Κ 24

Homework 2: Ημερομηνία παράδοσης: 1/6/2010

Σημείωση: Για τα προγράμματα που θα γράψετε, παρακαλείστε να παρουσιάσετε ένα σύντομο αρχείο README που να εξηγεί τη λογική του προγράμματος σας. Οδηγίες για την υποβολή της άσκησης υπάρχουν στη σελίδα του μαθήματος.

Υπεύθυνοι για την άσκηση αυτή (ερωτήσεις και βαθμολόγηση, κλπ) είναι: Nikos Chondros (n.chondros@di.uoa.gr) and George Balatsouras (gbalats@gmail.com).

Problem 1: Multi-threaded network server (50 points). Γράψτε έναν πολυνηματικό εξυπηρετητή δικτύου σε C που ονομάζεται poller ο οποίος εκτελεί μια δημοσκόπηση για το «καυτό» ζήτημα της ημέρας. Το πρόγραμμα θα το τρέχετε από τη γραμμή εντολής με τα ακόλουθα ορίσματα

```
prompt> poller [portnum] [numWorkerThreads] [bufferSize] [hotIssueFileName]
[poll-log]
```

Συγκεκριμένα τα ορίσματα είναι:

- **portnum:** ο αριθμός θύρας που ακουει ο εξυπηρετητής
- **numWorkerThreads:** ο αριθμός νημάτων εργατών που θα δημιουργήσει για να κάνει τη δημοσκόπηση. Πρέπει να είναι > 0.
- **bufferSize:** το μέγεθος ενός ενταμιευτή που θα κρατά συνδέσεις από πελάτες που περιμένουν να εξυπηρετηθούν. Πρέπει να είναι > 0.
- **hotIssueFileName:** το όνομα ενός αρχείου που περιέχει το ζήτημα στο οποίο θα γίνει η δημοσκόπηση
- **poll-log:** το όνομα ενός αρχείου όπου θα αποθηκεύονται τα ονόματα και οι ψήφοι των χρηστών, μαζί με κάποια στατιστικά για τις συνδέσεις που δημιουργούνται ανάμεσα στον εξυπηρετητή και πελάτες.

Για παράδειγμα, αν τρέξετε το πρόγραμμα σας με τα ακόλουθα ορίσματα:

```
prompt> poller 5634 8 16 hotIssue-May1.txt pollLog-May1.txt
```

τότε ο εξυπηρετητής θα ακουει στη θύρα 5634, θα δημιουργήσει 8 νήματα εργάτες, θα χρησιμοποιήσει έναν ενταμιευτη που κρατά μέχρι 16 συνδέσεις που περιμένουν εξυπηρέτηση, θα διαβάζει το ζήτημα από το αρχείο hotIssue-May1.txt και θα αποθηκεύει τα αποτελέσματα της δημοσκόπησης (ονόματα, ψήφους) στο αρχείο pollLog-May1.txt.

Στην υλοποίηση σας, θα πρέπει να έχετε ένα νήμα-αρχηγό (master thread) που ξεκινά δημιουργώντας numWorkerThreads νήματα-εργάτες. Το master thread θα δέχεται συνδέσεις από πελάτες με την accept κλήση συστήματος και θα τοποθετεί τους περιγραφείς αρχείων που αντιστοιχούν στις συνδέσεις σε έναν ενταμιευτή συγκεκριμένου μεγέθους (που ορίζεται από το bufferSize). Το νήμα-αρχηγός ΔΕΝ θα διαβάζει από τις συνδέσεις που δέχεται. Απλώς, όποτε δέχεται κάποια σύνδεση θα τοποθετεί τον περιγραφέα αρχείου στον ενταμιευτή και θα συνεχίζει να δέχεται επόμενες συνδέσεις.

