIS\Qsync\SPYROS-PERSONAL-SPACE @ NAS\mywork\mycourses\CEID\CEID_NE4117_DS\TANENBAUM-STUFF\python-codes\CH4\FIG-4-30_rabbitMQ
> python .\rabbitMQ_message-filtering_PUBLISHER.py ORCH1 WORKER1 ASSIGNED TASK-ID: ORCH1.01 TASK LOAD: 2 SECONDS
============================================================
[PUBLISHER ORCH1] Publisher's Process Name = .\rabbitMQ_message-filtering_PUBLISHER.py
------------------------------------------------------------
        PUBLISHER ID: ORCH1
        MESSAGE ID: ORCH1.3750
        MESSAGE TAG: WORKER1
        BODY: ASSIGNED TASK-ID: ORCH1.01 TASK LOAD: 2 SECONDS
------------------------------------------------------------
[PUBLISHER ORCH1] New Message just published.
============================================================

PS C:\Users\Spyros KONTOGIANNIS\Qsync\SPYROS-PERSONAL-SPACE @ NAS\mywork\mycourses\CEID\CEID_NE4117_DS\TANENBAUM-STUFF\python-codes\CH4\FIG-4-30_rabbitMQ
>
```

WORKER1

```
> python .\rabbitMQ_message-filtering_SUBSCRIBER.py WORKER1 WORKER1 WORKERS
============================================================
[SUBSCRIBER WORKER1] Awaiting Messages with TAGS in [ WORKER1 WORKERS ALL ]
============================================================
[SUBSCRIBER WORKER1] Just received a new message:
------------------------------------------------------------
        PUBLISHER ID: ORCH1
        MESSAGE ID: ORCH1.3963
        MESSAGE TAG: WORKERS
        BODY: TASK-ID: ORCH1.01 TASK LOAD: 2 SECONDS
------------------------------------------------------------
Total number of received Messages for SUBSCRIBER WORKER1 is: 1
------------------------------------------------------------
...simulating the execution of some other local work, for 3 seconds...
============================================================
[SUBSCRIBER WORKER1] Just received a new message:
------------------------------------------------------------
        PUBLISHER ID: ORCH1
        MESSAGE ID: ORCH1.3750
        MESSAGE TAG: WORKER1
        BODY: ASSIGNED TASK-ID: ORCH1.01 TASK LOAD: 2 SECONDS
------------------------------------------------------------
Total number of received Messages for SUBSCRIBER WORKER1 is: 2
------------------------------------------------------------
...simulating the execution of some other local work, for 1 seconds...
```

WORKER2

```
> python .\rabbitMQ_message-filtering_SUBSCRIBER.py WORKER2 WORKER2 WORKERS
============================================================
[SUBSCRIBER WORKER2] Awaiting Messages with TAGS in [ WORKER2 WORKERS ALL ]
============================================================
[SUBSCRIBER WORKER2] Just received a new message:
------------------------------------------------------------
        PUBLISHER ID: ORCH1
        MESSAGE ID: ORCH1.3963
        MESSAGE TAG: WORKERS
        BODY: TASK-ID: ORCH1.01 TASK LOAD: 2 SECONDS
------------------------------------------------------------
Total number of received Messages for SUBSCRIBER WORKER2 is: 1
------------------------------------------------------------
...simulating the execution of some other local work, for 2 seconds...
```

(5) Ο ενεργός εργάτης αποδέχεται την ανάθεση και, μόλις ολοκληρώσει την εξυπηρέτηση, στέλνει μήνυμα επιβεβαίωσης εκτέλεσης του καθήκοντος στον ενορχηστρωτή, με τον εξής μορφότυπο:

<worker id> <orchestrator tag> COMPLETED TASK ID: <task id >

WORKER1

```
> python .\rabbitMQ_message-filtering_PUBLISHER.py WORKER1 ORCH1 TOTAL LOAD: 12 seconds
============================================================
[PUBLISHER WORKER1] Publisher's Process Name = .\rabbitMQ_message-filtering_PUBLISHER.py
------------------------------------------------------------
        PUBLISHER ID: WORKER1
        MESSAGE ID: WORKER1.6608
        MESSAGE TAG: ORCH1
        BODY: TOTAL LOAD: 12 seconds
------------------------------------------------------------
[PUBLISHER WORKER1] New Message just published.
