# Ενσωματωμένα Συστήματα Πραγματικού Χρόνου

Εργασία 2η: Producer Consumer Implementation with Timer

# Author

Όνομα: Πορτοκαλίδης Σταύρος, ΑΕΜ: 9334, email: [stavport@ece.auth.gr](mailto:stavport@ece.auth.gr)

Code: <https://github.com/sportokalidis/producer-consumer-pi>

# Περιγραφή Προβλήματος

Σκοπός της 2ης εργασίας είναι η υλοποίηση μιας ουράς εξυπηρέτησης, με χρήση πολλαπλών threads στο ρόλο των producers και consumers όπως είχε γίνει και στα πλαίσια της πρώτης εργασίας. Η διαφορά έγκειται στο γεγονός ότι στην 2η εργασία καλούμαστε να υλοποιήσουμε ένα timer ο οποίος με μια συγκεκριμένη περίοδο, θα καλείται και θα προσθέτει διεργασίες στην ουρά. Τέλος, καλούμαστε να αντιμετωπίσουμε και το πρόβλημα της χρονικής μετατόπισης (time drifting), ώστε να πραγματοποιείται εισαγωγή των διεργασιών σύμφωνα με την περίοδο που έχει οριστεί.

# Εξήγηση Υλοποίησης

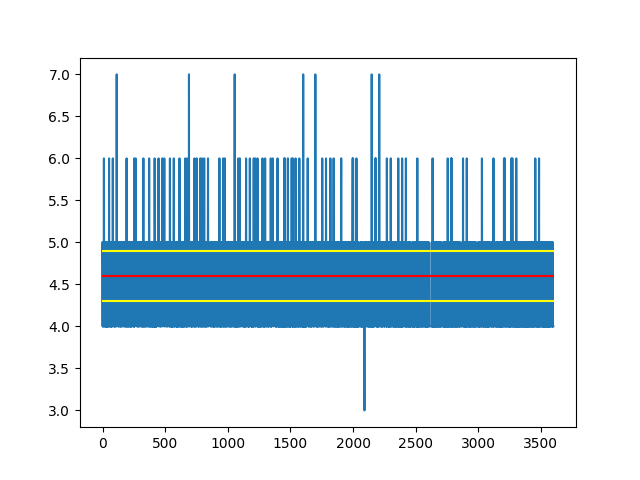
Για την υλοποίηση του timer δημιουργούμε ένα struct με όνομα Timer το οποίο περιέχει όλες τις απαραίτητες λειτουργίες όπως ζητείται από την εκφώνηση (period, taskToExecute, StartDelay, κτλ). Αρχικά, στην main() function θεωρώντας ως δεδομένα τον χρόνο εκτέλεσης και την περίοδο του Timer υπολογίζουμε τον αριθμό των task που θα κληθούν να εξυπηρετήσουν οι consumers. Στην συνέχεια, δημιουργούμε όλα τα consumers threads πρώτα (θα περιμένουν στο wait, εφόσον οι ούρα είναι άδεια), και έπειτα δημιουργούμε ένα αντικείμενο timer, στο οποίο αφού περάσουμε όλα τα απαραίτητα δεδομένα, μέσω της functions start(timer T), δημιουργούμε όλα τα producer threads. H ρουτίνα που τρέχουν τα threads consumers είναι πανομοιότυπη με αυτήν της 1ης εργασίας, ωστόσο η ρουτίνα που εκτελούν τα producer threads έχει ορισμένες διαφοροποιήσεις για να προσαρμοστούμε στο ζητούμενο της 2ης εργασίας. Η κύρια διαφοροποίηση που υπάρχει είναι ότι τα threads στο τέλος της ρουτίνας κάνουν ένα sleep(). Ο χρόνος τον οποίο κάνει sleep το thread producer δεν είναι σταθερός. Ουσιαστικά, μετράμε τον χρόνο από την στιγμή που ξύπνησε από το προηγούμενο sleep, μέχρι και την στιγμή που ξανά έκανε insert στην ουρά (drift time). Έτσι, για να διατηρήσουμε σταθερή την περίοδο που μπαίνουν οι διεργασίες στην ουρά, βάζουμε τον producer να κάνει sleep για *period – drift time*.