Η δουλειά των νημάτων-εργατών είναι να διαβάζουν τις αιτήσεις από τους περιγραφείς αρχείων και να εξυπηρετούν τους πελάτες. Ένα νήμα-εργάτης ξυπνά όταν υπάρχει τουλάχιστον ένας περιγραφέας στον ενταμιευτή (δηλαδή τουλάχιστον ένας πελάτης έχει συνδεθεί στον εξυπηρετητή). Όταν ξυπνά, το νήμα-εργάτης, στέλνει ένα MENU μήνυμα στον πελάτη που περιέχει

What would you like to do?
Choice 1: Vote on Issue of the Day
Choice 2: See Poll Results Collected thus far
Choice 3: Quit
Enter your choice:

Μετά, διαβάζει από τον περιγραφέα αρχείου την αίτηση του πελάτη, την επεξεργάζεται, και επιστρέφει την απόκριση στον πελάτη γράφοντας στον περιγραφέα αρχείου. (Δείτε παρακάτω για περισσότερες λεπτομέρειες για την αλληλεπίδραση ανάμεσα στο νήμα-εργάτης και στον πελάτη). Το νήμα-εργάτης μετά συνεχίζει με την επόμενη σύνδεση.

Το νήμα-αρχηγός και τα νήματα-εργάτες έχουν σχέση παραγωγού-καταναλωτή και έτσι στην υλοποίηση σας θα πρέπει οι προσβάσεις τους στον κοινό ενταμιευτή να συγχρονίζονται. Συγκεκριμένα, το νήμα-αρχηγός πρέπει να μπλοκάρεται και να περιμένει όταν ο ενταμιευτής είναι γεμάτος ενώ ένα νήμα-εργάτης πρέπει να περιμένει αν ο ενταμιευτής είναι άδειος. Με αυτή τη προσέγγιση, αν υπάρχουν περισσότερα νήματα-εργάτες από ενεργές συνδέσεις, τότε κάποια από τα νήματα-εργάτες θα μπλοκάρονται, περιμένοντας νέες συνδέσεις να φτάσουν στον εξυπηρετητή, ενώ αν υπάρχουν περισσότερες ενεργές συνδέσεις από νήματα-εργάτες, τότε οι συνδέσεις (δηλαδή οι αντίστοιχοι περιγράφεις αρχείων) θα πρέπει να αποθηκεύονται στον ενταμιευτή μέχρι που να υπάρχει ένα διαθέσιμο νήμα-εργάτης.

Σε αυτήν την εργασία, θα πρέπει να χρησιμοποιήσετε μεταβλητές συνθήκης στην υλοποίηση σας. **Αν η υλοποίηση σας κάνει οτιδήποτε busy-waiting, θα υπάρξει σημαντική ποινή.**

Για κάθε σύνδεση, θα καταγράφετε κάποια στατιστικά στο poll-log αρχείο . Η συνάρτηση gettimeofday() ίσως σας φανεί χρήσιμη. Συγκεκριμένα θα καταγράψετε:

- **Stat-req-arrival:** The arrival time, as first seen by the master thread
- **Stat-req-dispatch:** The dispatch interval (the duration between the arrival time and when the connection was picked up by a worker thread)

Επίσης να καταγράψετε και τα ακόλουθα στοιχεία για κάθε νήμα-εργάτης:

- **Stat-thread-id:** The id of the responding worker thread (numbered 0 to numWorkerThreads-1)
- **Stat-thread-count:** The total number of client connections this worker thread has handled

Για κάθε σύνδεση που χειρίζεται το νήμα-εργάτης με αριθμό i, ο εξυπηρετητής σας θα αποθηκεύει στο poll-log αρχείο τα στοιχεία για την σύνδεση και τα τρέχοντα στοιχεία για το νήμα i.

Problem 2: User-interface client (20 points): Γράψτε ένα πρόγραμμα πελάτη σε C για την αλληλεπίδραση ανάμεσα στον χρήστη και τον εξυπηρετητή σας. Ο πελάτης θα συνδέεται μέσω TCP με τον εξυπηρετητή, θα λαμβάνει ένα MENU μήνυμα και θα εμφανίζει στον χρήστη το payload του μηνύματος:

What would you like to do?
Choice 1: Vote on Issue of the Day
Choice 2: See Poll Results Collected thus far
Choice 3: Quit
Enter your choice:

Αν ο χρήστης πληκτρολογήσει 1, τότε ο εξυπηρετητής ζητά από τον πελάτη να του στείλει το όνομα του χρήστη και ελέγχει αν έχει ήδη ψηφίσει ο χρηστής. Αν ναι, τότε ο εξυπηρετητής στέλνει μήνυμα στον πελάτη ότι δεν επιτρέπεται ο χρήστης να ξαναψηφίσει «*Sorry, you have already participated in the poll. You cannot vote again.*» και ο πελάτης το εμφανίζει στον χρήστη. Αν όχι, τότε εμφανίζει ο πελάτης το ζήτημα που λαμβάνει από τον εξυπηρετητή στον χρήστη και στέλνει τη ψήφο (YES ή NO) του χρήστη στον εξυπηρετητή που τη καταγράφει.

Αν ο χρήστης πληκτρολογήσει 2, τότε ο εξυπηρετητής ζητά από τον πελάτη να του στείλει το όνομα του χρήστη και ελέγχει αν έχει ψηφίσει ο χρήστης. Αν ναι, τότε επιτρέπεται να δει τα αποτελέσματα που έχει συλλέξει μέχρι στιγμής, και ο εξυπηρετητής τα στέλνει στον πελάτη που τα παρουσιάζει στον χρήστη. Αν όχι, τότε ο εξυπηρετητής στέλνει ένα μήνυμα «*Sorry you cannot see the poll results. You have not yet voted on the Issue of the Day*» στον πελάτη που το εμφανίζει στον χρήστη.

Αν ο χρήστης πληκτρολογήσει 3, τότε ο πελάτης τερματίζει τη σύνδεση του με τον εξυπηρετητή.

Ο εξυπηρετητής θα πρέπει να κρατήσει κάποια στοιχεία για την δημοσκόπηση: τα ονόματα χρηστών που έχουν συμμετάσχει μέχρι στιγμής στη δημοσκόπηση μαζί με τις ψήφους τους, τον αριθμό των χρηστών που έχουν ψηφίσει YES, και τον αριθμό των χρηστών που έχουν ψηφίσει NO. Στην υλοποίηση, θα πρέπει να αποθηκεύεται τα πρώτα δυο στοιχεία στο αρχείο poll-log (είτε καθώς στέλνονται από τους πελάτες, είτε περιοδικά, είτε στο τέλος αν πάρει σήμα τερματισμού ο εξυπηρετητής – επιλέγετε εσείς και εξηγείτε στο README αρχείο την επιλογή σας.) Μπορείτε επίσης να κρατήσετε αυτά τα στοιχεία σε οποιαδήποτε δομή δεδομένων σας εξυπηρετεί. Καθώς τα νήματα που εξυπηρετούν τους χρήστες θα πρέπει να ενημερώνουν τα δεδομένα που κρατά ο server, **θα πρέπει να υλοποιήσετε και τον κατάλληλο συγχρονισμό ώστε να γίνονται τροποποιήσεις στα δεδομένα με σωστό τρόπο**.

Problem 3: Batch client for testing purposes (30 points) Γράψτε έναν πολυνηματικό πελάτη σε C που ονομάζεται pollSwayer που παίρνει τέσσερα ορίσματα: το όνομα του εξυπηρετητή στον οποίο θα συνδεθεί (serverName), τον αριθμό θύρας που ακουει ο εξυπηρετητής (portNumber), τον αριθμό ψήφων που θα στείλει στον εξυπηρετητή (numVotes), και το είδος ψήφου (YES ή NO) που θα στείλει (voteType). Το pollSwayer θα παράγει numVotes ψεύτικα ονόματα χρηστών και για κάθε ψεύτικο όνομα θα δημιουργεί ένα νήμα που θα συνδέεται στον εξυπηρετητή και θα του στέλνει ψήφο τύπου voteType από για το όνομα. Μπορείτε επίσης να τρέξετε το πρόγραμμα αυτό από πολλαπλά διαφορετικά

κελύφη συγχρόνως για να βρείτε και να διορθώσετε λάθη συγχρονισμού στον εξυπηρετητή σας.

