ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΤΟΜΕΑΣ ΛΟΓΙΚΟΥ ΤΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ 1
ΠΡΩΤΗ ΑΣΚΗΣΗ

ΤΣΑΡΝΑΣ ΒΑΣΙΛΕΙΟΣ (ΑΜ: 5444) -
ΡΟΥΜΠΕΪΔΗΣ ΑΛΕΞΑΝΔΡΟΣ(ΑΜ: 5409)

Εισαγωγικά: Η αναφορά αυτή περιγράφει περιληπτικά το πρόγραμμα που υλοποιεί την 1η εργαστηριακή άσκηση με έμφαση στις προγραμματιστικές συμβάσεις που αποφασίστηκαν κατά την διάρκεια της ανάπτυξής της.

***Μετά τον τερματισμό του** server **εκτελέστηκε η εντολή** ipcs*
ΚΑΝΕΝΑΣ** ipc **πόρος δεν μένει αποθηκευμένος στην κύρια μνήμη!
***ΠΡΙΝ εκτελέσετε** make **διαβάστε το** README **στο** directory **του** project*

Γενικά: Σε κάθε πελάτη αντιστοιχεί ένας ακέραιος-id (μεγαλύτερος του 0), ο οποίος αποθηκεύεται στο πλάνο των θέσεων μιας ζώνης, το οποίο είναι ένας πίνακας ακεραίων (το id του πελάτη που έχει κλείσει μια θέση i στην ζώνη X, αποθηκεύεται στο planX[i]). Μια κενή θέση έχει τιμή 0 στην θέση στον πίνακα πλάνου ζώνης που της αντιστοιχεί. Οι θέσεις στο θέατρο κλείνονται ΠΑΝΤΑ σειριακά. Η κράτηση ΔΕΝ θεωρείται επιτυχής όταν ο πελάτης δεν έχει αρκετά λεφτά στον λογαριασμό του (έλεγχος τράπεζας) ή όταν δεν υπάρχουν θέσεις στην ζώνη που επέλεξε. Αφού κλείσουν όλες οι θέσεις στο θέατρο ένα μύνημα αποτυχίας εμφανίζεται αυτόματα.

ΔΙΑΡΘΡΩΣΗ ΚΩΔΙΚΑ:

Για το πρόγραμμα του server έχουμε τα αρχεία serveros.c (περιλαμβάνει την main()), functions.c (περιλαμβάνει ορισμούς συναρτήσεων και signal handlers) και headeros.h, το οποίο είναι αρχείο κεφαλίδας και περιέχει δηλώσεις global μεταβλητών, συναρτήσεων και δηλώσεις των signal handlers.
Για το πρόγραμμα του client έχουμε το αρχείο clientos.c. Δεν χρησιμοποιήθηκε ξεχωριστό αρχείο κεφαλίδας εδώ γιατί έχουμε λίγα σε αριθμό #include, τα οποία είναι διαφορετικά από αυτά του server.
Εκτελώντας την εντολή make στο directory του project γίνονται αυτόματα οι απαραίτητες μεταγλωτίσσεις και συνδέσεις και ξεκινάει την λειτουργία του ο server.
Το run_tests.sh είναι ένα script το οποίο ξεκινά πολλούς clients μαζί. Εκτελείται σε νέο terminal window.

***Επικοινωνία** Client – Server:* Στόχος μας είναι η ανάπτυξη μιας εφαρμογής που κλείνει εισητήρια σε θέατρο "τηλεφωνικώς". Για την χρηματική συναλλαγή μεσολαβεί μια

τραπεζική εταιρία. Το θέατρο έχει 10 τηλεφωνητές και η τράπεζα 4 τερματικά. Τόσο η λειτουργεία των τηλεφωνητών, όσο και η λειτουργία των τερματικών της τράπεζας προσομοιώνονται σε ΕΝΑ πρόγραμμα – SERVER. Ο SERVER με μια "άπειρη" for ακούει, με την χρήση ενός listen socket νέες αιτήσεις από τυχόν πελάτες. Το πως θα εξυπηρετηθεί κάθε νέα αίτηση, ανατρέξτε παρακάτω (θέμα δημιουργίας διεργασιών).
Ως προς το πρόγραμμα του CLIENT, υπάρχουν 2 τρόποι με τους οποίους μπορεί να τρέξει: Από το terminal μπορούμε να το τρέξουμε ως