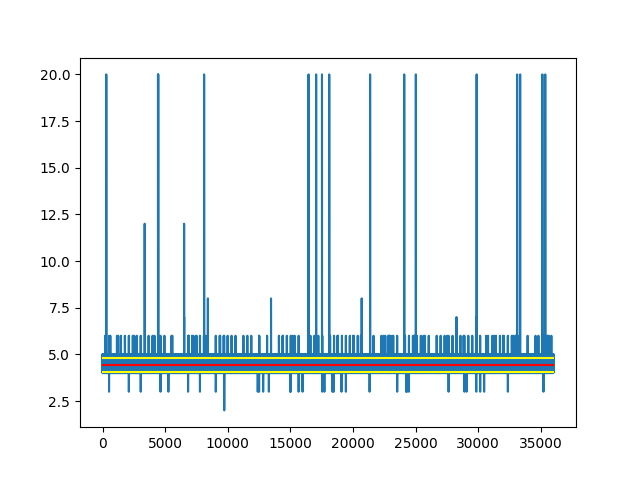
# Testing

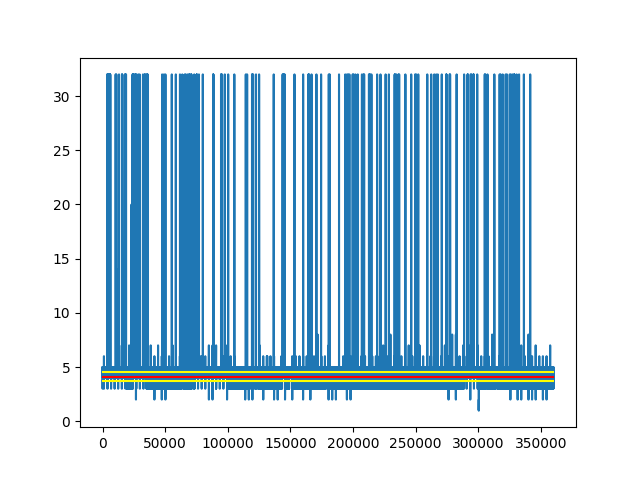
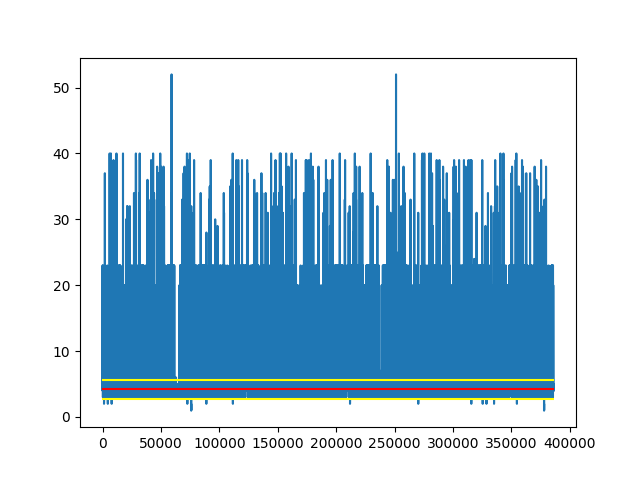
Ως επαλήθευση για την ορθότητα της υλοποίησης έχουμε το γεγονός ότι το πρόγραμμα τερματίζει (δηλαδή κάνουν return όλα τα threads) και οι ουρά είναι empty. Αυτό σε συνδυασμό ότι ο αριθμός των αποτελεσμάτων είναι ο αναμενόμενος και ότι εκτέλεση διαρκεί για την αναμενόμενη χρονική διάρκεια (μια ώρα), μας υποδηλώνει ότι εξυπηρετούνται όλες οι διεργασίες και με τον σωστό ρυθμό εισαγωγής στην ουρά.

