Σημειώσεις ΛΣ

Διεργασίες

1. Γιατί οι υπολογιστές παρέχουν εντολές όπως η TSL αφού υπάρχει η λύση του Peterson;

Οι υπολογιστές που σχεδιάζονται με πρόβλεψη για πολλούς επεξεργαστές διαθέτουν τέτοιες εντολές.
Οι πράξεις της ανάγνωσης και της αποθήκευσης στη λέξη αυτή είναι αδιαίρετες → κανείς επεξεργαστής δεν μπορεί να προσπελάσει τη λέξη αυτή μέχρι την ολοκλήρωση της εντολής. Όταν η CPU εκτελεί αυτή την εντολή, κλειδώνει το δίαυλο μνήμης (memory bus), ώστε να αποτρέψει την προσπέλαση της λέξης μνήμης από άλλες CPU μέχρι να τελειώσει την εργασία της

2. Όταν μια διεργασία χρησιμοποιεί την εντολή TSL δεν ξέρει πόσο θα περιμένει μέχρι να μπει στην κρίσιμη περιοχή. Γιατί;

```
enter_region:
     TSL REGISTER, LOCK   | Αντέγραψε το κλείδωμα στον
                          | καταχωρητή και θέσε τιμή 1
     CMP REGISTER, #0     | Είχε το κλείδωμα τιμή 0;
     JNE enter_region     | Αν όχι τότε κάποια άλλη είναι
                          | στην κρίσιμη περιοχή – loop
     RET                  | Επιστροφή: επιτεύχθηκε η είσοδος

leave_region:
     MOVE LOCK, #0        | Θέσε τιμή 0 στο κλείδωμα
     RET                  | Επιστροφή
```

Πριν μπει επομένως μια διεργασία στην κρίσιμη περιοχή της καλεί την *enter_region* η οποία εκτελεί αναμονή με απασχόληση μέχρι να ελευθερωθεί το κλείδωμα (*spin lock*) και μόλις γίνει αυτό αποκτά τον έλεγχο του κλειδώματος και επιστρέφει.

Όταν μία διεργασία θέλει να μπει στην κρίσιμη περιοχή της, ελέγχει αν αυτό επιτρέπεται. Αν δεν επιτρέπεται η διεργασία εκτελεί συνεχώς έναν περιορισμένο βρόχο αναμένοντας να ελευθερωθεί η είσοδος στην κρίσιμη περιοχή. Όπως φαίνεται και από τον παραπάνω κώδικα δεν είναι δυνατό να γνωρίζει πόσο ακριβώς χρειάζεται να περιμένει. Πάντως οι τεχνικές αναμονής με απασχόληση (*busy waiting*) χρησιμοποιούνται μόνο όταν υπάρχει η σχετική βεβαιότητα ότι η αναμονή θα είναι σύντομη.

3. Αναπτύξτε ένα αλγόριθμο (που χρησιμοποιεί την TSL) ο οποίος θα εγγυάται ότι από την στιγμή που θα ζητήσει μία διεργασία να μπει στην κρίσιμη περιοχή, μέχρι τη στιγμή που θα το καταφέρει, κάθε άλλη διεργασία θα μπει το πολύ μέχρι μία φορά στην κρίσιμη περιοχή.

```
enter_region:
     TSL REGISTER, LOCK   | Αντέγραψε το κλείδωμα στον
                          | καταχωρητή και θέσε τιμή 1
     CMP REGISTER, #0     | Είχε το κλείδωμα τιμή 0;
     JNE enter_region     | Αν όχι τότε κάποια άλλη είναι
                          | στην κρίσιμη περιοχή – loop
     MOVE IN, #0          | Θέσε τιμή 0 στον έλεγχο
     RET                  | Επιστροφή: επιτεύχθηκε η είσοδος

leave_region:
     MOVE LOCK, #0        | Θέσε τιμή 0 στο κλείδωμα
     TSL REGISTER, IN     | Αντέγραψε τον έλεγχο στον
                          | καταχωρητή και θέσε τιμή 1
     CMP REGISTER, #0     | Είχε ο έλεγχος τιμή 0;
     JNE leave_region     | Αν όχι τότε κάποια άλλη δεν έχει
                          | μπει στην κρίσιμη περιοχή – loop
     RET                  | Επιστροφή: μπήκαν όλες
```

Χρησιμοποιούμε δηλαδή μια δεύτερη αναμονή με απασχόληση η οποία μας εξασφαλίζει ότι κάθε διεργασία θα μπει το πολύ μέχρι μία φορά στην κρίσιμη περιοχή μέχρι να μπουν όλες (λειτουργεί δηλαδή ως φράγμα – barrier).

