Ενσωματωμένα Συστήματα Πραγματικού Χρόνου

Σκοπός της παρούσας εργασίας είναι η συλλογή δεδομένων συναλλαγών για διάφορα σύμβολα μέσω της πλατφόρμας του finnhub. Στόχος μας είναι η συνεχής λειτουργία του προγράμματος στο Raspberry Pi καθώς επίσης και η καταγραφή, η αποθήκευση και η επεξεργασία των δεδομένων που μας στέλνει το finnhub, σε πραγματικό χρόνο. Για την υλοποίηση όλων των παραπάνω χρησιμοποιήθηκαν πολλαπλά νήματα και πιο συγκεκριμένα βασίστηκα στο παράδειγμα του producer-consumer ενώ επίσης χρησιμοποιήθηκε και η προτεινόμενη βιβλιοθήκη libwebsockets για την δημιουργία του Web Socket.

Η εργασία και ο κώδικας βρίσκονται στο παρακάτω link https://github.com/chrisdardas/RTES-PROJECT-2024

Σύντομη περιγραφή του τρόπου υλοποίησης:

Βασικός στόχος είναι η υλοποίηση της συνάρτησης **ws_callback()** μέσω της οποίας διαχειρίζονται όλες οι λειτουργίες του Web Socket. Με την συνάρτηση αυτά τα νήματα producer πραγματοποιούν την σύνδεση δημιουργώντας το Web Socket, κάνουν subscribe στα αντίστοιχα σύμβολα που βρίσκονται στον πίνακα subscriptions, δέχονται τα JSON αρχεία από το Web Socket και κάνουν το parsing χρησιμοποιώντας την βιβλιοθήκη **jansson.h** βάζοντας στην συνέχεια τα trade objects στην κοινή ουρά και τέλος είναι υπεύθυνα να ξαναδημιουργήσουν το socket σε περίπτωση σφάλματος ή να κλείσουν το socket όταν επιθυμούμε να σταματήσουμε την εκτέλεση του προγράμματος. Επιπλέον, η συνάρτηση **client()** είναι υπεύθυνη για την δημιουργία του context του Web Socket καθώς επίσης και για την δημιουργία του client, ενώ ακόμη μπορεί να κληθεί και από την συνάρτηση **reconnect()** σε περίπτωση που υπάρξει κάποιο σφάλμα και κλείσει η σύνδεση μας. Στην συνέχεια μέσα στο main πρόγραμμα κάνω initialize το **mutex_candlestick** το οποίο γίνεται locked και δεν επιτρέπει το νήμα **quarter_mvg_avg** να σώσει τους κινούμενους μέσους όρους έως ότου έρθει το condition done που σηματοδοτεί πως τα candlesticks έχουν σώσει τα δεδομένα τους για το λεπτό οπότε μπορεί πλέον και ο κινούμενος μέσος όρος να σωθεί για το τρέχον λεπτό. Επιπλέον, κάνω initialize το **connection_mutex** το οποίο αφήνει μόνο ένα νήμα producer να κάνει την σύνδεση και να καλέσει το API του finnhub κάθε φορά ώστε να μην έχουμε θέματα με πολλαπλά API calls. Τέλος το condition minute έρχεται μέσω του **timer**(τον οποίο θα περιγράψω στην συνέχεια) ξυπνώντας το νήμα που είναι υπεύθυνο για την δημιουργία των candlesticks του τρέχοντος λεπτού. Επιπρόσθετα, μέσα στην main αρχικοποιώ τους **file pointers** και τους βάζω όλους σε λειτουργία μόνο για γράψιμο, ενώ επίσης δημιουργώ τα νήματα **producer**, **consumer**, **candlestick_con** και **mvg_avg_con** τα οποία σε περίπτωση σφάλματος θα στείλουν στο process που τα δημιούργησε ένα σήμα SIGINT.

