ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
http://www.cslab.ece.ntua.gr

Λειτουργικά Συστήματα

7ο εξάμηνο, Ακαδημαϊκή περίοδος 2010-2011

Επαναληπτική Εξέταση ακ. έτους 2010-2011 – Λύσεις

Διάρκεια: 2.5 ώρες

Θέμα 1 (25%)

Δίνεται το ακόλουθο τμήμα κώδικα:

```
int i, j, n, fd[3][2];
pid_t p[3];

for (i = 0; i < 3; i++) pipe(fd[i]);

for (i = 0; i < 3; i++) {
    p[i] = fork();
    if (p[i] == 0) {
        for (j = i; j < 3; j++)
            close(fd[j][1]);
        sleep(5);

        n = 3 - i;
        if (i > 0)
            write(fd[i - 1][1], &n, sizeof(n));

        read(fd[i][0], &n, sizeof(n));
        for (j = 0; j < n; j++)
            if (fork() == 0)
                Fn(i, j);

        wait(NULL);
        Fn(i, 5);
    }
}
```

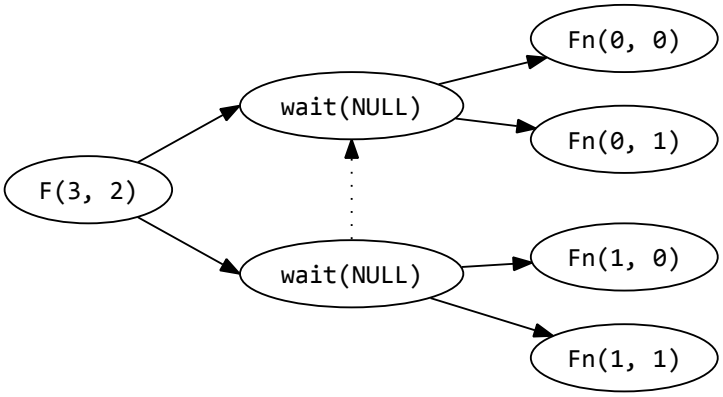
```
n = 2;
write(fd[2][1], &n, sizeof(n));

kill(p[2], SIGKILL);
close(fd[1][1]);
/* A */
wait(NULL);
Fn(i, n);
```

Οι κλήσεις συστήματος δεν αποτυγχάνουν κι η `Fn()` *δεν επιστρέφει ποτέ. Θεωρήστε ότι η αρχική διεργασία φτάνει στο σημείο* `/* A */` *σε πολύ λιγότερο από 5 δευτερόλεπτα. Δεδομένου ότι η* `Fn()` *δεν επιστρέφει ποτέ, τελικά οι διεργασίες που δημιουργεί το παραπάνω πρόγραμμα έρχονται σε μια* μόνιμη *κατάσταση: το δέντρο διεργασιών μένει σταθερό για πάντα, και κάθε διεργασία εκτελεί συγκεκριμένη συνάρτηση από τον κώδικά της.*

α. (12%) *Για την* τελική *κατάσταση: σχεδιάστε το δέντρο διεργασιών που προκύπτει. Εξηγήστε* συνοπτικά *το σκεπτικό πίσω από τη μορφή του.*

Το δέντρο διεργασιών που προκύπτει φαίνεται στο παρακάτω σχήμα:



Ή αρχική διεργασία κατασκευάζει 3 pipes, και γεννά 3 παιδιά. Κάθε παιδί περιμένει για 5 δευτερόλεπτα. Το i-ιοστό παιδί (εκτός από το `p[0]`) υπολογίζει την τιμή $n = 3 - i$ και τη γράφει στο pipe υπ'αριθμό $i - 1$. Έπειτα, διαβάζει από το i-οστό pipe μια τιμή που καθορίζει πόσα παιδιά κατασκευάζει αμέσως μετά. Κάθε ένα από τα παιδιά του, εγγόνια της αρχικής, μπλοκάρει μέσα στην `Fn()`.
Τι συμβαίνει με το παιδί `p[2]`; Επειδή ο πατέρας έχει προλάβει να φτάσει στο σημείο `/* A */` σε πολύ λιγότερο από 5 δευτερόλεπτα, σκοτώνεται, γι'αυτό και δεν εμφανίζεται στο παραπάνω σχήμα, που δείχνει τη μόνιμη κατάσταση. Οπότε, δεν προχωράει ποτέ κάτω από τη `sleep()`. Η `read(fd[1],...)` στο παιδί `p[1]` επιστρέφει την τιμή `0` (EOF), γιατί όλοι όσοι είχαν ανοιχτό το άκρο εγγραφής `fd[1][1]` το έχουν κλείσει: ο πατέρας το κλείνει ρητά, το παιδί `p[0]` το κλείνει πριν από τη `sleep()`, το παιδί `p[2]` το κλείνει γιατί σκοτώνεται. Οπότε, αφού η `read()` δεν επηρεάζει την τιμή του `n`, μένει $n = 3 - 1 = 2$ και το παιδί με PID `p[1]` κατασκευάζει 2 παιδιά.

