ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

Λειτουργικά Συστήματα Ι

Εργασία 2

Φώτιος Διονυσόπουλος ΑΜ:5753

Email: dionys@ceid.upatras.gr

Εισαγωγικά

Σκοπός της 2^{ης} εργασίας ήταν ο συγχρονισμός διεργασιών και νημάτων έτσι ώστε να εκτυπώνονται μηνύματα στην έξοδο με τη σωστή σειρά.

Ο συγχρονισμός των διεργασιών έγινε με Semaphores , ενώ ο συγχρονισμός των Threads με Mutexes και Condition Variables.

Syn_Process_1 :

Περιγραφή λογικής :

Αρχικά, οι σεμαφόροι χρησημεύουν σε δύο σενάρια:

-Για αμοιβαίο αποκλεισμό

-Για συγχρονισμό διεργασιών

Στην περίπτωση του Syn_Process_1 οι σημαφόροι χρησημοποιήθηκαν για αμοιβαίο αποκλεισμό. Επομένως, δίνουμε αρχική τιμή στη σεμαφόρο την τιμή 1(χρησημοποιώντας τη συνάρτηση semctl με τις κατάλληλες παραμέτρους) .
Στη συνέχεια, κάνουμε “fork” την βασική διεργασία.Μπαίνοντας στο critical region, καταβάζουμε τη σεμαφόρο. Σε αυτή την περίπτωση το region αυτό είναι η συνάρτηση Display , ηοποία βρίσκεται και στον γονέα και στο παιδί. Άρα, πρίν μπούμε στη κρίσημη περιοχή κατεβάζουμε τη σεμαφόρο και όταν βγούμε , την ανεβάζουμε.

Πρέπει να σημειωθέι οτι αφού δεν μας ενδοιαφέρει πιο μήνυμα θα εκτυπωθεί πρώτο χρειαζόμαστε μόνο μία σεμαφόρο και όχι δύο.

Τέλος, με τις κατάλληλες παραμέτρους(IPC_RMID) στη συνάρτηση semctl καταστρέφουμε τη σημαφόρο.

Προβλήματα & Λύσεις :

Το βασικότερο πρόβλημα σε αυτή τη υλοποίηση ήταν η εύρεση των απαραίτητων συναρτήσεων και παραμέτρων του , καθώς και ο καθορισμός του Critical Region.

Syn_process_2 :

Περιγραφή λογικής :

Στο δεύτετο ερώτημα σκοπός ήταν η εκτύπωση του μηνύματος “abcd” 10 φορές .Η βασική διαφορά με το προηγούμενο ήταν οτι σε αυτή την περίπτωση έπρεπε να λειτουργούν δύο σημαφόροι έτσι ώστε να εκτυπώνεται πρώτα το ab (γονέας) και έπειτα το cd (παιδί).

Στην αρχή κάνουμε 1 την τιμή του σεμαφόρου του γονέα έτσι ώστε να εκτυπωθεί πρώτα το “ab” και όχι το “cd”.Μπαίνοντας στο Critical Region όπου βρίσκεται το “ab”, μέσα στην For, κατεβάζουμε τη σεμαφόρο του παιδιού έτσι ώστε να μην μπορεί να εκτελεστεί και αφού εκτυπωθεί ένα “ab” , ανεβάζουμε τη σεμαφόρο του παιδιού έτσι ώστε να τρέξει το παιδί.Το παιδί “ειδοποιείται” και κατεβάζει τη σεμαφόρο του γονέα ετσι ώστε να μην εκτυπωθεί το “ab”, εκτυπώνει στο stdout το “cd” και στη συνέχεια ανεβάζει τη σεμαφόρο του γονέα.Η διαδικασία αυτή εκτελείτε 10 φορές, δίνοντας το επιθυμητό αποτέλεσμα.

Προβλήματα & Λύσεις :

Το βασικότερο πρόβλημα σε αυτή τη υλοποίηση ήταν, όπως και στο Syn_process_1 , η εύρεση των απαραίτητων συναρτήσεων και παραμέτρων του , καθώς και ο καθορισμός του Critical Region.Επίσης, σημαντικό πρόβλημα ήταν ηαποφυγή Race Condition, καθώς θέλουμε να εμφανίζεται πρώτα το “ab”.Λύση σε αυτό το πρόβλημα έδωσε η χρήση δύο σημαφόρων.

