Κωνσταντίνος Τσιάρας AM: 1115201500165

Προγραμματισμός Συστήματος

Εργασία 3η

Δομή της εργασίας:

Εκτός από το παρών README, ο παραδοτέος φάκελος περιέχει το *webcreator.sh*, ένα Makefile και τους υποφακέλους *common*, *jobExecutor*, *mycrawler* και *myhttp* ο καθένας εκ των οποίων περιέχει αρχεία .c και .h. Εκτός από τον *common*, οι υπόλοιποι φάκελοι περιέχουν επίσης ένα Makefile έκαστος το οποίο μεταγλωττίζει το ομώνυμο πρόγραμμα. Το Makefile που βρίσκεται στο ανώτερο επίπεδο περιλαμβάνει target "*all*" καθώς και targets για τα επιμέρους προγράμματα, καλώντας εσωτερικά τα Makefiles αυτών. Τέλος, σημειώνεται ότι το Makefile του *mycrawler* επίσης καλεί αυτό του jobExecutor καθώς αυτός απαιτείται για τη συνολική λειτουργία του πρώτου.
Η κλήση τόσο του *myhttpd* όσο και του *mycrawler* δύναται να γίνει είτε από τον φάκελο του καθενός αντίστοιχα είτε, σε ανώτερο επίπεδο, απ' τον φάκελο του παραδοτέου.

a) Web site creator

- Οριακές τιμές *w* και *p*: Προκειμένου να είναι δυνατή η ύπαρξη *f* μοναδικών εσωτερικών links και *q* μοναδικών εξωτερικών links όπως περιγράφει η εκφώνηση, ο webcreator απαιτεί *w>1* και *p>2*. Οι έλεγχοι αυτοί γίνονται με κατάλληλα regular expressions.
- Για παραγωγή *k* τυχαίων αριθμών στο διάστημα [*a*,*b*] γίνεται χρήση της εντολής Linux "*shuf -i a-b -n k*".
- Τα ονόματα όλων των sites κατασκευάζονται πριν την κατασκευή των ίδιων των sites και αποθηκεύονται στη μεταβλητή *sitelist*. Με αλλεπάλληλη χρήση εντολών *grep* και *shuf* στη «λίστα» αυτή προκύπτουν για κάθε site τα links τα οποία πρέπει να περιέχει.
- Τα παραγόμενα links είναι root-relative (δηλαδή της μορφής "*../siteY/pageZ_XXXX.html*").
- Κάθε φορά που δημιουργείται incoming link προς κάποιο site, το site αυτό προστίθεται στη μεταβλητή *linked*. Αν μετά την δημιουργία όλων των sites το πλήθος αυτών στην *sitelist* είναι ίσο με το πλήθος των μοναδικών sites στην linked τότε αυτό συνεπάγεται ότι όλα έχουν τουλάχιστον ένα incoming link, αλλιώς η διαφορά τους δίνει το πλήθος όσων δεν έχουν incoming link. Σε κάθε περίπτωση εκτυπώνεται σχετικό μήνυμα.

b) Web server

Μεταγλώττιση: $make\ myhttpd$

❖ Sockets:

- Το διάβασμα από το socket γίνεται με επαναληπτική κλήση της *read()* σε buffer σταθερού μεγέθους *BUFSIZ* έως ότου αυτή να επιστρέψει 0. Σε κάθε επανάληψη τα περιεχόμενα του buffer αντιγράφονται σε έναν δεύτερο buffer ο οποίος διατηρεί το συνολικό μήνυμα και στον οποίο γίνεται κάθε φορά *realloc()* αυξάνοντας το μέγεθός του κατά *BUFSIZ*.
- Στο main loop του προγράμματος γίνεται χρήση της *poll()* έτσι ώστε ο server να μην μπλοκάρει περιμένοντας να διαβάσει από ένα συγκεκριμένο socket. Όπως περιγράφει η εκφώνηση, αν φτάσει request στο *serving_port* το αναθέτει σε κάποιο απ' τα διαθέσιμα threads ενώ αν φτάσει εντολή στο *command_port* την αναλαμβάνει το ίδιο το main thread.

❖ Threads:

- Έχει δοθεί προσοχή για χρήση αποκλειστικά thread-safe συναρτήσεων στα κομμάτια του κώδικα που τρέχουν πολλά threads ταυτοχρόνως. Για το λόγο αυτό χρησιμοποιήθηκαν για παράδειγμα (όπου χρειάζονταν) οι συναρτήσεις *strtok_r()* και *gmtime_r()*.
- Σε κάθε επανάληψη του main loop του server γίνεται έλεγχος για threads που έχουν τερματίσει πρόωρα (με χρήση της *pthread_tryjoin_np()*). Εάν κάποιο τέτοιο βρεθεί τότε κατασκευάζεται εκ νέου.