# Μετρήσεις και Αποτελέσματα

* Drift Time

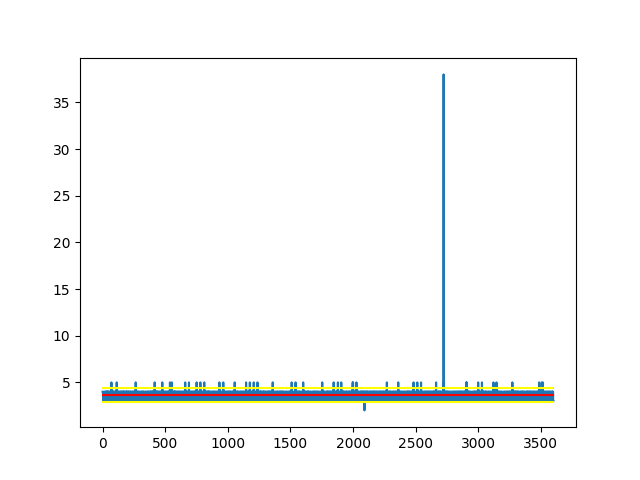
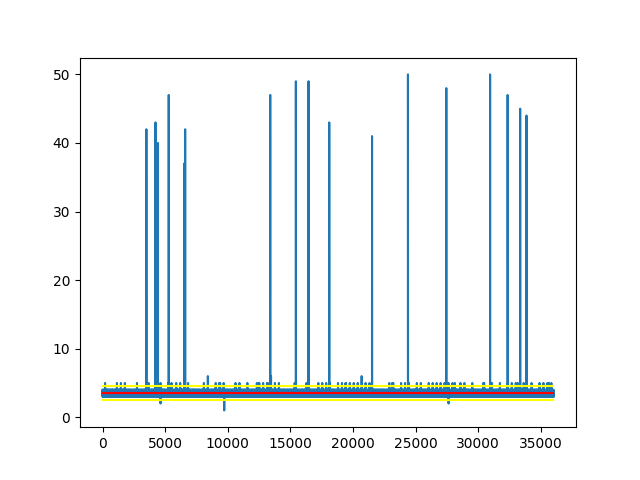


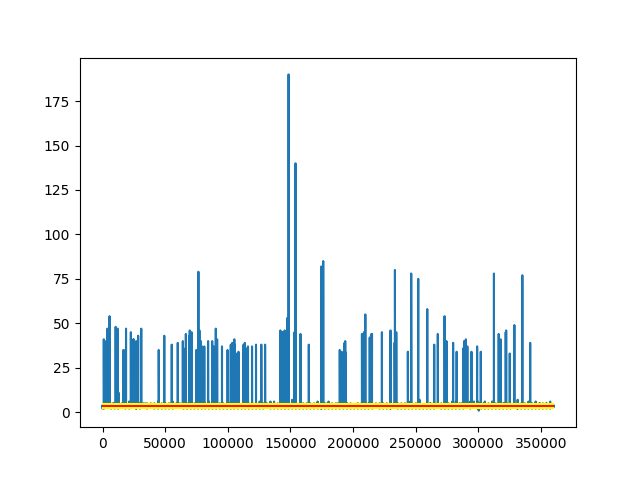
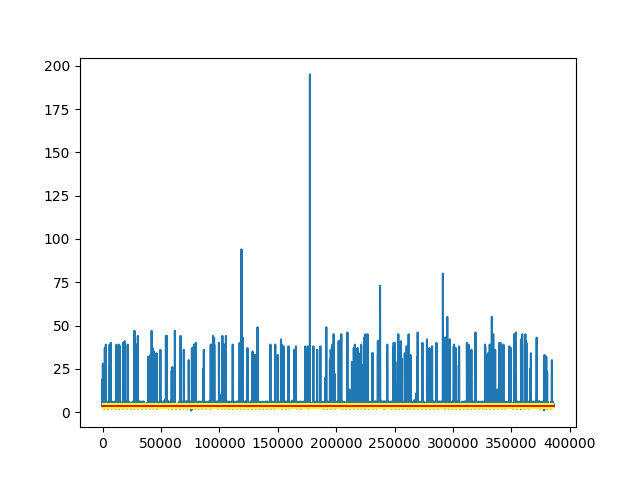




|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Period | Mean | Median | Min | Max | Std |
| 1 | 4.6 | 5 | 3 | 7 | 0.29 |
| 0.1 | 4.4 | 4 | 2 | 20 | 0.36 |
| 0.01 | 4.11 | 4 | 1 | 32 | 0.43 |
| All | 4.20 | 4 | 1 | 52 | 1.48 |