Για την διαχείριση αυτού του σήματος έχω υλοποιήσει έναν **signal handler** ο οποίος στην περίπτωση του **SIGINT** καταστρέφει το Web Socket κάνει **broadcast** τα **condition signals** σχετικά με την κοινή ουρά αλλά και αυτά σχετικά με τα candlesticks και το moving average έτσι ώστε να μην κολλήσουν τα νήματα και τέλος καλεί την συνάρτηση **cleanup()** ή οποία είναι υπεύθυνη για την αποδέσμευση της δυναμικής μνήμης καθώς επίσης και για το κλείσιμο των **file pointers** και για την καταστροφή της κοινής ουράς. Πέρα από αυτή την λειτουργία ο **signal handler** είναι επίσης υπεύθυνος να λειτουργεί και ως timer ο οποίος μετά από την πάροδο ενός λεπτού λαμβάνει το σήμα **SIGALRM**, κλειδώνει το **mutex_candlestick**

ΧΡΗΣΤΟΣ-ΑΛΕΞΑΝΔΡΟΣ ΔΑΡΔΑΜΠΟΥΝΗΣ (ΑΕΜ : 10335)

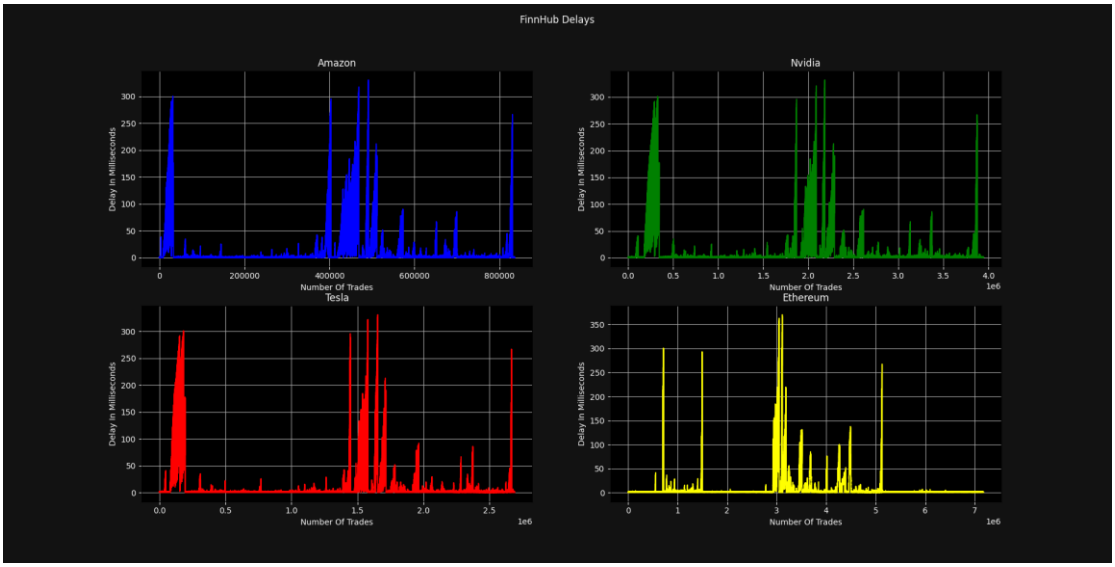
και στέλνει μέσω του condition σήμα ώστε να ξεκινήσει το νήμα που είναι υπεύθυνο για το σώσιμο των candlesticks να αρχίζει να τα αποθηκεύει στους αντίστοιχους φακέλους.

