Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Πανεπιστήμιο Αθηνών

Λειτουργικά Συστήματα (Κ22) / Περίοδος 2021-2022

1η Εργασία

Ονομ/νυμο: Δεικτάκης Μιχάλης

ΑΜ: 1115200800018

Αρχικά, η εργασία μου αποτελείται από δύο αρχεία σε γλώσσα C , ένα αρχείο επικεφαλίδων και ένα makefile, συν δύο txt αρχεία τα οποία χρησιμοποίησα για tests.

Στον κώδικά μου έχω αποφασίσει να χρησιμοποιήσω 4 Posix named semaphores, επίσης το shared memory που θα χρειαστούν ο server και οι clients είναι ένα struct που θα περιέχει μια μεταβλητή τύπου int και έναν array από 100 το πολύ χαρακτήρες με σύνολο 104 bytes.

Το αρχείο defines.h περιέχει κάποιες αρχικοποιήσεις.

Το όνομα του αρχείου client.c που θα χρησιμοποιήσει ο πατέρας , τα ονόματα των named semaphores που θα χρησιμοποιήσουν και τα 2 προγράμματα , τα permitions και flags για άνοιγμα και κλείσιμο των semaphores όπως επίσης και το struct που θα αναπαραστήσει το shared memory segment.

Το αρχείο Server.c

Ξεξινώντας λοιπόν από το Server-Πατέρα, τα ορίσματα που θα δέχεται από τη γραμμή εντολών είναι 3: πρώτο το όνομα του αρχείου κειμένου , μετά το πλήθος των Clients και τέλος το πλήθος των δοσολειψιών που θα κάνει κάθε Client.

Πρώτα γίνεται ένας έλεγχος αν ο αριθμός arguments είναι σωστός αλλιώς επιστρέφεται κωδικός λάθους και το πρόγραμμα τερματίζει. Συνεχίζοντας κρατάμε σε μεταβλητές τα δωσμένα ορίσματα.

Αμέσως μετά γίνεται η αρχικοποίηση της διαμοιραζόμενης μνήμης και η σύνδεση της με τη μεταβλητή shmem , με το σχετικό και απαραίτητο έλεγχο αποτυχίας, όπου και πάλι θα τερματίσουμε το πρόγραμμα.

Στη συνέχεια θα αρχικοποιήσουμε και ανοίξουμε τους 4 semaphores που θα χρειαστούμε πάλι με αντίστοιχο έλεγχο αποτυχίας.

Μετά ανοίγουμε το αρχείο , διαβάζουμε τις γραμμές του και αποθηκεύουμε το πλήθος στη μεταβλητή linecount. Θα χρειαστεί να μετατρέψουμε τις μεταβλητές K και linecount σε strings έτσι ώστε να μπορέσουμε να τις περάσουμε στους clients ως arguments. Να σημειώσω εδώ πως χρησιμοποίησα δύο char arrays μεγέθους 10 εφόσον ένας int των 4 byte σε 32bit σύστημα μπορεί να έχει μέγεθος μέχει 10 ψηφία. Θα μπορούσα να φτιάξω μια συνάρτηση που θα κάνει την μετατροπή σε ακριβώς όσα ψηφία χρειάζεται αλλά δεν νομίζω πως είναι το ζητούμενο της εργασίας και θα χρειαζόταν περαιτέρω χρόνος.

Εδώ ξεκινά η δημιουργία τών παιδιών-clients. Καλούμε λοιπόν την fork() K φορές και αναλόγως αν το pid == 0 δηλαδή αν πρόκεται για client καλούμε την execl() έτσι ώστε να εκτελεστεί το αρχείο client περνόντας του ως ορίσματα: πρώτα το shared memory id, μετά το πλήθος γραμμών του αρχείου και τέλος το πλήθος των requests που θα στείλει.

Αν κάποια fork() αποτύχει ο πατέρας τερματίζει το πρόγραμμα.

Αν τώρα κάποιο παιδί αποτύχει στην execl στέλενεται kill signal για να τερματίσει η λειτουργία του και μειώνουμε το πλήθος των παιδιών κατά 1 και τελικά κάνει επιστρέφει κωδικό error στην exit().

Εδώ ξεκινούν οι δοσοληψίες.

Αρχικά κατά την δημιουργία των semaphores αρχικοποιούνται όλοι με τιμή 0 (έτσι θα πρέπει το εκάστοτε πρόγραμμα να κάνει up το σημαφόρο για να προχωρήσει η δοσοληψία).

Θα χρησιμοποιήσουμε 2 σημαφόρους για την είσοδο και έξοδο από το critical section αυτοί είναι οι sem_clients και sem_done , και 2 σημαφόρους για την επίτευξη της δοσοληψίας αυτοί είναι οι sem_proc και sem_req.

Ο server ξεκινάει loop για K*N φορές(σημείωση στο τέλος του readme σχετικά με αυτό).