Syn_Thread_1 :

Το 3^{ο} μέρος της άσκησης διαφοροποείται αρκετά απο τα δύο προηγούμενα αφού αυτή τη φορά χρησημοποιήθηκαν Threads αντί για Processes.

Στο 3^{ο} μέρος χρησημοποιήθηκε ένα Mutex (δυαδική σεμαφόρος) .Αρχικά, δημιουργούνται δύο Thread, το ένα συνδέεται με τη συνάρτηση που εκτυπώνει το “ Hello world” και το άλλο με τη συνάρτηση που εκτυπώνει το “ Kalimera kosme”.

Μόλις, δημιουργηθούν τα Threads επικρατεί Race Condition και τα δύο threads “αγωνίζονται” προκειμένου να αποκτήσουν τη σεμαφόρο.Όταν κάποιο απο τα δύο Threads εκτελέσει τη συνάρτησή του, τότε κλειδώνει το mutex(pthread_mutex_lock) και εκτυπώνει, μέσω της display, το επιθυμητο μήνυμα.Μόλις το εκτυπώσει , ξεκλειδώνει τη σεμαφόρο(pthread_mutex_unlock).

Προβλήματα & Λύσεις :

Το βασικότερο πρόβλημα ήταν ο χειρισμός του mutex.Τη λύση σε αυτό το πρόβλημα έδωσαν οι συναρτήσεις pthread_mutex_lock και pthread_mutex_unlock “κλειδώνουν” και “ξεκλειδώνουν” τη σημαφόρο αντίστοιχα.Επίσης, πρόβλημα ήταν το γεγονός οτι τα threads(lightweight process) χρησημοποιούν όλα τις πηγές(resources) της διεργασίας.Χρησημοποιούν δηλαδή: κοινή μνήμη, αρχεία κτλ. , κάτι που δυσχεραίνει την κατάσταση.

Syn_Thread_2 :

Στο 4^{ο} και τελευταίο μέρος της άσκησης, σκοπός ήταν, όπως και στο Syn_Process_2 , η εκτύπωση του “abcd” 10 φορές.

Προφανώς, πρόκειται, ξανά, για το πρόβλημα παραγωγού-καταναλωτή.Χρησημοποιήθηκαν ένα Mutex, καθώς και δύο Condition Variable.

Η διαδικασία εκτέλεσης:

- Ο παραγωγός “κοιμάται” όσο ο Buffer είναι γεμάτος και “ξυπνάει” τον καταναλωτή όταν σταματά να είναι άδειος
- Ο καταναλωτής “κοιμάται” όσο ο Buffer είναι άδειος και “ξυπνάει” τον παραγωγό όταν σταματά να είναι γεμάτος

Αναλυτικότερα :

Η μία Condition Variable χρησημοποιήθηκε για να “ξυπνάει “ τον παραγωγό , και η άλλη για τον καταναλωτή.Πρίν εκτυπωθεί το “ab” , κλειδώνουμε τη σημαφόρο και αμέσως μετά κλειδώνουμε την Condition Variable(pthread_cond_wait).Χρησημοποείται επίσης, ένας Buffer ο οποίος αποθηκεύει πληροφορία(τον αριθμο της επανάληψης).Όσο ο Buffer είναι διάφορος του μηδενός και αφού εκτυπωθεί το μήνυμα , απελευθερώνεται το Thread και στη συνέχεια ξεκλειδώνεται η σεμαφόρος(pthread_mutex_unlock).Έπειτα, ο καταναλωτής παίρνει τη σκυτάλη και επαναλαμβάνει την παραπάνω διαδικασία, αλλά με τη διαφορά οτι, αυτή τη φορά, περιμένει όσο ο Buffer είναι ίσος με το μηδέν .

Προβλήματα & Λύσεις :

Ένα πρόβλημα στην παραπάνω διαδικασία είναι το γεγονός ότι ο παραγωγός και ο καταναλωτής μπορεί να προσπελάσουν ταυτόχρονα τον μετρητή(της επανάληψης) και να “χαθούν ” κάποιες επαναλήψεις, ουσιαστικά ”χάθηκε” το Wakeup call.Τη λύση σε αυτό, δίνει η σημαφόρος η οποία δεν επιτρέπει να αλλάξουν ταυτόχρονα την τιμή του μετρητή ο παραγωγός και ο καταναλωτής.