$>CLIENTOS manual
(μέσω του stdin επιτρέπουμε τον χρήστη να πληκτρολογήσει ο ίδιος το μύνημα που θα στείλει στον server, για να κλείσει εισητήρια), η αλλιώς μπορούμε να το τρέξουμε ως

$>CLIENTOS auto
(με αυτόν τον τρόπο παράγεται ένα μύνημα με τυχαία πεδία, με την χρήση της rand()).

Ο client με την χρήση της connect συνδέεται με τον server. Το μύνημα που στέλνουμε κάθε φορά από τον client στον server είναι ένα string με συγκεκριμένη μορφοποίηση:
id_pelati<SPACE>zone<SPACE>number_of_tickets<SPACE>balance<SPACE>

π.χ "1234 A 2 1500 ", όπου 1234 ο κωδικός του πελάτη, A η επιλεγμένη ζώνη, 2 ο αριθμός εισητηρίων και 1500 το υπόλοιπο λογαριασμού του πελάτη. ΠΡΟΣΟΧΗ ΣΤΑ ΚΕΝΑ!

Ο server απαντάει στον client με ένα από τα επόμενα μυνήματα σε κάθε περίπτωση (τα μυνήματα φαίνονται στο terminal window που τρέχουμε τους client):

<u>επιτυχής κράτηση:</u>
Reservation complete!
Your id: %d
Your seats (in the specified zone): %d to %d (αριθμημένες θέσεις απο, μεχρι)
Money paid: %d

<u>υπόλοιπο λογαριασμού ανεπαρκές:</u>
Your balance value can't afford the reservation!

<u>Δεν υπάρχουν αρκετές θέσης στην επιλεγμένη ζώνη:</u>
There are not enough seats in the specified zone.

<u>Το θέατρο είναι γεμάτο (επιστρέφεται σε όλους τους νέους clients ανεξεραίτως):</u>
Theatre seats are all full!

<u>Δημιουργία διεργασιών:</u> Όταν το θέατρο είναι γεμάτο ΚΑΜΜΙΑ νέα διεργασία δεν δημιουργείται! Όταν υπάρχουν ακόμα θέσεις έχουμε τα εξης: Όταν ο SERVER δέχεται αίτημα από έναν CLIENT, δημιουργείται με χρήση της fork() ένα νέο στιγμιότυπο-διεργασία του SERVER για να εξηπυρετήσει την αίτηση. Η διεργασία SERVER-γονέας, θα συνεχίσει να "ακούει" για νέα αιτήματα. Η διεργασία παιδί θα προσομοιώνει την λειτουργία του

τηλεφωνητή. Αφού ο τηλεφωνητής πρέπει "ταυτόχρονα" να συνδεθεί με τερματικό της τράπεζας και να κλείσει θέσεις, πρέπει να κάνουμε μια νέα fork() μέσα στον τηλεφωνητή. Το νέο παιδί που θα προκύψει θα προσομοιώνει την λειτουργία του τερματικού της τράπεζας και θα εκτελείται ταυτόχρονα με το "κλείσιμο θέσεων" στον τηλεφωνητή (που τώρα γίνεται διεργασία πατέρας).