Τα νήματα producer, όπως ανέφερα και προηγουμένως είναι πολύ σημαντικά για το πρόγραμμα καθότι είναι αυτά που πραγματοποιούν την σύνδεση με το finnhub δημιουργώντας το **Web Socket** και είναι αυτά που καλούν το API του finnhub μέσω της συνάρτησης **lws_service()** οπού ανάλογα το case εκτελείται και η αντίστοιχη περίπτωση από την συνάρτηση **ws_callback()**. Στην περίπτωση που λαμβάνουμε δεδομένα καλείται η συνάρτηση **process_message()** η οποία είναι υπεύθυνη για το parsing του JSON που μας στέλνει το finnhub. Αρχικά φορτώνω το μήνυμα που έλαβα και ελέγχω άμα είναι JSON array. Στην περίπτωση που δεν είναι σημαίνει πως δεν έχω λάβει δεδομένα από το finnhub αλλά κάποιο ping. Μετά από πολλές δοκιμές παρατήρησα πως σε περίπτωση που υπάρξει κάποιο error το finnhub, παρόλο που έχω κάνει establish την σύνδεση μου στέλνει κάποια pings για αρκετή ώρα. Για να το αντιμετωπίσω, έβαλα έναν counter ο οποίος μετράει τα pings και άμα έρθουν 3 συνεχόμενα θεωρώ πως έχω χάσει το connection και ενημερώνω το αντίστοιχο flag προκειμένου ένα από τα νήματα producer να δημιουργήσει εκ νέου την σύνδεση με το finnhub. Άμα τώρα βρισκόμαστε στην περίπτωση όπου λαμβάνουμε ένα JSON array τότε εξάγω τα δεδομένα που με ενδιαφέρουν χρησιμοποιώντας τις συναρτήσεις της βιβλιοθήκης και τα σώζω σε μια δομή τύπου trade όπου στην συνέχεια σώζεται στην κοινή ουρά.

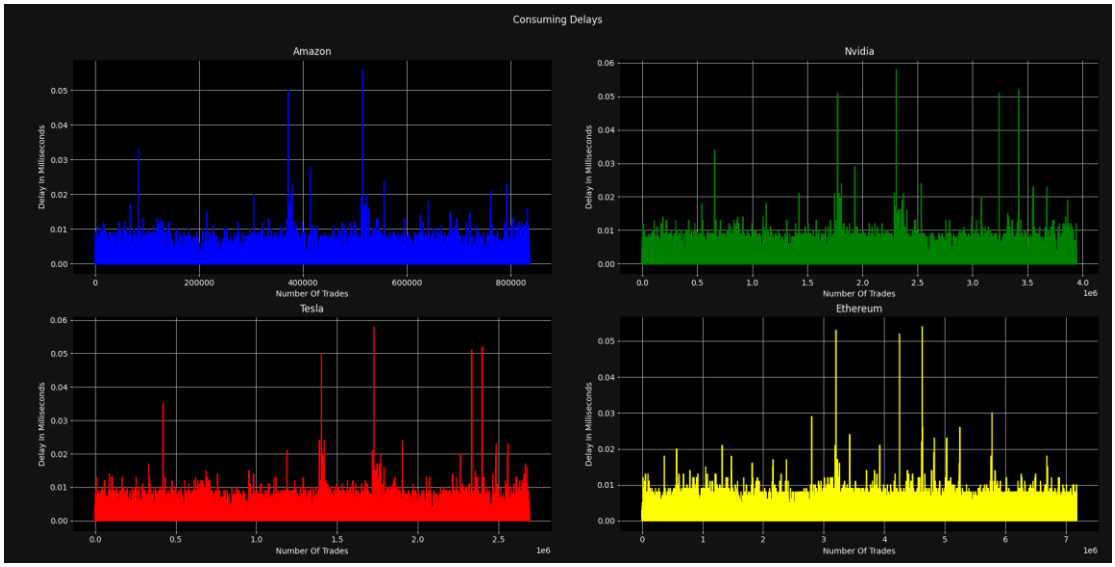
Τα νήματα consumer είναι υπεύθυνα για να λαμβάνουν τα trade objects από την ουρά, να τα σώζουν στο κατάλληλο αρχείο και παράλληλα να δημιουργούν το κατάλληλο candlestick σε πραγματικό χρόνο έως ότου χτυπήσει ο **timer** και σωθούν από το νήμα που εκτελεί την ρουτίνα **minute_candlestick()**. Μέσα σε αυτή την ρουτίνα σώζεται το candlestick που έχει φτιαχτεί στο συγκεκριμένο λεπτό, υπολογίζεται ο συνολικός όγκος των συναλλαγών καθώς και ο κινούμενος μέσος όρος για το συγκεκριμένο λεπτό και τέλος αρχικοποιούνται ξανά τα candlesticks για το επόμενο λεπτό. Επιπλέον, ορισμένες φορές το finnhub σταματάει να στέλνει δεδομένα για 2 ή και περισσότερα σύμβολα, οπότε για να διορθώσω αυτή την ενέργεια, κλείνω την σύνδεση και την ξαναφτιάχνω. Άμα αυτό συμβεί 3 φορές συνεχόμενα, τότε θεωρώ πως το χρηματιστήριο είναι κλειστό και σταματώ να αποσυνδέομαι και να ξαναφτιάχνω το Web Socket. Τέλος το νήμα που είναι υπεύθυνο για την εκτέλεση της ρουτίνας **quarter_mvg_avg()** σώζει τους κινούμενους μέσους όρους κάθε συμβόλου στα αντίστοιχα αρχεία.

