ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

ΤΜΗΜΑ: ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ/ΚΩΝ & ΜΗΧ/ΚΩΝ Η/Υ

Εργασία για το μάθημα «Ενσωματωμένα συστήματα πραγματικού χρόνου»

Μάρτιος 2025

Ονοματεπώνυμο: Αθανάσιος Γκόβτζιας
E-mail: gkovatha@ece.auth.gr
ΑΕΜ: 10525

Εισαγωγή

Σκοπός της εργασίας που καλούμαστε να υλοποιήσουμε είναι η δημιουργία ενός συστήματος που θα λαμβάνει ασύγχρονα τη πληροφορία συναλλαγών από τη σελίδα OKX public channel (wss://ws.okx.com:8443/ws/v5/public) για τα παρακάτω σύμβολα:

BTC-USDT, ADA-USDT, ETH-USDT, DOGE-USDT,
XRP-USDT, SOL-SDT, LTC-USDT, BNB-USDT

Για κάθε σύμβολο πρέπει να υπάρχουν 3 αρχεία, ένα για τις συναλλαγές, ένα για το κινούμενο μέσο όρο και ένα για τις τιμές του Pearson correlation. Επίσης, πρέπει να υπάρχει κι ένα αρχείο στο οποίο θα καταγράφεται το ποσοστό του χρόνου που η CPU μένει αδρανής. Το πρόγραμμα πρέπει να τρέξει στο user space του Raspberry Pi (Raspberry Pi ZERO 2W) για τουλάχιστον 48 ώρες συνεχόμενα.
Σύνδεσμος GitHub: GkovAtha/RTES_2025

Ανάλυση προγράμματος (client_okx.c)

Για την υλοποίηση του προγράμματος χρησιμοποιήθηκε η γλώσσα c και η βιβλιοθήκη libwebsockets για την επικοινωνία με τη σελίδα okx. Αρχικά, εισάγουμε στο κώδικα όλες τις κατάλληλες βιβλιοθήκες για την υλοποίηση μας. Χρησιμοποιήθηκε η πιο πρόσφατη έκδοση της βιβλιοθήκης libwebsocket 4.3 σε συνδυασμό με την OpenSSL έκδοση 1.1.1, καθώς η πιο πρόσφατη έκδοση OpenSSL 3.0 δεν ήταν συμβατή με τη libwebsocket με αποτέλεσμα να μην μπορεί να τρέξει το πρόγραμμα. Στη συνέχεια έχουμε την υλοποίηση της κυκλικής ουράς που χρησιμοποιείται για να αποθηκεύσει προσωρινά τα δεδομένα συναλλαγών που λαμβάνονται από το websocket. Το websocket λαμβάνει νέες συναλλαγές και τις αποθηκεύει στην ουρά μέχρι ένα νήμα να λάβει τα δεδομένα και να τα επεξεργαστεί. Για να εξασφαλίσουμε ότι η ουρά μπορεί να χρησιμοποιηθεί με ασφάλεια από πολλαπλά νήματα έχουμε εισάγει μηχανισμό συγχρονισμού με mutex. Όταν το πρόγραμμα τερματιστεί υπάρχει κατάλληλη συνάρτηση cleanup η οποία εξασφαλίζει ότι οι πόροι απελευθερώνονται σωστά. Αμέσως μετά, το κομμάτι κώδικα που ακολουθεί οργανώνει και αποθηκεύει τα δεδομένα για τα ζητούμενα σύμβολα. Για κάθε σύμβολο, αποθηκεύονται σε μια λίστα οι συναλλαγές που λαμβάνουμε, το ιστορικό των τελευταίων τιμών του κινούμενου μέσου όρου και το timestamp που αντιστοιχεί σε κάθε τιμή. Η λίστα διατηρεί μόνο τις πρόσφατες τιμές και αυτές που είναι παραπάνω από 15 λεπτά τις διαγράφει. Στη συνέχεια, έχουμε τις απαραίτητες συναρτήσεις για τη καταγραφή