Επικοινωνία διεργασιών (shared memory, signals): Η shared memory δομείται με την βοήθεια ενός struct για να έχουμε εύκολη πρόσβαση από κάθε διεργασία που έχει τα κατάλληλα δικαιώματα. Στην περίπτωση μας, που χρησιμοποιούμε fork() όλες οι διεργασίες server που δημιουργούμε έχουν τα ίδια δικαιώματα. Το struct παραπάνω έχει ως πεδία οτιδήποτε απαραίτητο για το κλείσιμο εισητηρίων (π.χ. Πλάνο θέσεων για κάθε ζώνη, λογαριασμός τράπεζας και θεάτρου και λοιπή πρόσθετη πληροφορία για την επεξεργασία αποτελεσμάτων- ανατρέξτε το αρχείο header για περισσότερες πληροφορίες). Τα signals χρησιμοποιούνται στο πρόγραμμα αυτό για την επικοινωνία μεταξύ διεργασίας τράπεζας και διεργασίας τηλεφωνητή με τον εξής τρόπο:

Ο τηλεφωνητής "περιμένει" να λάβει μύνημα από την τράπεζα για το αν συνεχίσει ή οχι. Αν η κάρτα του πελάτη είναι OK, τότε η διεργασία-τράπεζα στέλνει με την kill() ένα SIGTSTP. Ο handler που έχουμε ορίσει για το συγκεκριμένο σήμα δεν κάνει τίποτε, επιτρέποντας έτσι τον τηλεφωνητή να συνεχίσει την εκτέλεση του. Αλλιώς, η τράπεζα στέλνει στον τηλεφωνητή ένα SIGTERM, του οποίου ο handler θα ΤΕΡΜΑΤΙΣΕΙ τον τηλεφωνητή, κάνοντας τα κατάλληλα clean-up operations (αυξηση σημαφόρου τηλεφωνητών κατά 1, βλ. Παρακάτω).

Άλλη χρήση των signals: Ο τερματισμός του αρχικού προγράμματος SERVER (με την χρήση του SIGINT- περιλαμβάνεται και ελευθέρωση των ipc πόρων του λειτουργικού), η περιοδική κλήση συναρτήσεων (μεταφορά χρημάτων ανά 30 secs στον server, μύνημα συγνώμης στον πελάτη – client program) με την χρήση της alarm() και του SIGALRM.

Υλοποίηση χρόνων: Τα t_seatfind, t_cardcheck, t_wait και t_tranfer υλοποιούνται με χρήση της συνάρτησης sleep(). Σημειώνουμε εδώ ότι, όσων αφορά τα t_seatfind και t_cardcheck, για να προσομοιωθεί σωστά ο παραλληλισμός μεταξύ των διεργασιών τράπεζας και τηλεφωνητή, θα βάλουμε στην sleep() που αντιστοιχεί στην διαδικασία του τηλεφωντή t_seatfind = 4 και όχι 6. (ΑΝΑΤΡΕΞΤΕ στην επικοινωνία διεργασιών παραπάνω). Όλα τα παραπάνω προκύπτουν από την ιδέα ότι όταν έχουμε 2 διεργασίες που πρέπει να εκτελεστούν παράλληλα, και η μια τελειώνει σε 2 secs (bank), ενώ η άλλη σε 6 secs (η bank τελειώνει νωρίτερα και πρέπει να ενημερώσει τον τηλεφωντή, ενώ αυτός τρέχει), το response time που περιμένουμε ως χρήστες πρέπει να είναι συνολικά 6 secs! Για περισσότερες λεπτομέρειες δείτε τον κώδικα.

Σημείωση: Στο πρόγραμμα αυτό το t_seatfind είναι τελείως εικονικό και προστίθεται για τις ανάγκες της άσκησης μέσα στον κώδικα του τηλεφωνητή. Οι θέσεις κλείνονται πάντα με προκαθορισμένο τρόπο στο critical section της προσπέλασης στην κοινή μνήμη. Αν βάζαμε sleep(4) στο μπλοκ κώδικα που είναι κλειδωμένο με τον δυαδικό σημαφόρο θα

έπρεπε να περιμένουμε για ΚΑΘΕ ΜΙΑ διεργασία ξεχωριστά να κάνει sleep(4) ΤΗΝ ΦΟΡΑ! Αντιλαμβάνεστε ότι όταν έχουμε να τρέξουμε 500 (και παραπάνω) clients ταυτόχρονα, το throughput είναι απαγορευτικά μικρό...