β. (4%) *Για κάθε κόμβο του δέντρου, γράψτε την κλήση συστήματος ή συνάρτηση που εκτελεί η αντίστοιχη διεργασία, μαζί με τα ορίσματά της.*

Οι διεργασίες-εγγόνια μπλοκάρουν στην `Fn()`. Οι διεργασίες-παιδιά μπλοκάρουν στη `wait(NULL)` γιατί κανένα από τα παιδιά τους δεν έχει πεθάνει. ο πατέρας *περνά* την `wait(NULL)` γιατί το παιδί με PID `p[2]` έχει ήδη πεθάνει, και μπλοκάρει στην `Fn(3, 2)`.

γ. (4%) *Συμπληρώστε το δέντρο διεργασιών ώστε να φαίνεται η διαδιεργασιακή επικοινωνία: για κάθε μεταφορά δεδομένων που συνέβη, σχεδιάστε ένα διακεκομμένο βέλος από τη διεργασία-αποστολέα στη διεργασία-παραλήπτη. Πάνω στο βέλος γράψτε την τιμή ή τιμές που μεταφέρονται κάθε φορά.*

Η μόνη μεταφορά που ολοκληρώνεται είναι η μεταφορά της τιμής `2` από τη διεργασία με PID `p[1]` στη διεργασία με PID `p[0]`.

δ. (5%) *Απαντήστε* συνοπτικά, *όχι πάνω από δύο γραμμές, στα ακόλουθα:*

i. Τι κάνει η κλήση συστήματος `wait()`;

ii. Πότε μια διεργασία γίνεται zombie; Τι πρέπει να συμβεί για να εξαφανιστεί από το σύστημα;

iii. Τι συμβαίνει στα ανοιχτά αρχεία μιας διεργασίας όταν αυτή πεθάνει κανονικά κι όταν σκοτωθεί;

iv. Τι επιστρέφει η `read()` σε κάποιον που διαβάζει από ένα pipe όταν κανείς δεν έχει πλέον ανοιχτό το άκρο εγγραφής;

i. Μπλοκάρει τη διεργασία που την καλεί μέχρι να πεθάνει ένα από τα παιδιά της.

ii. Μια διεργασία γίνεται zombie από τη στιγμή που πεθάνει μέχρι ο πατέρας της να κάνει `wait()`. Για να εξαφανιστεί από το σύστημα πρέπει ο πατέρας της να κάνει `wait()` ή να πεθάνει. Αν ο πατέρας της πεθάνει, θα υιοθετηθεί αμέσως από την `init`, η οποία κάνει `wait()` συνεχώς.

iii. Ό,τι και να γίνει κλείνουν, σαν να είχε εκτελέσει η ίδια `close()`.

iv. Όταν κανείς δεν έχει ανοιχτό το άκρο εγγραφής ενός pipe, η `read()` επιστρέφει την τιμή `0`, δηλαδή δηλώνει κατάσταση EOF.

Θέμα 2 (25%)

α. (5%) *Δύο φίλοι πηγαίνουν για ψάρεμα με μια βάρκα, έστω ότι αναπαρίστανται από τις διεργασίες `friend_A` και `friend_B`. Καθένας από τους δύο κάνει τα εξής: περπατά μέχρι την αποβάθρα (συνάρτηση `walk_to_dock()`), αν ο άλλος δεν είναι ήδη εκεί τον περιμένει (σημείο `...rendezvous...`), μπαίνει στη βάρκα (`board()`) και φεύγουν μαζί.*

```
void friend_A()
{
    walk_to_dock();
    ...rendezvous...
    board();
}
```

```
void friend_B()
{
    walk_to_dock();
    ...rendezvous...
    board();
}
```

Στα σημεία που υποδηλώνονται (`...rendezvous...`), σας ζητείται να υλοποιήσετε σχήμα συγχρονισμού, μόνο με χρήση `signal` και `wait()` σε κατάλληλα αρχικοποιημένους σημαφόρους, έτσι ώστε οι φίλοι να μπαίνουν στη βάρκα μόνο όταν κι οι δύο έχουν φτάσει στην αποβάθρα.

