Λειτουργικά Συστήματα Ι

Αναστασία Παπαρροδοπούλου 3873

Χαράλαμπος Λογδανίδης 3776

Ασκηση 1

Στην άσκηση αυτή καλούμαστε να υλοποιήσουμε ένα σύστημα τηλεφωνητών για την αγορά θέσεων σε ένα θέατρο. Μια διεργασία που τρέχει στο background (server) αναλαμβάνει να βρει τις κενές θέσεις, να χρεώσει τους πελάτες και να πραγματοποίησει την κράτηση ενώ οι πελάτες συνδέονται στο server μέσω μιας διεργασίας client.

Η υλοποίηση της άσκησης έγινε με χρήση των εργαλείων:

1. **fork** για την δημιουργία καινούργιων διεργασιών.
2. **shared memory** για την αποθήκευση των κοινόχρηστων δεδομένων μεταξύ των διεργασιών.
3. **sockets** για την επικοινωνία του client με το server.
4. **semaphores** για τον συγχρονισμό των διεργασιών και την προστασία των κοινών δεδομένων.
5. **signals** για τον τερματισμό των διεργασιών, την ελευθέρωση πόρων από διεργασίες που έχουν τερματίστει (αποφυγή zombie processes), καθώς και την επικοινωνία μεταξύ διεργασιών όπως αναλύεται παρακάτω.

Shared memory

Η Shared memory αποτελείται από τις παρακάτω μεταβλητές:

1. **int company_account:** Αποθηκεύει τα χρήματα που έχει συγκεντρώσει η εταιρία.
2. **int theater_account**: Αποθηκεύει τα χρήματα που έχουν μεταφερθεί στον λογαριασμό του θεάτρου.
3. **int full:** Περιέχει τον αριθμό τον θέσεων που έχουν δεσμευτεί. Το θέατρο είναι γεμάτο αν *full==thesis_num
4. **sem_t sem_tilef:** Σημαφόρος για τους τηλεφωνητές.
5. **sem_t sem_bank:** Σημαφόρος για την τράπεζα.
6. **sem_t sem_data:** Σημαφόρος για τα δεδομένα της κοινής μνήμης.
7. **thesis_t plano:** struct για τις θέσεις του θεάτρου.
8. **statistics statist:** struct για τα στατιστικά στοιχεία που ζητούνται.

Τα struct που χρησιμοποιούνται είναι τα ακόλουθα:

thesis_t

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

statistics

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

Οπου :

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

Διεργασίες

Όλες οι διεργασίες δημιουργούνται με χρήση της συνάρτησης fork().

Κατά την εκκίνηση του, ο server δημιουργεί μια δεύτερη διεργασία για την μεταφορά χρημάτων. Η διεργασία αυτή περιμένει κάνοντας sleep για 30 δευτερόλεπτα. Έπειτα μεταφέρει τα χρήματα και επαναλαμβάνει την διαδικασία. Έτσι πριν από την σύνδεση κάποιου client ο server αποτελείται ήδη από δύο διεργασίες.

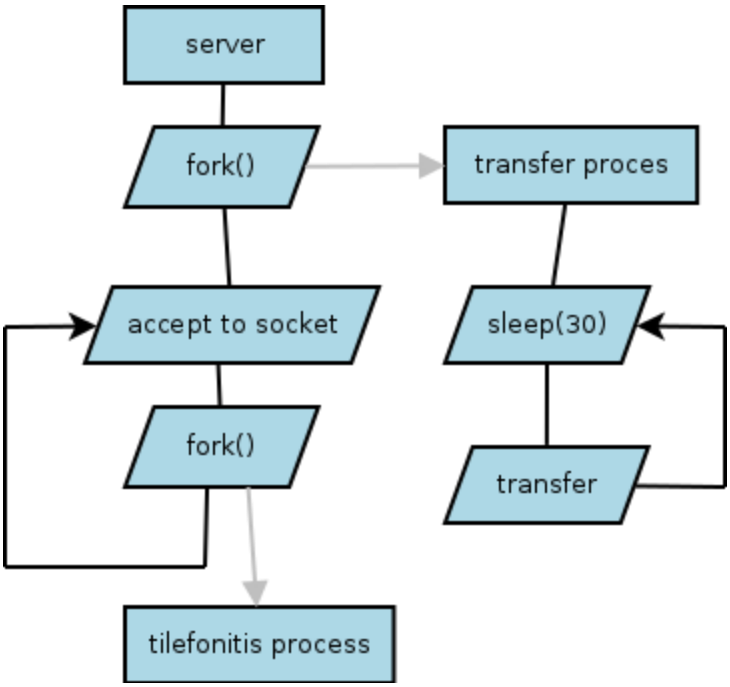
Μετά την σύνδεση του client, ο server δημιουργεί μια τρίτη διεργασία-τηλεφωνητή όπου αναλαμβάνει την εξυπηρέτηση.

