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

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ 1
ΔΕΥΤΕΡΗ ΑΣΚΗΣΗ

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

Εισαγωγικά: Η αναφορά αυτή περιγράφει περιληπτικά το πρόγραμμα που υλοποιεί την 2η εργαστηριακή άσκηση με έμφαση στις προγραμματιστικές συμβάσεις που αποφασίστηκαν κατά την διάρκεια της ανάπτυξής της.

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 serveros2.c (περιλαμβάνει την main()), functions2.c (περιλαμβάνει ορισμούς συναρτήσεων και signal handlers) και headeros2.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. *Σημείωση: Το πρόγραμμα του client είναι το ίδιο με της 1ης άσκησης.*
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 λειτουργία 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 threads).
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 pthreads:</u> Όταν ο SERVER δέχεται αίτημα από έναν CLIENT, δημιουργείται με χρήση της pthread_create() ένα νέο thread του SERVER για να εξηπυρετήσει την αίτηση. Ο SERVER (main() thread), θα συνεχίσει να “ακούει” για νέα αιτήματα μέχρι να την κλείσουμε με το χέρι (ctrl-c). Το thread που δημιουργήθηκε 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 pthread_create() μέσα στον τηλεφωνητή. Το νέο thread που θα προκύψει θα προσομοιώνει την λειτουργία

του τερματικού της τράπεζας και θα εκτελείται ταυτόχρονα με το “κλείσιμο θέσεων” στον τηλεφωνητή.

Επικοινωνία threads: Η shared memory δομείται με την βοήθεια ενός struct. Το struct αυτό το δηλώνουμε σε global scope έτσι ώστε να είναι “ορατό” σε όλα τα threads. Δεν χρειάζεται να κάνουμε τίποτα παραπάνω, όπως στο inter process communication, γιατί τα threads μοιράζονται το ίδιο adress space. 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).
Για την επικοινωνία μεταξύ τηλεφωνητή και τράπεζας, χρησιμοποιούμε ένα νέο struct sent (δείτε το αρχείο κεφαλίδας για λεπτομέρειες). Κάθε τηλεφωνητής φτιάχνει το δικό του struct και περνάει ένα pointer σε αυτό, στην κλήση της pthread_create(), όταν δημιουργεί το thread-τράπεζα. Έτσι το thread-τράπεζα μπορεί να τροποποιήσει τις κατάλληλες τιμές και να δώσει απάντηση στον τηλεφωνητή (μέσω του struct αυτού). Σημείωση: Χρησιμοποιώντας την pthread_join() (στο σώμα της συνάρτησης του τηλεφωνητή) εξασφαλίζουμε ότι ο τηλεφωνητής ΔΕΝ θα τελειώσει νωρίτερα από την τράπεζα.

Χρήση των POSIX signals περιλαμβάνουν: Ο τερματισμός του αρχικού προγράμματος SERVER (με την χρήση του SIGINT- περιλαμβάνεται και clean-up κώδοκας), 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 nanosleep(). Σημειώνουμε εδώ ότι, όσων αφορά τα t_seatfind και t_cardcheck, για να προσομοιωθεί σωστά ο παραλληλισμός μεταξύ των threads τράπεζας και τηλεφωνητή, θα βάλουμε στην nanosleep() που αντιστοιχεί στην διαδικασία του τηλεφωντή t_seatfind = 4 και όχι 6. (ΑΝΑΤΡΕΞΤΕ στην επικοινωνία threads παραπάνω). Όλα τα παραπάνω προκύπτουν από την ιδέα ότι όταν έχουμε 2 threads που πρέπει να εκτελεστούν παράλληλα, και το ένα τελειώνει σε 2 secs (bank), ενώ το άλλο σε 6 secs (η bank τελειώνει νωρίτερα και πρέπει να ενημερώσει τον τηλεφωντή, ενώ αυτός τρέχει-περιμένει), 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 (στα posix threads χρησιμοποιούμε ένα απλο pthread_mutex) θα έπρεπε να περιμένουμε για ΚΑΘΕ ΕΝΑ thread ξεχωριστά να κάνει nanosleep(4) ΤΗΝ ΦΟΡΑ!
Αντιλαμβάνεστε ότι όταν έχουμε να τρέξουμε 500 (και παραπάνω) clients ταυτόχρονα, το throughput είναι απαγορευτικά μικρό...

