Φροντιστήριο: 19/10/2015
Παράδοση: 6/11/2015

Treasure Room Simulation

Σε αυτήν την άσκηση θα εξοικειωθείτε με τη δημιουργία και το χειρισμό threads, shared memory, και UNIX sockets για την επικοινωνία μεταξύ διαφορετικών διεργασιών, καθώς επίσης και με semaphores/mutexes για τον συγχρονισμό διαφορετικών διεργασιών και τον αμοιβαίο αποκλεισμό.

Συγκεκριμένα, θα χρειαστεί να κατασκευάσετε ένα πρόγραμμα που θα προσομοιώνει την λειτουργία ενός παιχνιδιού σε μία αίθουσα θησαυρού όπου δύο ομάδες πρέπει να ανταγωνιστούν για να νικήσουν. Ο προσομοιωτής αυτός θα έχει τα εξής χαρακτηριστικά:
Την αίθουσα θησαυρού (server) και δύο ομάδες (clients), την ομάδα Α και την ομάδα Β. Σκοπός του παιχνιδιού είναι κάποια ομάδα να προλάβει να αδειάσει την αίθουσα που περιέχει Ν χρυσά νομίσματα.

> Ο αριθμός Ν χρυσών νομισμάτων θα δίνεται από την κονσόλα:
> π.χ.: ./treasure_room 100

Αρχικά, ο server θα δημιουργεί ένα UNIX socket και θα περιμένει κάποιον client να συνδεθεί (χρησιμοποιώντας τις κλήσεις συστήματος socket(), bind(), listen() και accept()). Ο server μπορεί να δέχεται παράλληλα πολλούς clients με τη χρήση threads.

Η αίθουσα του θησαυρού θα γεννά ένα κλειδί (srand) που θα οδηγεί σε shared memory. Κάθε ομάδα στη συνέχεια θα επιχειρεί να εισέλθει. Για να εισέλθει θα πρέπει να παράγονται δύο τυχαίοι αριθμοί Α και Β από (1 έως 10). Αρχικά η κάθε ομάδα που επιχειρεί να μπει θα περιμένει με την χρήση της sleep() για Α δευτερόλεπτα, ώστε καμία ομάδα να μην εκμεταλλεύεται μονίμως την αίθουσα. Μόλις ξυπνήσει θα εισέρχεται στην αίθουσα για Β δευτερόλεπτα και θα αφαιρεί ένα νόμισμα για κάθε δευτερόλεπτο που είναι μέσα.

Για την πρώτη φορά που κάποια ομάδα θα προσπαθήσει να μπει θα πρέπει να ζητήσει πρώτα το κλειδί. Για να συνδεθεί κάποια ομάδα, θα χρησιμοποιεί ένα UNIX socket (με κλήσεις συστήματος socket() και connect()) για να ζητήσει το κλειδί της από την αίθουσα. Αφού λάβει από το socket, (κλήση συστήματος recv()) τότε θα πρέπει να κάνει attach το shared memory (επίσης με τις κλήσεις σε shmget() και shmat()) χρησιμοποιώντας και το κλειδί της αίθουσας.

Επειδή **δεν γίνεται να είναι δύο ομάδες ταυτόχρονα μέσα** θα πρέπει επίσης να ελέγχεται εάν η αίθουσα είναι άδεια ή βρίσκεται η άλλη ομάδα μέσα.

Συγχρονισμός ομάδων:

Όπως είπαμε δεν γίνεται να είναι ταυτόχρονα δύο ομάδες στην αίθουσα θησαυρού. Για αυτό το λόγο, όταν τρέχουν και οι δύο ομάδες και προσπαθούν να μπουν στην αίθουσα, πρέπει να συγχρονίζονται σωστά έτσι ώστε μόνο όταν έχει τελειώσει ο χρόνος παραμονής της μίας ομάδας, να μπορεί να μπει η άλλη, ή να επιχειρήσει να ξαναμπεί η ίδια. Για να διασφαλίσουμε τα παραπάνω κάνουμε χρήση semaphores/mutexes.

