Λειτουργικά Συστήματα

2021 - 2022

2η Εργαστηριακή Άσκηση

Μέρος 1 [75 μονάδες]

Ερώτημα Α [25]: (Α) Δημιουργήστε ένα πρόγραμμα στο οποίο μία διεργασία στο Linux/Unix παράγει άλλες Ν θυγατρικές της. Κάθε διεργασία i (1<=i<=N), διαβάζει την τιμή που είναι αποθηκευμένη στη ρίζα ενός κοινά διαμοιραζόμενου Heap tree (min heap ή max heap, δεν παίζει ρόλο) το οποίο αποθηκεύει 10*N ακέραιους **διακριτούς** αριθμούς στο διάστημα [1, 10*N], και στη συνέχεια την προσθέτει σε μία κοινά διαμοιραζόμενη μεταβλητή int *p, με αρχική τιμή 0. Για παράδειγμα, αν N=10, τότε το shared memory heap-tree αποθηκεύει όλους τους διακριτούς αριθμούς στο διάστημα [1, 100], δηλαδή 1,2,3, …..,99, 100. Το heap είναι προσπελάσιμο από τον shared root pointer. Συγχρονίστε τις παραπάνω διεργασίες, ώστε στο τέλος η μεταβλητή *p να περιέχει το άθροισμα των τιμών όλων των διακριτών αριθμών στο διάστημα [1, 10*N]. Ποιος είναι ο συνολικός χρόνος υπολογισμού του παραπάνω αθροίσματος;

(Β) Αν το Heap tree δεν είναι shared, προκύπτει το ίδιο αποτέλεσμα; Τι παρατηρείτε στο συνολικό χρόνο υπολογισμού;

*****Μετρήστε τους κύκλους της CPU κάνοντας χρήση της συνάρτησης clock(). Στη συνέχεια διαιρέστε με την built-in σταθερά CLOCKS_PER_SEC ώστε να πάρετε τον αριθμό των δευτερολέπτων.**

Ερώτημα Β [20]: Γράψτε πρόγραμμα που να επιλύει με χρήση σημαφόρων το πρόβλημα των αναγνωστών και των εγγραφέων, ώστε να αποφεύγεται η παρατεταμένη στέρηση. Δηλαδή, αν υπάρχει επιθυμία και από εγγραφείς και από αναγνώστες να προσπελάσουν τη βάση ταυτόχρονα, θα την προσπελάζουν εναλλάξ, ένας προς έναν, π.χ., πρώτα ένας αναγνώστης, στη συνέχεια ένας εγγραφέας, στη συνέχεια και πάλι ένας αναγνώστης, κ.ο.κ.

Ερώτημα Γ [10]: Σε ένα πρόγραμμα στον kernel του ΛΣ, πραγματοποιούνται 5 κλήσεις συτήματος (Systems Calls: SC1, SC2, SC3, SC4 και SC5). Οι εντολές SC1, SC2 και SC3 εκτελούνται αρχικά (είναι οι αρχικές δραστηριότητες του προγράμματος) και μπορούν να εκτελούνται παράλληλα (με οποιαδήποτε σειρά). Η έναρξη της εντολής SC4 μπορεί να πραγματοποιηθεί μόνο όταν ολοκληρωθεί η εκτέλεση των εντολών SC1 και SC2. Η SC5 είναι η τελική εντολή του προγράμματος και η έναρξή της μπορεί να πραγματοποιηθεί μόνο όταν ολοκληρωθεί η εκτέλεση των εντολών SC3 και SC4.

Ζητείται να υλοποιήσετε κώδικα που να αναπαριστά (με τη μέγιστη δυνατή παραλληλία) τη σειρά εκτέλεσης των κλήσεων συστήματος του παραπάνω προγράμματος. Μην παραλείψετε να δηλώσετε και να αρχικοποιήσετε τους σημαφόρους που θα χρησιμοποιήσετε.

Θεωρείστε ότι κάθε εντολή SC1...SC5, εκτελεί μία εντολή του συστήματος της αρεσκείας σας . π.χ. system("ls -l")ή system("ps -l") κ.τ.λ...

Σημείωση 1: Οι σημαφόροι που θα χρησιμοποιήσετε θα πρέπει να αρχικοποιηθούν σε κατάλληλες τιμές. Αυτές μπορεί να είναι και αρνητικές.

