ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν
Πανεπιστήμιον Αθηνών

Μάθημα
Λειτουργικά Συστήματα

1η εργασία

Όνομα
Κυλάφη Χριστίνα-Θεανώ

ΑΜ
1115201200077

Αθήνα, 2017

Η άσκηση έχει υλοποιηθεί σε γλώσσα C, στο πρόγραμμα “sublime text” ενώ παράλληλα δοκιμαζόταν σε περιβάλλον linux μέσω απομακρυσμένης σύνδεσης μέσω τερματικού macOS.

Στο φάκελο, περιλαμβάνονται τα εξής αρχεία εκτός από το readme :
- **makefile** : με τη χρήση της εντολής “make” γίνεται η μεταγλώττιση
- **main.c** : περιλαμβάνει τον αλγόριθμο επίλυσης του προβλήματος, τις δομές, κλπ
- **semun.c** : συναρτήσεις χειρισμού σημαφόρων
- **semun.h** : αρχείο κεφαλίδας

Παράγεται το εκτελέσιμο “**feeder**”. Η σύνταξη για την εκτέλεσή του, είναι:
feeder <**πλήθος διεργασιών**> <**μέγεθος πίνακα**>.
Έχω προσθέσει ελέγχους λανθασμένης σύνταξης, για λιγότερα, μη αριθμητικά, ή λανθασμένα ορίσματα.

Το πρόγραμμα, τυπώνει πληροφορίες για τη διαδικασία της δημιουργίας των διεργασιών, τα στοιχεία των διεργασιών, τον κινητό μέσο όρο κάθε διεργασίας και τέλος τον επιτυχημένο τερματισμό των παιδιών και την απελευθέρωση της μνήμης.

Για την επίλυση του προβλήματος της εργασίας, δημιουργήθηκαν και χρησιμοποιήθηκαν :
- **Διαμοιραζόμενη μνήμη** μεγέθους struct που περιέχει έναν int και ένα timestamp (σε nanosecond)

- **Σημαφόροι** (readers, feeder, rwait, last, writewait)
  - **readers** : φροντίζει ώστε όσο γράφει ο feeder να μη διαβάζουν οι αναγνώστες, παρά μόνο αφού τελειώσει την εγγραφή στοιχείου στη μνήμη. Επίσης, με την κατάλληλη λειτουργία (operation) της συνάρτησης semop(), διασφαλίζει το να διαβάσει κάθε διεργασία μόνο 1 φορά το στοιχείο που γράφει στη μνήμη κάθε φορά ο feeder.
  - **feeder** : αποκλεισμός εγγραφής στη διαμοιραζόμενη μνήμη από τον feeder πριν τελειώσουν την ανάγνωση και οι N διεργασίες - αναγνώστες.
  - **writewait** : mutex μεταξύ των διεργασιών που έχουν διαβάσει τον πίνακα από το feeder, ώστε να γράφει μία μία στο αρχείο, τα τελικά αποτελέσματα.

- **Συναρτήσεις χειρισμού** των παραπάνω σημαφόρων με τη βοήθεια των έτοιμων βιβλιοθηκών. Οι συναρτήσεις αυτές, θέτουν την τιμή των σημαφόρων (int set_sem( )), τους διαγράφουν (int delete_sem( )), κατεβάζουν την τιμή (int sem_down( )) και αντίστοιχα την ανεβάζουν (int sem_up( ) , int sem_up_val( )) όταν αυτό είναι απαραίτητο. Υλοποίησα επίσης μία επιπλέον συνάρτηση, τη sem_zero( ), η οποία δηλώνει πως η συγκεκριμένη διεργασία που την καλεί, πρέπει να περιμένει το συγκεκριμένο σημαφόρο με τον οποίο κλήθηκε, να αποκτήσει τιμή 0, και μέχρι τότε μπλοκάρεται. Αυτή η λειτουργία, χρησιμοποιήθηκε για να διασφαλίσει πως κάθε αναγνώστης διαβάζει 1 φορά το κάθε στοιχείο από τη διαμοιραζόμενη μνήμη.

Το πρόγραμμα:
Ξεκινώντας, παίρνουμε και ελέγχουμε τα arguments από το command line. Αν είναι ορθά, τότε δημιουργείται πίνακας που κρατείται στην κοινή μνήμη readers - feeder. Ο πίνακας του feeder, όπως θα δούμε στη συνέχεια, γεμίζει αργότερα, αφού δημιουργηθούν οι διεργασίες παιδιά, πριν ξεκινήσουν την ανάγνωση. Έπειτα, δημιουργείται με τις κατάλληλες συναρτήσεις, shared memory segment για το οποίο κρατάμε το id στην κοινή μνήμη, ώστε στη συνέχεια να γίνει attached στην περιοχή μνήμης τόσο του feeder όσο και του κάθε reader (θα μπορούσε να γίνει και στην αρχή attached, αλλά προτίμησα να το κάνω ξεχωριστά).