<u>Αποτελέσματα εκτέλεσης του προγράμματος:</u>

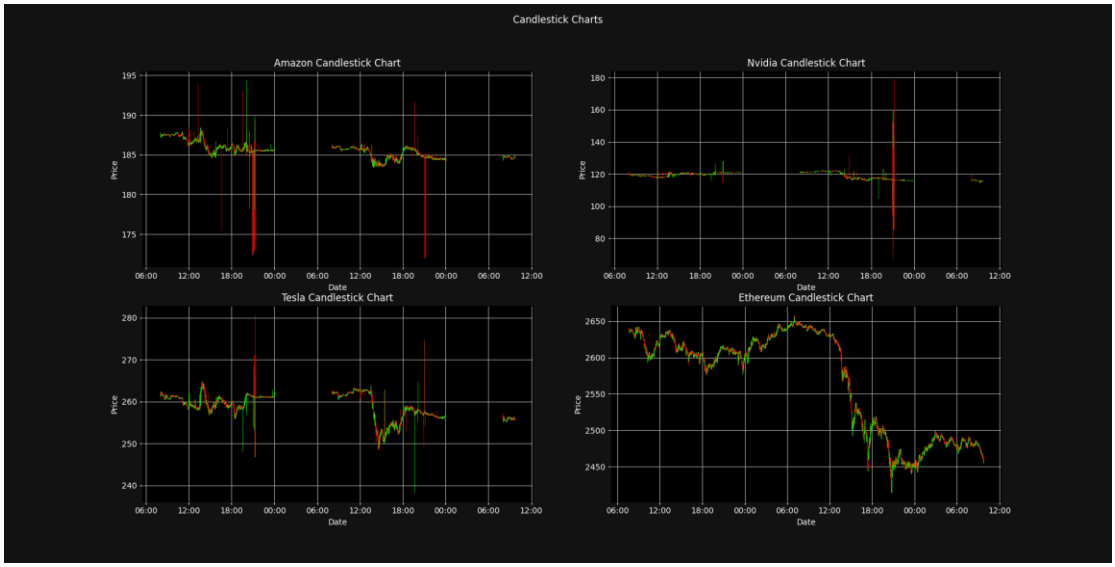
Παρακάτω παραθέτω τα σχετικά γραφήματα μετά από την εκτέλεση του προγράμματος για **50,173 ώρες** στο Raspberry Pi. Αρχικά παραθέτω τα γραφήματα σχετικά με τα delays που αφορούν το finnhub καθώς επίσης και τα delays που αφορούν το πόσο γρήγορα γίνονται consumed από την στιγμή που μπαίνουν στην ουρά. Στην συνέχεια παραθέτω τα γραφήματα με τα candlesticks καθώς επίσης και τις χρονικές διαφορές του επόμενου με το προηγούμενο candlestick. Τέλος παραθέτω την μεταβολή των κινούμενων μέσων ορών καθώς επίσης και τις χρονικές διαφορές του επόμενου με τον προηγούμενο κινούμενο μέσο όρο.



