Ανάπτυξη Λογισμικού Πληροφορικής
Χειμερινό Εξάμηνο 2014 - 2015

“Σύστημα Ανάλυσης Κοινωνικών δικτύων” part 3

Ημερομηνία παράδοσης: 08/07/2014

ΠΕΡΙΕΧΟΜΕΝΑ

Γενική περιγραφή

Σε αυτό το επίπεδο της εργασίας θα υλοποιήσετε δυο από τους πιο γνωστούς αλγορίθμους που χρησιμοποιούνται για να βρίσκουν κοινότητες (communities) σε γράφους.
Στα κοινωνικά δίκτυα σχηματίζονται κοινότητες με φυσικό τρόπο. Εύρεση κοινοτήτων σε ένα δίκτυο σημαίνει ουσιαστικά να χωρίσεις το δίκτυο σου σε υποομάδες. Δεν υπάρχει μία μόνο σωστή απάντηση στο πως χωρίζεται ένα δίκτυο σε ομάδες. Ακόμα και ο ορισμός του τι σημαίνει “σωστό” είναι σχετικός.
Μελετώντας τις κοινότητες ενός δικτύου είναι ένας τρόπος για να καταλάβουμε καλύτερα τον τρόπο που λειτουργεί το δίκτυο. Μπορούμε να κοιτάξουμε για συγκεκριμένες δομές και μοτίβα ή να ανακαλύψουμε τα “φυσικά” όρια κάθε κοινότητας, χωρίς να βάλουμε δικούς μας τεχνητούς περιορισμούς.

Περιγραφή παραδοτέων τρίτου επιπέδου

Σε αυτό το κομμάτι της εργασίας θα πρέπει να ανακαλύψετε communities που έχουν δημιουργηθεί σε forums του κοινωνικού μας δικτύου. Θα περιορίσετε την αναζήτηση στα top-N forums. Για να ταξινομήσετε τα forum θα χρησιμοποιήσετε ως κριτήριο το πλήθος των μελών του.

Κοινότητες (communities) σε κοινωνικά δίκτυα

Προεργασία

Πριν τη εφαρμογή των αλορίθμων για την εύρεση των communities θα χρειαστεί για κάθε forum από τα top-N να δημιουργήσετε το γράφο (g_f) με τα μέλη του και τις σχέσεις γνωριμίας τους.
Το κομμάτι της προεργασίας θα πρέπει να γίνει παράλληλα με τη χρήση νημάτων. Στο πρώτο στάδιο θα υπολογίσετε για κάθε forum το μέγεθος του και θα αποφασίσετε ποια ειναι τα top-N. Στο δεύτερο στάδιο θα δημιουργήσετε ένα g_f γράφο για κάθε ένα από τα top-N forums.
Μετά την παρουσίαση των αλγορίθμων θα βρείτε περισσότερες πληροφορίες για τα threads και για τη βιβλιοθήκη που θα χρησιμοποιήσετε.

Παρακάτω υπάρχουν οι περιγραφές των 2 αλγορίθμων που θα υλοποιήσετε. Και οι δυο αλγόριθμοι δέχονται ως είσοδο ένα γράφο και επιστρέφουν τα communities του.

Αλγόριθμος Clique percolation method (CPM)

Ο αλγόριθμος αυτός ανήκει στην κατηγορία των αλγορίθμων εύρεσης κοινοτήτων που σχετίζονται με τις ιδιότητες των κόμβων (Node-Centric Community Detection). Ο αλγόριθμος χρησιμοποιεί τις κλίκες του γράφου ως ενδιάμεσο βήμα για να βρει τις κοινότητες.

Κλίκα : είναι ένας γράφος που για κάθε ζευγάρι κόμβων του υπάρχει μια ακμή που να τους συνδέει (για μη κατευθυνόμενους γράφους).
k*-κλίκα* : είναι ο υπογράφος μεγέθους k που είναι κλίκα.