μιας συναλλαγής. Ελέγχουμε αρχικά σε ποιο σύμβολο αντιστοιχεί η συναλλαγή και σε περίπτωση που δεν είναι από τα ζητούμενα εμφανίζει σφάλμα. Αφού βρει το σύμβολο δημιουργεί η ανοίγει ένα .log αρχείο (πχ BTC-USDT_transactions.log) και καταγράφει τα στοιχεία της συναλλαγής. Αφού καταγράψει και κλείσει το αρχείο, η συναλλαγή αποθηκεύεται σε μια λίστα ώστε να είναι διαθέσιμη για τους υπόλοιπους υπολογισμούς. Υπάρχουν και τα αντίστοιχα μηνύματα σε περίπτωση σφάλματος. Το επόμενο κομμάτι κώδικα παρέχει βασικές λειτουργίες για τη διαχείριση συναλλαγών, δηλαδή καθαρισμό παλιών συναλλαγών, υπολογισμό του κινούμενου μέσου όρου όπως και το ποσοστό αδράνεια της CPU. Όλες οι παραπάνω λειτουργίες είναι σχεδιασμένες για να λειτουργούν σε πολλαπλά νήματα. Στη συνέχεια, έχουμε τη διαχείριση μια σύνδεσης websocket. Στέλνουμε αρχικά μηνύματα εγγραφής(subscriptions) για τα σύμβολα που θέλουμε δημιουργώντας ένα μήνυμα σε μορφή json. Επιπλέον, υπάρχει συνάρτηση η οποία διαχειρίζεται όλα τα γεγονότα που σχετίζονται με τη σύνδεση websocket όπως η αποστολή και λήψη δεδομένων αλλά και η διαχείριση σφαλμάτων. Επιπλέον, υπάρχει κατάλληλο νήμα το οποίο παρακολουθεί τη ροή δεδομένων. Αν δεν έχουν ληφθεί δεδομένα για ένα συγκεκριμένο χρονικό διάστημα που έχουμε ορίσει εμείς, θεωρεί ότι η σύνδεση έχει χαθεί και ενεργοποιεί διαδικασία επανασύνδεσης. Στη συνέχεια, έχουμε δυο βασικά νήματα τα οποία διαχειρίζονται τη σύνδεση websocket. Το πρώτο νήμα είναι υπεύθυνο για τη διατήρηση τη σύνδεσης websocket, επανασυνδέεται αυτόματα αν η σύνδεση χαθεί και προσθέτει τις συναλλαγές που λαμβάνει στην ουρά ενώ το δεύτερο νήμα αφαιρεί τις συναλλαγές από την ουρά κα τις καταγράφει σε αρχείο και στη μνήμη. Αυτή η συνεργασία των δυο thread βασίζεται στο μοντέλο παραγωγού – καταναλωτή (producer – consumer) όπου το websocket thread παράγει δεδομένα και το logger thread τα καταναλώνει. Το επόμενο κομμάτι κώδικα περιγράφει ένα thread το οποίο εκτελείται κάθε λεπτό και είναι υπεύθυνο για τις παρακάτω διεργασίες: υπολογισμό μέσου όρου και όγκου συναλλαγών, υπολογισμό συσχετίσεων Pearson και υπολογισμό και καταγραφή του CPU idle. Τέλος, έχουμε τη βασική λειτουργία του προγράμματός (main function). Αρχικοποιέι το πρόγραμμα μας και δημιουργεί τα 4 νήματα που εξηγήσαμε παραπάνω. Το πρόγραμμα εκτελείται σε ένα βρόγχο while μέχρι να τερματιστεί από το χρήστη. Αφού τερματιστεί από το χρήστη, τερματίζονται όλα τα νήματα, καθαρίζονται όλοι οι πόροι που είχε δεσμεύσει το πρόγραμμα και εμφανίζεται μήνυμα ότι το πρόγραμμα τερματίστηκε επιτυχώς.