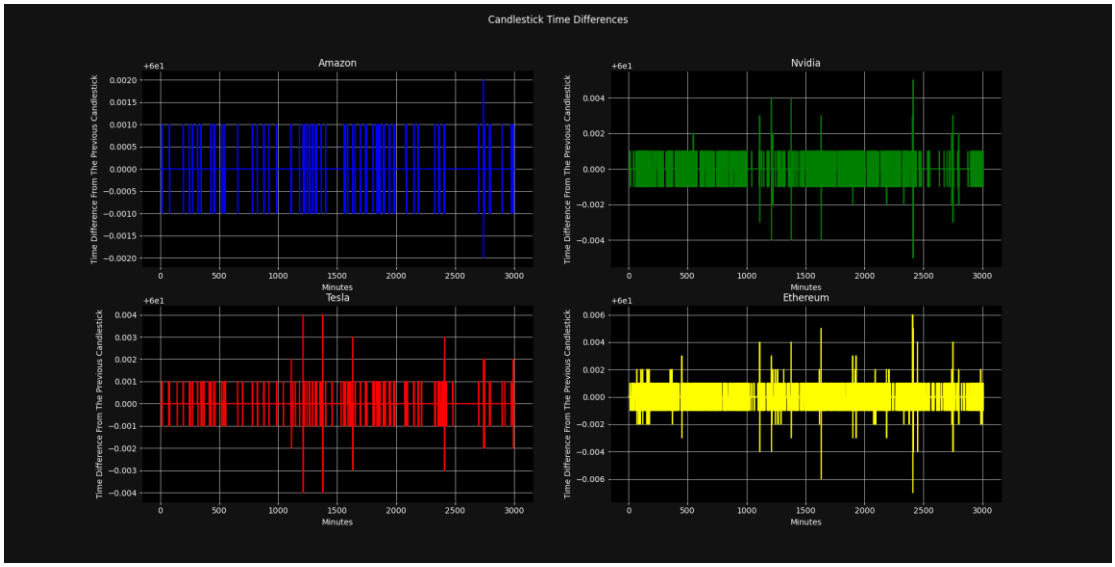
Εικόνα 1 Finnhub Delays for every Symbol



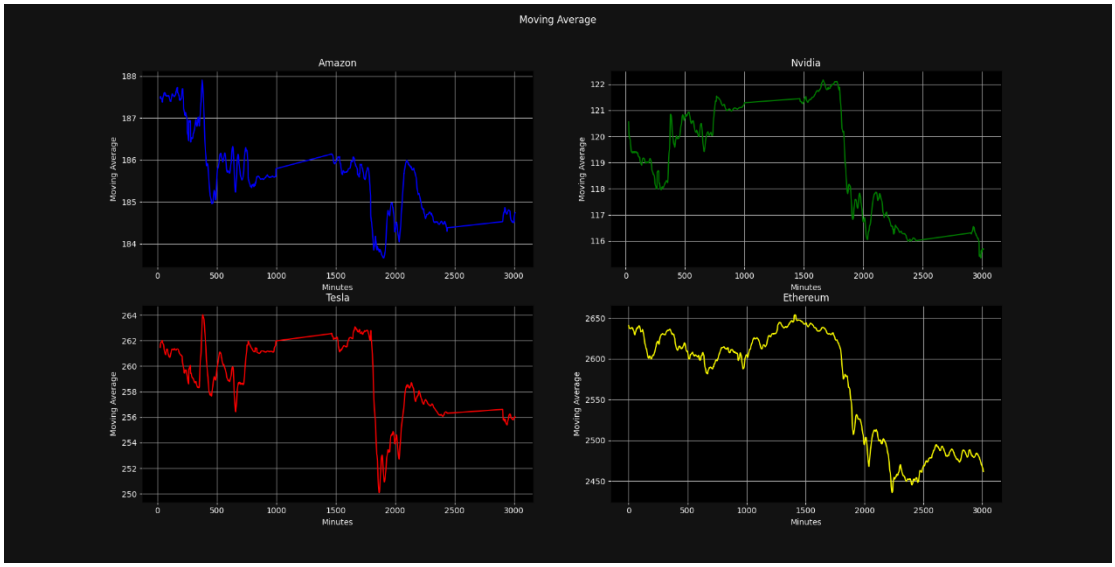
Εικόνα 2 Consuming Delays for every Symbol