Ο αλγόριθμος παίρνει ως παράμετρο τη μεταβλητή k που αντιστοιχεί στο μέγεθος των κλικών που θα χρειαστεί να υπολογίσουμε.

Αλγόριθμος CPM
Βήμα 1. Εύρεση όλων των k-κλικών στον αρχικό γράφο.
Βήμα 2. Δημιουργία νέου γράφου με υπερ-κόμβους που αντιστοιχούν στις k-κλίκες του Βήματος 1.

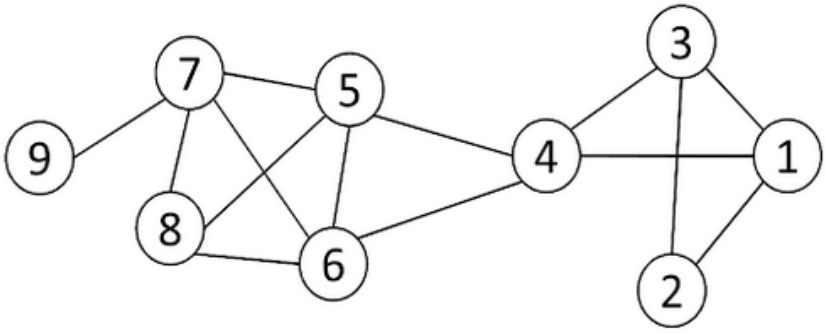
Βήμα 3. Εισαγωγή ακμών στο νέο γράφο. Μια ακμή δημιουργείται μεταξύ δύο υπερ-κόμβων εφόσον έχουν k-1 κοινούς κόμβους.
Βήμα 4. Εύρεση των συνεκτικών γραφημάτων του νέου γράφου. Κάθε συνεκτικό γράφημα αποτελεί μια κοινότητα.

- *Communities* cliquePercolationMethod(int k, Graph* g);*

Το Communities είναι μια δομή που περιέχει για κάθε community το id του (που δημιουργήσατε εσεις) και τα ids των χρηστών που περιέχει.

Παράδειγμα

Σας δίνετε ως είσοδος ο παρακάτω γράφος με παράμετρο k=3.

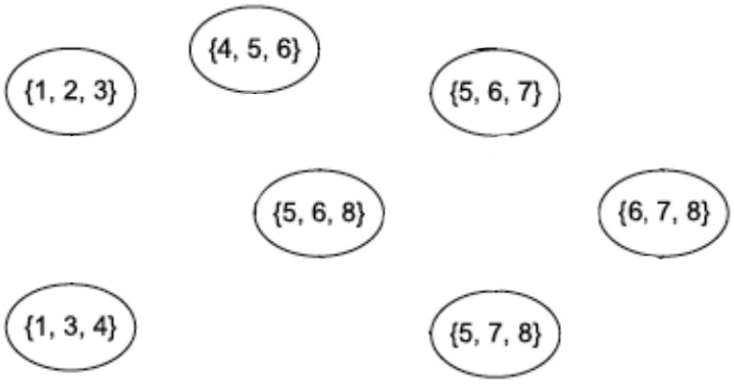


Σχήμα 1.

Βήμα 1: Υπολογισμός όλων των 3-κλικών:

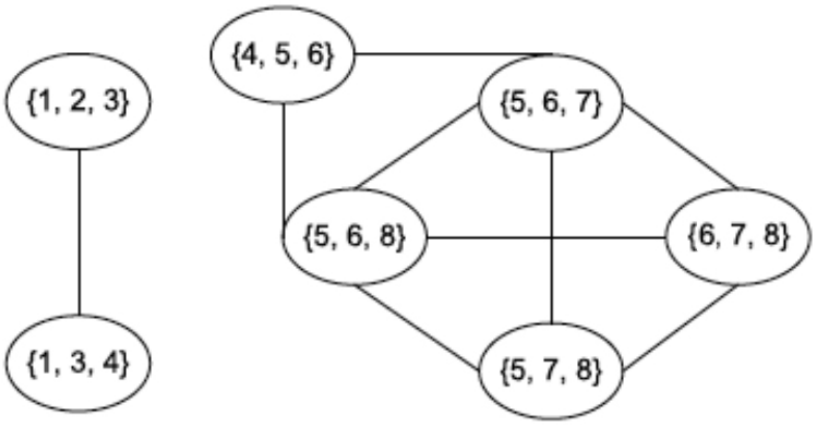
- (1,2,3)
- (1,3,4)
- (4,5,6)
- (5,6,7)
- (5,6,8)
- (5,7,8)
- (6,7,8)

