ΕΡΓΑΣΙΑ 2024 -ΕΝΣΩΜΑΤΩΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΡΑΓΜΑΤΙΚΟΥ ΧΡΟΝΟΥ

Στόχος της εργασίας είναι να υλοποιήσουμε ένα σύστημα που θα λαμβάνει ασύγχρονα τη πληροφορία συναλλαγών για ένα σύνολο από σύμβολα από το finnhub. Έπειτα θα καταγράφουμε σε αρχεία τις συνναλαγές την στιγμή που πραγματοποιούνται και στο λεπτό κάθε λεπτό θα καταγράφουμε για κάθε συμβολο το candlestick και τον κινούμενο μέσο όρο των τιμών συναλλαγών και συνολικό όγκο των πιο πρόσφατων 15 λέπτων. Επομένως θα έχουμε 3 αρχεία μέσα στα οποία θα είναι καταγεγραμμένη η πληροφορία. Θα χρησιμηποιηθεί το Rasberry Pi για να “τρέξει” το πρόγραμμά μας. Για την καταγραφή και επεξεργασία των παραπάνω δεδομένων χρησιμοποιήθηκαν threads. Παρακάτω θα περιγράψω τον τρόπο υλοποίησης της εργασιας.

Ξεκινώντας λοιπόν γράφω όλες τις βιβλιοθήκες που θα χρειαστώ και ορίζω μερικές global μεταβλητές οι οποίες είναι απαραίτητες για την επιτυχή σύνδεση. Έπειτα υλοποιώ τις συναρτήσεις που ασχολούνται με την αρχικοποίηση της ουράς αλλά και με την προσθήκη και την αφαίρεση στοιχείου από την ουρά του καθώς κάθε σύμβολο έχει την δικιά του ουρά. Πιο συγκεκριμένα για την προσθήκη και την αφαίρεση στοιχείου , χρησιμοποιώ την **pthread_mutex_lock()** και την **pthread_mutex_unlock()** έτσι ώστε να εξασφαλίσω ότι όταν ο producer (main) θα κάνει enqueue στην ουρά , ο consumer(thread) δεν θα μπορεί να κάνει dequeue και το αντίστροφο. Στην συνέχεια υλοποιώ τις συναρτήσεις που πραγματοποιουν την καταγραφη των δεδομένων στα εκάστοτε αρχεία. Αν το μέγεθος των αρχείων είναι μηδενικό (**ftell(file)** == 0), πράγμα που σημαίνει ότι είναι κενά, τότε δηλώνω τι αντιπροσωπεύει η κάθε στήλη του αρχείου, αν όχι αποθηκεύω τα δεδομένα. Στην συνάρτηση **write_candlestick_to_file()** αρχικά είχα υλοποιήσει να υπάρχει ένα μήνυμα μαζί με το candlestick όταν δεν έχω trades σε κάποιο λεπτό. Αυτό όμως μου δυσκόλεψε την επεξεργασία των δεδομένων στο Matlab και γιαυτό το λόγο το έχω βάλει σε σχόλια. Έπειτα υλοποιώ την **calculate_time_difference()**, η οποία επιστρέφει την απόκλιση από την