Αρχεία καταγραφής

Αρχείο συναλλαγών

Μορφή εγγραφής:
```
timestamp,price,volume
```

Παράδειγμα:
```
2025-03-15 12:01:00,596.98695652,2.32524000
```

Αρχείο κινούμενου μέσου όρου

Μορφή εγγραφής:
```
timestamp,moving_average,total_volume
```

Παράδειγμα:
```
2025-03-15 12:01:00,596.98695652,10.32524000
```

Αρχείο συσχέτισης

Μορφή εγγραφής:
```
timestamp,correlation,best_symbol,best_time
```

Παράδειγμα:
```
2025-03-15 12:01:00,0.9876,ETH-USDT,2025-03-15 12:00:00
```

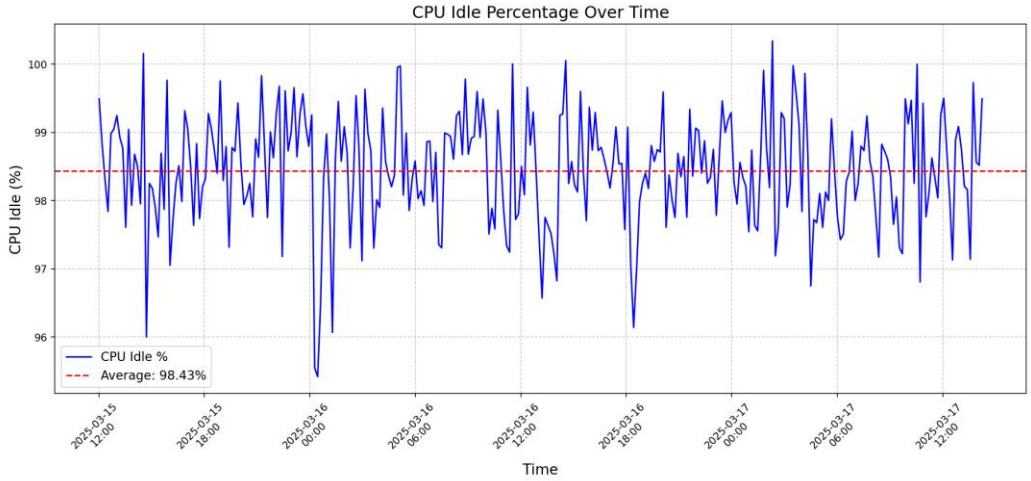
Αρχείο CPU

Μορφή εγγραφής:
```
timestamp,cpu_idle_percentage
```

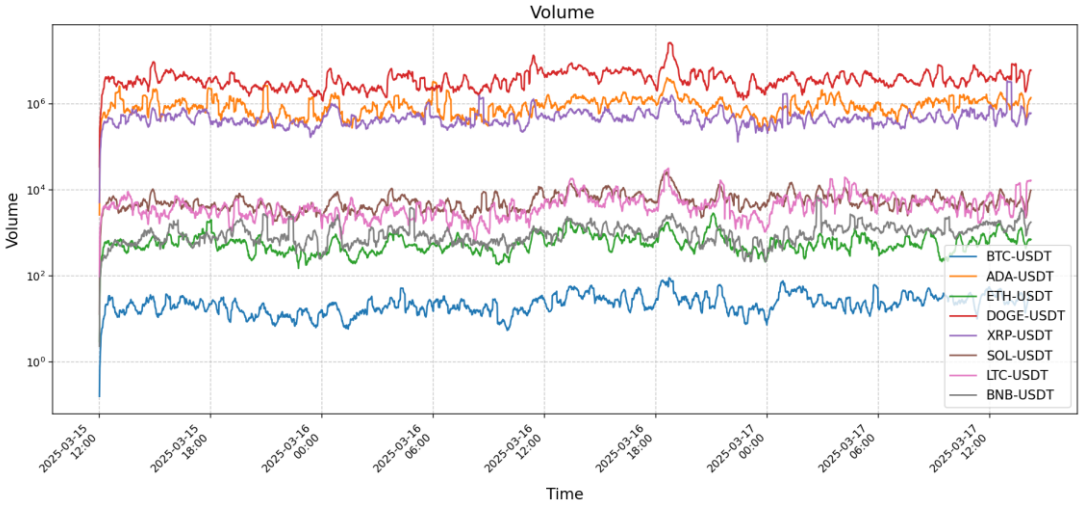
Παράδειγμα:
```
2025-03-15 12:01:00,85.23
```

Διαγράμματα

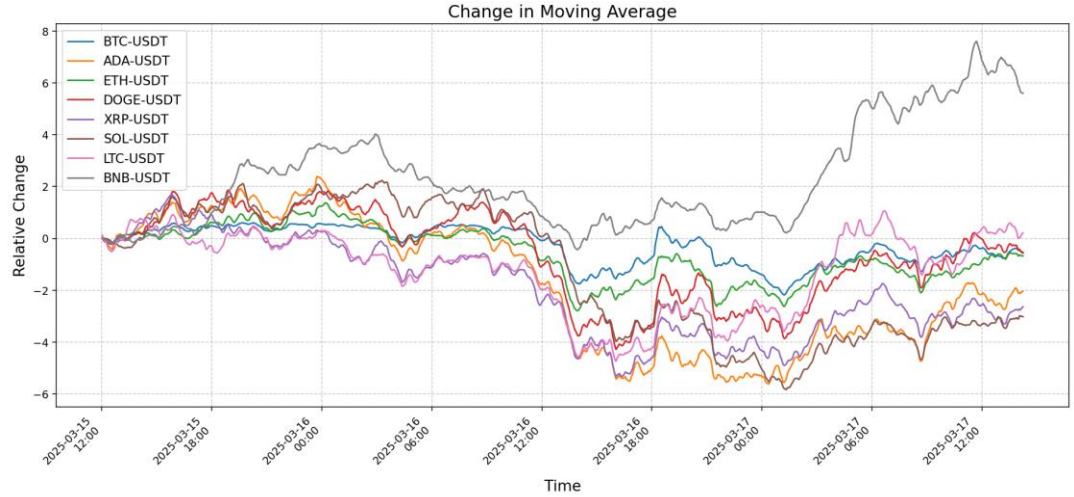
Παρακάτω ακολουθούν διαγράμματα για την οπτική ανάλυση των δεδομένων. Για τη δημιουργία των διαγραμμάτων χρησιμοποιήθηκαν προγράμματα γραμμένα σε python για την ανάγνωση των αρχείων και την εξαγωγή των δεδομένων σε μορφή διαγράμματος.



Διάγραμμα CPU idle



Διάγραμμα μεταβολής όγκου για κάθε σύμβολο



Διάγραμμα μεταβολής κινούμενου μέσου όρου για κάθε σύμβολο