Πανεπιστήμιο Πατρών
Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Λειτουργικά Συστήματα

Αναφορά Πρώτης Άσκησης – Ακαδημαϊκό έτος 2012-2013

Αφροδίτη Αλεβιζοπούλου ΑΜ: 3879
Χρήστος Κλαουδιάνης ΑΜ: 3677

Για την ανάπτυξη του κώδικα της άσκησης χρησιμοποιήσαμε μια "divide and conquer" είδους λογική. Σπάσαμε το πρόβλημα σε διάφορα μικρά κομμάτια τα οποία στη συνέχεια αναπτύχθηκαν σε διακριτά βήματα. Πιο συγκεκριμένα:

- Ξεκινήσαμε φτιάχνοντας τη σύνδεση μεταξύ εξυπηρετητή-πελάτη. Θέλαμε να επιβεβαιώσουμε ότι η επικοινωνία τους ήταν εφικτή. Έγινε χρήση των συναρτήσεων read() και write() τόσο στον κώδικα του εξυπηρετητή όσο και στου πελάτη με σκοπό την μεταξύ τους ανταλλαγή μηνυμάτων.
- Έπειτα, δουλέψαμε τον κώδικα του εξυπηρετητή έτσι ώστε να μπορεί να δουλεύει με παράλληλες συνδέσεις, μέσω του fork(). Δηλαδή, ο εξυπηρετητής έπρεπε να δουλεύει στο παρασκήνιο (background) και για κάθε αίτηση που θα λαμβάνει να δημιουργεί μια νέα διεργασία που θα εξυπηρετεί την αίτηση. Ο κώδικάς μας δημιουργεί μια νέα διεργασία για κάθε πίτσα της παραγγελίας του πελάτη.
- Στη συνέχεια, φτιάξαμε κώδικα με σκοπό να ελέγξουμε τον τρόπο λειτουργίας των σημαφόρων. Είδαμε τον τρόπο με τον οποίο ανοίγεις, κλείνεις και μοιράζεσαι έναν σημαφόρο.
- Έπειτα, έπρεπε να ενώσουμε τη λογική των σημαφόρων με τον υπόλοιπο κώδικα. Το shared resource είναι το πόσοι διαθέσιμοι ψήστες και διανομείς υπάρχουν ανά πάσα στιγμή. Θέτοντας την αρχική τιμή των σεμαφόρων στο 10 και κάνοντας απλά sem_wait/sem_post γίνεται αυτόματα ο έλεγχος των διαθέσιμων ψηστών και διανομέων (αν υπάρχει διαθέσιμος η τιμή του σεμαφόρου είναι πάνω από ο οπότε η sem_wait δεν περιμένει, ειδάλλως περιμένει ώσπου κάποιος άλλος να καλέσει sem_post, δηλαδή να απελευθερώθει ψήστης/διανομέας).
- Το τελικό βήμα ήταν η ανάπτυξη ενός script (run_tests.sh) το οποίο αναλαμβάνει να ελέγξει τη λειτουργία του εξυπηρετητή, τρέχοντας παράλληλα 100 πελάτες, όπου ο καθένας του δίνει 15 τυχαίες παραγγελίες.

Κατά την ανάπτυξη του εξυπηρετητή αντιμετωπίσαμε διάφορα προβλήματα. Άξιο αναφοράς είναι η διαχείριση και η απελευθέρωση των socket και των semaphores και κυρίως η λογική διαμοιρασμού πόρων ανάμεσα στα forked children.
Χαρακτηριστικό παράδειγμα αποτελεί το ότι κατά τη διάρκεια ανάπτυξης του δοκιμαστικού σεναρίου κελύφους λάβαμε συχνά το λάθος ότι έχουμε πολλά ανοιχτά αρχεία από την accept. Το πρόβλημα πήγαζε από τη λαθασμένη διαχείρηση των socket με αποτέλεσμα να ξεπερνάμε γρήγορα το όριο των 1024 αρχείων (ulimit -n) επειδή δε τα κλείναμε σωστά.