Βήμα 2: Δημιουργία υπερ-κόμβων.



Σχήμα 2.

Βήμα 3: Υπολογισμός των k-1 κοινών κόμβων ανά ζευγάρι για τη δημιουργία των ακμών.



Σχήμα 3.

Βήμα 4: Υπολογισμός των συνεκτικών γραφημάτων με αποτέλεσμα τα communities του αρχικού γράφου:

- (1,2,3), (1,3,4)
- (4,5,6), (5,6,7), (5,7,8), (6,7,8), (5,6,8)

Αλγόριθμος Girvan–Newman (με κριτήριο τερματισμού)

Ο αλγόριθμος ανήκει στην κατηγορία των ιεραρχικών αλγορίθμων (Divisive Hierarchical Clustering) και ανακαλύπτει κοινότητες αφαιρώντας σταδιακά ακμές από το δίκτυο. Χρησιμοποιεί το μέτρο "*edge betweenness*" για να βρει ποιες ακμές του γράφου είναι πιο κεντρικές και άρα πιο πιθανό να βρίσκονται μεταξύ δύο κοινοτήτων.

Edge betweenness μιας ακμής: διαισθητικά ορίζεται ως ο αριθμός των κοντινότερων μονοπατιών στα οποία συμμετέχει η ακμή αυτή. Έστω V το σύνολο των κόμβων του γράφου. Έστω $s,t \in V$ και e μια ακμή του γράφου. Τότε η τιμή της edge betweenness για τη συγκεκριμένη ακμή δίνεται από τον παρακάτω τύπο.

$$edgeBetweenness(e) = \sum_{s<t} \frac{\sigma_{st}(e)}{\sigma_{s,t}}$$

Για να τερματίσει ο αλγόριθμος χρησιμοποιεί την έννοια του *modularity*. Όσο μεγαλύτερο είναι το *modularity*, τόσο "καλύτερες" είναι οι κοινότητες που έχουν βρεθεί.

Τύπος modularity:

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(C_i, C_j)$$

Όπου m = ο αριθμός των ακμών σε μη κατευθυνόμενο γράφο

A_{ij} = είναι 1 αν οι κόμβοι i και j είναι γειτονικοί, αλλιώς 0.

$\delta(.)$ = η συνάρτηση δίνει αποτέλεσμα 1 αν οι δύο κόμβοι i και j ανήκουν στην ίδια κοινότητα, και 0 διαφορετικά.

k_i, k_j = βαθμός των κόμβων i και j αντίστοιχα.

C_i, C_j = η κοινότητα του κόμβου i και j αντίστοιχα.

Αλγόριθμος Girvan–Newman

Βήμα 1. Υπολογίστε το edge-betweenness centrality για όλες τις ακμές του γράφου

Βήμα 2. Βρείτε και αφαιρέστε την ακμή με το μεγαλύτερο edge-betweenness centrality.

Βήμα 3. Ξανα-υπολογίστε το edge-betweenness για τις υπολειπόμενες ακμές του γράφου.