Χρησιμοποιούμε δύο σημαφόρους:

```
sA = semaphore(0);
sB = semaphore(0);
```

```
void friend_A()
{
    walk_to_dock();
    signal(sB);
    wait(sA);
    board();
}
```

```
void friend_B()
{
    walk_to_dock();
    signal(sA);
    wait(sB);
    board();
}
```

β. (10%) *Έστω ότι τρεις φίλοι `friend_A`, `friend_B`, `friend_C` πηγαίνουν για ψάρεμα με βάρκα που χωράει τρία άτομα. Ομοίως με το προηγούμενο ερώτημα, υλοποιήστε κατάλληλο σχήμα συγχρονισμού* μόνο *με κλήσεις `signal()` και `wait()` σε σημαφόρους, ώστε να μπαίνουν στη βάρκα μόνο όταν κι οι τρεις έχουν φτάσει στην αποβάθρα.*

Χρησιμοποιούμε τρεις σημαφόρους. Κάθε ένας από τους φίλους που φτάνει, ειδοποιεί τους άλλους δύο, και περιμένει μέχρις ότου έχει λάβει δύο ειδοποιήσεις στο δικό του σημαφόρο για να επιβιβαστεί:

```
sA = semaphore(0);
sB = semaphore(0);
sC = semaphore(0);
```

```
void friend_A()
{
    walk_to_dock();
    signal(sB);
    signal(sC);
    wait(sA);
    wait(sA);
    board();
}
```

```
void friend_B()
{
    walk_to_dock();
    signal(sA);
    signal(sC);
    wait(sB);
    wait(sB);
    board();
}
```

```
void friend_C()
{
    walk_to_dock();
    signal(sA);
    signal(sB);
    wait(sC);
    wait(sC);
    board();
}
```

γ. (10%) *Έστω ότι N φίλοι πηγαίνουν για ψάρεμα, με μία βάρκα που χωράει N επιβάτες. Μπαίνουν στη βάρκα* μόνο *όταν* όλοι *έχουν φτάσει στην αποβάθρα. Ένας από αυτούς ορίζεται καπετάνιος· αφού μπει στη βάρκα, την οδηγεί (`steer_boat()`).*
Όλοι εκτελούν ακριβώς την ίδια συνάρτηση:

```
shared int N;

void friend()
{
    bool captain = false;

    walk_to_dock();
    ...rendezvous...
    board();
    if (captain)
        steer_boat();
}
```

Σας ζητείται η υλοποίηση σχήματος συγχρονισμού στο σημείο που υποδεικνύεται, κοινό για όλες τις συναρτήσεις, έτσι ώστε να τηρούνται οι κανόνες τις εκδρομής. Ο κώδικάς σας πρέπει να θέτει τη σημαία `captain` μόνο *σε μία από τις διεργασίες, της δικής σας επιλογής. Μπορείτε να χρησιμοποιήσετε κλήσεις* `signal()` *και* `wait()` *σε κατάλληλα αρχικοποιημένους σημαφόρους, καθώς και μεταβλητές μοιραζόμενες ανάμεσα στις διεργασίες. Το πλήθος των φίλων σας δίνεται στη μεταβλητή* `N`*.*

Υπόδειξη: *Οι φίλοι κρατούν σε κατάλληλα αρχικοποιημένη μοιραζόμενη μεταβλητή πόσοι έχουν φτάσει μέχρι τώρα στην αποβάθρα.*

```
shared int N;

shared int count = 0;
lock = semaphore(1);
barrier = semaphore(0);

void friend()
{
    bool captain = false;

    walk_to_dock();
    wait(lock);
    ++count;
    if (count == N) {
        captain = true;
        signal(barrier);
    }
    signal(lock);

    wait(barrier);
    signal(barrier);

    board();
    if (captain)
        steer_boat();
}
```