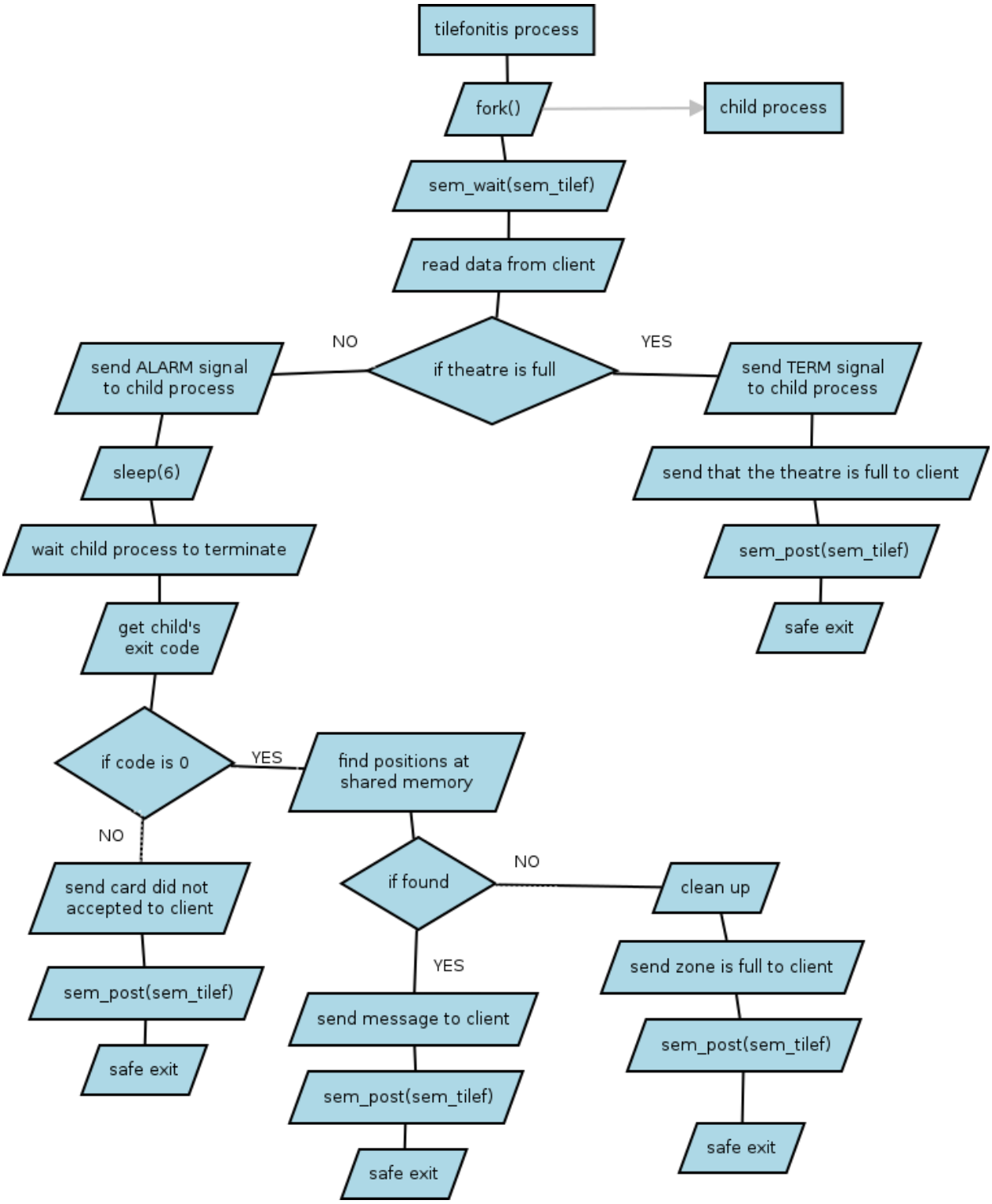
Ο τηλεφωνητής με την σειρά του δημιουργεί ένα child process που στέλνει μηνύματα στον client κάθε 10 δευτερόλεπτα ζητώντας του συγγνώμη για την καθυστέρηση. Ταυτόχρονα η διεργασία του τηλεφωνητή περιμένει την σειρά της στον αντίστοιχο σημαφόρο. Όταν έρθει η σειρά της, ο τηλεφωνητής ειδοποιεί το child process του με ένα μήνυμα alarm. Το child process λαμβάνοντας το μήνυμα ξεκινάει τον έλεγχο της κάρτας ενώ παράλληλα ο τηλεφωνητής αναλαμβάνει την εύρεση της θέσης. Το child process τερματίζει με τιμή 0 σε περίπτωση που η κάρτα γίνει αποδέκτη και με τιμή 1 στην αντίθετη περίπτωση.