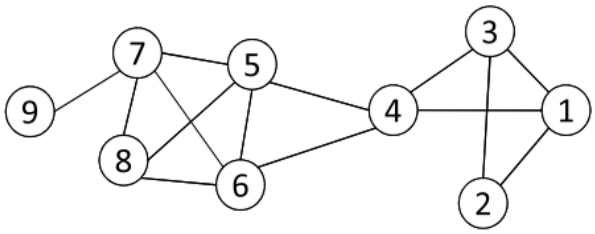
Βήμα 4. Υπολογίστε το modularity του γράφου και αν δεν ισχύει το κριτήριο τερματισμού, επιστρέψτε στο βήμα 2.

Κριτήριο τερματισμού: όταν το modularity γίνει μεγαλύτερο από κάποια τιμή X ή πάρει τη μεγαλύτερη του τιμή.

- *Communities* GirvanNewman(double modularity, Graph* g);*

Παράδειγμα

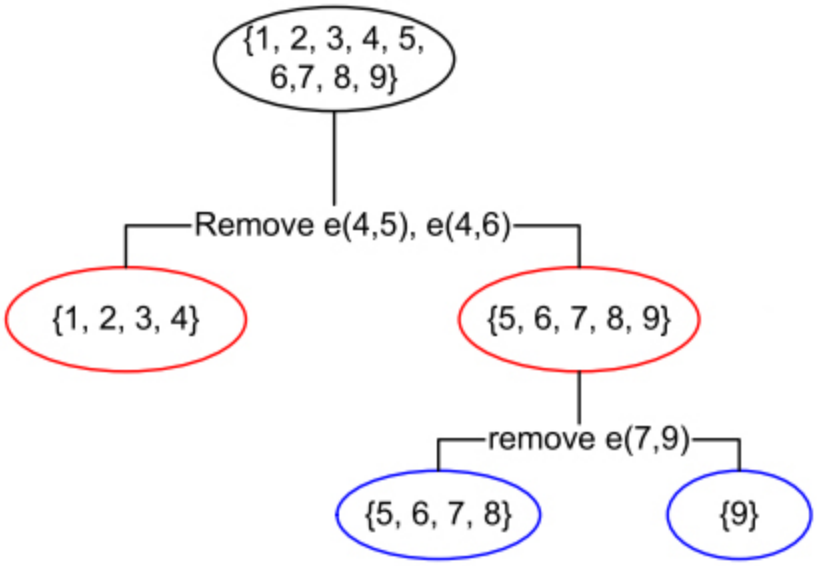
Σας δίνετε το παρακάτω δίκτυο. Οι αρχικές τιμές edge betweenness έχουν υπολογιστεί και δίνονται στον πίνακα που φαίνεται δίπλα στο γράφημα.



	1	2	3	4	5	6	7	8	9
1	0	4	1	9	0	0	0	0	0
2	4	0	4	0	0	0	0	0	0
3	1	4	0	9	0	0	0	0	0
4	9	0	9	0	10	10	0	0	0
5	0	0	0	10	0	1	6	3	0
6	0	0	0	10	1	0	6	3	0
7	0	0	0	0	6	6	0	2	8
8	0	0	0	0	3	3	2	0	0
9	0	0	0	0	0	0	8	0	0

Σχήμα 4.

Στο πρώτο βήμα θα αφαιρεθεί η ακμή $e(4,5)$ που έχει το μεγαλύτερο edge betweenness. Επειδή το modularity δεν έχει φτάσει στο όριο του κατωφλίου ο αλγόριθμος θα ξανα-υπολογίσει τα edge betweenness και θα συνεχίσει. Η επαναληπτική διαδικασία μέχρι τον τερματισμό φαίνεται στο δενδροδιάγραμμα που ακολουθεί.



