ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Λειτουργικά Συστήματα 2021-2022

Αναφορά Εργασίας

Διονύσιος Ρηγάτος – P3200262

Χριστόφορος Παπαποστόλου – P3150208

Θέμα:

Το θέμα της εργασίας είναι η δημιουργία ενός πολυνηματικού προγράμματος το οποίο θα αναλαμβάνει το random generation πελατών και της επιλογής εισιτηρίων τους και θα κάνει simulate την διαδικασία κράτησης στο τηλεφωνικό κέντρο ενός θεάτρου, λαμβάνοντας υπόψη τυχών προβλήματα όπως η ανεπάρκεια θέσεων και η αποτυχία μίας πληρωμής με πιστωτική κάρτα.

Εισαγωγή:

Αρχικά, στο header file (p3200262-p3150208-res.h) γίνεται η εισαγωγή όλων των απαραίτητων βιβλιοθηκών και η δήλωση σταθερών σύμφωνα με την εκφώνηση, καθώς και ένα customer struct το οποίο κρατάει πληροφορίες κράτησης για κάθε πελάτη. Έπειτα, στην κορυφή του main source file μας (p3200262-p3150208-res.c) κάνουμε include to header file καθώς και την δήλωση καθολικών μεταβλητών και mutexes/conds.

Θα γίνει η χρήση των παρακάτω mutexes & conditionals, με τα self-explanatory σχόλια:

```
pthread_mutex_t operator_lock_a;        // Locks Zone A
pthread_mutex_t operator_lock_b;        // Locks Zone B
pthread_mutex_t operator_wait_lock;     // Locks the operator wait condition
pthread_mutex_t cashier_wait_lock;      // Locks the cashier wait condition
pthread_mutex_t payment_lock;           // Locks the payment variable
pthread_mutex_t output_lock;            // Locks the output
pthread_mutex_t free_seats_lock;        // Locks the free_per_row array
pthread_mutex_t time_lock;              // Locks the time variables

pthread_cond_t operator_cond;           // Condition variable for operator wait
pthread_cond_t cashier_cond;            // Condition variable for cashier wait
```

Η χρήση μεταβλητών-μετρητών:

- Για τον αριθμό διαθέσιμων τηλεφωνητών (αρχ. από .h)
- Για τον αριθμό διαθέσιμων ταμιών (αρχ. από .h)
- Για το πλήθος των πελατών (αρχ. από command line arguments)
- Αποτυχίες εύρεσης θέσεων
- Αποτυχίες πληρωμών
- Χρόνου αναμονής
- Χρόνου κλήσης

Καθώς και η μεταβλητή του σπόρου που θα δοθεί στην είσοδο.

Τέλος, γίνεται η δήλωση τριών πινάκων. Ενός για το πλάνο των θέσεων (seat_plan), ενός πίνακα-μετρητή (free_per_row) που θα χρησιμοποιηθεί για να αποφευχθεί η άσκοπη διερεύνηση σειρών που δεν έχουν αρκετές διαθέσιμες θέσεις και ενός πίνακα που έχει μέσα του τα customer structs (customers).

Helper Functions & Notes:

```
void mutex_lock(pthread_mutex_t* lock);
void mutex_unlock(pthread_mutex_t* unlock);
```

Κλειδώνουν/Ξεκλειδώνουν τα mutexes ανάλογα, κάνοντας έλεγχο στο RC. Εμφανίζουν μήνυμα λάθους και τερματίζουν το νήμα σε περίπτωση σφάλματος. Από δω και πέρα για κάθε αναφορά σε κλείδωμα/ξεκλείδωμα mutex/cond/broadcast υπονοείται ότι γίνεται ο απαραίτητος έλεγχος του RC code, είτε με την χρήση συναρτήσεων (mutexes) είτε με if (conditionals/broadcasts), και καλείται η pthread_exit σε περίπτωση σφάλματος.

```
void init_mutex_conds();
void destroy_mutex_conds();
```

Initialization/destruction mutexes/conditionals με τους απαραίτητους ελέγχους.

```
void free_memory();
```

Απελευθέρωση dynamically allocated μνήμης. Καλείται κάθε φόρα πριν τερματίσει το κύριο πρόγραμμα, είτε κανονικά είτε όχι.

