Στην παρούσα αναφορά τα *mutliproc/multithread_1* και *multiproc/multithread_2* εχουν ομαδοποιηθεί στην εξηγησή τους καθώς ακολουθούν σχεδόν όμοια λογική και υλοποίηση.

Multiproc/Multithread 1

Η αρχική δοκιμαστική υλοποίηση του *multiproc_1* ήταν μονάχα η δημιουργία ενός σταθερού αριθμού διεργασιών που καλούσαν την *display* με παράμετρο το όνομα του εκτελέσιμου (*argv[0]*). Σκοπός της δοκιμής αυτής ήταν η επιβεβαίωση της ιδιότητας που έχουν οι διεργασίες παιδια να κληρονομούν τις εντολές εισόδου του κυρίου προγράμματος. Καθώς όμως δεν υπήρχαν ακόμη semaphores για συγχρονισμό, οι χαρακτήρες εμφανίζονταν ανακατεμένοι όπως ακριβώς προέβλεψε η εκφώνηση της άσκησης.

Για τον συγχρονισμό των διεργασιών χρησιμοποιήθηκε named semaphore (με την κλήση της *sem_open*), εφόσον υπάρχει ανάγκη για πρόσβαση απο διεργασίες-παιδιά. Καθώς ο semaphore που χρησιμοποιούμε παίρνει μόνο δυο τιμές (δηλαδή είναι binary semaphore) και η ουσιαστική λειτουργία του είναι όμοια με αυτή ενός mutex, ονομάστηκε mutex.

Έχοντας λύσει το κύριο πρόβλημα της άσκησης, το μόνο που απέμεινε είναι η πλήρης υλοποίηση της *multiproc_1*. Αυτό επιτυγχάνεται με την εξής λογική: ο αριθμός των εντολών εισόδου που βρίσκονται στον πίνακα *argv* είναι *argc*. *argv[0]* είναι το όνομα του εκτελέσιμου και *argv[1]* ο αριθμός των επαναληπτικών κλήσεων της *display* απο κάθε διεργασία παιδί. Συμπερασματικά ο αριθμός των διεργασιών που θέλουμε να δημιουργήσουμε είναι *argc-2*. Καθε διεργασία n καλεί την *display* για το *argv[n+2]*.

Σημαντικό επίσης είναι και ο καθορισμός της κρίσιμης περιοχής μέσα στις διεργασίες. Εφόσον δέν μας ενοχλεί οι διεργασίες να εμφανίζουν τα αποτελέσματα τους πλεκτά (π.χ για *./multiproc_1* 3 University Patras να εμφανίστει μια φορά “University”, τρείς φορές “Patras” και τέλος δύο φορές “University”), η κρίσιμη περιοχη βρίσκεται εντός του loop που καλεί επανειλλημένα την *display* και οχι εκτός.

Με ακριβώς την ίδια λογική υλοποιείται και η *multithread_1*, με την διαφορά οτι αντί για binary semaphore χρησιμοποιούμε mutex που παρέχεται απο τα pthreads. Επίσης αφού τα thread σε αντίθεση με τις διεργασίες δεν κληρονομούν τις μεταβλητές που ορίζονται στην *main* και η αρχική συνάρτηση που εκτελεί το thread δέχεται μόνο ένα όρισμα, προκύπτει η ανάγκη ορισμού μίας struct ως όρισμα της αρχικής συνάρτησης του thread που θα περιλαμβάνει τον αριθμο των επαναληπτικών κλήσεων της *display* (*argv[1]*) και την λέξη που θα εμφανίζει η *display*(*argv[n]*).

Multiproc/Multithread 2

Τα συγκεκριμένα προγράμματα έχουν όμοια λειτουργία με τα προηγούμενα, με την προσθήκη της κλησης της συνάρτησης *init* για την οποία απαιτείται να ολοκληρωθεί αυστηρά πριν αρχισουν οι κλήσεις των *display*.

Αφού οι *display* εξαρτώνται ήδη απο ένα mutex απο τις προηγούμενες ασκήσεις, η λογική για την υλοποίηση του προβλήματος είναι η εξής: το mutex θα αρχικοποιείται ως κλειδωμένο, με τιμή μηδέν. Επιπλέον θα υπάρχει μία συνθήκη που θα ελέγχει πότε ακριβώς έχουν ολοκληρωθεί οι κλήσεις των *init*. Όποια διεργασία ολοκληρώσει την *init* της τελευταία, ξεκλειδώνει το mutex ετσι ώστε να μπορούν να αρχίσουν οι κλήσεις των *display* απο τις διεργασίες κανονικά.

Η αρχική ιδέα ήταν απλώς να προστεθεί μια συνθήκη if μεσα στο loop που δημιουργεί τις διεργασίες παιδιά. Άν ο μετρητης του loop ηταν στην μέγιστη τιμή, τότε η διεργασία ξεκλείδωνε το mutex. Η λυση αυτή ομως ήταν λάνθασμένη, καθώς δεν υπάρχει εγγύηση οτι η τελευταία διεργασία που καλείται θα είναι και αυτή που θα φτάσει τελευταία στην συνθήκη που ορίστηκε, με αποτέλεσμα το mutex να ξεκλείδωνε πρόωρα και να δημιουργόταν race condition.

Έτσι προέκυψε η ανάγκη ορισμου δυο νέων semaphore: τον imutex, που ουσιαστικά λειτουργεί κανονικά ως mutex αποκλειστικά για τις κλήσεις των *init*, και τον sem, εναν counting semaphore που αρχικοποιείται με τιμή ίση με τον αριθμό και μειώνεται κατα ένα (καλείται σε αυτόν η *sem_wait*) μετά απο κάθε ολοκληρωμένη εκτέλεση της *init*. Όταν η τιμή του sem φτάσει το μηδέν, τότε εκπληρώνεται η συνθήκη μας και μπορούμε να ξεκλειδώσουμε τον mutex. Ο λόγος που δημιουργήσαμε τον imutex αντι να χρησιμοποιήσουμε τον υπάρχων mutex, είναι επειδή ο mutex πρεπει να παραμείνει κλειδωμένος μεχρι να ολοκληρωθούν όλες οι κλήσεις της *init*.

Η κρίσιμη περιοχή που ελέγχει ο imutex πρέπει εκτός απο την *init* να περιλαμβανει και την *sem_wait* του semaphore. Με την υλοποίηση αυτή αποφεύγονται τα race condition που εμφανίστηκαν πρίν και το πρόγραμμα τρέχει όπως πρέπει.

Η υλοποίηση της *multithread_2* είναι ακριβώς ίδια, με την διαφορά οτι πέρα απο την χρήση struct για την σωστή προσπέλαση των τιμών στα thread, τα binary semaphores εχουν αντικατασταθεί απο mutex, και το counting semaphore αντικαταστάθηκε απο μια global μεταβλητη στην οποία έχουν πρόσβαση όλα τα thread που δημιουργούνται απο την *main*.