Συγχρονισμός threads (mutexes – condition variables): Οι πόροι που διαθέτει το σύστημα που προσομοιώνουμε είναι περιορισμένοι. Στο threaded version του server μας, θα προσομοιώσουμε ΑΚΡΙΒΩΣ την λειτουργία των semaphores, με την χρήση mutex σε συνδυασμό με condition variables. Με λόγια, έχουμε έναν μετρητή για τα τερματικά τις τράπεζας και έναν για τους τηλεφωντές. Πριν μπεί ένα thread σε αυτά τα sections, ελέγχει τον μετρητή που αντιστοιχεί στο section. Αν υπάρχει διαθέσιμος πόρος (ενας ή περισσότεροι), τον καταλαμβάνει. Αλλιώς, περιμένει, μέχρι κάποιο άλλο thread να το ενημερώσει όταν βρεθεί ελεύθερος πόρος. Όταν ένα thread βγει από το section του τηλεφωνητή ή της τράπεζας, ο πόρος που είχε πιάσει ελευθερώνεται και στην συνέχεια ενημερώνεται ένα από τα άλλα threads που περιμένουν για πόρο (pthread_cond_sgnal). Οι “ενημερώσεις” γίνονται με την χρήση των condition variables. Το mutex το χρησιμοποιούμε για να προστατέψουμε την τιμή του μετρητή και της condition variable.

Κατά την διάρκεια εκτέλεσης του SERVER, στο τερματικό εμφανίζονται τακτικά οι τιμές των counters, τόσο για τους τηλεφωνητές, όσο και για τις τράπεζες, έτσι ώστε να είμαστε σίγουροι για την ομαλή προσομοιώση των σημαφόρων της προηγούμενης άσκησης.

ΣΗΜΑΝΤΙΚΟ: Στην υλοποίηση της άσκησης αυτής, όποτε δεσμεύεται ένας τηλεφωνητής ή τερματικό, ο αντίστοιχος counter ΑΥΞΑΝΕΤΑΙ κατά 1. Στην προηγούμενη άσκηση, όπου χρησιμοποιούσαμε sem_wait(), η τιμή του σημαφόρου μειωνόταν κατα 1. ΄Αρα, όταν πχ όλοι οι τηλεφωντές είναι ελεύθεροι, η τιμή του counter θα είναι 0 (σε αυτήν την άσκηση) και όχι 10 (όπως ήταν η τιμή του σημαφόρου)!!!

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), 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:
2396 2396 2396 5624 5624 6081 6081 6081 6081 481 481 481 7136 7136 7136 3669 3669
2618 2618 8046 8046 7728 7728 7728 373 373 373 373 4790 4790 2100 2100 3523 3523
3523 3523 2935 1810 1810 1810 1810 9304 9304 9304 7618 7618 7618 4319 4319 4319