Κάθε φορά που γίνεται σε αναφορά αναμονής χρόνου διαστήματος, π.χ. 1-5 sec, αναφερόμαστε σε κλήση της sleep() με είσοδο ψευδοτυχαίο από την rand_r. Η χρήση mutex σε output οθόνης και καθολικές μεταβλητές επίσης μπορεί να εννοηθεί.

Main Function:

Στην αρχή εκτέλεσης του προγράμματός μας γίνεται ο έλεγχος εγκυρότητας εισόδου καθώς και η μετατροπή των χαρακτήρων σε integers έτσι ώστε να χρησιμοποιηθούν ορθά. Στη συνέχεια γίνεται δημιουργία των mutexes/conds και το initialization των πινάκων, καθώς και το δυναμικό allocation μνήμης στον πίνακα με τα customer structs.

Προσεχώς, καλείται το for loop οπού και γίνεται η ανάθεση ενός unique ID σε κάθε πελάτη-νήμα καθώς και η δημιουργία τους, και καλείται η συνάρτηση make_reservation η οποία αναλαμβάνει την διαδικασία της κράτησης. Κάθε πελάτης μετά από τον 1ο περιμένει ένα τυχαίο διάστημα πριν καλέσει (δημιουργηθεί το νήμα του)΄. Εν τέλει γίνεται και το join των νημάτων.

Τέλος εμφανίζονται όλα τα στατιστικά (και τα έσοδα) στην παρακάτω μορφή και το πλάνο θέσεων. Η δυναμική μνήμη απελευθερώνεται και τα mutex/conds καταστρέφονται.

```
Welcome to the Derigni (amphi)theatre!
Now taking reservations for the Operating Systems S/S 2022 Class.
Reserve your seat (and up to 4 extra for your friends) or you will be standing!
```

```
Reservations are now complete! PROFIT: 5910€
Statistics:
        Successful reservations                 : 88.00%
        Failed reservations - NO SEATS AVAILABLE : 6.00%
        Failed reservations - PAYMENT FAILED     : 6.00%
        Average wait time                       : 11.31 sec
        Average call time                       : 26.19 sec
```

```
-----SEAT PLAN FOR TODAYS LECTURE-----
[Zone A / Row 1 / Seat 1 / Client 9]
[Zone A / Row 1 / Seat 2 / Client 9]
[Zone A / Row 1 / Seat 3 / Client 9]
[Zone A / Row 1 / Seat 4 / Client 9]
[Zone A / Row 1 / Seat 5 / Client 11]
[Zone A / Row 1 / Seat 6 / Client 11]
```

Παραδείγματα εξόδου

Make Reservation:

Η συνάρτηση αυτή είναι η κύρια συνάρτηση που διαχειρίζεται τις κρατήσεις του πελατών παράλληλα (άρα η συνάρτηση που τρέχει πολυνηματικά), και παίρνει σαν argument το struct του πελάτη με τις πληροφορίες του. Επί' τόπου δημιουργείται ένα μοναδικό local_seed και γίνονται οι βασικές αρχικοποιήσεις, όπως την εισιτηρίων (randomly generated 1-5). Στη συνέχεια ξεκινάει η διαδικασία. Λαμβάνονται δείγματα χρόνου από την αρχή της κλήσης και από την αρχή της αναμονής για τον τηλεφωνητή. Με την χρήση mutex & conditional variables, ο κάθε πελάτης περιμένει στην περίπτωση που όλοι οι τηλεφωνητές βρίσκονται σε κλήση, με την χρήση μιας while loop, του μετρητή operators_available και της pthread_cond_wait(&operator_cond, &operator_wait_lock) με τα ανάλογα cond/mutex. Όταν ένας τηλεφωνητής τελειώσει, ο τηλεφωνητής που ελευθερώθηκε (απελευθέρωση στην operator_call()) αναλαμβάνει τον επόμενο πελάτη.

