ΣΤΑΥΡΙΝΑΚΗΣ ΠΑΝΑΓΙΩΤΗΣ
A.M.: 236217
email: stavrinakis@ceid.upatras.gr
Όλα τα ερωτήματα είναι λυμένα και λειτουργούν σωστά.
Το πρόβλημα μας ήταν το ότι έτρεχαν ταυτόχρονα οι διεργασίες, με αποτέλεσμα να μην παράγεται το επιθυμητό αποτέλεσμα. Για το πρώτο ερώτημα που απαιτεί semaphores χρησιμοποιώ sem_open() για να ανοίξω μια ήδη υπάρχουσα. Στο πρώτο πεδίο ορίζουμε το όνομά της, στο δεύτερο το O_CREAT ώστε να τη δημιουργήσει αν δεν υπάρχει, στο τρίτο τα permissions(που απαιτούν και το <sys/stat.h>) και στο τέταρτο την τιμή της σεμαφόρας. Με την sem_wait() μειώνεται η τιμή της semaphore, άμα γίνει 0 τότε είναι η διεργασία έτοιμη για εκτέλεση, ενώ αν είναι -1 μπλοκάρεται. Μόλις τελειώσει την εκτέλεση της η διεργασία με το sem_post() αυξάνουμε την τιμή της semaphore και προχωράμε στην επόμενη. Με αυτόν τον τρόπο οι εκτελέσεις γίνονται ξεχωριστά χωρίς να υπάρχει πρόβλημα. Τέλος, κλείνουμε τη σεμαφόρα με sem_close(), με sem_unlink() σβήνουμε το όνομα της για να μην έχει διατηρηθεί όταν την ξαναχρησιμοποιήσουμε διότι θα δημιουργήσει προβλήματα στην εκτέλεση και με sem_destroy() την καταστρέφουμε. Επίσης, έβαλα τις πρόσθετες εντολές μέσα στη for() ώστε να υπάρχει πιθανότητα να εμφανιστούν εναλλάξ και όχι πρώτα η μια και μετά η άλλη. Στον δεύτερο κώδικα η μόνη διαφορά είναι ότι μόλις εκτελεστεί η μια διεργασία στην συνέχεια να εκτελεστεί οπωσδήποτε η δεύτερη. Οπότε, φτιάχνουμε 2 σεμαφόρες. Ενεργοποιείται η πρώτη και μόλις τελειώσει αντί να αλλάξουμε την τιμή της ενεργοποιούμε την δεύτερη. Για αυτό και ορίζουμε την μια στην τιμή 0 ώστε με sem_post() να γίνει 1 και όχι 2. Για το δεύτερο ερώτημα χρησιμοποιήσαμε pthreads. Παρόμοια με τις σεμαφόρες μόνο που απαιτεί functions για να λειτουργήσει. Το pthread_mutex_lock() παρόμοιο με το sem_wait() όπως και το pthread_mutex_unlock() με το sem_post(), μέχρι να γίνει unlock δεν εκτελείται άλλη διεργασία. Δημιουργούμε το thread με την εντολή pthread_create(), το πρώτο πεδίο το όνομα, το δεύτερο τα τυχόν χαρακτηριστικά, το τρίτο το start_routine και το τέταρτο το argument. Στη προκειμένη περίπτωση δεν τα χρειαζόμαστε και για αυτό είναι NULL. Το pthread_join() περιμένει για το thread να τερματιστεί και μετά επιστρέφει. Ο δεύτερος κώδικας θέλει πάλι 2 threads, όμως εδώ χρησιμοποιούμε condition για να ελέγξουμε ποιο thread εκτελείται και να μην υπάρξει race condition.