============================================================

PS C:\Users\Spyros KONTOGIANNIS\Qsync\SPYROS-PERSONAL-SPACE @ NAS\mywork\mycourses\CEID\CEID
> python .\rabbitMQ_message-filtering_PUBLISHER.py WORKER1 ORCH1 COMPLETED TASK ID: ORCH1.01
============================================================
[PUBLISHER WORKER1] Publisher's Process Name = .\rabbitMQ_message-filtering_PUBLISHER.py
------------------------------------------------------------
        PUBLISHER ID: WORKER1
        MESSAGE ID: WORKER1.7353
        MESSAGE TAG: ORCH1
        BODY: COMPLETED TASK ID: ORCH1.01
------------------------------------------------------------
[PUBLISHER WORKER1] New Message just published.
============================================================
```

ORCH1

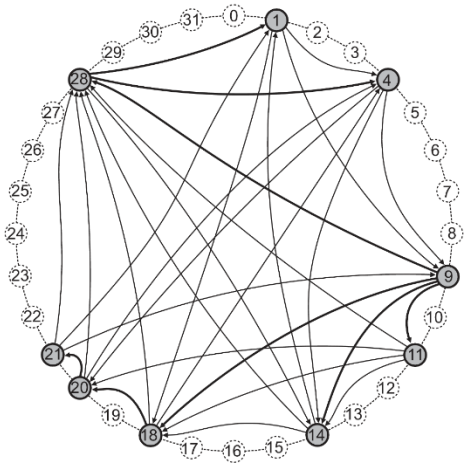
```
> python .\rabbitMQ_message-filtering_SUBSCRIBER.py ORCH1 ORCH1 ORCHESTRATORS
============================================================
[SUBSCRIBER ORCH1] Awaiting Messages with TAGS in [ ORCH1 ORCHESTRATORS ALL ]
============================================================
[SUBSCRIBER ORCH1] Just received a new message:
------------------------------------------------------------
        PUBLISHER ID: WORKER1
        MESSAGE ID: WORKER1.6608
        MESSAGE TAG: ORCH1
        BODY: TOTAL LOAD: 12 seconds
------------------------------------------------------------
Total number of received Messages for SUBSCRIBER ORCH1 is: 1
------------------------------------------------------------
...simulating the execution of some other local work, for 4 seconds...
============================================================
[SUBSCRIBER ORCH1] Just received a new message:
------------------------------------------------------------
        PUBLISHER ID: WORKER2
        MESSAGE ID: WORKER2.6043
        MESSAGE TAG: ORCH1
        BODY: TOTAL LOAD: 13 seconds
------------------------------------------------------------
Total number of received Messages for SUBSCRIBER ORCH1 is: 2
------------------------------------------------------------
...simulating the execution of some other local work, for 4 seconds...
============================================================
[SUBSCRIBER ORCH1] Just received a new message:
------------------------------------------------------------
        PUBLISHER ID: WORKER1
        MESSAGE ID: WORKER1.7353
        MESSAGE TAG: ORCH1
        BODY: COMPLETED TASK ID: ORCH1.01
------------------------------------------------------------
Total number of received Messages for SUBSCRIBER ORCH1 is: 3
------------------------------------------------------------
...simulating the execution of some other local work, for 4 seconds...
```

2.3. Υπολογισμός Ελάχιστων/Μέγιστων Ενδείξεων Θερμοκρασίας (30 μονάδες)

Έστω ότι, για σύμπαν αναγνωριστικών $U = \{00,01,02,\ldots,30,31\}$ μας δίνεται ένα σύνολο $S \subseteq U$ από αισθητήρες θερμοκρασίας, με διψήφια δεκαδικά αναγνωριστικά από το U, καθώς και 9 κατανεμημένες διεργασίες με τα (μοναδικά) αναγνωριστικά $P = \{01,04,09,11,14,18,20,21,28\}$. Επίσης, έστω ότι η επικοινωνία μεταξύ των διεργασιών και των αισθητήρων γίνεται αποκλειστικά και μόνο μέσω κάποιου ανταλλακτηρίου μηνυμάτων.

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

Θεωρούμε ότι οι 9 διεργασίες είναι οργανωμένες σε ένα δίκτυο επικάλυψης όπως φαίνεται στο ακόλουθο σχήμα.



Κάθε διεργασία έχει τη δική της λίστα έξω-γειτόνων στο δίκτυο επικάλυψης (με τους 9 γκρι κόμβους), προς τους οποίους επιτρέπεται να αποστείλει μηνύματα, είτε ξεχωριστά είτε σε όλους μαζί (όμως μόνο σε αυτούς). Πχ, η

λίστα OUT01 = {04,09,18}, υποδεικνύει τους έξω-γείτονες της P1, η OUT04 = {09,14,20} τους έξω-γείτονες της P4, κ.ο.κ.

