Λειτουργικά Συστήματα Ι

Αναστασία Παπαρροδοπούλου 3873
Χαράλαμπος Λογδανίδης 3776

Άσκηση 2

Η άσκηση αυτή, ακολουθώντας το ίδιο μοτίβο με την προηγούμενη, υλοποιεί ένα σύστημα κρατήσεων θέσεων σε ένα θέατρο. Τις κρατήσεις τις πραγματοποιεί ένας server που τρέχει στο background. Σε αυτόν συνδέονται clients κάνοντας αιτήσεις για τις θέσεις που επιθυμούν. Ο server αναλαμβάνει να βρει τις κατάλληλες θέσεις και να κάνει τη χρέωση στην πιστωτική κάρτα του πελάτη (client). Με κάθε είσοδο ενός client, ένα νέο thread δημιουργείται για να τον εξυπηρετήσει. Επειδή οι τηλεφωνητές του θεάτρου είναι μονο 10, αν συνδεθούν στο σύστημα πάνω από 10 clients τότε κάποια threads μπλοκάρουν σε ένα condition μέχρι να ύπαρξει ξανά ελεύθερος τηλεφωνητής. Με τον ίδιο τρόπο λειτουργεί και ο έλεγχος της πιστωτικής κάρτας. Οι υπάλληλοι της τράπεζας είναι μόνο 4, οπότε περισσότερες από 4 αιτήσεις περιμένουν σε ένα άλλο condition αντίστοιχα.

Παρακάτω ακολουθούν περισσότερες λεπτομέρειες για την υλοποίηση.

Δομές δεδομένων (structs)

Για την άσκηση χρησιμοποιήθηκαν οι ακόλουθες δομές που διαμοιράζονταν μεταξύ των threads:

Πίνακες των θέσεων:

```
struct thesis_t
{
    int A[A_NUM];
    int B[B_NUM];
    int C[C_NUM];
    int D[D_NUM];
} plano;
```

Στατιστικά στοιχεια:

```
struct statistics
{
    int fail;
    int succeded;
    time_t wait_time;
    time_t e3ipiretisi_time;
    int transfer[100];
} statist;
```

Όπου:

fail: Ο αριθμός των αποτυχημένων προσπαθειών.
succeded: Ο αριθμός των επιτυχημένων κρατήσεων
wait_time: Ο χρόνος αναμονής για τον τηλεφωνητή.
e3ipiretisi_time: Ο χρόνος εξυπηρέτησης από τη στιγμή που έχει αρχίσει η επικοινωνία με τον τηλεφωνητή μέχρι το τέλος τις συνομιλίας.
transfer: Ο πίνακας με τα ποσά από τις μεταφορές χρημάτων που έγιναν στον λογαριασμό του θεάτρου.

Μεταβλητές:

```
int kratisi_id
int company_account
int theater_account
int full;
```

Όπου :

kratisi_id: Ένας αύξων αριθμός για το id της κάθε κράτησης.
company_account: Ο τραπεζικός λογαριασμός της εταιρίας. Περιέχει τα χρήματα που μεταφέρθηκαν από τους πελάτες.
theater_account : Ο τραπεζικός λογαριασμός του θεάτρου. Περιέχει τα χρήματα που μεταφέρθηκαν στο θέατρο από το λογαριασμό της εταιρίας.
full: Αποθηκεύει το πόσες θέσεις έχουν δεσμευτεί. Έτσι ο έλεγχος για το αν το θέατρο είναι γεμάτο γίνεται πολύ γρήγορα.(full==THESIS_NUM).

‘Ολες οι παραπάνω μεταβλητές μπορούν να θεωρηθούν *ευαίσθητες* αφού περισσότερα από 1 threads έχουν πρόσβαση σε αυτές. Γι’ αυτό το λόγο προστατεύονται με χρήση mutex (συγκεκριμένα το data_mutex) .

Threads και συγχρονισμός

Με την εκκίνηση του server, ένα νέο thread δημιουργείται για την μεταφορά των χρημάτων από την εταιρία στο θέατρο. Το thread αυτό περιμένει 30 δευτερόλεπτα, πραγματοποιεί τη μεταφορα και επαναλαμβάνει τη διαδικασία.

