ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
Τμήμα Πληροφορικής και Τηλεπικοινωνιών
3η Εργασία - Τμήμα: Άρτιων Αριθμών Μητρώου
K24: Systems Programming – Εαρινό Εξάμηνο ‘20

Κωστόπουλος Σταύρος
Α.Μ.: 1115201700068

Η υλοποίηση του προγράμματος της 2ης Εργασίας, προκύπτει από την σύνθεση δεκατριών (13) πηγαίων αρχείων της μορφής .c (Το πρόγραμμα αναπτύχθηκε στην γλώσσα προγραμματισμού C).Τα δεκατρία αυτά αρχεία είναι τα παρακάτω:

- master.c
- functions.c
- hashtable.c
- kouvades.c
- queries.c
- tree.c
- topk.c
- worker.c
- dateandlist.c
- whoClient.c
- whoServer.c
- whoserverfunctions.c
- whoclientfunctions.c

Θα εξηγήσω την υλοποίηση της εργασίας μου,σε ένα περιγραφικό χρονοδιάγραμμα.

Αρχικά, κάνουμε compile τα πηγαία αρχεία:

$ make

Ύστερα γίνεται εκκίνηση του server,με την κλήση του εκτελέσιμου αρχείου whoServer,όπως φαίνεται παρακάτω:

$ ./whoServer -q 2000 -s 4080 -w 2 -b 5

Ο whoServer μόλις ξεκινήσει, δημιουργεί numThreads threads,τα οποία περιμένουν στοιχεία να αποθηκευτούν στο circular buffer(aka. pool),για να τα κάνουν obtain και να τα διαχειριστούν αναλόγως.Ύστερα ο whoServer κάνει bind στο queryPortNum και στο statisticsPortNum,και μπαίνει σε loop.Στο loop αυτό,μέσω της συνάρτησης

network_accept_any,ο whoServer περιμένει αιτήματα για να κάνει accept() σε οποιοδήποτε από τα δύο ports(με την χρήση του select() system call) .Όταν ο whoServer δεχθεί μία σύνδεση,τοποθετεί το file descriptor που επέστρεψε η accept στο circular buffer,για να το χρησιμοποιήσει το thread που θα το αναλάβει.

1. Για τη συνάρτηση network_accept_any, κατανόησα και εμπιστεύθηκα δημόσια πηγή απο τον παγκόσμιο ιστό:
https://stackoverflow.com/questions/15560336/listen-to-multiple-ports-from-one-server/15560580
2. Η πηγή που εμπιστεύθηκα για την υλοποίηση του Circular Buffer είναι η διαφάνειες απο την ιστοσελίδα του μαθήματος(lecture13))

Ενώ ο whoServer περιμένει να δεχθεί αιτήματα για συνδέσεις, και βρίσκεται σε αναμονή,γίνεται εκκίνηση του προγράμματος master,με την παρακάτω εντολή:

$ valgrind --trace-children=yes ./master -w 4 -b 8 -s 195.134.65.85 -p 4080 -i Database

(Είναι σημαντικό να γίνει χρήση της valgrind γιατί χωρίς αυτή είναι πιθανό -όχι σίγουρο- ένας από τους workers να σταματήσει να δουλεύει.Προσπάθησα να το λύσω,αλλά από την στιγμή που με την χρήση της valgrind δουλεύει και το master πρόγραμμα καλύπτει μόνο το 10% της βαθμολογίας,δεν επέμεινα)

Το πρόγραμμα master δημιουργεί τους workers,οι οποίοι (όπως και στην δεύτερη εργασία) μέσω named pipes διαβάζουν τα ονόματα των φακέλων που τους ανατίθενται και το ServerIP και ServerPort.Έπειτα αφού κατασκευάσουν τα summary stats,επιχειρούν μέσω του serverIP και serverPort να κάνουν connect() στον whoServer.Μόλις το καταφέρει ένας worker αυτό,στέλνει τα summary stats στον whoServer μαζί με κάποιο τυχαίο διαθέσιμο port number στο οποίο θα ακούει ο συγκεκριμένος worker.Ύστερα,περιμένει αίτημα για να κάνει accept() μια σύνδεση στο συγκεκριμένο Port.