Δημιουργούνται οι 3 σημαφόροι (δημιουργήθηκαν σε ένα σετ και όχι ξεχωριστά - στον καθένα αντιστοιχεί ένας αριθμός για το χειρισμό του με τις συναρτήσεις), και αρχικοποιούνται:

- **readers** : σε 0, ώστε μέχρι ο feeder να τις κάνει up στο N , οι αναγνώστες να περιμένουν τον feeder να τελειώσει την εγγραφή.
- **feeder** : σε 0.
- **writewait** : σε 1, ώστε όταν ο πρώτος αναγνώστης καλέσει down σε αυτή, να του επιτραπεί να γράψει στο αρχείο, ενώ οι υπόλοιπες να μπλοκάρονται.

Αφού ανοίξουμε το αρχείο στο οποίο θα γράψουν αργότερα οι διεργασίες, ξεκινάμε τη δημιουργία παιδιών διεργασιών (αναγνωστών) με τη συνάρτηση fork().
Με μία συνθήκη στο pid της διεργασίας παιδιού (pid = 0), μπαίνει το πρόγραμμα στο αντίστοιχο τμήμα του κώδικα.

- Διεργασίες **παιδιά** (**αναγνώστες**) : Με το που δημιουργούνται, γράφουν στο τερματικό τα στοιχεία τους μέσω της συνάρτησης whois( ) και επισυνάπτουν το memory segment στην περιοχή μνήμης τους. Έπειτα, για M φορές (όσα είναι και τα στοιχεία του πίνακα δηλαδή που πρέπει να αντιγραφεί στη δομή της), κατεβάζουν αρχικά τον readers , διαβάζουν το στοιχείο που βρίσκεται εκείνη τη στιγμή στη διαμοιραζόμενη μνήμη, καταγράφουν την ώρα και κάνουν την ανανέωση του running average. Στη συνέχεια, η χρήση της συνάρτησης sem_zero( ), φροντίζει πως ακριβώς N διεργασίες θα διαβάσουν το συγκεκριμένο στοιχείο, και μάλιστα όλες από μία μόνο φορά, δηλαδή πως καμία διεργασία δε θα πάρει τη θέση άλλης σε αυτό το “παράθυρο” των N διεργασιών. Αυτό συμβαίνει, γιατί με την sem_down( ) αρχικα, ο σημαφόρος readers από N μειώνεται μετά από κάθε αναγνώστη, και με τη sem_zero( ) όλες εκτός από την τελευταία διεργασία μπλοκάρονται, μέχρι εκείνη τη χρονική στιγμή που η τελευταία θα φτασει τον readers στο 0, όπου και απελευθερώνονται όλες. Στη συνάρτηση sem_zero( ) αξιοποιείται η λειτουργία 0 της semop( ) όπως αναφέραμε παραπάνω, η οποία μπλοκάρει κάθε καλούμενη διεργασία, μέχρι η τιμή του σημαφόρου να γίνει 0. Μετά την απελευθέρωση όλων των διεργασιών, η κάθε μια ανεβάζει το σημαφόρο feeder από μία φορά, ξεμπλοκάροντας τον feeder, επιτρέποντάς του να γράψει, αφού πλέον έχει γίνει και η ν-οστή ανάγνωση του στοιχείου. Οι αναγνώστες μπλοκάρονται στον readers, περιμένοντας τον feeder να γράψει το νέο στοιχείο στη μνήμη , κι αφού τελειώσει το critical section του, να ανεβάσει στο N τους readers, επιτρέποντάς τους να διαβάσουν το νέο στοιχείο και να επαναληφθεί η παραπάνω διαδικασία M (πλήθος στοιχείων πίνακα) φορές συνολικά, ώσπου να δημιουργηθεί ο ίδιος πίνακας σε όλες τις διεργασίες.