Θεωρήσαμε πως ο συνολικός χρόνος κάθε παραγγελίας είναι ο χρόνος από τη στιγμή που λαμβάνουμε την παραγγελία μέχρι και τη στιγμή που βρίσκουμε διαθέσιμο διανομέα. Εάν αυτό το χρονικό διάστημα είναι μεγαλύτερο από T_VERY_LONG = 50 msec, τότε στέλνουμε στον πελάτη και μια δωρεάν μπύρα μαζί με την παραγγελία.

Δομές

Η βασική δομή που χρησιμοποιήσαμε είναι η δομή order (ορισμένη στο order.h).
Η δομή χρησιμοποιείται τόσο για την ανταλλαγή μηνυμάτων ανάμεσα στον εξυπηρετητή και τον πελάτη αλλά και εσωτερικά στον κώδικα του εξυπηρετητή ανάμεσα στις συναρτήσεις που διαχειρίζονται την παραγγελία. Η δομή αποτελείται από 4 πεδία. Τις πίτσες που θέλει ο πελάτης, την απόστασή του (πληροφορίες που στέλνει ο πελάτης) και 3 πεδία που χρησιμοποιεί μόνο ο εξυπηρετητής: το pid της παραγγελίας, το χρόνο που λάβαμε την παραγγελία και τέλος το αν ο πελάτης δικαιούται έξτρα μπύρα επειδή η παραγγελία ξεπέρασε το VERY LONG TIME.

Εξωτερικές δομές που χρησιμοποιήσαμε εκτενώς είναι η struct sockaddr_un και η struct timeval:
- struct sockaddr_un : Χρησιμοποιείται για να παραμετροποιηθεί το socket κατάλληλα. Τόσο από το server για το που θα ακούει (listen) και θα δέχεται αιτήσεις (accept) όσο και από τον πελάτη για να δηλώσει που θέλει να συνδεθεί (connect).
- struct timeval. Χρησιμοποιήθηκε εκτενώς για να παρακολοθούμε την εξέλιξη της παραγγελίας. Αν και είχε αναφερθεί ότι μπορούμε να χρησιμοποιούμε alarm για τον αν έχει αργήσει η παραγγελία, θεωρήσαμε πιο αποτελεσματικό να ελέγχουμε στο τέλος της παραγγελίας αν έχει αργήσει και κατάλληλα να προσθέτουμε μια μπύρα.

Σήματα

Τα σήματα που χρειαστήκαμε είναι:
- SIGCHLD:
  Στο βασικό πρόγραμμα το θέτουμε σε ignore για να μπορούν να σταματήσουν τα forked child κατάλληλα (ειδάλως έπρεπε να κάνουμε wait στο τέλος)
- SIGTERM:
- SIGINT:
  Και τα δύο αυτά τα στέλνουμε στη terminate ετσι ώστε να μπορούμε να απελευθερώσουμε τους semaphores αν τερματιστεί ο εξυπηρετητής με καποιο από τα δυο.
- SIGUSR1:
  Δεν εμφανίζεται στον τελικό κώδικα. Στις αρχικές εκδόσεις του εξυπηρετητή χρησιμοποιούσαμε τη pause() για να περιμένει το fork της παραγγελίας να τερματίσουν τα παιδιά. Στην τελική έκδοση το έχουμε αντικαταστήσει με κλήσεις στη waitpid ώσπου να τερματίσουν όλα τα παιδιά.

Οδηγίες:

Η εντολή make δημιουργεί δύο εκτελέσιμα αρχεία, το pizzaria (=server) και το client. Αν τρέξουμε τον client χωρίς ορίσματα στέλνει μια default παραγγελία. Αν θελήσουμε να δώσουμε custom παραγγελία τρέχουμε τον client και του δίνουμε 4 ορίσματα με τη σειρά που αναφέρονται:
<αριθμός μαργαρίτα> <αριθμός πεπερόνι> <αριθμός σπέσιαλ> <απόσταση από πιτσαρία>

Ο πελάτης θα στείλει την παραγγελία μόνο αν ο συνολικός αριθμός των πιτσών είναι το πολύ τρεις.
Το script run_tests.sh τρέχει τον server στο background και εκτελεί 100 παράλληλους πελάτες που ο καθένας στέλνει 15 τυχαίες παραγγελίες. Δίνοντας ένα όρισμα στο script μπορούμε να καθορίσουμε τον αριθμό των πελατών που θα τρέξουν παράλληλα.