Λειτουργικά Συστήματα

Άσκηση 2

Ονοματεπώνυμο : Κυριακού Ανδρόνικος

AM : 5806

email: kyriakou@ceid.upatras.gr

Τα τμήματα της άσκησης τα οποία υλοποιήθηκαν και περνάνε με επιτυχία όλα τα τεστ του αυτόματου διορθωτή είναι τα syn_process_1 , syn_process_2 , syn_thread_1 και syn_thread_2. Παρακάτω καταγράφονται με περισσότερες λεπτομέρειες οι υλοποιήσεις και τα προβλήματα τα οποία παρουσιάστηκαν κατά την διάρκεια επίλυσης.

Syn_process_1

Το ερώτημα αυτό χρησιμοποιεί δυο διεργασίες (πατέρας , παιδί) προκειμένου να προσπελαστεί ένας κοινός πόρος (η συνάρτηση display) και να εκτυπωθούν μηνύματα στην κονσόλα. Για να αποφευχθούν race conditions μεταξύ των δυο διεργασιών επιλέχθηκαν οι σημαφόροι ως εργαλείο συγχρονισμού. Είναι γνωστό ότι οι σημαφόροι σε λειτουργικά συστήματα τύπου Unix λειτουργούν ατομικά , δηλαδή όλες οι λειτουργίες που γίνονται πάνω σε αυτούς δεν διασπώνται αλλά τερματίζονται πριν ξεκινήσει κάποια άλλη διεργασία. Παράλληλα ,γνωρίζουμε ότι οι αφαιρετικές αλλαγές που γίνονται στις τιμές του σημαφόρου , υλοποιούν εσωτερικά μια δομή επιλογής. Πιο συγκεκριμένα όταν γίνει προσπάθεια αφαίρεσης τιμής από την τιμή του σημαφόρου, πρέπει το τελικό αποτέλεσμα να είναι τουλάχιστον 0 αλλιώς η διεργασία μπαίνει σε κατάσταση αναμονής, ενώ όταν προστίθεται τιμή , δεν υλοποιείται κάποιος έλεγχος. Έτσι , με βάση τα παραπάνω , η υλοποίηση που έγινε , αρχικοποιεί ένα σημαφόρο στο 1 (γίνεται με χρήση της δομής τύπου sembuf up καθώς οι σημαφόροι που δημιουργούνται σε Unix έχουν αυτόματα αρχική τιμή 0 , διαφορετικά μπορούσε να γίνει με χρήση της SETVAL) και κάθε διεργασία πριν καλέσει την συνάρτηση display προσπαθεί να κάνει down τον σημαφόρο . Αν η άλλη διεργασία δεν έχει τελειώσει με την χρήση της display , τότε η τιμή του σημαφόρου θα είναι 0 και η πρώτη διεργασία θα μπλοκαριστεί. Τέλος κάθε διεργασία αφού τελειώσει με την χρήση της display κάνει τον σημαφόρο up προκειμένου να επιτρέψει την συνέχεια του προγράμματος.

Syn_process_2

Η λογική υλοποίησης του ερωτήματος αυτού είναι κοινή με το παραπάνω , δηλαδή οι σημαφόροι χρησιμοποιήθηκαν με τον ίδιο ακριβώς τρόπο και για τους ίδιους λόγους . Η μόνη διαφοροποίηση είναι ότι θέλουμε η διεργασία πατέρας να στέλνει ένα σήμα στο παιδί ότι το ab τυπώθηκε και να του επιτρέπει να τυπώσει το cd μια μόνο φορά. Το παραπάνω μπορεί να επιτευχθεί απλά με χρήση μιας μεταβλητής

ακεραίου η οποία παίρνει 2 τιμές . Το πρόβλημα που υπήρξε ήταν ότι κάθε διεργασία έχει δικό της αντίγραφο των μεταβλητών και άρα δεν μπορούσαν να παρατηρηθούν από την μία αλλαγές που έκανε η άλλη. Η λύση του προβλήματος ήταν η χρήση shared memory στην οποία αποθηκεύτηκε μια ακέραια μεταβλητή την οποία κάθε διεργασία τροποποιούσε μετά την κλήση της display. Όσο κάθε διεργασία δεν κατείχε τη CPU , κοιμόταν με χρήση της usleep για 1 microsecond και μετά έκανε πάλι έλεγχο μήπως ήταν η σειρά της. (Σημ : Προτιμήθηκε η usleep σε σχέση με την sleep για να μην μένει ο επεξεργαστής άπραγος για μεγάλο διάστημα.)

Syn_thread_1

Για το ερώτημα αυτό , μετατράπηκε ο κώδικας που δινόταν προκειμένου να υλοποιεί την ίδια λειτουργικότητα με χρήση threads . Δημιουργήθηκε μια συνάρτηση (printer) η οποία καλείται από τα δυο δημιουργημένα threads κάθε φορά με διαφορετικό όρισμα. Αυτό έγινε προκειμένου να εισαχθεί η χρήση των mutexes. Όπως και στο syn_process_1 , πριν από την χρήση της συνάρτησης display η οποία αποτελεί τον κοινό πόρο , γίνεται ένα κλείδωμα του mutex και μετά από την χρήση ένα ξεκλείδωμα. Έτσι , αν την ώρα που το ένα thread χρησιμοποιεί την display, το δεύτερο προσπαθήσει να την χρησιμοποιήσει , θα δει ότι είναι κλειδωμένη και θα ξαναπέσει σε ύπνο.

Syn_thread_2

Τέλος , η λογική του ερωτήματος αυτού είναι όμοια με του syn_process_2 , δηλαδή ότι το ένα thread πρέπει να δώσει σήμα στο άλλο ότι τερμάτισε την εκτέλεση του και ότι ήρθε η σειρά του. Για την υλοποίηση αυτού του ερωτήματος , δημιουργήθηκαν δυο συναρτήσεις οι οποίες έχουν παρόμοια λογική αλλά καθορίζουν την λειτουργία των δυο διαφορετικών threads. Σε αυτή την περίπτωση δεν εμφανίστηκε το θέμα των μη κοινών μεταβλητών καθώς τα threads έχουν τον ίδιο χώρο διευθύνσεων μεταξύ τους και έτσι μια απλή δήλωση μιας global μεταβλητής αρκεί για να μπορεί να προσπελαστεί και από τα δύο threads. Αυτή η global μεταβλητή αποτελεί και τον κοινό πόρο στον οποία οι condition variables αναφέρονται. Όσο αυτή η μεταβλητή είναι στην μια κατάσταση το ένα thread θα χρησιμοποιήσει την display και μετά με χρήση της pthread_cond_signal θα ξυπνήσει το άλλο thread. Ύστερα , με χρήση της pthread_cond_wait , το πρώτο thread θα κοιμηθεί μέχρι να τελειώσει την λειτουργία του το δεύτερο thread. Όλα τα παραπάνω φυσικά δεν θα λειτουργούσαν αν δεν υπήρχε ένας mutex προκειμένου να αποκλείει την ταυτόχρονη προσπέλαση – τροποποίηση της κοινής μεταβλητής.