Την ίδια στιγμή τα numThreads threads του whoServer,έχουν διαβάσει τα summary statistics,τα εκτυπώνουν,και παίρνοντας το worker port number και χρησιμοποιώντας την IP address του Worker (από το δεύτερο όρισμα της accept() μέσα στην network_accept_any συνάρτηση) επιχειρούν να κάνουν connect() με τον worker.Μόλις κάνουν connect,με την χρήση mutex για να μην προκύψουν race conditions τοποθετούν το file descriptor της σύνδεσης αυτής σε μία λίστα από workerfd_list structs,που το κάθε Node περιέγχει: την τιμή του file descriptor,έναν δείκτη στο επόμενο node,και ένα pthread_mutex για να εξασφαλίζεται η ασφαλής χρήση του συγκεκριμένου file descriptor,και η παραλληλία.

Αφού έγιναν όλα αυτά,είμαστε έτοιμοι να κάνουμε εκκίνηση του προγράμματος whoClient,με την παρακάτω εντολή:

$ ./whoClient -q queryfile.txt -w 3 -sp 2000 -sip 195.134.65.85

Το πρόγραμμα whoClient μόλις ξεκινήσει,δημιουργεί numThreads threads για να εξυπηρετήσει numThreads queries από το query file.Αφού δημιουργήθουν τα Threads(όπου

το καθένα έχει αναλάβει από ενα query)περιμένουν,και όταν δημιουργηθούν όλα τα numThreads threads,προσπαθούν όλα μαζί αν συνδεθούν στον whoServer.Ο whoServer με την accept,αποθηκεύει τα whoClient file descriptors στο Circular Buffer.Το κάθε thread του whoServer αναλαμβάνει από ένα whoClient file descriptor κάνοντας obtain από το Circular Buffer.Μέσω αυτού του whoClient file descriptor,διαβάζει ένα query και χρησιμοποιώντας τα mutex που έχει κάθε node του workerfd_list στέλνει μόνο αυτός σε έναν συγκεκριμένο worker,αναμένοντας την απάντηση του.΄Οταν διαβάσει την απάντηση του worker,περιμένει άδεια για να χρησιμοποιήσει τον επόμενο,και οποιοδήποτε άλλο thread μπορεί(ελεγχόμενα και αυτό)να γράψει ενα query σε αυτόν τον worker που μόλις απάντησε.Το κάθε thread με αυτή τη διαδικασία συλλέγει τις απαντήσεις από τους workers και διαμορφώνοντας μια τελική απάντηση(όπως έκανε ο diseaseAggregator στην δεύτερη εργασία) την εκτυπώνει μαζί με το query που είχε δεχθεί και στέλνει την απάντηση στο thread του whoClient που είχε στείλει το query ,και αυτό ελεγχόμενα επίσης τυπώνει το query με τα αποτελέσματα του.(Τα threads του whoServer που εκπλήρωσαν κάποιο query,προσπαθούν πάλι να κάνουν obtain κάποιο ελεύθερο στοιχείο από το Circular Buffer)

Ο whoClient επαναλαμβάνει αυτή τη διαδικασία με την δημιουργία των threads,μέχρι να εκπληρώσει όλα τα ερωτήματα του query file.

Ο whoServer και οι workers,τυπικά,περιμένουν πάντα για κάποιο query ,για αυτό το λόγο τερματίζουν μόνο με SIGINT signal (Ctrl-C) (^C).Ο whoClient μόλις πάρει απαντήσεις για όλα τα ερωτήματα του queryfile,απελευθερώνει την δεσμευμένη μνήμη και τερματίζει.