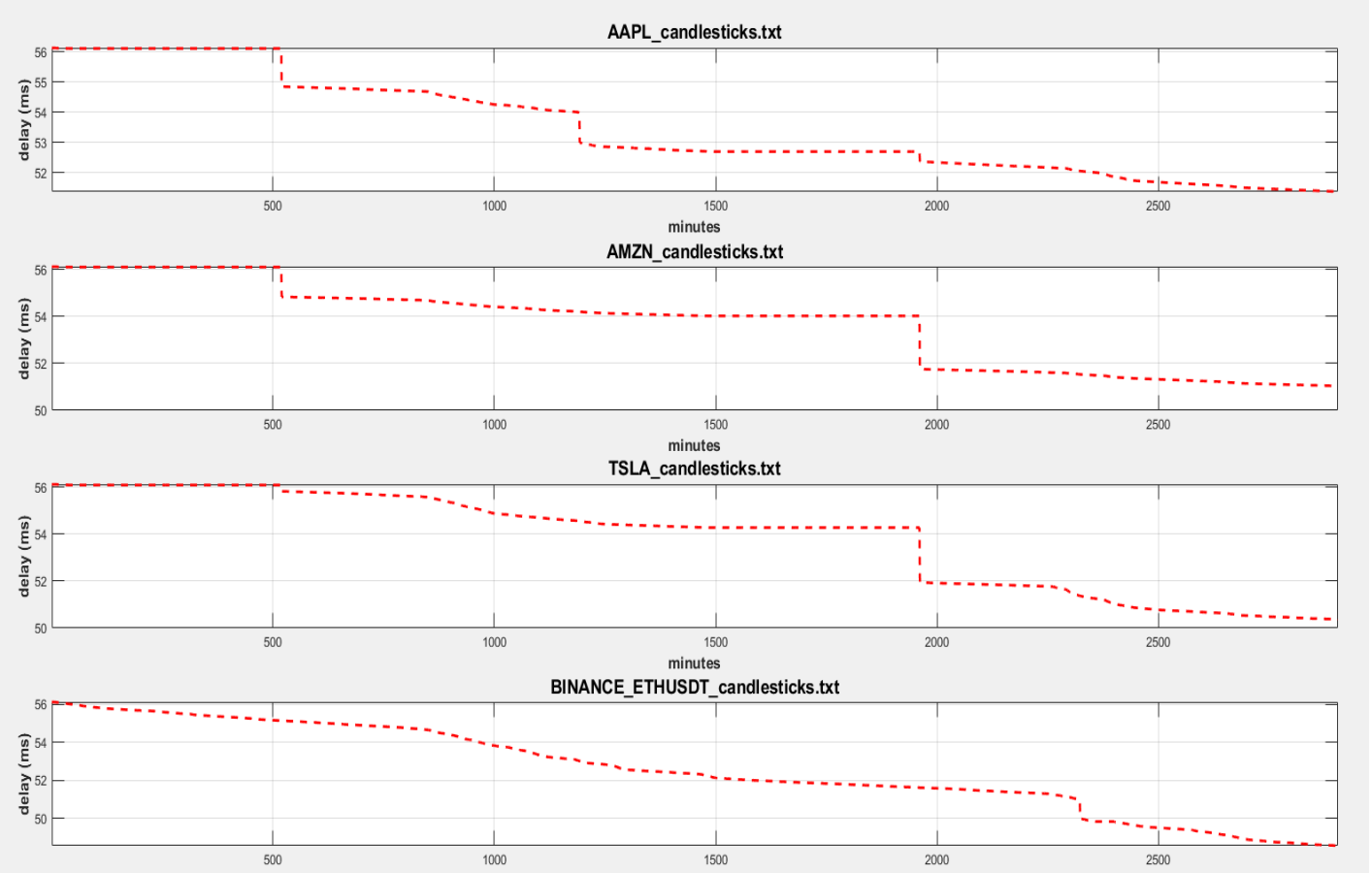
επιθυμητή επεξεργασία των συναλλαγών που είναι στο τέλος κάθε λεπτού(59s 999999us) με την πραγματική στιγμή που επεξεργάζονται. Όλη η επεξεργασία των δεδομένων που συλλέγω από το finnhub πραγματοποιείται στην **process_trades()**, όπου αφού καλέσουμε την **calculate_time_difference()** για να πάρουμε την χρονικη απόκλιση που αναφέρθηκε παραπάνω, χρησιμοποιούμε την **pthread_mutex_lock()**, έτσι ώστε να διασφαλίσουμε ότι μόνο ο consumer του κάθε συμβόλου θα έχει πρόσβαση στην ουρά του, μην επιτρέποντας έτσι στον producer να προσθέσει δεδομένα στην ουρά. Έχοντας διασφαλίσει το παραπάνω δημιουργούμε το candlestick. Στην περίπτωση που δεν έχω συναλλαγές για ένα σύμβολο, επέλεξα να μεταφέρω το candlestick του προηγούμενου λεπτού, δείχνοντας έτσι ότι υπάρχει σταθερότητα στην αγορά καθώς η τιμή του παραμένει ίδια. Στη συνέχεια αποθηκεύω τα τελευταία 15 candlesticks σε μια λίστα και όταν αυτή η λίστα γεμίσει, αφαιρείται το παλιότερο candlestick, έτσι ώστε να προσθεθεί το καινούριο. Όσο αναφορά τον υπολογισμό του moving_average, διατρέχω κάθε στοιχείο που βρίσκεται στην ουρά και υπολογίζω το