Ο σημαφόρος `lock` προστατεύει τη μοιραζόμενη μεταβλητή `count`, όπου τηρείται ο αριθμός των φίλων που έχουν φτάσει μέχρι τώρα στην αποβάθρα. Καπετάνιος ορίζεται ο τε-

λευταίος που φτάνει. Προσοχή, ο έλεγχος γίνεται κρατώντας το `lock`. Αφού αυξήσει το `count`, κάθε φίλος περιμένει στο σημαφόρο `barrier`. Ο καπετάνιος κάνει `signal()`, οπότε κάποιος περνάει από το barrier. Αυτός που πέρασε κάνει `signal()` εκ νέου, οπότε τελικά περνάνε όλοι.

Θέμα 3 (25%)

α. (5%) *Τι είναι εναλλαγή περιβάλλοντος (context switch); Τι πληροφορίες αποθηκεύει και επαναφέρει το ΛΣ για ένα context switch; Πώς διαφέρει η εναλλαγή περιβάλλοντος ανάμεσα σε διεργασίες από την εναλλαγή ανάμεσα σε νήματα της ίδιας διεργασίας;*

Η εναλλαγή περιβάλλοντος είναι η διαδικασία στην οποία το ΛΣ αλλάζει τη διεργασία που εκτελείται στη CPU: σώζει την κατάσταση της τρέχουσας διεργασίας στο PCB της, επαναφέρει την κατάσταση της προς εκτέλεση διεργασίας από το PCB της και περνά τον έλεγχο σε αυτή, από το σημείο στο οποίο είχε σταματήσει την τελευταία φορά που ήταν στον επεξεργαστή. Το ΛΣ αποθηκεύει κι επαναφέρει ανάμεσα στα άλλα: την τιμή του Program Counter, τις τιμές των καταχωρητών, την κατάσταση της MMU του επεξεργαστή (τη ρυθμίζει ώστε να αναφέρεται στον πίνακα σελίδων της διεργασίας). Η εναλλαγή περιβάλλοντος ανάμεσα σε νήματα της ίδιας διεργασίας είναι φθηνότερη, γιατί όλα τα νήματα τρέχουν στο ίδιο περιβάλλον μνήμης: δεν χρειάζεται αλλαγή της κατάστασης της MMU, η οποία συμβαίνει σε context switch ανάμεσα σε διαφορετικές διεργασίες.

β. (5%) *Αναφέρατε τέσσερα κριτήρια αξιολόγησης για έναν αλγόριθμο χρονοδρομολόγησης. Περιγράψτε πολύ συνοπτικά πώς ορίζεται το καθένα.*

i. **Βαθμός χρησιμοποίησης ΚΜΕ:** ποσοστό χρόνου που είναι απασχολημένη η ΚΜΕ.

ii. **Ρυθμός διεκπεραίωσης:** Αριθμός διεργασιών που ολοκληρώνονται στη μονάδα του χρόνου.

iii. **Χρόνος ολοκλήρωσης:** Ο συνολικός χρόνος που χρειάζεται μια διεργασία από την υποβολή ως την ολοκλήρωσή της.

iv. **Χρόνος απόκρισης:** Χρόνος από την υποβολή μιας διεργασίας μέχρι την παραγωγή της πρώτης απόκρισης από αυτή προς τον χρήστη.

γ. (10%) *Σχολιάστε την επίδραση του μεγέθους του κβάντου χρόνου που χρησιμοποιείται για χρονοδρομολόγηση στα εξής:*

i. Αποκρισιμότητα του συστήματος

ii. Επιβάρυνση της ΚΜΕ με λειτουργίες χρονοδρομολόγησης

iii. Hit rate της κρυφής μνήμης (cache)

Αν θέλετε μπορείτε να χρησιμοποιήσετε παραδείγματα.

i. Μεγάλο κβάντο μειώνει την αποκρισιμότητα. Αν πιέσω ένα πλήκτρο κι ο editor μου γίνει `READY`, θα περάσει αρκετή ώρα μέχρι να ολοκληρωθούν τα κβάντα χρόνου των υπολοίπων διεργασιών για να τρέξει αυτός.

ii. Μικρό κβάντο χρόνου αυξάνει την επιβάρυνση της ΚΜΕ: αντί να εκτελεί χρήσιμο κώδικα διεργασιών, εκτελεί πολύ συχνά context switches. Μεγαλύτερο κβάντο χρόνου μειώνει το ποσοστό του χρόνου που χάνεται σε context switches.