Η λεπτομερής συμπεριφορά του εξυπηρετητή και πελάτη περιγράφεται με την ακόλουθη μηχανή πεπερασμένων καταστάσεων (finite state machine) και ψευδοκώδικα. (Ευχαριστίες στον Νικο Χονδρο.) Τα ονόματα των καταστάσεων και των μηνυμάτων που ανταλλάζονται εμφανίζονται με κεφαλαία γράμματα. Send/wait συμβολίζει την αποστολή / λήψη ενός μηνύματος συγκεκριμένου τύπου. Payload αναφέρεται στα δεδομένα που μεταφέρει ένα μήνυμα.

```
SERVER behavior (state machine):
        STATE_INITIAL
                Send MENU
                wait VOTE-BEGIN/POLL-RESULTS-BEGIN
                switch(input)
                        case VOTE-BEGIN
                                send CREDENTIALS-REQUEST
                                state = STATE_VOTE_CREDENTIALS
                        case POLL-RESULTS-BEGIN
                                send CREDENTIALS-REQUEST
                                state = STATE_RESULTS_CREDENTIALS

        STATE_VOTE_CREDENTIALS
                wait CREDENTIALS-RESPONSE
                lookup payload (name)
                if not found
                        //need to keep the name as session-level state
                        send VOTE-SUBJECT
                        state=STATE_VOTE_SELECTION
                else
                        send ERROR-VOTE-AGAIN
                        state=STATE_INITIAL

        STATE_VOTE_SELECTION
                wait VOTE
                //read session-level state: name
                register name/vote in server database
                        // uniqueness of name has to be re-checked here
                        // (while the locks are held); otherwise, you may
                        // get errors on the data structures you maintain
                        // while testing: suggest a SLEEP of 100-500ms right here,
                        //  as this will help multiple clients clash easier

                send VOTE-REGISTERED
                state=STATE_INITIAL

        STATE_RESULTS_CREDENTIALS
                wait CREDENTIALS-RESPONSE
                lookup payload (name)
                if found
```

send POLL-RESULTS
state=STATE_INITIAL
else
send ERROR-NOT-VOTED
state=STATE_INITIAL

CLIENT behavior (completely driven by remote input from server –problem 2):
wait remote input
switch(input)
case MENU
display payload (menu)
ask selection (1,2,3)
submit VOTE-BEGIN/POLL-RESULTS-BEGIN or quit
case CREDENTIALS-REQUEST
ask name from user
send CREDENTIALS-RESPONSE
case VOTE-SUBJECT
display payload (issue of the day)
ask YES/NO from user
send VOTE
case ERROR-VOTE_AGAIN, ERROR-NOT-VOTED
display appropriate message
wait for ENTER from user
case POLL-RESULTS
display payload (results)
wait for ENTER from user

BATCH CLIENT behaviour (problem 3)
start <n> threads, each one autonomous
at each thread:
connect
wait MENU
send VOTE-BEGIN
wait CREDENTIALS-REQUEST
send CREDENTIALS-RESPONSE
wait VOTE-SUBJECT (or ERROR-VOTE-AGAIN)
send VOTE
wait VOTE-REGISTERED
disconnect

Extra credit (25 points) Να τροποποιήσετε τον εξυπηρετητή σας ώστε κάθε φορά που περισσότερα από *idleWorkerThreads* ελεύθερα νήματα-εργάτες περιμένουν να αφαιρέσουν περιγραφέα αρχείου από τον ενταμιευτή και ο ενταμιευτής έχει τουλάχιστον *upperBufThreshold* περιγραφείς αρχείων, τότε να μπλοκάρεται το νήμα-αρχηγός ώστε να μη προσθέτει άλλους περιγραφείς αρχείων, μέχρι ο ενταμιευτής να έχει το πολύ *lowerBufThreshold* περιγραφείς αρχείων ή να μην υπάρχουν ελεύθερα νήματα εργάτες.

Το πρόγραμμα σας να ονομάζεται poller2 και να παίρνει τρία πρόσθετα ορίσματα: το **idleWorkerThreads, το upperBufThreshold, και το lowerBufThreshold.** [Ευχαριστίες στον George Balatsoura.]