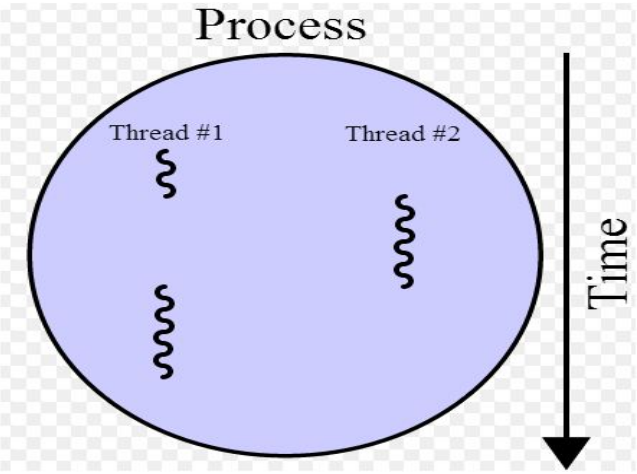
Σχήμα 5.

Βελτιστοποιήσεις

Σε όποια σημεία παρατηρήσετε ότι μπορούν να γίνουν αλλαγές που να βελτιστοποιήσουν την απόδοση των αλγορίθμων (είτε με τη χρήση δομών, είτε με τη χρήση νημάτων), μπορείτε να τις υλοποίησετε.

Πολυνηματισμός

Ένα νήμα είναι μια ελαφριά διεργασία. Η υλοποίηση των νημάτων και των διεργασιών διαφέρει από το ένα λειτουργικό σύστημα στο άλλο. Σε κάθε διεργασία υπάρχει τουλάχιστον ένα νήμα. Αν μέσα σε μια διεργασία υπάρχουν πολλαπλά νήματα, τότε αυτά διαμοιράζοντε την ίδια μνήμη και πόρους αρχείων.



Σχήμα 6.

Ένα νήμα διαφέρει από μια διεργασία ενός πολυεπεξεργαστικού λειτουργικού συστήματος στα εξής:

- οι διεργασίες είναι τυπικώς ανεξάρτητες, ενώ τα νήματα αποτελούν υποσύνολα μιας διεργασίας.
- οι διεργασίες περιέχουν σημαντικά περισσότερες πληροφορίες κατάστασης από τα νήματα, ενώ πολλαπλά νήματα μιας διεργασίας μοιράζονται την κατάσταση της διεργασίας, όπως επίσης μνήμη και άλλους πόρους.
- οι διεργασίες έχουν ξεχωριστούς χώρους διευθυνσιοδότησης, ενώ τα νήματα μοιράζονται το σύνολο του χώρου διευθύνσεων που τους παραχωρείται.
- η εναλλαγή ανάμεσα στα νήματα μιας διεργασίας είναι πολύ γρηγορότερη από την εναλλαγή ανάμεσα σε διαφορετικές διεργασίες.

Η πολυνημάτωση αποτελεί ένα ευρέως διαδεδομένο μοντέλο προγραμματισμού και εκτέλεσης διεργασιών το οποίο επιτρέπει την ύπαρξη πολλών νημάτων μέσα στα πλαίσια μιας και μόνο διεργασίας. Τα νήματα αυτά μοιράζονται τους πόρους της διεργασίας και μπορούν να εκτελούνται ανεξάρτητα. Το γεγονός ότι επιτρέπει μια διεργασία να εκτελείται παράλληλα σε ένα σύστημα με πολλαπλούς πυρήνες αποτελεί ίσως την πιο ενδιαφέρουσα εφαρμογή της συγκεκριμένης τεχνολογίας.

Το πλεονέκτημα ενός πολυνηματικού προγράμματος είναι ότι του επιτρέπει να εκτελείται γρηγορότερα σε υπολογιστικά συστήματα που έχουν πολλούς επεξεργαστές, επεξεργαστές με πολλούς πυρήνες, ή κατά μήκος μιας συστοιχίας υπολογιστών. Για να μπορούν να χειραγωγηθούν σωστά τα δεδομένα, τα νήματα θα πρέπει ορισμένες φορές να συγκεντρωθούν σε ένα ορισμένο χρονικό διάστημα έτσι ώστε να επεξεργασθούν τα δεδομένα στη σωστή σειρά.