iii. Πολύ μικρό κβάντο χρόνου συνεπάγεται πολύ συχνά context switches. Αυτό σημαίνει ότι κάθε διεργασία δεν θα έχει την ευκαιρία να χτίσει ένα καλό σύνολο εργασίας (working set) μέσα στην cache, αφού κάθε φορά που γίνεται context switch η νέα διεργασία πετάει έξω από την cache τα δεδομένα της προηγούμενης. Οπότε, πολύ μικρό κβάντο χρόνου συνεπάγεται πολύ μικρό hit rate στην cache.

δ. (5%) *Πότε μια διεργασία λέγεται I/O-bound και πότε CPU-bound; Πού κατατάσσονται οι εξής διεργασίες και γιατί; (α′) Ο vi όταν επεξεργάζεστε ένα κείμενο (β′) Το Photoshop όταν η εικόνα είναι μικρότερη της φυσικής μνήμης (γ′) Το 3D Studio όταν υπολογίζει μια φωτορεαλιστική σκηνή (δ′) Το Photoshop όταν η εικόνα δεν χωράει στη φυσική μνήμη (ε′) Μια μηχανή σκακιού*

Μια διεργασία λέγεται I/O-bound όταν ο χρόνος εκτέλεσής της καθορίζεται από το ρυθμό εξυπηρέτησης λειτουργιών Ε/Ε. Μια διεργασία λέγεται CPU-bound όταν ο χρόνος εκτέλεσής της καθορίζεται από το ρυθμό εκτέλεσης υπολογισμών από την ΚΜΕ. Μια I/O-bound διεργασία κάνει μεγάλα ξεσπάσματα Ε/Ε που χωρίζονται από πολύ μικρά ξεσπάσματα ΚΜΕ, μια CPU-bound διεργασία κάνει μεγάλα ξεσπάσματα ΚΜΕ που χωρίζονται από πολύ μικρά ξεσπάσματα Ε/Ε. (α′) I/O-bound, περιμένει συνεχώς τον χρήστη σε `read()` (β′) CPU-bound, κάνει υπολογισμούς με τα pixels της εικόνας (γ′) CPU-bound, υπολογίζει τις σκιάσεις για τη σκηνή (δ′) I/O-bound, θέλει να κάνει υπολογισμούς αλλά συνεχώς προκαλεί page faults (ε′) CPU-bound, ψάχνει το δέντρο των διαφορετικών επιλογών, αποτιμώντας διαφορετικές θέσεις πάνω στη σκακιέρα.

Θέμα 4 (25%)

α. (10%) *Απαντήστε* συνοπτικά*, όχι πάνω από δύο-τρεις γραμμές:*

i. Με ποιον τρόπο βοηθά η υποστήριξη modify bit από το υλικό την υλοποίηση αποδοτικού μηχανισμού για σελιδοποίηση κατ'απαίτηση από το ΛΣ;

ii. Τι θα γινόταν αν μπορούσε μια διεργασία χρήστη να ελέγξει τον χρονιστή (timer) του συστήματος; πώς αποτρέπεται αυτό το ενδεχόμενο;

iii. Υπάρχει περίπτωση μια εικονική διεύθυνση να είναι έγκυρη διεύθυνση για μια διεργασία (π.χ. έχει επιστραφεί από τη `malloc()`), και παρ'όλα αυτά η αντίστοιχη σελίδα να μην διαθέτει έγκυρη απεικόνιση στον πίνακα σελίδων; Τι συμβαίνει αν η διεργασία κάνει αναφορά στη συγκεκριμένη σελίδα;

i. Όταν το ΛΣ χρειάζεται να αντικαταστήσει μια σελίδα, αν το modify bit δεν είναι αναμμένο, ξέρει ότι δεν χρειάζεται να γράψει το περιεχόμενο του πλαισίου μνήμης στο δίσκο, πριν διαβάσει μια νέα σελίδα εκεί.

ii. Θα μπορούσε να τρέχει για πάντα, σταματώντας το μηχανισμό χρονοδρομολόγησης. Το ενδεχόμενο αυτό αποτρέπεται γιατί οι εντολές για χειρισμό του χρονιστή είναι προνομιούχες (privileged) κι εκτελούνται μόνο από κατάσταση επόπτη.

iii. Ναι υπάρχει. Όταν, για παράδειγμα, η συγκεκριμένη σελίδα είναι έγκυρη για τη διεργασία, αλλά αυτή τη στιγμή είναι στο δίσκο, οπότε δεν έχει έγκυρη απεικόνιση στον πίνακα σελίδων. Όταν η διεργασία κάνει αναφορά, θα προκληθεί page fault και το ΛΣ θα φτιάξει τον πίνακα σελίδων αφού φέρει τη σελίδα από το δίσκο.