- Η **γονεϊκή** διεργασία (**feeder**) : Όμοια, ο feeder παρουσιάζει τα στοιχεία του, επισυνάπτει κι εκείνος το memory segment, γεμίζει με τυχαίες τιμές τον πίνακα και μπαίνει στην επανάληψη για M φορές. Εκεί λοιπόν αντιγράφει το κάθε στοιχείο στη μνήμη, καταγράφει και περνά την ώρα (χρονοσφραγίδα μεταφοράς) στη μνήμη (η μνήμη αποτελείται από ένα struct, ορισμένο να χωράει έναν int και μία τιμή timestamp σε nanosecond). Με το που τελειώνει λοιπόν την εγγραφή (critical section), ανεβάζει στο N τον readers, στον οποίο εχουν μπλοκαριστεί οι αναγνώστες, επιτρέποντας σε όλους ταυτόχρονα να διαβάσουν το νέο στοιχείο . Έπειτα, κατεβάζει τον feeder N φορές, μπλοκάροντας έτσι τον εαυτό του από το να γράψει πριν τελειώσουν όλοι οι αναγνώστες.

Ύστερα από το γέμισμα του πίνακα των διεργασιών με τα στοιχεία του feeder μέσω της μνήμης, οι διεργασίες μία μία (χρήση mutex writewait για να μην “πέσει η μία πάνω στην άλλη”) γράφουν τα δεδομένα τους στο αρχείο, τυπώνουν στο τερματικό τον κινητό μέσο όρο καθυστέρησής τους (μετράται σε nanoosecond για μεγαλύτερη ακρίβεια, αλλά για λόγους εμφάνισης στο τερματικό, κάνω μία διαίρεση, μετατρέποντάς το σε microsecond) η κάθε μιά, και τερματίζουν με τη συνάρτηση exit( ).

Τέλος, η μητρική διεργασία περιμένει να τερματίσουν και τα N παιδιά διεργασίες μέσω της συνάρτησης wait( ), ώστε να κλείσει το αρχείο και να κάνει απελευθέρωση της μνήμης (δομές: διαμοιραζόμενη μνήμη, σετ σημαφόρων, πίνακας).

Πειραματισμός

Ύστερα από 9 ενδεικτικές εκτελέσεις του προγράμματος “**feeder**” με διαφορετικό πλήθος διεργασιών (N) και στοιχείων πίνακα (M), συγκεντρώσαμε και παραθέτουμε τα αποτελέσματα στον **Πίνακα 1**. Συγκεκριμένα, σε κάθε εκτέλεση, καταγράφεται ο κινητός μέσος όρος λήψης στοιχείων της διεργασίας που τερμάτισε πρώτη (A) , τελευταία (B), με το μικρότερο μέσο (C) και με το μεγαλύτερο μέσο (D).

Πίνακας 1

			A	B	C	D
#	N (number of readers)	M (size of array)	Running Average (microsecond)			
			First process to exit	Last process to exit	Lowest	Highest
1	5	30000	0,264	0,351	0,264	0,997
2	5	10000	0,635	6,631	0,635	11,405
3	5	3001	3,436	21,447	3,436	21,447
4	10	20000	1,150	2,275	0,372	6,351
5	10	3001	5,981	12,822	5,981	17,092
6	100	6000	112,959	138,545	106,353	155,588
7	500	10000	707,109	651,783	543,283	865,390
8	500	4000	831.358	781,661	456,788	978,100
9	1000	3001	1.527,577	1.498,761	1.089,428	1.986,356

Όπως παρατηρούμε από τις παραπάνω εκτελέσεις, όσο αυξάνεται το πλήθος των διεργασιών (με σταθερό μέγεθος πίνακα), τόσο αυξάνεται ο μέσος χρόνος καθυστέρησης μεταφοράς - επικοινωνίας με τη διαμοιραζόμενη μνήμη.

Αυτό γίνεται ιδιαίτερα εμφανές από τις δοκιμές 3, 5, 9 όπου με το ίδιο μέγεθος πίνακα (3001 στοιχεία), για 3 τιμές (5, 10, 1000) με μεγάλη διαφορά, έχουμε αξιοσημείωτη διαφορά στην ελαχιστη καθυστέρηση (3,436, 5,981, 1.082,428 microseconds).

Επί προσθέτως, για μικρές τιμές του πλήθους των διεργασιών, όσο αυξάνεται το μέγεθος του πίνακα, τόσο μειώνεται ο κινητός μέσος καθυστέρησης λήψης (δοκιμές 1,2,3 και 4,5).
Όταν το πλήθος διεργασιών γίνει 500 (δοκιμές 7,8), τότε βλέπουμε ότι οι διαφορές του μέσου, όταν αυξάνουμε το μέγεθος του πίνακα είναι σχεδόν μηδαμινές (ειδικά αν σκεφτούμε ότι βρισκόμαστε σε κλίμακα microsecond), καθώς οι τιμές των δύο γραμμών- δοκιμών 7,8 του παραπάνω πίνακα , σχεδόν συμπίπτουν.