Ένα ακόμα πλεονέκτημα της πολυδιεργασίας (και κατ' επέκταση της πολυνημάτωσης) ακόμα και στους απλούς επεξεργαστές (με έναν πυρήνα επεξεργασίας) είναι η δυνατότητα για μια εφαρμογή να ανταποκρίνεται άμεσα. Έχοντας ένα πρόγραμμα που εκτελείται μόνο σε ένα νήμα, αυτό θα φαίνεται ότι κολλάει ή ότι παγώνει, στις περιπτώσεις που εκτελείται κάποια μεγάλη διεργασία που απαιτεί πολύ χρόνο. Αντίθετα σε ένα πολυνηματικό σύστημα, οι χρονοβόρες διεργασίες μπορούν να εκτελούνται παράλληλα με άλλα νήματα που φέρουν άλλες εντολές για το ίδιο πρόγραμμα, καθιστώντας το άμεσα ανταποκρίσιμο.

Για την υλοποίηση των πολυνηματικών λειτουργιών θα χρησιμοποήσετε τη Pthreads που είναι μια POSIX API C βιβλιοθήκη. Για τη χρήση της πρέπει να συμεριλαμβάνετε στα αρχεία του κώδικα σας το <pthread.h> και να χρησιμοποήσετε κατα το compilation το -pthread flag.

Οι βασικές ρουτίνες των PThreads είναι οι εξής:

- int **pthread_create**(pthread_t *thread, const pthread_attr_t *attr, void*(*start_routine)(void), void *arg)
- void **pthread_exit**(void *value_ptr)
- int **pthread_join**(pthread_t thread, void **value_ptr)

Ο στόχος της ρουτίνας ***pthread_create*** είναι να δημιουργήσει ένα νέο νήμα και αφού αρχικοποιήσει τα χαρακτηριστικά αυτού, το κάνει διαθέσιμο για χρήση. Στην μεταβλητή *thread* επιστρέφετε το αναγνωριστικό του νήματος που μόλις δημιουργήθηκε. Με βάση την τιμή στο πεδίο *attr* θα αρχικοποιηθούν τα χαρακτηριστικά του νέου νήματος. Στο όρισμα *start_routine*

ορίζεται η ρουτίνα που θα εκτελέσει το νέο thread όταν δημιουργηθεί και στο πεδίο *arg* θα ορισθούν οι παράμετροι αυτής.

Η ρουτίνα ***pthread_exit*** θα τερματίσει ένα ήδη υπάρχον νήμα και θα αποθηκεύσει την κατάσταση τερματισμού για όσα άλλα νήματα θα προσπαθήσουν να συνενωθούν με αυτό. Επιπρόσθετα ελευθερώνει όλα τα δεδομένα του νήματος, συμπεριλαμβανομένων και της στοίβα του νήματος. Είναι σημαντικό τα αντικείμενα συγχρονισμού νημάτων ,όπως τα mutexes και οι μεταβλητές κατάστασης (condition variables), που κατανέμονται στη stack του νήματος, να καταστραφούν πριν κλήση της ρουτίνας pthread_exit.

Η ρουτίνα ***pthread_join*** μπλοκάρει το τρέχον thread μέχρι να τερματίσει το thread που προσδιορίζεται από το πεδίο *thread*. Η κατάσταση τερματισμού του thread αυτού επιστρέφεται στο πεδίο *value_ptr*. Αν το συγκεκριμένο thread έχει ήδη τερματίσει (και δεν είχε προηγουμένως αποσπαθεί), το τρέχον thread δεν θα μπλοκαριστεί.

Υλοποίηση

Η χρήση των threads μπορεί να γίνει με δύο τρόπους είτε δημιουργώντας καινούργια threads για κάθε παράλληλο κομμάτι, είτε με την υλοποίηση ενός job scheduler [Παράρτημα 1]. Η δεύτερη επιλογή προσθέτει μια πολυπλοκότητα στην υλοποίηση αλλά διευκολύνει τη χρήση των threads στον υπόλοιπο κώδικα και κάνει την εύρεση σφαλμάτων πολύ πιο εύκολη. Η επιλογή του τρόπου που θα χρησιμοποιήσετε τα threads αφήνεται σε εσάς.