Συγχρονισμός διεργασιών (semaphores): Οι πόροι που διαθέτει το σύστημα που προσομοιώνουμε είναι περιορισμένοι. Για αυτό θα χρησιμοποιήσουμε ένα σημαφόρο για τους τηλεφωνητές (ξεκινάει από την τιμή 10), με τον οποίο περικλείουμε το κωμάτι κώδικα που αντιστοιχεί στην λειτουργία του τηλεφωνητή. Αντίστοιχα για την τράπεζα (ξεκινάει από 4). Στο critical section που περιλαμβάνει προσπέλαση στην shared memory, θα χρησιμοποιήσουμε ένα mutex (2δικός σημαφόρος), ώστε ΜΟΝΟ ΜΙΑ διεργασία την προσπελαύνει κάθε φορα. Για το πως οργανώνοται τα sections τηλεφωνητών-τράπεζας-μνήμης, ανατρέξτε στον κώδικα του server.

Έλεγχος ορθότητας: Κατ αρχήν, χρησιμοποιείται ένα bash script run_tests.sh το οποίο δημιουργεί πολλούς clients ΤΑΥΤΟΧΡΟΝΑ. Για την εύρεση του ποσοστού των αποτυχημένων παραγγελιών προσμετρούνται ΜΟΝΟ οι αποτυχημένες παραγγελίες λόγω έλλειψης χρημάτων. Με αυτό τον τρόπο μπορούμε να επαληθεύσουμε το 10% αποτυχίας που μας δίνεται στην εκφώνηση (και ναι, επαληθεύεται)! Αφού το θέατρο γεμίσει (παρατηρούμε το κατάλληλο μύνημα στο terminal που τρέξαμε τους clients) μπορούμε να τερματίσουμε τον server με ctrl-C και να δούμε όλα τα τελικά αποτελέσματα. Παρακάτω έχουμε το output τερματισμού του server, αφού εκτελέσαμε sript που δημιουργεί 500 clients ταυτόχρονα (λόγω των υψηλών διαστάσεων τον οθονών του υπολογιστή, δεν εμφανίζεται καλά η screenshot εικόνα pasted στο αρχείο αυτό. Δείτε την εικόνα png στο directory του project για screenshot εκτέλεσης). Πρώτα εμφανίζεται το πλάνο για κάθε ζώνη (οι ακέραιοι αντιστοιχούν σε πελάτες), μετά ο αριθμός κλεισμένων θέσεων σε κάθε ζώνη (100, 130, 180 ,230), η τιμή των σημαφόρων συγχρονισμού (για να βεβαιωθούμε ότι έχουν ίδια τιμή με την αρχική), το ποσοστό αποτυχημένων παραγγελιών, ο μέσος χρόνος αναμονής και εξυπηρέτησης ανά πελάτη και ο πίνακας μεταφοράς χρημάτων από την εταιρία στο θέατρο, ανα 30 secs. 0 στον πίνακα αυτό σημαίνει ότι δεν μεταφέρθηκαν λεφτά μέσα σε διάστημα 30 secs.

Zone A:
6417 7103 7103 8245 8245 8245 8245 3445 4945 8633 8633 5032 5734 2142 2142 2142 42 42 42 42 4410
6629 6629 6629 105 2310 2310 3556 3556 9760 9760 9760 9760 1516 1516 1516 1516 2975 2975 2975 1190
1190 2902 2902 2902 2902 4504 4504 4504 78 78 1766 1766 1766 1766 6980 7460 7460 4211 4211 4211 767
767 2706 2706 8470 8470 8470 3942 3942 3942 3942 1036 8649 8649 8649 8649 9002 9002 6472 3189 3189
3189 7576 7576 7576 7576 3151 4712 4712 4712 4712 4172 4172 4323 4323 4323 7793 8733 9960
Zone B:
4222 4222 4222 9196 9196 9220 9220 515 515 515 515 1584 1584 1584 6697 5504 5504 5504 5504 2553
2553 2553 5700 7055 7055 7055 204 204 3815 3815 3026 3026 8611 8611 8736 8736 8736 8736 7533 7533
7533 514 514 514 514 7458 7458 7458 2594 2594 2594 6050 6050 2226 2226 6702 5312 5312 9876 6131
6131 2701 2701 2701 788 788 4157 4157 4157 8238 8238 8238 8238 8166 8166 8166 5449 6338 6338 2905
2905 2905 1843 1843 1843 5194 5194 2220 2220 2220 2220 6483 3677 3677 3677 4335 4335 8968 8968
8968 8968 7494 7494 7494 7494 8212 8212 8212 8212 4098 1638 1638 1638 1335 1335 1335 1335 9501
9208 9208 9208 3950 3950 3950 3950 2753 2753 2753 9142 9142
Zone C:

2529 1981 1981 1238 1238 7731 7731 7731 7731 7352 7352 7352 7352 8738 8738 8738 4456 4456 4456
4456 1417 1417 4117 4117 5664 5664 5664 3245 3245 3245 8263 8263 3173 3173 9410 9410 9410 9410
5891 8039 8039 9480 9480 9480 9480 5864 7011 7011 7011 7011 8576 8197 8197 5045 7633 7633 7633
1704 1704 2952 2952 2952 5448 5448 5448 5448 6671 4114 4114 4114 1919 1919 9872 9872 7915 7915
7915 7915 8170 8170 8131 9217 9217 9217 500 500 1843 1843 1843 1843 1773 1773 1773 8424 8424 8424
2992 2992 2992 2992 2143 3773 3773 3773 389 1516 1516 8315 6354 6354 6354 6354 1274 8751 8751 8751
8751 3335 5557 5557 5448 5448 5091 5091 8628 8628 1025 1025 1025 8564 8564 8564 8564 2501 2501
2501 2501 3768 3768 3768 5613 5613 5613 8993 8993 8993 1462 1462 1462 5394 5394 5394 925 3681 3681
1416 1416 1416 7670 7670 7670 7670 5923 5923 5923 322 322 5883 1072 1072 1072 1072 8979 8979 9817
9817 725 8238 6186 9480
Zone D:
8376 8376 8376 8997 8997 8997 731 9874 9874 702 702 702 165 165 165 165 6922 9524 9524 9524 5399
5399 839 839 839 1980 1980 1980 1980 1094 1094 2997 2997 2997 636 636 636 636 6782 6782 9502 9502
9502 9502 6046 872 296 296 296 6753 6753 6753 3259 4200 4200 4200 5473 5473 5473 8863 8863 8863
9228 1992 1992 1992 6091 6091 6091 5015 5015 5015 6742 6742 6742 2284 2284 2284 2284 8968 8968
8137 1058 1058 1058 1058 3866 3866 3866 3866 3405 3405 3405 3405 6481 6481 6481 3011 3856 3856
2116 2116 2404 2404 9650 5068 5068 5068 5068 2518 9582 9582 9582 9582 7902 4526 4526 4526 4526
5472 5472 5472 5472 1919 1919 3840 3840 3840 3840 3783 7577 4905 4905 4905 4905 7303 7303 7303
6882 6882 1860 1860 1797 9053 9053 9053 9053 9940 9940 3811 7725 7725 7725 7835 7835 7835 7835
6993 6993 7473 7473 7473 9663 3518 3518 5477 5477 5477 9699 2041 2041 2041 6826 6826 6826 6826
4552 3098 3098 3098 3098 7509 7509 7509 5098 5098 5098 5098 9693 9693 4237 4237 4237 8105 8105
8105 334 334 334 334 3606 6104 6104 8569 8569 8569 8569 8253 8253 8253 8253 1434 5974 6243 6243
6688 2186 7613 7613 7613 7613 7150 7150 5697 5697 8834 159 159 159 159
A:100 B:130 C:180 D:230
nthl: 10
nbank: 4
mutex: 1
Overall orders: 500
Failed orders: 56
Fail percentage: 11.200000
Customers waited to connect for (average): 140.684000 secs
Customer (average) serve time: 5.532000 secs
Money tranfered to theatre account (every 30 secs):
->2770
->3940
->4170
->3970
->3400
->2600
->800
->750
->700
->300
TOTAL MONEY GATHERED: 23400
Cya buddy!