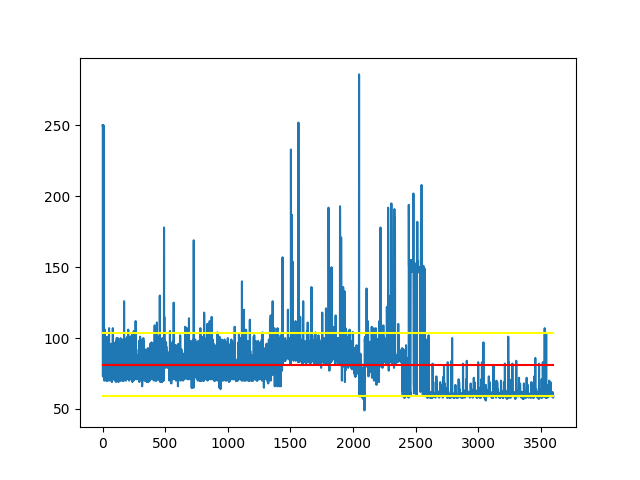
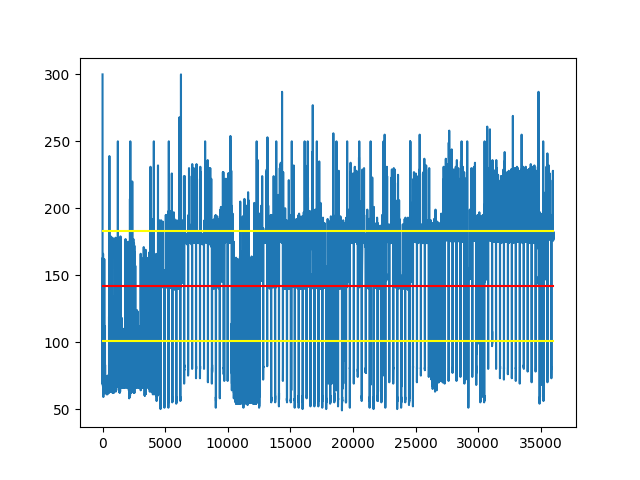
* Producer Waiting Time

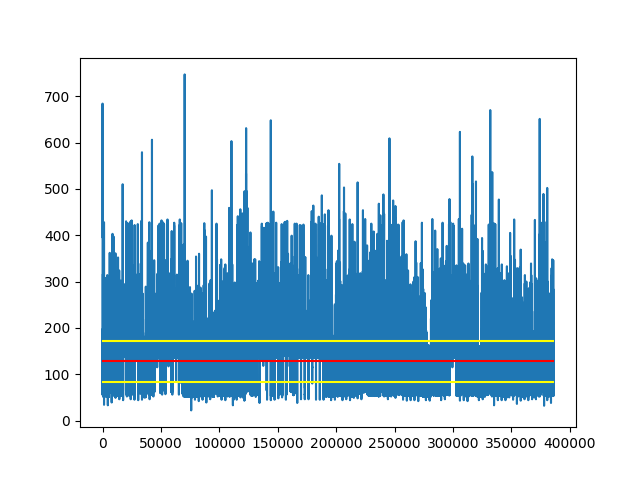


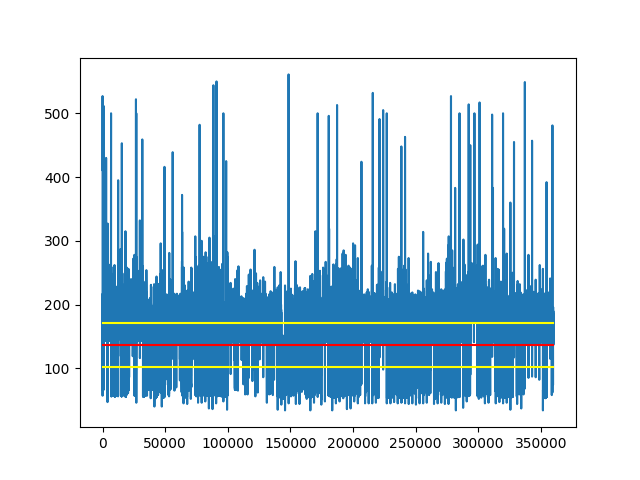


|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Period | Mean | Median | Min | Max | Std |
| 1 | 81.3 | 81 | 49 | 289 | 22.5 |
| 0.1 | 141.7 | 147 | 49 | 300 | 41.1 |
| 0.01 | 136.9 | 144 | 34 | 561 | 34.1 |
| All | 128.1 | 144 | 22 | 747 | 44.2 |

* Time in Queue







|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Period | Mean | Median | Min | Max | Std |
| 1 | 3.6 | 4 | 2 | 38 | 0.76 |
| 0.1 | 3.5 | 4 | 1 | 50 | 1.03 |
| 0.01 | 3.5 | 4 | 1 | 190 | 0.98 |
| All | 1 | 4 | 1 | 195 | 0.99 |

* CPU Usage

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | User Time | System Time | Real Time | CPU Usage |
| T=1 | 0m0.736s | 0m2.432s | 60m2.325s |  |
| T=0.1 | 0m3.567s | 0m3.785s | 60m9.314s |  |
| T=0.01 | 0m20.147s | 0m40.457s | 60m12.156s |  |
| All | 0m22.892s | 0m42.342s | 60m14.843s |  |

# Σχολιασμός Αποτελεσμάτων