Λειτουργικά Συστήματα Ι, Ασκηση 1

Σπύρος Σειμένης 5070

16 Νοεμβρίου 2012

1 Εισαγωγή

Η πλήρης λειτουργία του κώδικα περιγράφεται αναλυτικά με σχόλια μέσα στα αρχεία. Παρακάτω περιγράφω τις δομές που έχω χρησιμοποιήσει και μια εκτέλεση του προγράμματος, δεν θα αναφερθώ σε λεπτομέρειες υλοποιήσης.

2 Περιγραφή λειτουργίας

2.1 common.h

Στο παραπάνω αρχείο υπάρχουν:

- Δήλωση κοινών βιβλιοθηκών που χρειάζονται τα άλλα δύο αρχεία
- directives ελέγχου _STACKOP_ και _DEBUG_ που ελέγχουν προαιρετικές λειτουργίες του server
  - _STACKOP_ αν δηλωθεί ο server μπορεί να χειριστεί παραπάνω παραγγελίες απο το MAX_ORDERS (2.3.4). Είναι ενεργοποιημένο απο deafult.
  - _DEBUG_ αν δηλωθεί ο server θα εκτυπώνει παραπάνω πληροφορίες για την λειτουργία του. Δεν είναι ενεργοποιημένο απο default.
- Συναρτήσεις ελέγχου fatal και debug
- Γενικές μεταβλητές του προγράμματος (σημειωμένες με /*——*/)

2.2 client.c

Ενέργειες του client:

- Άνοιγμα κοινού socket για επικοινωνία με τον server
- Δημιουργία παραγγελίας
  - Αν υπάρχουν ορίσματα δημιουργεί μια τυχαία παραγγελίας
  - Αν όχι, ο χρήστης δίνει είσοδο
- Παραμένει ανοιχτός για να λαμβάνει μηνύματα του server (πχ cocacoles ή έλλειψη μνήμης) και κλείνει μόλις λάβει το μήνυμα DONE

2.3 server.c

Η κεντρική main του server το μόνο που κάνει είναι συνεχώς να ακούει για παραγγελίες και μόλις λάβει μία παραγγελία κάνει fork για την διαχείρισή της (2.3.1) και συνεχίζει να κάνει accept. Ο έλεγχος για καθυστερημένες παραγγελίες αντί να είναι βάρος του server είναι αρμοδιότητα του fork της κάθε παραγγελίας.

2.3.1 Fork παραγγελίας

Εσωτερικά το κάθε παιδί λαμβάνει την παραγγελία απο τον client και την κάνει parse σε ένα πίνακα κωδικών (0-2).Αφού λάβει την παραγγελία εκκινεί τον timer.

Έπειτα προσθέτει την παραγγελία στην λίστα παραγγελιών σε αναμονή pending (λεπτομέρειες στο section 2.3.3) και ξεκινάει η διαδικασία ανάθεσης σε ψήστη. Η παραγγελία εκτελεί ένα loop μέχρι να δώσει επιτυχώς όλες τις πίτσες της σε ψήστες. Κάθε φορά προσπαθεί να κατεβάσει το semaphore που ελέγχει τους ψήστες(αρχικοποιείται με NBAKERS), αν τα καταφέρει θα πει ότι υπήρχε διαθέσιμος ψήστης και κάνει fork έναν ψήστη (2.3.2). Μόλις δώσει όλες τις πίτσες της σε ψήστες βγαίνει απο το loop ανεξαρτήτως αν έχει τελειώσει το ψήσιμο και περιμένει τους ψήστες(παιδιά της) να τελειώσουν. Μόλις τελειώσει η διαδικασία του ψησίματος, το order διαγράφεται απο την pending list και εισάγεται στην ready list. Απο κει και μετά ακολουθείται η ίδια διαδικασία για ανάθεση σε delivery.

2.3.2 Baker/delivery process

Το process του baker με αυτό του delivery είναι πανομοιότυπα στην λειτουργία με τις εξής διαφορές:

- ότι το κάθε ένα περιμένει διαφορετικούς χρόνους όπως είναι λογικό
- γράφουν σε ξεχωριστές shared memories που ελεγχονται με ξεχωριστούς semaphores
- έχουν διαφορετικούς semaphores ελέγχου pending->sem_res για ψήστες, ready->sem_res για deliveries

Γενικά ένας baker/delivery κάνει απλά sleep για τον απαραίτητο χρόνο έπειτα ενημερώνει το orderκάνοντας -1 το status του πατέρα του στην shared memory.Αφού τελειώσει αυξάνει τον αντίστοιχο semaphore sem_res.