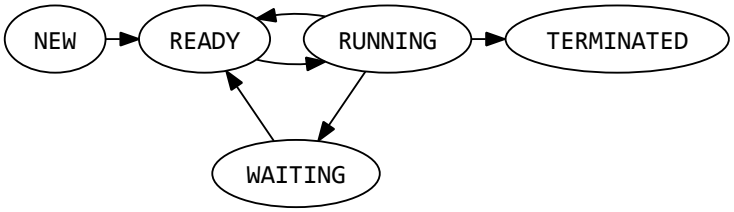
β. (10%)

i. Σχεδιάστε ένα ενδεικτικό διάγραμμα μετάβασης καταστάσεων διεργασίας σε ΛΣ.

ii. Από ποια σε ποια κατάσταση μεταβαίνει μια διεργασία όταν εκτελεί `sleep()`; Πότε φεύγει από την κατάσταση στην οποία μπαίνει με `sleep()` και σε ποια πηγαίνει;

iii. Μια διεργασία είναι "Υπό Εκτέλεση" στον επεξεργαστή και πηγαίνει σε κατάσταση "Έτοιμη" χωρίς να εκτελέσει κλήση συστήματος. Τι συνέβη; Πότε θα φύγει από εκεί;

iv. Μια διεργασία είναι "Υπό Εκτέλεση" και γίνεται "Σε Αναμονή" χωρίς να εκτελέσει κλήση συστήματος. Τι συνέβη; Πότε θα φύγει από εκεί;



i.

ii. Μεταβαίνει από την κατάσταση `RUNNING` στην κατάσταση `WAITING`. Φεύγει από `WAITING` και γίνεται `READY` όταν περάσει ο χρόνος που ζήτησε (γίνεται interrupt από τον χρονιστή και το ΛΣ βλέπει ότι πρέπει να την ξυπνήσει).

iii. Τελείωσε το κβάντο χρόνου της. Θα φύγει από `READY` όταν επιλεγεί για εκτέλεση από τον χρονοδρομολογητή.

iv. Έκανε page fault και πρέπει να έρθει η σελίδα στην οποία αναφέρθηκε από το δίσκο. Θα φύγει από `WAITING` όταν ο δίσκος κάνει interrupt ότι μετέφερε την απαιτούμενη σελίδα, οπότε το ΛΣ θα την ξανακάνει `READY`.

γ. (5%) *Έστω συγκεκριμένος υπολογισμός που εκτελείται με δεδομένο αριθμό πλαισίων F σε σύστημα εικονικής μνήμης και προκαλεί p σφάλματα σελίδας. Αν αυξήσουμε τον αριθμό των πλαισίων σε $F' > F$, υπάρχει περίπτωση ο νέος αριθμός σφαλμάτων σελίδας να είναι $p' > p$; θεωρήστε στρατηγική αντικατάστασης σελίδας*

i. First-In, First-Out (FIFO)

ii. Least Recently Used (LRU)

Δικαιολογήστε συνοπτικά *την απάντησή σας.*

i. Ναι, υπάρχει, είναι το παράδοξο του Belady: υπάρχουν ακολουθίες αναφοράς που με μεγαλύτερο αριθμό πλαισίων προκαλούν περισσότερα σφάλματα σελίδας, όταν η στρατηγική αντικατάστασης είναι FIFO.

ii. Αποκλείεται, γιατί σε κάθε σημείο της ακολουθίας αναφορών το σύνολο των σελίδων που είναι στη μνήμη για p' πλαίσια (οι τελευταίες p' σελίδες στις οποίες έγινε πιο πρόσφατα αναφορά) θα είναι πάντα υπερσύνολο του συνόλου σελίδων που είναι στην μνήμη για p πλαίσια, αν $p' > p$.