Δημιουργήστε το πρωτόκολλο επικοινωνίας, μέσω ενός κατάλληλα κατασκευασμένου ανταλλακτηρίου, που θα επιτρέπει σε κάθε διεργασία P να αποστέλλει μηνύματα, είτε μαζικά σε όλους τους έξω-γείτονές της, είτε σε συγκεκριμένους έξω-γείτονές της. Αντίστοιχα, η P θα μπορεί να παραλαμβάνει μηνύματα που προορίζονται γι' αυτήν, στο ίδιο ανταλλακτήριο.

2.3.2. Δημιουργία Παλμών Χρονισμού και Δειγματοληψίας Τιμών Αισθητήρων Θερμοκρασίας

Θα πρέπει να κατασκευάσετε μια ρουτίνα συντονισμού, πχ, με όνομα **HeartbitAndTemperatureGenerator (HBTG)** που αναλαμβάνει τα εξής καθήκοντα:

1. **Παραγωγή παλμών συγχρονισμού (heartbits):** Μετά τον εναρκτήριο παλμό HB(0), κάθε επόμενος παλμός HB(K+1) παράγεται, όχι νωρίτερα από SamplingInterval δευτερόλεπτα (για κάποια αυθαίρετη τιμή, πχ, SamplingInterval = 30sec), και αμέσως μόλις όλες οι άλλες διεργασίες έχουν απαντήσει με επιβεβαίωση ότι παρέλαβαν ήδη τον παλμό HB(K) και τις αντίστοιχες ενδείξεις θερμοκρασιών από τους αισθητήρες τους.
2. **Προσομοίωση δειγματοληψίας θερμοκρασιών:** Στην αρχή κάθε νέου παλμού, η ρουτίνα HBTG προσομοιώνει τη δειγματοληψία θερμοκρασιών από τους 32 αισθητήρες του κατανεμημένου μας συστήματος παράγοντας τυχαίες τιμές στο διάστημα [0,40] (με ακρίβεια 1 δεκαδικού ψηφίου). ΠΡΟΣΟΧΗ: Η διεργασία HBTG θα πρέπει να δέχεται ως είσοδο μια παράμετρο που διευκρινίζει αν η (τυχαία) δειγματοληψία θα πρέπει να επαναλαμβάνεται στην αρχή κάθε παλμού, ή αν θα πρέπει να δημιουργηθεί μόνο στον παλμό HB(0) και απλά να στέλνονται τα ίδια δείγματα σε κάθε επόμενο παλμό. Εναλλακτικά, μπορείτε, όσο η HBTG θα εκτελείται, να επιτρέπετε στον χρήστη να δίνει από το πληκτρολόγιό του εντολή (πατώντας συγκεκριμένο πλήκτρο) για δημιουργία νέων δειγμάτων θερμοκρασιών, στον αμέσως επόμενο παλμό.
3. **Αποστολή δειγμάτων θερμοκρασιών στις υπεύθυνες διεργασίες:** Η ρουτίνα HBTG, σε κάθε νέο παλμό που δημιουργεί, αποστέλλει προς κάθε διεργασία P μήνυμα

   < CurrentHeartBit , SensorSamples(P) >

   με την τιμή του νέου παλμού (ως χρονοσφραγίδα) και μια πλειάδα δειγμάτων θερμοκρασίας, για όλους τους αισθητήρες που διαχειρίζεται η P.

2.3.3. Καταγραφή Τοπικών Ενδείξεων και Υπολογισμός Καθολικών Εκτιμήσεων για Ελάχιστες/Μέσες/Μέγιστες Τιμές Θερμοκρασίας

Οι 9 διαφορετικές διεργασίες (μια για κάθε γκρι κόμβο) αναλαμβάνουν την ευθύνη να λειτουργούν ως οι ακμοσυσκευές που χειρίζονται τους δικούς τους αισθητήρες. Κάθε διεργασία P αναλαμβάνει τους αισθητήρες που έχουν αναγνωριστικό μεγαλύτερο ή ίσο με το δικό της αναγνωριστικό, και μικρότερο από το αναγνωριστικό της αμέσως επόμενης διεργασίας κατά μήκος του μεγάλου κύκλου στο δίκτυο επικάλυψης. Για παράδειγμα, η διεργασία P_1 έχει στην ευθύνη της τους αισθητήρες του συνόλου SENSORS(1) = {01,02,03}, η διεργασία P_4 έχει στην ευθύνη της τους αισθητήρες του συνόλου SENSORS(4) = {04,05,06,07,08}, ενώ η διεργασία P_{28} αναλαμβάνει τους αισθητήρες του συνόλου SENSORS(28) = {28,29,30,31,00}.