4319 6901 6901 9137 3070 3070 2849 2849 2849 2849 3711 3711 5108 5108 5108 4110
4110 1527 1527 8065 8065 7816 7816 7816 7816 2259 2259 8165 6077 6077 823 823 4209
4209 4209 4209 350 350 350 350 1591 2442 2442 2442 1449 1449 6085 6085 1718 1718
Zone B:
8412 8412 4732 4732 4732 4732 3138 3138 3138 3138 1083 1358 1358 1358 6961 6961
6961 6961 7340 831 831 8313 8313 1541 1541 1816 5943 9362 9362 7689 7689 7689 7169
7169 7169 8442 8442 8442 8442 5692 5692 20 20 20 20 8828 7571 7571 7495 7495 7495
8747 8423 8423 8423 4740 1286 5626 5626 5626 2841 2841 2841 8723 7950 7950 7950
7950 3498 3498 3498 3498 2280 2280 2280 2280 2911 2911 2911 2911 4286 3746 3746
3746 8078 8078 8078 1607 8725 8725 8725 5463 4102 4102 9229 9229 9229 9229 7178 669
669 669 669 8728 8728 8728 1252 1252 2672 2672 2672 2672 3545 3545 3545 3545 3707
3707 3707 1842 5078 5078 5078 5078 4749 4749 4749 1979 926 926
Zone C:
9338 9338 9338 9338 731 6280 6280 6280 6280 5382 5382 5382 5382 6027 6027 8487 8487
8487 8487 5682 5682 3924 3924 3924 5911 7273 4133 4133 4133 4133 2498 2498 9228
6859 6859 6859 5107 5107 5107 8998 8998 8998 8998 9821 9821 9821 9821 3768 3768
2461 8707 8707 4094 4094 4094 4622 4622 4622 4622 5054 5054 5054 3013 3013 3013
3013 2841 2841 2841 5659 5659 8800 8800 8800 2055 2055 8477 8477 5317 2036 8846
9992 9992 3901 3901 7494 7494 396 396 396 5461 5461 3900 3900 3900 3900 8666 8666
1616 9383 9383 9094 9094 9094 6218 6218 6218 3835 3835 3835 3835 28 28 3327 3327
3327 3327 7258 874 2070 2070 2492 2492 2492 6221 7687 1471 1471 2264 2264 2264 5248
5248 5248 5248 2269 2269 2269 1334 1334 1108 1108 4291 8600 8600 9253 9253 9253
9253 2098 2098 2098 7938 7938 6912 6912 424 424 6822 6822 6822 384 8325 8449 8449
5969 5969 466 5555 4228 4228 6067 6067 6067 6067 3522 8617 9849 9849 7982
Zone D:
1580 1580 1580 6969 6969 6969 6969 3047 3047 3047 3047 2687 2687 2687 2687 7455
7455 7455 7455 5259 5259 5259 7451 7451 7451 7451 4851 4851 4851 7503 7503 7503
7503 9220 9220 2992 2992 2992 351 351 4393 4393 4393 5101 5101 705 705 1454 2786
2786 2786 5520 5520 5520 4680 7159 4516 4516 4516 4516 6010 6010 6010 6721 6721
6721 6721 2802 2802 2802 3162 3162 3162 3162 3972 3972 3972 3553 3553 4932 4932
4932 8026 8026 4159 4159 4159 3883 3883 3883 3883 2357 2357 83 2755 2755 2755 707
707 9504 9504 9504 9504 6663 9207 1657 7151 7151 7151 4501 4501 4501 4501 3029 3029
8870 8870 8870 8870 586 586 586 586 7875 7875 7875 4827 4827 728 9777 4607 4607 4607
4607 7241 7241 7241 6776 6776 910 5585 5585 4203 4203 4203 7617 7617 508 508 508
4572 4572 4572 5840 1608 1608 1608 1608 2942 2942 6624 6624 6624 6624 8939 1929
1929 1929 3135 4151 4151 4151 4151 2615 2615 2615 2615 1411 1411 906 906 8059 8059
8059 8059 8388 8388 8388 3170 4753 4753 8761 3490 3490 3490 8615 8615 8615 4623
4623 4623 4623 1893 1893 2672 2672 2672 2672 514 514 6740 6740 9460 9460 9460 561
561 561 561 1034 1034 1034 5407 5407 5407 5407 698 698 698 698
A:100 B:130 C:180 D:230
Overall orders: 463
Failed orders: 34
Fail percentage: 7.343413
Customers waited to connect for (average): 153.179266 secs
Customer (average) serve time: 6.069930 secs
Money tranfered to theatre account (every 30 secs):
->3400
->4405
->3705
->4120
->3710
->1075
->635
->800
->1450
->100
->0
TOTAL MONEY GATHERED: 23400