Πρώτα κάνει post τον sem_clients και έτσι δίνει access σε όποιο παιδί προλάβει να μπει στο critical section. Αμέσως το πρώτο παιδί που θα προλάβει κάνει down τον sem_clients και μπαίνει στο critical section και ζητάει τυχαία γραμμή τοποθετόντας τον αριθμό στην μεταβλητή lineNo του shared memory κάνοντας με τη σειρά του up to sem_proc.Μόλις γίνει up τον κάνει down ο πατέρας και διαβάζει τη lineNo ψάχνει στο αρχείο και περνάει στη μεταβλητή line της shared memory την αντίστοιχη γραμμή και αμέσως κάνει up τον sem_req οπότε γνωστοποιεί στο παιδί οτι έχει απαντήσει στο request και με τη σειρά του το παιδί κάνει down τον σημαφόρο και μόλις εκτυπώσει τη γραμμή κάνει up τον sem_done γνωστοποιώντας τον πατέρα οτι τελείωσε και βγαίνει από το critical section και μπαίνει στο loop για το επόμενο request περιμένοντας μαζί με τα άλλα παιδιά. Αμέσως ο πατέρας κάνει down τον sem_done και ξεκινάει κι αυτός νέο loop για να δώσει πρόσβαση για το επόμενο request.

Μόλις τελειώσουν όλα τα requests ο πατέρας κάνει wait τα pids και παίρνει τα exit status τους. Κλείνει το αρχείο κειμένου και κάνει unlink τους 4 semaphores, detach το shared memory και τέλος κάνει remove το memory segment.

Το αρχείο client.c

Ξεκινάει δηλώνοντας τις μεταβλητές που θα χρειαστεί παρακάτω.

Στην μεταβλητή mem_id,totallines,requests περνάει τα arguments που δέχτηκε αντίστοιχα δηλαδή το id του shared memory , το πλήθος των γραμμών του αρχείου και το πλήθος των request που θα ζητήσει.

Στη συνέχεια κάνει attach το shared memory και κάνει open τους 4 σημαφόρους. Αρχικοποιούνται και οι μεταβλητές που θα χρειαστούμε για το μέσο χρόνο ενός αιτήματος.

(Εδώ αρχικά κρατούσα το χρόνο από την υποβολή ενός αιτήματος μέχρι την λήψη απάντησης και το πρόσθετα κάθε φορά σε μια μεταβλητή result και διαιρώντας με τα request έβγαζα το χρόνο αλλά αυτό ήταν πάντα 0 με 12 δεκαδικά ψηφία) Οπότε εδώ υπολογίζω το χρόνο που περνάει από το πρώτο request μέχρι να ολοκληρωθεί και το τελευταίο και βγάζω το μέσο χρόνο διαιρώντας με τα request.

Σε αυτό το σημείο ο client ξεκινάει ένα for loop και για κάθε request εκτελούνται οι εντολές που έχω περιγράψει παραπάνω.

Τέλος εκτυπώνεται ο μέσος χρόνος, το παιδί κάνει close τους σημαφόρους και detatch την shared memory.

Σχολιασμός για το loop των requests από τον πατέρα και παραδοχές.

Αρχικά προσπάθησα με 2 διαφορετικούς τρόπους από το for loop K*N που θεωρούσα πιο σωστούς:

1. Με χρήση while(waitpid(-1,NULL,WNOHANG) == 0)

   Με αυτό τον τρόπο όσο όλα τα παιδιά δεν έχουν δώσει exit status το loop συνεχίζει και πιστεύω θα ήταν καλύτερη υλοποίηση όμως σε περιπτώσεις ο πατέρας προλάβαινε για ελάχιστα να μπει στο loop πριν το παιδί επιστρέψει, αμέσως μετά επέστρεφε το παιδί και κολούσε ο πατέρας στο sem_wait().

2. Με χρήση while(True) και sem_timedwait()

   Σε αυτή την περίπτωση ουσιαστικά είχα στο shared memory μια μεταβλητή int done. Κάθε φορά που ένα παιδί τελείωνε ή πριν επιστρέψει exit(1) θα έκανε increment αυτή τη μεταβλητή κατα 1. Αντίστοιχα ο πατέρας στη γραμμή 163 θα είχε sem_timedwait() και αναλόγως αν έκανε fail θα κοίταγε αν η μεταβλητή done == K τότε θα έκανε break το loop αν όχι θα έκανε continue. Αυτό μου δούλευε κανονικά εκτός από μεγάλα K και N όπου έκανε exit λόγω failure σε sem_post.

Παραδοχές:

Δεν κάνω handle περίπτωση που μια γραμμή έχει παραπάνω από από 100 χαρακτήρες αφού δίνεται από την εκφώνηση πως τα αρχεία θα έχουν μέχρι τόσες γραμμές(χρησιμοποιώ fgets αντί fgetc) και δεν θεώρησα πως ήταν το ζητούμενο της εργασίας.