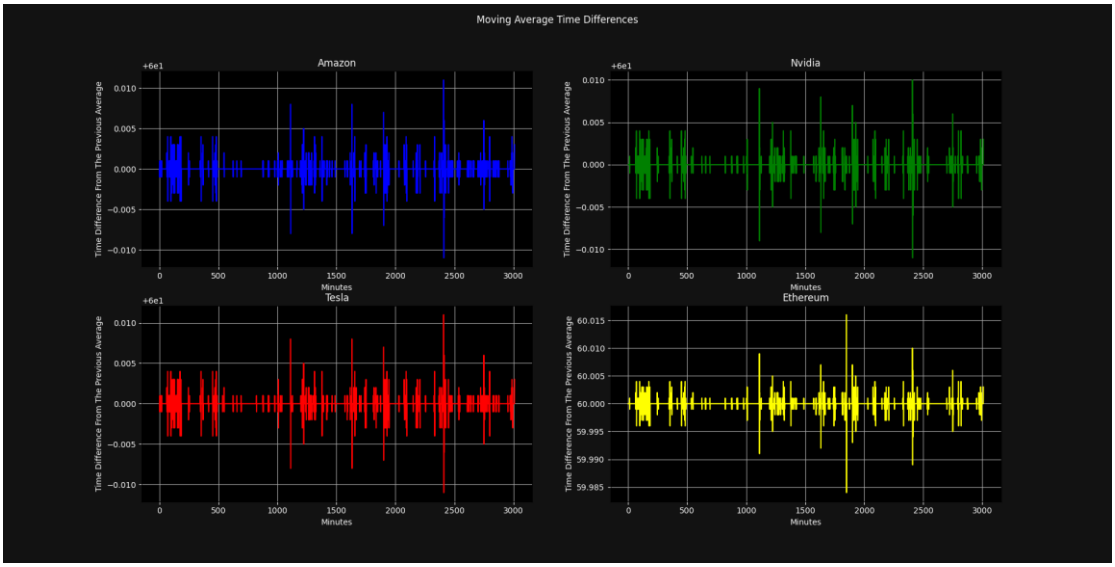
Εικόνα 3 Candlesticks for every Symbol



Εικόνα 4 Candlestick Time Differences for every Symbol

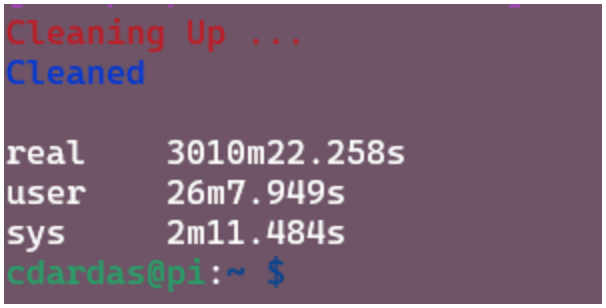


Εικόνα 5 Moving Average for every Symbol



Εικόνα 6 Moving Average Time Differences for every Symbol

Για να καταγράψω τον χρόνο που η CPU έμεινε αδρανής, έτρεξα το πρόγραμμά μου με την εντολή time και τα αποτελέσματα που έλαβα μετά το πέρας εκτέλεσης του προγράμματος μου είναι τα εξής :



Αυτό μας δείχνει ότι το πρόγραμμα μου έτρεχε για **3010 λεπτά και 22 δευτερόλεπτα** δηλαδή για **180.622,258** δευτερόλεπτα ενώ ο χρόνος που ήταν απασχολημένη η CPU είναι ο user time + sys time δηλαδή για **1.699, 433** δευτερόλεπτα. Επομένως ο συνολικός χρόνος που η CPU δεν ήταν είναι αδρανής είναι $\frac{\mathbf{1.699,433}}{\mathbf{180.622,258}} \times \mathbf{100}\% = \mathbf{0.94}\%$ άρα η CPU ήταν αδρανής για το $\mathbf{99,06}\%$ του χρόνου εκτέλεσης του προγράμματος, πράγμα που είναι λογικό και αναμενόμενο καθότι στην πλειονότητα του χρόνου εκτέλεσης περιμένουμε για **I/O** operations