❖ Συγχρονισμός πόρων:

- Η πρόσβαση στις μεταβλητές που αποθηκεύονται τα στατιστικά (*pages_serves* και *bytes_served*) γίνεται μέσω mutexes καθώς ανανεώνονται από πολλαπλά threads που τρέχουν παράλληλα και είναι υπαρκτός ο κίνδυνος race conditions.
- Όταν κάποιος client/request φτάνει στον server αυτός (main thread) κατασκευάζει ένα file descriptor γι' αυτό (μέσω της *accept()*) και, αφού το προσθέσει στην λίστα *fdList*, κάνει broadcast στο condition variable *fdList_cond*. Καθένα από τα threads του thread pool τρέχει σε ένα infinite loop κάνοντας *wait()* σε αυτό το condition variable. Όταν κάποιο κάνει acquire τότε αναλαμβάνει τον πρώτο file descriptor της *fdList* τον οποίο και αφαιρεί από τη λίστα. Έπειτα, και προτού προχωρήσει με το να επεξεργάζεται το request που μόλις έλαβε, ελέγχει αν η λίστα άδειασε και αν όχι τότε κάνει εκ νέου broadcast στο *fdList_cond* για να αναλάβουν άλλα διαθέσιμα threads τα υπόλοιπα requests που βρίσκονται σε εκκρεμότητα.

❖ Διάφορα:

- Το πρόγραμμα απαιτεί από τον χρήστη να δώσει non-privileged ports (δηλαδή μεγαλύτερα από 1023) ως *serving_port* και *command_port* για αποφυγή συγκρούσεων με ports που βρίσκονται ήδη σε χρήση. Απαιτεί επίσης τα δύο αυτά ports να είναι διαφορετικά μεταξύ τους.
- Γίνεται signal handling των διαχειρίσιμων fatal signals ώστε σε περίπτωση που φτάσει κάποιο τέτοιο στον server αυτός να τερματίζει αφού πρώτα απελευθερώσει ό,τι μνήμη έχει δεσμεύσει και ικανοποιήσει όσα τυχόν ανοιχτά requests.

c) Web crawler

Μεταγλώττιση: $make\ mycrawler$

Κλήση: Το starting_URL αναμένεται, όπως έχει ζητηθεί, να είναι της μορφής "*http://host:port/...*"

Οι επιλογές υλοποίησης που περιγράφονται παραπάνω για τον server ισχύουν επίσης (όπου εφαρμόζονται) και για τον crawler. Επιπροσθέτως:

❖ Σχεδιαστικές επιλογές:

- Στην περίπτωση που το *save_dir* υπάρχει ήδη (π.χ. από προηγούμενη εκτέλεση του crawler) και για αποφυγή ανάμειξης των αρχείων μεταξύ των εκτελέσεων, το προηγούμενο *save_dir* μετονομάζεται σε "<*save_dir*>*_verX*".
- Μόλις ολοκληρωθεί το crawling και αφού ξέρουμε ότι δεν πρόκειται να προστεθεί καινούργιο link στη σχετική ουρά, θα ήταν ανούσιο να έχουμε τα threads να περιμένουνε σε ένα condition που δεν πρόκειται να πραγματοποιηθεί. Για το λόγο αυτό, με την ολοκλήρωση του crawling τερματίζουν και τα threads, αφήνοντας ενεργό μόνο το main thread για να δέχεται commands.
- Η ολοκλήρωση του crawling είναι επίσης η στιγμή που εκκινείται ο jobExecutor, εφόσον έχει κατέβει έστω και 1 σελίδα. Αν δεν κατέβηκε καμία σελίδα τότε δεν υπάρχει λόγος να χρησιμοποιηθεί ο jobExecutor και προσπάθεια για εντολή SEARCH αποτυγχάνει με σχετικό μήνυμα.

❖ Ενοποίηση με jobExecutor:

- Στον jobExecutor δεν έγιναν σχεδόν καθόλου αλλαγές από την προηγούμενη εργασία και ο crawler πρακτικά τον αντιμετωπίζει σαν "black box". Η κλήση του γίνεται με *fork()* και *exec()*, αλλά πριν από αυτό ανοίγονται δύο pipes που θα επιτρέπουν την επικοινωνία του με τον crawler και η διεργασία-παιδί (που πρόκειται να «γίνει» ο jobExecutor καλώντας την *exec()*) πρώτα ανακατευθύνει το stdin και το stdout της στα αντίστοιχα άκρα των δύο pipes. Όλο αυτό συμβαίνει μία μόνο φορά στο πρόγραμμα, αυτόματα με την ολοκλήρωση του crawling.
- Κάθε φορά που δημιουργείται ένας καινούργιος φάκελος της μορφής *siteX* από τον crawler, το path αυτού προστίθεται στην λίστα *dirfileList* (εδώ πάλι γίνεται χρήση σχετικού mutex για αποφυγή race conditions). Μετά το τέλος του crawling τα paths αυτά γράφονται σε ένα αρχείο *dirfile.txt* το οποίο δίνεται σαν όρισμα στην κλήση του jobExecutor.
- Οι αλλαγές που έγιναν στον jobExecutor περιορίζονται στην εκτύπωση ενός συγκεκριμένου χαρακτήρα ('<') για την σηματοδότηση της λήξης των αποτελεσμάτων της "/search" και την προσθήκη της συνάρτησης *ignoreHTMLTags()* για τον ομώνυμο λόγο.
- Όπως και στον jobExecutor, έτσι παραμένει και εδώ η επιπρόσθετη λειτουργικότητα για SEARCH άνω των 10 λέξεων.