συνολικό άθροισμα των τιμών και τον συνολικο όγκο, επιτρέποντας μου έτσι να υπολογίσω και το moving_average διαιρώντας το συνολικό αθροισμα των τιμών με τον συνολικό όγκο. Ίσως η πιο σημαντική συνάρτηση είναι η **callback_f()**, η οποία γίνεται triggered κάθε φορά που συμβαίνει ένα event στο Websocket. Έχουν υλοποιηθεί διάφορα cases, όπως όταν συνδεόμαστε επιτυχώς με τον server (*LWS_CALLBACK_CLIENT_ESTABLISHED*), όταν ο server μας στέλνει ένα μήνυμα (*LWS_CALLBACK_CLIENT_RECEIVE*) και όταν συμβαίνει κάποιο error στην σύνδεση ή όταν κλείνει η σύνδεση, όπου και στις 2 αυτές περιπτώσεις καλούμε την reconnect_websocket(), έτσι ώστε να ξανασυνδεθούμε στον server. Πιο αναλυτικά για το case *LWS_CALLBACK_CLIENT_RECEIVE*, αρχικά ελέγχουμε αν τα δεδομένα που πήραμε είναι πίνακας. Αν ναι, τότε είναι trade data και αφού απομονώσουμε τα δεδομένα που θέλουμε (τιμή, σύμβολο,ποσότητα, timestamp), τα προσθέτουμε στην ουρά του συμβόλου που επεξεργάζεται τη συγκεκριμένη χρονική στιγμή. Αν όχι, τότε λαμβάνουμε ping. Σε περίπτωση που έρθουν 3 συνεχόμενα pings, πραγματοποιώ επανασύνδεση καλώντας την **reconnect_websocket()** . Στην συγκεκριμλενη