Κάθε τηλεφωνητής «ψάχνει» για θέση (sleep) για 5-13 δευτερόλεπτα με την κλήση της **operator_call()** (βλεπε παρακάτω). Εάν δεν βρεθούν αρκετές συνεχόμενες θέσεις στην ζώνη, το κόστος που οφείλει ο πελάτης(c->cost) θα είναι 0 και θα τερματίζει με ανάλογο μήνυμα η κλήση, προσθέτοντας στις καθολικές μεταβλητές που αφορούν χρόνο κλήσης και αναμονής ανάλογα. Εάν βρέθηκαν θέσεις, η operator_call() θα έχει μεταβάλλει την τιμή του c->cost σε x>0, άρα ο πελάτης μεταφέρεται στον αναμονή για ταμία (ανάλογα της αναμονης για operator με χρήση conds/mutex και while loop). Όταν ένας ταμίας αναλάβει τον πελάτη, με την κλήση της **cashier_call**, χρειάζεται 4-8 δευτερόλεπτα για να ανταποκριθεί για το εάν η πληρωμή πέτυχε η όχι. Εάν η πληρωμή πετύχει, το ποσό προστίθεται στον λογαριασμό μας (global BALANCE, με mutex) και εμφανίζεται μήνυμα επιτυχίας. Τέλος, προστίθενται οι ανάλογοι χρόνοι στις καθολικές μεταβλητές wait_time & call_time (με mutex) και η κλήση τερματίζει.

```
[CID: 31] Reservation successful! [Seats: 1 - 3, Row: 18, Zone: B || Cost: 60]
[CID: 32] Reservation successful! [Seats: 9 - 10, Row: 17, Zone: B || Cost: 40]
[CID: 33] Reservation successful! [Seats: 4 - 6, Row: 18, Zone: B || Cost: 60]
[CID: 37] Reservation successful! [Seats: 1 - 5, Row: 3, Zone: A || Cost: 150]
[CID: 34] Reservation successful! [Seats: 6 - 8, Row: 3, Zone: A || Cost: 90]
[CID: 36] Reservation successful! [Seats: 7 - 9, Row: 18, Zone: B || Cost: 60]
```

```
[CID: 12] Reservation failed! Payment unsuccessful - [Seats: 1 - 2, Row: 14, Zone: B || Cost: 40]
```

```
[CID: 94] Reservation failed! No 5 consecutive seats are available in Zone B.
```

Παραδείγματα εξόδου

Operator Call:

Κατά την διάρκεια της κλήσης, δημιουργείται ένα τοπικό seed βάσει του client number και τον αριθμό εισιτηρίων που πήρε. Στην συνέχεια, κάνουμε randomly generate $x \in [0.0, 1.0]$ το οποίο με πιθανότητα 30% (x <= 0.3) θα καθορίσει την ζώνη του πελάτη σαν A, η με πιθανότητα 70% σαν B. Ανάλογα την ζώνη, κλειδώνουμε mutex που δηλώνει ότι γίνεται επεξεργασία δεδομένων σε κομμάτι του πίνακα που ανήκει στην ανάλογη ζώνη. Εδώ χρησιμοποιούνται 2 διαφορετικά mutex, ένα για την ζώνη A και ένα για την B, έτσι ώστε να μην κλειδώνονται άσκοπα κομμάτια του πίνακα που δεν θα χρησιμοποιηθούν καθώς γίνεται μια κράτηση (π.χ. δεν χρειάζεται να είναι κλειδωμένη η ζώνη B εάν εγώ κάνω κράτηση στην A, γιατί τα indexes των πινάκων δεν θα φτάσουν ποτέ στην B). Εκεί υπολογίζουμε το κόστος, το οποίο εξαρτάται από το κόστος της ζώνης, τον αριθμό των εισιτήριών και το εάν βρέθηκαν η θέσεις (κλήση βοηθού **reserve_seat()**), με c->cost > 0 να δηλώνει ότι βρέθηκαν οι θέσεις, δεδομένο που επεξεργάζεται αργότερα η make_reservation(). Στη συνέχεια αποδεσμεύεται ο operator με την χρήση του pthread_cond_broadcast(&operator_cond) και operators_available++ και των ανάλογων mutex.