Απλή υλοποίηση

Στο πρώτο κομμάτι της προεργασίας θα δημιουργήσετε #forums threads και σε κάθε ένα θα αναθέσετε την καταμέτρηση των μελών ενός forum. Το main thread αφού φτιάξει τα thread θα περιμένει με χρήση της pthread_join να τελειώσουν όλα τα threads την εκτέλεση τους. Κάθε thread θα τοποθετεί σε μια κοινή δομή το αποτέλεσμα της καταμέτρησης. Στο τέλος της εκτέλεσης των threads ξέρουμε τα top-N forums.
Στο δεύτερο κομμάτι το main thread θα δημιουργήσει N threads και σε κάθε ένα θα αναθέσει ένα απο τα top-N forums. Κάθε thread θα δημιουργήσει ένα g_f και θα τον αποθηκεύσει σε μια κοινή δομή. Το main thread θα περιμένει μέχρι να τελειώσουν όλα τα threads.
Για την αποθήκευση των αποτελεσμάτων των threads θα πρέπει να δημιουργήσετε μια κοινή δομή, να την δίνετε ως όρισμα στα threads και να φροντίσετε για την ασφαλή προσπέλαση της.

Συγχρονισμός

Όταν δυο thread χρησιμοποιούν τις ίδιες δομές θα πρέπει να βρείτε ένα τρόπο να συγχρονίσετε την πρόσβαση σε αυτές, ώστε να εξασφαλίσετε συνέπεια στο αποτέλεσμα των ενεργειών που εκτελούν τα threads. Η POSIX παρέχει τους mutexes.
Οι mutexes έχουν δυο καταστάσεις locked, unlocked. Χρησιμοποιώντας ένα mutex ανά κοινή δομή, μπορούμε να περιορίσουμε την πρόσβαση σε αυτή σε ένα μόνο thread κάθε στιγμή.

Οι βασικές ρουτίνες των POSIX mutexes είναι οι εξής:

- int **pthread_mutex_init**(pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr)
- int **pthread_mutex_lock**(pthread_mutex_t *mutex)
- int **pthread_mutex_unlock**(pthread_mutex_t *mutex)
- int **pthread_mutex_destroy**(pthread_mutex_t *mutex)

Όσο κάποιο thread έχει κλειδωμένο ένα mutex τότε αν κάποιο άλλο thread προσπαθήσει να κλειδώσει τον ίδιο mutex, το δεύτερο thread θα “κολλήσει” μέχρι να ξεκλειδωθεί. Άρα αν η πρόσβαση στις κοινές δομές γίνεται ανάμεσα σε κλείδωμα και ξεκλείδωμα του αντίστοιχου mutex αποφεύγουμε τις ταυτόχρονες αλλαγές και τα λάθη που μπορεί να προκαλέσουν.

Παράρτημα 1

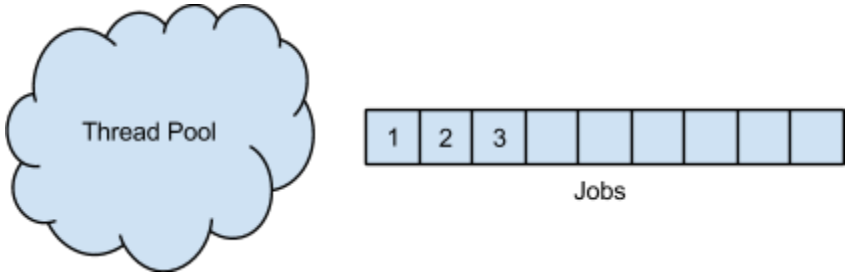
Job Scheduler

Εργασία (Jobs)

Εργασία (Job) είναι μια ρουτίνα κώδικα η οποία θέλουμε να εκτελεστεί από κάποιο thread πιθανότατα παράλληλα με κάποια άλλη. Μπορούμε να ορίσουμε οτιδήποτε θέλουμε ως job και να το αναθέσουμε στον χρονοπρογραμματιστή.