Σημείωση 2: Ορθές λύσεις (δηλ. λύσεις που εξασφαλίζουν κατά την παράλληλη εκτέλεση των δύο διεργασιών τους προηγουμένως αναφερθέντες περιορισμούς) που δεν χρησιμοποιούν τον ελάχιστο δυνατό αριθμό σημαφόρων θα είναι αποδεκτές κατά την αξιολόγηση αλλά δεν θα λάβουν το μέγιστο της βαθμολογίας του ερωτήματος.

Ερώτημα Δ [20]: Ο ιδιοκτήτης ενός πάρκινγκ, προκειμένου να βελτιώσει και να εκσυγχρονίσει την επιχείρησή του, εγκατέστησε μια σειρά από αυτόματες μηχανές στάθμευσης που συνδέονται μεταξύ τους δικτυακά και έχουν τη δυνατότητα να χρησιμοποιούν διαμοιραζόμενη μνήμη. Πλέον, ένας πελάτης με το αυτοκίνητό του θα εκδώσει - μέσω μιας μηχανής στην είσοδο του πάρκινγκ - ένα εισιτήριο στάθμευσης το οποίο θα αναγράφει και την ακριβή θέση στάθμευσης. Αν δεν υπάρχει χώρος στάθμευσης, υποθέτουμε πως ο πελάτης/αυτοκίνητο περιμένει σε ένα ειδικό χώρο, που για λόγους απλότητας δεχόμαστε πως είναι απεριόριστος, μέχρι να ελευθερωθεί μια θέση. Με το τέλος της στάθμευσης, ο πελάτης θα κατευθυνθεί σε μια αυτόματη μηχανή, θα εισαγάγει το εισιτήριο και το αντίτιμο της στάθμευσης και η μηχανή θα αποδεσμεύσει τη θέση προκειμένου να χρησιμοποιηθεί από άλλο αυτοκίνητο. Στη συνέχεια δίνεται ο κώδικας που χρησιμοποιείται στις αυτόματες μηχανές για τη διαχείριση των θέσεων στάθμευσης.

```
                    shared integer free_s=N;
                    shared boolean free_a [1..N] αρχικά όλες οι θέσεις TRUE;
                    const N;
 Enter_p( )                                  Leave_p(free_p)
{                                           {
    integer free_p;                             region free_s do
    region free_s do                            {
    {                                               free_s := free_s + 1;
       await(free_s > 0);                       }
       free_s := free_s – 1;                    region free_a do
    }                                           {
       region free_a do                             free_a[free_p]:= TRUE;
      {                                         }
         free_p:= Επιλογή_Θέσης(free_a);    }
         if έγκυρη free_p {
           free_a[free_p] := FALSE;
           τύπωσε εισιτήριο στάθμευσης με
                    θέση free_p;
         }
      }
}
```

Στον παραπάνω κώδικα παρουσιάζονται οι δυο βασικές συναρτήσεις που χρησιμοποιούνται για τη διαχείριση των θέσεων στάθμευσης. Ως εργαλεία αμοιβαίου αποκλεισμού χρησιμοποιούνται κρίσιμες περιοχές και κρίσιμες περιοχές υπό συνθήκη. Η συνάρτηση **Επιλογή_Θέσης**(free_a) βρίσκει και επιστρέφει τον αριθμό μιας κενής θέσης. Δεν χρειάζεται να σας απασχολήσει περαιτέρω η συγκεκριμένη συνάρτηση. Η διαδικασία **Leave_p**(free_p) δέχεται ως όρισμα τον αριθμό θέσης στάθμευσης του εισιτηρίου που εισάγει ο πελάτης.

Ο ανωτέρω κώδικας πάσχει από ένα λάθος ως προς τη διαχείριση των θέσεων του πάρκινγκ. Ένας πελάτης θα αντιληφθεί το λάθος αυτό γιατί π.χ. θα αναγκαστεί να περιμένει συνεχώς την εκτύπωση του εισιτηρίου παρόλο που βλέπει πως υπάρχει μια ελεύθερη θέση (δηλαδή ενώ υπάρχει μια ελεύθερη θέση, αυτή δεν διατίθεται). Πιο συγκεκριμένα, το λάθος οφείλεται στον τρόπο με τον οποίο διαχειρίζονται/χρησιμοποιούνται οι δυο διαμοιραζόμενες μεταβλητές στη διαδικασία **Leave_p( )**. Βρείτε το λάθος και δώστε ένα αντίστοιχο σενάριο εκτέλεσης του αλγορίθμου που να τεκμηριώνει την απάντησή σας. Υλοποιείστε τον αλγόριθμο αυτό.