2.3.3 Λίστα παραγγελιών

Η λίστα παραγγελιών υλοποιείται με μια διπλά συνδεδεμένη λίστα για γρηγορότερους χρόνους σβησίματος και εισαγωγής. Κάθε κόμβος κρατάει το status της παραγγελίας και το pid της διεργασίας που την έχει αναλάβει.

Η λίστα αποθηκεύεται στην shared memory και ξεκινάει καταλαμβάνει χώρο απο την αρχή της shared memory μέχρι MAX_ORDERS*μέγεθος του κόμβου order_info. Επειδή χρησιμοποιώ 2 λίστες για πιο γρήγορους χρόνους ώστε να μπορούν delivery και bakers να προσπελαύουν ταυτόχρονα τις απαραίτητες πληροφορίες, χρειάζεται για κάθε λίστα να διατηρώ τα στοιχεία της σε ένα struct list_info.

Συνολικά την shared memory που κάνω allocate την χωρίζω σε 2 τμήματα ένα για τους bakers και ένα για τους deliveries. Κάθε τμήμα αποτελείται απο τα εξής:

- Την παραπάνω λίστα, η pending για το ένα, η ready για το άλλο.
- Ένα struct list_info το οποίο αποθηκεύει τις πληροφορίες που χρειάζονται για να ελέγχω την λίστα και το τμήμα γενικά.
- Την στοίβα διαχείρισης μνήμης(2.3.4).
- Τον semaphore για τον έλεγχο πρόσβασης στο τμήμα
- Τους semaphores για τον έλεγχο της στοίβας(2.3.4).
- Τον semaphore για το resource που ελέχγχει η κάθε λίστα (bakers για την pending, deliverys για την ready)

2.3.4 Στοίβα διαχείρισης μνήμης

Χωρίς την προσθήκη της στοίβας μόλις ο server δεχτεί παραπάνω απο MAX_ORDERS θα πάψει να δέχεται παραγγελίες. Για να μπορώ να παραχωρώ συνέχεια μνήμη η εισαγωγή στην shared memory γίνεται ως εξής. Μέχρι να καλυφθεί το max_orders η παραγγελίες εισάγονται στην λίστα σειριακά (μεταβλητή offset). Μόλις τελειώσει το πρώτο γέμισμα της λίστας (offset == MAX_ORDERS) οι υπόλοιπες παραγγελίες εισάγονται σε διεύθυνση που κάνουν pull απο το stack. Μόλις μια παραγγελία διαγράφεται απο μια λίστα κάνει push την θέση της στο stack.

2.3.5 Καθυστερημένες παραγγελίες

Ο έλεγχος των καθυστερημένων παραγγελιών γίνεται χρησιμοποιώντας POSIX timers λόγω των ευκολιών που παρέχουν. Με το που ξεκινάει ένα fork παραγγελίας, αφού λάβει την παραγγελία του ξεκινάει και ένας timer. Αυτός ο timer μόλις περάσει tverylong χρόνος θα αναλάβει να στείλει στην παραγγελία ένα sigalarm με το οποίο ενημερώνεται ο client ότι θα πάρει cocacola, και θα ξαναθέσει τον εαυτό του να ξαναστείλει σήμα σε tverylong χρόνο. Για να μην επηρεάζει το σήμα την λειτουργία του προγράμματος και κυρίως να μην διακόπτει τις sem_wait, γίνεται mask πριν τα critical section του κώδικα. Η ευκολία που παρέχουν οι POSIX timers είναι η συνάρτηση timer_getoverrun με την οποία ξέρω πόσες φόρες έληξε ο timer ενώ το σήμα ήταν blocked. Έτσι με το που επιτρέπω το σήμα ξανά στέλνω ξεχωριστά στον client όσες κοκα κόλες χρειάζεται, ώστε να είναι όσο πιο ακριβής γίνεται η καθυστέρηση.