Με την είσοδο κάποιου client, ένα νέο thread τηλεφωνητή αναλαμβάνει να τον εξυπηρετήσει. Καθώς οι τηλεφωνητές είναι μόνο 10, σε περίπτωση που τα thread τηλεφωνητών ξεπεράσουν αυτόν τον αριθμό, περιμένουν σε ένα condition. Θέλοντας να αποφύγουμε τα threads να περιμένουν επάπειρον και επιθυμώντας να στέλνεται ένα μήνυμα συγγνώμης στον client ανά 10 δευτερόλεπτα, χρησιμοποιήσαμε τη συνάρτηση pthread_cond_timedwait . Με τον τρόπο αυτό κάθε τηλεφωνητής ξυπνάει ανά 10 δευτερολεπτα, στέλνει το μήνυμα συγγνώμης και αν δεν εκπληρείται το condition περιμένει ξανά.

Ο κώδικας μοιάζει κάπως έτσι:

```
pthread_mutex_lock(&tilef_mutex);
while(tilef_num==0)
{
   int err;
   err=pthread_cond_timedwait(&tilef_cond,
         &tilef_mutex,&t_tilefonitis_time);
   if(err==ETIMEDOUT)
   {
      pthread_mutex_unlock(&tilef_mutex);
      write(ns,WAIT_MSG);
      pthread_mutex_lock(&tilef_mutex);
   }
}
pthread_mutex_unlock(&tilef_mutex);
```

Όταν έρθει η σειρά του τηλεφωνητή, αυτός δημιουργεί ένα νέο thread για τον έλεγχο της κάρτας ούτως ώστε ο έλεγχος κάρτας και θέσεων να γίνει παράλληλα. Μετά τον έλεγχο των θέσεων ο τηλεφωνητής περιμένει να ολοκληρωθεί και το thread της τράπεζας με την pthread_join. Η pthread_join επιστρέφει επίσης και το exit value του thread που τερμάτισε. Έτσι αν η τράπεζα επιστρέψει τιμή διάφορη του 0, τότε ο έλεγχος της κάρτας δεν ήταν επιτυχής. Τα threads της τράπεζας, ομοίως με τα threads τηλεφωνητών, περιμένουν τη σειρά τους σε ένα condition.

Τέλος αξίζει να σημειωθεί ότι τα threads τηλεφωνητών είναι σε detached state καθώς κανείς δεν περιμένει σε αυτά, ενώ τα threads της τράπεζας είναι joinable.

Τερματισμός του server

Ο server τερματίζει με interrupt ή με termination signal. Άμα λάβει ένα από τα δύο signals κλείνει το socket ώστε να μη γίνουν δεκτές άλλες συνδέσεις, περιμένει να ολοκληρωθούν οι συναλλαγές με τους ήδη συνδεδεμένους clients, κάνει cancel το thread μεταφοράς χρημάτων και τερματίζει με ασφάλεια.

Εκτέλεση

Αρχικά κάνουμε compile τον κώδικα με την εντολή **make**.

Για να εκτελέσουμε τον server γράφουμε

./server &

Για να εκτελέσουμε έναν client γράφουμε

./client $arithmos_isitirion $zoni $card

Παράδειγμα

./client 3 A 124

Αρχεία

server_thr.c : Ο κώδικας του server.
client.c : Ο κώδικας του client.
header.h : Κοινό header μεταξύ server και client.
utils.c : Κοινές συναρτήσεις μεταξύ server και client για δημιουργία τυχαίων αριθμών, ανάγνωση και εγγραφή, καθώς και sleep χωρίς να επηρεάζεται από interrupts.
Makefile : Αρχείο με οδηγίες για το compile του κώδικα.
run_tests.sh : Αρχείο για την εκτέλεση του server και του client.
report.pdf : Αναφορά άσκησης.

Αποτελέσματα

Ποσοστό αποτυχίας: 0.599064
Μέσος χρόνος αναμονής: 2.901716
Μέσος χρόνος εξυπηρέτησης: 5.859594
μεταφορές: 3590 3700 3695 3830 4425 2310 200 650 200 350 250 200

Όσο περισσότερο διαρκεί η εκτέλεση, το ποσοστό αποτυχίας αυξάνεται. Αυτό οφείλεται στο ότι κάποιες ζώνες γεμίζουν και τελικά όλο και περισσότερες κρατήσεις αποτυγχάνουν.
Επίσης πρέπει να σημειώσουμε ότι το script μας χρησιμοποιεί 15 clients ώστε να γίνει φανερός και ο χρόνος αναμονής.