Κάθε διεργασία P θα πρέπει να κάνει τα εξής:

1. **Καταγραφή και αποτύπωση τοπικών θερμοκρασιών.** Οι «τοπικές» θερμοκρασίες που αποστέλλονται στη διεργασία P (ως δείγματα των δικών της αισθητήρων) θα διατάσσονται κατ' αύξουσα σειρά και θα επιδεικνύονται στο τερματικό της συγκεκριμένης διεργασίας, μαζί με την ελάχιστη τιμή, τη μέση τιμή, και

τη μέγιστη τιμή τοπικής θερμοκρασίας. Σε κάθε νέο παλμό, θα γίνεται επικαιροποίηση αυτής της «τοπικής» εικόνας.

2. **Κατανεμημένος υπολογισμός (μέσω πλημμύρας) καθολικά ελάχιστης/μέσης/μέγιστης τιμής θερμοκρασίας.** Σε κάθε νέο παλμό, η διεργασία P, εκτός από τα τοπικά ελάχιστα/μέσα/μέγιστα, διατηρεί και μια τριάδα καθολικά ελάχιστων/μέσων/μέγιστων τιμών θερμοκρασίας. Προκειμένου να υπολογιστούν αυτές οι καθολικές τιμές, η P λαμβάνει υπόψη της τα δικά της τοπικά ελάχιστα/μέσα/μέγιστα, αλλά και τα καθολικά ελάχιστα/μέσα/μέγιστα που έλαβε (εντός του αμέσως προηγούμενου παλμού) από άλλες διεργασίες ως προς τις οποίες είναι έξω-γείτονας. Στη συνέχεια ενημερώνει τα δικά της καθολικά ελάχιστα/μέσα/μέγιστα λαμβάνοντας την ελάχιστη τιμή μεταξύ των ελαχίστων, τη μέγιστη τιμή μεταξύ των μεγίστων και τον μέσο όρο μεταξύ των μέσων. Αν προκύψει αλλαγή των καθολικών ελαχίστων/μέσων/μεγίστων τιμών που συντηρεί η ίδια, η P ενημερώνει (στον τρέχοντα παλμό) τους έξω-γείτονές της για τις νέες τιμές. Στο δικό της τερματικό, η P φροντίζει (εκτός από τις τοπικές ενδείξεις θερμοκρασίας) να επιδεικνύει και τις τρέχουσες εκτιμήσεις ελαχίστων/μέσων/μεγίστων τιμών θερμοκρασίας που διαθέτει.

3. ΠΑΡΑΔΟΣΗ ΕΡΓΑΣΙΑΣ

Δημιουργήστε τα εξής αρχεία:

(α) ΠΗΓΑΙΟΣ ΚΩΔΙΚΑΣ: Ένα συμπιεσμένο (ZIP) αρχείο, που να περιέχει όλα τα αρχεία του πηγαίου κώδικα που δημιουργήσατε για τις ανάγκες της εργασίας σας. Χρησιμοποιήστε την εξής ονοματολογία για το ZIP αρχείο που θα παραδώσετε: **NE4117_LAB1_<PROVIDE YOUR AM HERE>_SOURCE-FILES.zip**.

(β) ΓΡΑΠΤΗ ΑΝΑΦΟΡΑ: Για τη γραπτή αναφορά σας (σε DOCX ή/και σε PDF μορφή), χρησιμοποιήστε την εξής ονοματολογία: **NE4117_LAB1_<PROVIDE YOUR AM HERE>_REPORT.docx** ή **NE4117_LAB1_<PROVIDE YOUR AM HERE>_REPORT.pdf**.

Στη γραπτή αναφορά σας θα πρέπει να εξηγείτε, όσο πιο εμπεριστατωμένα και τεκμηριωμένα μπορείτε, όλες τις σχεδιαστικές σας επιλογές και να δίνετε:

- περιγραφή των βασικών μεθόδων και του σκεπτικού της υλοποίησής σας, μέσω και **ψευδοκώδικα** αλλά όχι τον αυτούσιο κώδικά σας, δεν χρειάζεται στην αναφορά.
- ένα υποτυπώδες **εγχειρίδιο χρήστη** για το πρόγραμμά σας, δίνοντας τη σύνταξη, την είσοδο, την έξοδο των μεθόδων που υλοποιήσατε.
- συγκεκριμένα παραδείγματα εκτέλεσης (με εικόνες από το τερματικό σας) για την επίδειξη των βασικών λειτουργιών του προγράμματός σας.