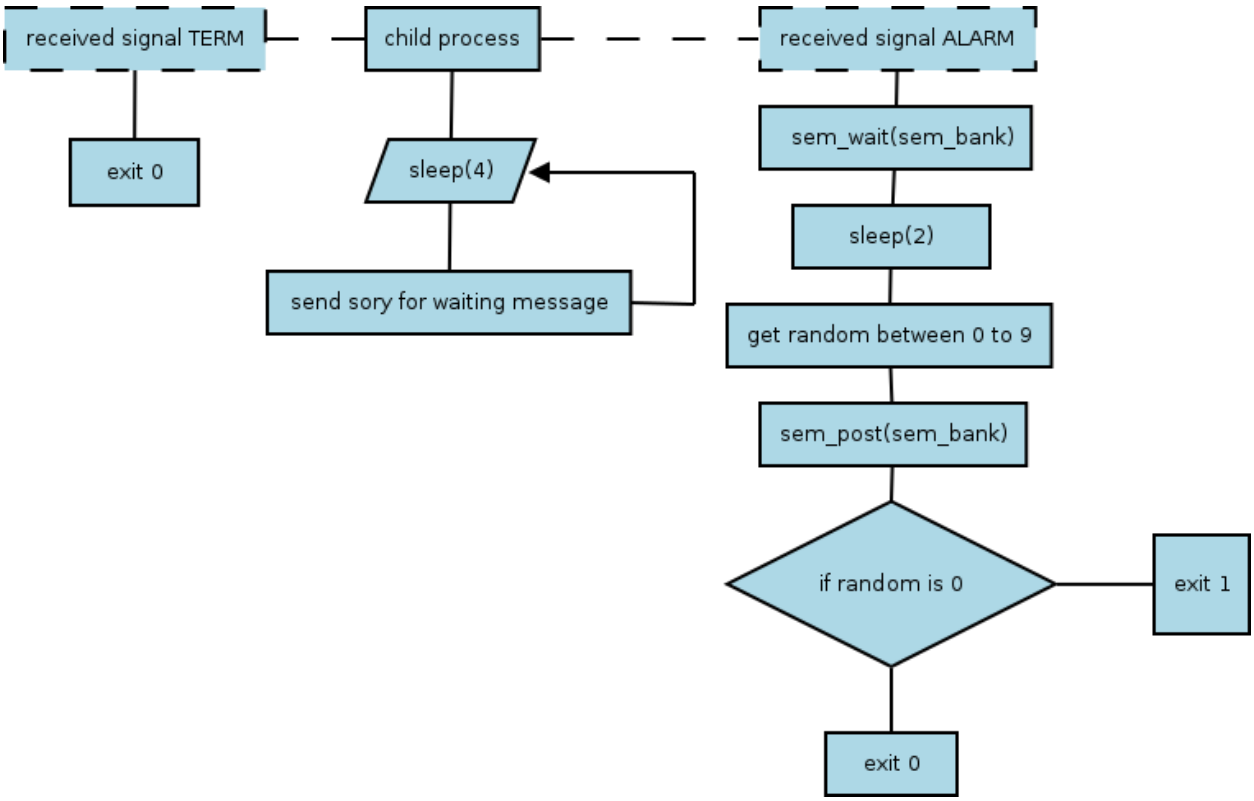
Η διεργασία-τηλεφωνητή κάνει sleep 6 δευτερόλεπτα για την εύρεση των θέσεων όπως έχει ζητηθεί. Έπειτα περιμένει το child process να τερματίσει και παίρνει το exit code του. Αν είναι μηδέν αποθηκεύει τις θέσεις στην κοινή μνήμη και ειδοποιεί τον client. Σε αντίθετη περίπτωση ειδοποιεί τον client ότι η κάρτα του δεν έγινε αποδεκτή.