Χρονοπρογραμματιστής Εργασιών (Scheduler) και Δεξαμενή Νημάτων (Thread Pool)

Ο χρονοπρογραμματιστής ουσιαστικά δέχεται δουλειές και αναλαμβάνει την αναθέτηση τους σε νήματα, για προσωρινή αποθήκευση των εργασιών χρησιμοποιεί μια ουρά προτεραιότητας. Έστω ότι έχουμε μια Δεξαμενή νημάτων (thread pool) και μια συνεχόμενη ροή από ανεξάρτητες εργασίες (jobs). Όταν δημιουργείται μια εργασία, μπαίνει στην ουρά προτεραιότητας του χρονοπρογραμματιστή και περιμένει να εκτελεστεί. Οι εργασίες εκτελούνται με την σειρά που δημιουργήθηκαν (first-in-first-out - FIFO). Κάθε νήμα περιμένει στην ουρά μέχρι να του ανατεθεί μια εργασία και, αφού την εκτελέσει, επιστρέφει στην ουρά για να αναλάβει νέα εργασία. Για την ορθή λειτουργία ενός χρονοπρογραμματιστή είναι απαραίτητη η χρήση σημαφόρων στην ουρά, ώστε να μπλοκάρουν εκεί τα νήματα, και κάποια κρίσιμη περιοχή (critical section), ώστε να γίνεται σωστά εισαγωγή και απολαβή εργασιών από την ουρά.



Σχήμα 7.

Condition Variables

΄Ενα ακομά εργαλείο συγχρονισμού που μπορεί να σας φανεί χρήσιμο ειναι τα condition variables. Ένα condition variable είναι ένα εργαλείο που επιτρέπει στα POSIX threads να αναβάλουν την εκτέλεση τους μέχρι μια έκφραση να γίνει αληθής. Δύο ειναι οι βασικές πράξεις πάνω σε ένα condition variable, wait που αδρανοποιεί το thread που την κάλεσε και η signal που ξυπνά ένα από τα thread που είναι απενεργοποιημένα πάνω στο ίδιο condition variable. Ένα condition variable χρησιμοποιείται ζευγάρι με ένα mutex.

Οι βασικές ρουτίνες των POSIX condition variable είναι οι εξής:

- int **pthread_cond_init**(pthread cond)t *cond, pthread_condattr_t *cond attr)
- int **pthread_cond_wait**(pthread_cond_t *cond, pthread_mutex_t *mutex)
- int **pthread_cond_signal**(pthread_cond_t *cond)
- int **pthread_cond_broadcast**(pthread_cond_t *cond)
- int **pthread_cond_destroy**(pthread_cond_t *cond)

Οι condition varibles χρησιμοποιούνται μέσα σε while loops:

```
mutex_lock(mtx1)
while( canRun ){
        cond_var_wait(condvar1, mtx1)
}
//do some work
cond_var_signal(condvar1)
mutex_unlock(mtx1)
```

Πριν τη χρήση του condvar1 κλειδώνουμε το mtx1. Μετά ελέγχουμε σε μια while μια συνθήκη που αν αποτιμηθεί ως αληθής σημαίνει ότι το thread δεν μπορεί να κάνει τη δουλειά του και

πρέπει να αδρανοποιηθεί με τη χρήση της cond_var_wait. Αλλιώς το thread δεν εισέρχεται στο εσωτερικό της while και εκτελεί τη δουλειά του. Χρησιμοποιεί τη signal για να ξυπνήσει ένα ακόμα thread, αν υπάρχει, που είναι αδρανοποιημένο στο ίδιο condition variable και ξεκλειδώνει το mtx1.