Όσο κάθε ομάδα είναι μέσα στην αίθουσα του θησαυρού θα αφαιρεί νομίσματα. Για να γνωρίζουμε ποιος είναι ο νικητής θα πρέπει να γνωρίζουμε ποια ομάδα βρίσκεται μέσα, τον αριθμό των νομισμάτων που έχει η αίθουσα και πόσα νομίσματα έχει πάρει κάθε ομάδα. Η ομάδα που θα

συγκεντρώσει τα περισσότερα νομίσματα κερδίζει. Πριν τερματιστεί ο server θα πρέπει να κλείνει το UNIX socket, καθώς και να αποδεσμεύει σωστά το shared memory segment.

Παράδειγμα για N = 10
Treasure room state:
Team A: 9
Team A: 8
Team A: 5
Team B: 4
...
Team A: 0

Οδηγίες
Treasure_room:
- Generate_key ()
- Room key is the shared memory segment key

While (coins > 0)
- Listen teams attempting to enter room
- If anyone makes it, pass the key
- Check number of coins

Say who the winner is and end game.
(Remember to close sockets and detach shared memory segments)

Team_a , team_b:
- Ask for key (first time only)
- If room is empty attempt to enter
  lock room
  coins --
  unlock room
  else -> wait

Reference - man pages
Ένα man page περιγράφει τον τρόπο λειτουργίας ενός προγράμματος, ενός system call ή μιας library function. Η εμφάνιση ενός man page γίνεται με τη χρήση της εντολής:

man(1)

Για να δείτε το man page (σε Linux) που αναφέρεται στη foo(N), κάνετε:

% man -S N foo

Ο συμβολισμός foo(N) αναφέρεται στο man page που περιγράφει τη foo στη κατηγορία (section) 'N'.

Λίστα με χρήσιμα man pages για την Άσκηση 2
Σας παραθέτουμε man pages με system calls που μπορεί να χρειαστείτε για την υλοποίηση της άσκησης. Η παρακάτω λίστα δεν είναι δεσμευτική. Μπορείτε να χρησιμοποιήσετε και εναλλακτικούς τρόπους.

socket(2)
bind(2)

```
listen(2)
accept(2)
connect(2)
read(2)
recv(2)
send(2)
write(2)
shutdown(2)
close(2)
shmget(2)
shmat(2)
shmctl(2)
shmdt(2)
pthread_create(3)
pthread_join(3)
pthread_exit(3)
sem_init(3)
sem_destroy(3)
sem_post(3)
sem_wait(3)
sem_getvalue(3)
pthread_mutex_lock(3)
pthread_mutex_unlock(3)
pthread_mutex_init(3)
pthread_mutex_destroy(3)
```

Παρατηρήσεις

1. Η άσκηση είναι ατομική. Τυχόν αντιγραφές μπορούν να ανιχνευθούν εύκολα από κατάλληλο πρόγραμμα και θα μηδενιστούν. Συμπεριλάβετε το όνομα σας και το λογαριασμό σας (account) σε όλα τα αρχεία.
2. Κατασκευάστε ένα αρχείο Makefile, έτσι ώστε πληκτρολογώντας make all να γίνεται η μεταγλώττιση (compilation) του προγράμματος και να παράγεται το εκτελέσιμο αρχείο. Επίσης πληκτρολογώντας make clean να καθαρίζονται όλα τα περιττά αρχεία, και να μένουν μόνο τα αρχεία που χρειάζονται για τη μεταγλώττιση.
3. Επιπλέον, γράψτε και ένα αρχείο readme.txt το πολύ 30 γραμμών που να περιέχει επεξηγήσεις για τον τρόπο υλοποίησης .
4. Τοποθετήστε σε ένα κατάλογο όλα τα αρχεία που χρειάζονται για την άσκηση 2. Παραδώστε τα παραπάνω αρχεία χρησιμοποιώντας το πρόγραμμα turnin (πληκτρολογήστε turnin assignment_2@hy345 *directory_name* από τον κατάλογο που περιέχει τον κατάλογο *directory_name* με τα αρχεία της άσκησης).
5. Σε πολλές περιπτώσεις τα ονόματα των συναρτήσεων βιβλιοθήκης είναι ενδεικτικά. Μπορείτε να χρησιμοποιήσετε όποια σας βολεύουν.