Μέρος 2 [25 μονάδες]

Ερώτημα Α [10]

Έστω έξι (6) διεργασίες οι οποίες φτάνουν σε ένα υπολογιστικό σύστημα με συνολικό χώρο μνήμης (για διεργασίες χρηστών) 620K. Στον ακόλουθο πίνακα φαίνονται οι στιγμές άφιξης των διεργασιών, η διάρκειά τους (απαιτούμενος χρόνος εκτέλεσης της κάθε μίας στην ΚΜΕ) και ο χώρος μνήμης που απαιτεί η κάθε μία για την εκτέλεσή της.

Διαδικασίες	Στιγμή Άφιξης	Διάρκεια	Μνήμη
P1	0	7	180K
P2	1	2	100K
P3	2	5	350K
P4	3	6	100K
P5	3	4	90K
P6	4	2	80

Αναπαραστήστε στον παρακάτω πίνακα την εικόνα της ΚΜΕ και της μνήμης του συστήματος κάθε χρονική στιγμή (ποιες διεργασίες βρίσκονται στην 'ουρά της ΚΜΕ' και ποια εκτελείται σε αυτήν κάθε χρονική στιγμή, ποια τμήματα μνήμης έχουν εκχωρηθεί σε κάθε διεργασία και ποια είναι ελεύθερα ('εικόνα μνήμης'), ποιες διεργασίες περιμένουν για εκχώρηση μνήμης ('ουρά μνήμης') κ.λ.π.), θεωρώντας ότι έχουμε ένα σύστημα μνήμης μεταβλητών διαιρέσεων χωρίς συμπίεση (compaction) και αλγόριθμο χρονοδρομολόγησης SRTF (Shortest Remaining Time First).

Χρονική Στιγμή	**Αφιξη**	**Εικόνα Μνήμης**	**ΚΜΕ**	**Ουρά Μνήμης**	**Ουρά ΚΜΕ**	**Τέλος**
0	P_1	<Οπή 620>	-	P_1	-	-
1	.....	.....	...	.....	...	-
2	.....	.....	...	.....	....	-

Σημείωση:

(i) Στον αλγόριθμο SRTF μια διεργασία που εκτελείται αφήνει την ΚΜΕ όταν τελειώσει η εκτέλεσή της ή όταν γίνει έτοιμη μια διεργασία που έχει μικρότερο χρόνο εκτέλεσης από τον εναπομείναντα χρόνο της εκτελούμενης διαδικασίας. Εάν περισσότερες από μία νέες διεργασίες έχουν μικρότερο χρόνο εκτέλεσης από τον εναπομείναντα χρόνο εκτέλεσης της τρέχουσας διεργασίας ο χρονοδρομολογητής θα επιλέξει να αντικαταστήσει την τρέχουσα διεργασία με την διεργασία εκείνη που έχει το μικρότερο χρόνο εκτέλεσης ή το μικρότερο εναπομείναντα χρόνο εκτέλεσης από όλες. Εάν υπάρχουν προς δρομολόγηση περισσότερες από μια διεργασίες με τον ίδιο μικρότερο (ή μικρότερο εναπομείναντα) χρόνο εκτέλεσης τότε ο χρονοδρομολογητής θα επιλέξει να δρομολογήσει τη διεργασία που έχει αφιχθεί πρώτη.

(ii) Μία 'μεγάλη' διεργασία που βρίσκεται στην ουρά μνήμης και περιμένει να της εκχωρηθεί μνήμη, δεν εμποδίζει άλλες μικρότερες διεργασίες να τους εκχωρηθεί μνήμη παρόλο που έχουν φτάσει αργότερα στην ουρά μνήμης (ανάθεση με υπερπήδηση).

Ερώτημα Β [10]

Θεωρήστε ένα σύστημα διαχείρισης μνήμης με σελιδοποίηση (paging), το οποίο έχει μέγεθος σελίδας / πλαισίου 1K bytes και διάστημα λογικών διευθύνσεων των 20 bits. Η φυσική μνήμη του συστήματος είναι ίση με 4MB. Έστω επίσης μια διεργασία η οποία αποτελείται από 6100 bytes και εισέρχεται στο σύστημα για εκτέλεση.