Στην περίπτωση που το θέατρο είναι γεμάτο ο τηλεφωνητής δεν περιμένει για την εύρεση των θέσεων. Αντίθετα στέλνει term signal στο child process του και ειδοποιεί άμεσα τον client ότι το θέατρο έχει γεμίσει.

Παρακάτω φαίνονται τα διαγράμματα για κάθε διεργασία.


server
fork()
transfer proces
accept to socket
sleep(30)
fork()
transfer
tilefonitis process


tilefonitis process
fork()
child process
sem_wait(sem_tilef)
read data from client
if theatre is full
NO
YES
send ALARM signal to child process
send TERM signal to child process
sleep(6)
send that the theatre is full to client
wait child process to terminate
sem_post(sem_tilef)
get child's exit code
safe exit
if code is 0
YES
find positions at shared memory
NO
send card did not accepted to client
if found
NO
clean up
sem_post(sem_tilef)
YES
send zone is full to client
safe exit
send message to client
sem_post(sem_tilef)
safe exit
sem_post(sem_tilef)
safe exit



Zombie processes

Για την αποφυγή τον zombie processes χρησιμοποιούμε την waitpid. Παρόλα αυτά zombie processes δημιουργούνται προσωρινά και διαγράφονται αργότερα. Αυτό συμβαίνει καθώς η διεργασία του τηλεφωνητή περιμένει για περισσότερα δευτερόλεπτα από τη διεργασία παιδί του. Ως αποτέλεσμα υπάρχει καθυστέρηση μέχρι να πάρει το exit code του παιδιού του. Η καθυστέρηση αυτή είναι αναμενόμενη και σε κάθε περίπτωση οι πόροι ελευθερώνονται.

Semaphores

Για την άσκηση χρησιμοποιήθηκαν τρεις σημαφόροι. Ο sem_tilef, ο sem_bank και ο sem_data.

Ο sem_tilef αντιπροσωπεύει τους τηλεφωνητές και αρχικοποιείται στο 10. Με αυτόν τον τρόπο μόνο 10 διεργασίες τηλεφωνητών μπορούν να είναι ενεργές κάθε φορά. Οι υπόλοιπες διεργασίες περιμένουν σε μια ουρά προτεραιότητας.

Ομοίως ο sem_bank αντιπροσωπεύει την τράπεζα που ελέγχει τις πιστωτικές κάρτες. Ο sem_bank είναι αρχικοποιημένος στο 4 και μόνο 4 διεργασίες ελέγχουν τις πιστωτικές κάρτες κάθε φορά.

Τέλος ο sem_data ελέγχει την πρόσβαση στην κοινή μνήμη.

Είναι αρχικοποιημένος στο 1 έτσι μόνο μια διεργασία να έχει πρόσβαση στα κρίσιμα δεδομένα κάθε φορά.

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

Παραδείγματος χάρη:
./client 3 A 124

Αρχεία

1. **server.c :** Ο κώδικας του server.
2. **client.c :** Ο κώδικας του client.
3. **header.h :** Κοινό header μεταξύ server και client.
4. **utils.c :** Κοινές συναρτήσεις μεταξύ server και client για δημιουργία τυχαίων αριθμών, ανάγνωση και εγγραφή καθώς και sleep χωρίς να επηρεάζεται από interrupts.
5. **Makefile :** Αρχείο με οδηγίες για το compile του κώδικα.
6. **run_tests.sh :** Αρχείο για την εκτέλεση του server και του client.
7. **Report.pdf**.

Αποτελέσματα

Ποσοστό αποτυχίας: 0.622864
Μέσος χρόνος αναμονής: 5.635828
Μέσος χρόνος εξυπηρέτησης: 5.946965
Μεταφορές:1130 1520 1490 1525 1580 1475 1440 1525 1525 1570 1420 1210 780 390 310 390 390 350 480 430 440 350 280 150 150 100 200 150 150 200 100 100 100

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.