Reserve Seat (OPERATOR CALL HELPER):

Η reserve_seat() ψάχνει σε κάθε σειρά στην ζώνη που επέλεξε ο πελάτης (limited by array indexes) για c->ticket ελεύθερες συνεχόμενες θέσεις. Επιτυγχάνουμε την διαδικασία με την χρήση του πίνακα free_per_row[row_num], όπου μας βοηθάει στο να ελέγξουμε για συνεχόμενες θέσεις μόνο σε σειρές όπου υπάρχουν αρκετές ελεύθερες (σε αυτό το στάδιο δεν ξέρουμε αν είναι συνεχόμενες). Στη συνέχεια ελέγχουμε από την πρώτη ελεύθερη θέση της σειράς εάν οι επόμενες c->ticket ελεύθερες είναι συνεχόμενες με την χρήση της **first_free_seat()** , η οποία επιστρέφει την πρώτη ελεύθερη θέση με c->ticket ελεύθερες μετά. Εάν δεν είναι τότε η c->seat είναι -1 και συνεχίζουμε το ψάξιμο σε αυτή τη σειρά. Εάν εν τέλει δεν βρεθούν c->ticket συνεχόμενες θέσεις εδώ, προχωράμε στις επόμενες σειρές. Εάν βρεθούν c->ticket συνεχόμενες θέσεις στην first_free_seat(), τότε το c->seat παίρνει το index της πρώτης ελεύθερης, τα εισιτήρια αφαιρούνται από τον free_per_row[c->row] (με mutex lock) και η συνάρτηση επιστρέφει 1, το οποίο κρατάει το γινόμενο που αναφέρθηκε στην operator_call() (στο c->cost) μη

μηδενικό. Εάν αποτύχουμε, μηδενίζεται το c->cost στην operator_call().

First Free Seat (RESERVE SEAT HELPER):

Η συνάρτηση αυτή ελέγχει εάν οι tickets επόμενες θέσεις από την πρώτη κενή είναι ελεύθερες. Εάν είναι, επιστρέφει το index της. Αλλιώς επιστρέφει -1. Η συνάρτηση αυτή είναι απαραίτητη καθώς μπορεί, λόγω παραλληλισμού, να έχουν κρατηθεί οι θέσεις 1-2-3-7-8-9-10 και να έχουμε τις 4-5-6 διαθέσιμες έπειτα από ακύρωση κράτησης με την cancel_reservation(), επομένως θα πρέπει να βρίσκουμε αυτές τις 3 συνεχόμενες θέσεις σε μελλοντικές κρατήσεις.

Cashier Call:

Η συνάρτηση αυτή αναλαμβάνει την πληρωμή και την εξασφάλιση της συναλλαγής. Με πιθανότητα 10% η συναλλαγή αποτυγχάνει, στην οποία περίπτωση κλειδώνεται η ζώνη που διάλεξε ο πελάτης (με το προαναφερόμενο mutex στην operator_call(), το operator_lock_a η το operator_lock_b) και οι θέσεις που είχε κλείσει αποδεσμεύονται από την cancel_reservation(). Επίσης εμφανίζεται ανάλογο μήνυμα σφάλματος της συναλλαγής.

Εάν η συναλλαγή πετύχει, κλειδώνουμε το mutex payment_lock και αυξάνουμε το BALANCE μας και εμφανίζουμε ανάλογο μήνυμα επιτυχίας συναλλαγής.

Cancel Reservation (CASHIER CALL HELPER):

Η συνάρτηση αυτή επαναφέρει τις θέσεις στο πλάνο σε 0, δηλαδή ελεύθερες, και αυξάνει (αφού κλειδώσει) τον μετρητή free_per_row[c->row].

Εν κατακλείδι, στην εργασία αυτή έχει γίνει χρήση πολλών αρθρωτών συναρτήσεων έτσι ώστε να ευνοούμαστε από την επαναχρησιμοποίηση κώδικα. Παράλληλα, υπάρχει συντηρητικός έλεγχος για response codes πιθανών σφαλμάτων και ανάλογη διαχείριση έτσι ώστε να μην συνεχίζεται μία διεφθαρμένη εκτέλεση. Τελευταίο και πιο σημαντικό, έχει γίνει χρήση ορθή mutex οπουδήποτε μπορεί να δημιουργηθεί race condition και τα δεδομένα παραμένουν αναλλοίωτα ανεξάρτητα με το εάν μία κράτηση ακυρωθεί η επιτύχει.