(α) Υπολογίστε την εσωτερική κλασματοποίηση την οποία προκαλεί η συγκεκριμένη διεργασία, αν φορτωθεί εξ' ολοκλήρου στη φυσική μνήμη του συστήματος. **[2]**

(β) Έστω ότι στον Πίνακα Σελίδων της διεργασίας υπάρχουν οι ακόλουθες εγγραφές, της μορφής (Αριθμός Λογικής Σελίδας, Αριθμός Πλαισίου Μνήμης – οι αριθμοί δίνονται στο δεκαδικό σύστημα):

(0,11), (1,12), (2,1), (3,15), (4,-), (5,8)

Σημείωση: ζεύγη της μορφής (x, -) δηλώνουν ότι η λογική σελίδα x δε βρίσκεται στη φυσική μνήμη.

Έστω επίσης οι ακόλουθες λογικές διευθύνσεις της ανωτέρω διεργασίας (δίνονται στο δεκαεξαδικό σύστημα):

(i) 00388_{16}

(ii) $0125F_{16}$

(iii) $015A4_{16}$

Υπολογίστε (δίνοντας τις απαντήσεις σας επίσης στο δεκαεξαδικό σύστημα) σε ποιες φυσικές διευθύνσεις αντιστοιχούν. **[8]**

Ερώτημα Γ [5]

Έστω η παρακάτω ακολουθία αναφοράς μίας διεργασίας:

2 5 8 1 8 7 5 1 8 2 4 2 1 3 6 4 7 5 3 7

Η διεργασία εκτελείται σε σύστημα που η μνήμη του διαθέτει τέσσερα (4) πλαίσια σελίδων, τα οποία αρχικά είναι κενά. Στον πίνακα που ακολουθεί δώστε την ακολουθία αναφοράς, σημειώνοντας **με μαύρο χρώμα** ανά χρονική στιγμή τις σελίδες που υπάρχουν στον πίνακα σελίδων και σημειώνοντας **με κόκκινο χρώμα** μόνο τους αριθμούς σελίδων στα σημεία στα οποία συμβαίνουν σφάλματα σελίδας για την πολιτική αντικατάστασης σελίδων LRU (Least Recently Used).

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0																				
1																				
2																				
3																				

Παράρτημα:

Υποδείξεις: Συμβουλευτείτε τους πηγαίους κώδικες των παραδειγμάτων, που βρίσκονται στο δικτυακό τόπο του μαθήματος στο eclass.

Η μεταγλώττιση ενός προγράμματος στο Linux/Unix γίνεται με τη χρήση της εντολής:

gcc -pthread –o <όνομα εκτελέσιμου> <όνομα αρχείου που περιέχει τον πηγαίο κώδικα>.

Η εντολή αυτή θα πρέπει να δοθεί από τερματικό ενώ ο τρέχων κατάλογος εργασίας θα πρέπει να είναι αυτός στον οποίο βρίσκεται αποθηκευμένος ο πηγαίος κώδικας του προγράμματος. (Η εντολή με την οποία μπορούμε να βρούμε τον τρέχοντα κατάλογο είναι η pwd ενώ μετακινούμαστε μεταξύ καταλόγων με την εντολή cd. Τα περιεχόμενα των καταλόγων βρίσκονται με την εντολή ls. Περισσότερες πληροφορίες για τη λειτουργικότητα και τις παραμέτρους των εντολών (αλλά και των κλήσεων συστήματος) μπορούν να βρεθούν μέσω της βοήθειας του Linux που δίνεται αν πληκτρολογήσουμε man <όνομα εντολής>.

Η εκτέλεση ενός προγράμματος γίνεται πληκτρολογώντας:
./<όνομα εκτελέσιμου>

Η εκτέλεση ενός προγράμματος στο παρασκήνιο γίνεται πληκτρολογώντας:
./<όνομα εκτελέσιμου>&

Καλή Επιτυχία!!!

Αριθμός μελών ανά ομάδα: *1 έως και 4.*
Ημερομηνία Παράδοσης: *Ημερομηνία Εξέτασης του μαθήματος στην εξεταστική Ιανουαρίου - Φεβρουαρίου*

Αποστολή εργασιών σε ZIP ή RAR μορφή (ο συμπιεσμένος φάκελος να περιέχει txt file με τα στοιχεία της ομάδας, pdf και C files) στο e-class καθώς και με e-mail στους διδάσκοντες:

makri@ceid.upatras.gr, aristeid@ceid.upatras.gr, sioutas@ceid.upatras.gr

Οι απαντήσεις που αφορούν κώδικα μέσα στο pdf της αναφοράς θα πρέπει να τεκμηριώνονται με screenshots από την εκτέλεση των προγραμμάτων.