συνάρτηση αρχικά ελέγχουμε πόσες προσπάθειες για reconnection έχουν γίνει. Σε περιπτωση που έχουν γίνει 10 προσπάθειες καλουμε την **terminate_program()**. Έπειτα ορίζουμε τις απαραίτητες μεταβλητές για να επιτύχει η σύνδεση. Σε περίπτωση αποτυχίας ξανακαλούμε την **reconnect_websocket()**, έχοντας κανει sleep(5), κάθως έπρεπε να προσέξω μην ξεπεράσω το όριο των API calls που έχει το finnhub.

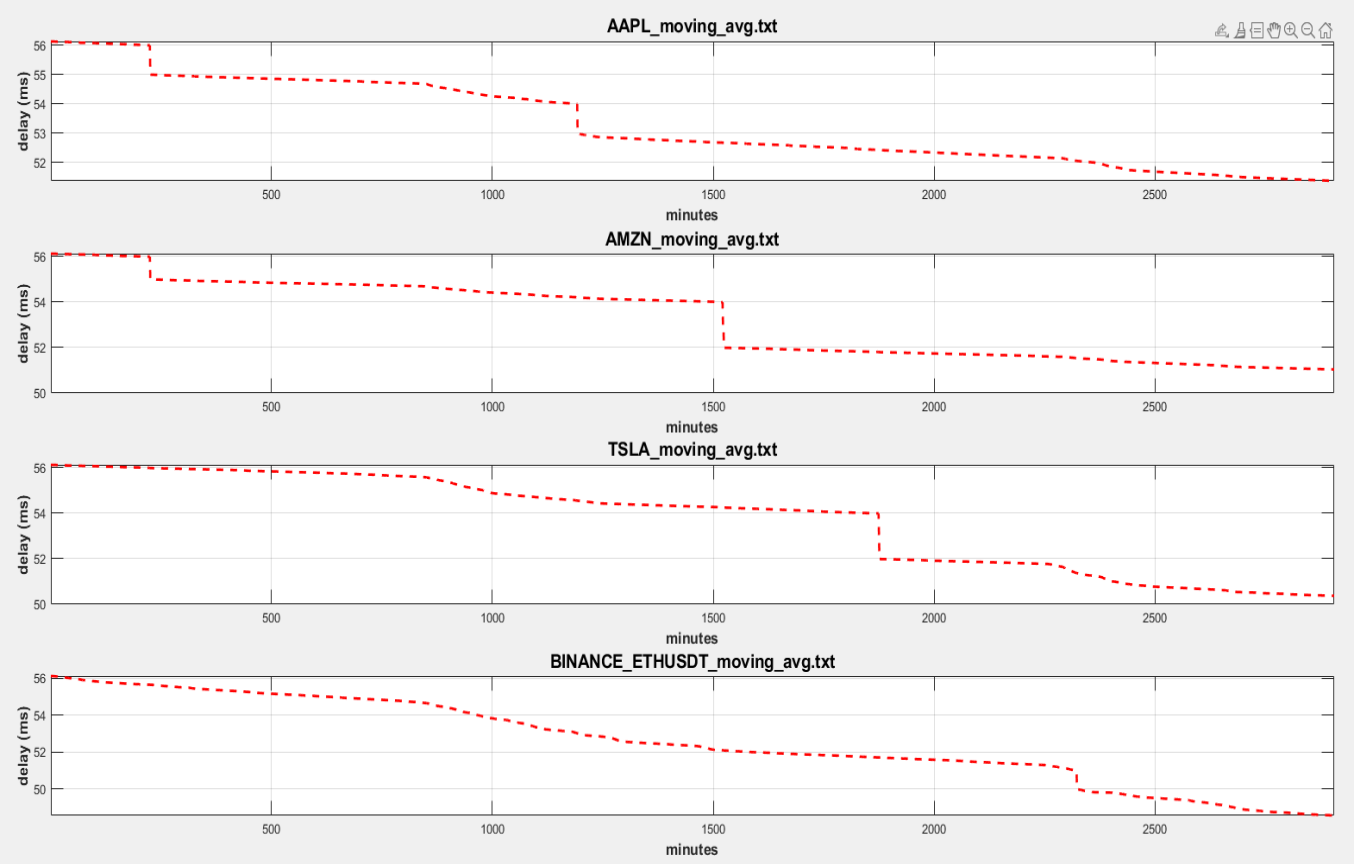
Για τον **consumer()** έχουμε να πούμε ότι, ελέγχει αν έχουν περάσει 60 sec από την τελευταία επεξεργασία. Αν έχουν περάσει 60 sec, καλείται η **process_trades()**. Ο consumer υπολογίζει το χρόνο που απομένει μέχρι το επόμενο λεπτό και κοιμάται έτσι ωστε να εξοικονομηθεί η χρήση της CPU. Από την αλλη πλευρά, η **main()** που αποτελεί τον producer καλεί για πρώτη φορά την **reconnect_websocket()** για να γίνει η σύνδεση και με την βοήθεια ενός ατέρμονα βρόχου, κοιτάει το websocket για δεδομένα ανά 1000ms και έπειτα γίνεται trigger η **callback_f()** ανάλογα με το case που θα έχουμε.

Παρακάτω παραθέτω διάγραμματα σχετικά με την χρονική απόκλιση ανάμεσα στην ιδανική επεξεργασία των δεδομένων (κάθε 59sec 999999us) και την πραγματική.

- Candlesticks



- Moving_average



- Για το ποσοστό που η CPU μένει αδρανής έχουμε το εξής :

```
^C
real    2905m21.403s
user    17m1.258s
sys     7m1.161s
```

Έχοντας πάρει τα παραπάνω τρέχοντας το πρόγραμμά με το εντολή **time ./final** ,
εκτελούμε τις παρακάτω πράξεις και παίρνουμε το CPU idle percentage

```
%%for the cpu idle

real_time = 2905*60 + 21.403;
user_time = 17*60 + 1.258;
sys_time = 7*60 + 1.161;

active_cpu_time = user_time + sys_time;
idle_time = real_time - active_cpu_time;
idle_percentage = (idle_time / real_time) * 100;

fprintf('CPU Idle Time Percentage: %.2f%%\n', idle_percentage);
```

```
Command Window
CPU Idle Time Percentage: 99.17%
```

ΚΟΥΝΣΟΛΑΣ ΧΡΗΣΤΟΣ

10345

https://github.com/kounsolas/realtime_systems.git