4. Δίδεται ο ακόλουθος κώδικας εισόδου / εξόδου κρίσιμης περιοχής. Είναι σωστός;

```
# define FALSE 0
# define TRUE 1
int turn = 0;
int blocked[2];
enter_region(int process)
{
   blocked[process]=TRUE;
   while(turn != process)
   {
        while(blocked[1-process]);
        turn = process;
   }
}
exit_region(int process)
{
   blocked[process] = FALSE;
}
```

Το πρόβλημα παρουσιάζεται αν έχουμε context switch στην εντολή:

```
blocked[process]=TRUE;
```

Τότε θα καταλήξουμε και με τις δύο θέσεις του πίνακα blocked να έχουν την τιμή TRUE με αποτέλεσμα και οι δύο διεργασίες να μπουν σε infinite loop (`while(blocked[1-process]);`) κι έτσι να μην μπει καμία στην κρίσιμη περιοχή.

5. Έστω η ατομική εντολή f_i(M,R) (fetch and increment), η οποία επιστρέφει στον καταχωρητή R την τιμή της θέσης μνήμης M και αυξάνει την M κατά 1. Γράψτε κώδικα ο οποίος ελέγχει την πρόσβαση στις κρίσιμες περιοχές με τη βοήθεια της εντολής αυτής. Συγκρίνετε και αναφέρατε πλεονεκτήματα και μειονεκτήματα των f_i και TSL

Η προφανής λύση είναι να αφαιρούμε 1 από το κλείδωμα (M) μετά από κάθε f_i(M,R). Κάτι τέτοιο όμως μπορεί να οδηγήσει στην συνθήκη ανταγωνισμού, όπου μία δεύτερη διεργασία θα εκτελούσε f_i(M,R) (λόγω context switch) ακυρώνοντας έτσι την αύξηση με την αφαίρεσή της, με αποτέλεσμα να υπάρχει ο κίνδυνος δύο διεργασίες να βρεθούν ταυτόχρονα στην κρίσιμη περιοχή τους. Μια εναλλακτική λύση είναι να επαναφέρουμε την τιμή του κλειδώματος σε μια σταθερή τιμή κατά την *leave_region,* αποφεύγοντας έτσι την προαναφερθείσα συνθήκη ανταγωνισμού.

```
enter_region:
     f_i(M,R)              | Αντέγραψε το κλείδωμα (M) στον
                           | καταχωρητή (R) και αύξησε την
                           | τιμή κατά 1
     CMP R, #0             | Είχε το κλείδωμα τιμή 0;
     JNE enter_region      | Αν όχι τότε κάποια άλλη είναι
                           | στην κρίσιμη περιοχή – loop
     RET                   | Επιστροφή: επιτεύχθηκε η είσοδος

leave_region:
     MOVE M, #0            | Θέσε τιμή 0 στο κλείδωμα
     RET                   | Επιστροφή
```

Όπως βλέπουμε και από των παραπάνω κώδικα η συμπεριφορά της f_i(M,R) είναι παρόμοια με αυτή της TSL. Ένα συγκριτικό πλεονέκτημά της είναι ότι καθώς δεν εναλλάσσεται μόνο μεταξύ των τιμών 1 και 0 μπορεί να χρησιμοποιηθεί έτσι ώστε να εξασφαλίζει ότι κάθε διεργασία θα μπει το πολύ μέχρι μία φορά στην κρίσιμη περιοχή μέχρι να μπουν όλες (λειτουργεί δηλαδή ως φράγμα – barrier), χωρίς να χρειάζεται και δεύτερη ενεργό αναμονή όπως συμβαίνει στην TSL (στην *leave_region*).

Το πλεονέκτημα που παρουσιάζουν εντοπίζεται σε πολυεπεξεργαστικά συστήματα με περισσότερες από μία CPU. Οι πράξεις της ανάγνωσης και της αποθήκευσης στη λέξη αυτή είναι αδιαίρετες → κανείς επεξεργαστής δεν μπορεί να προσπελάσει τη λέξη αυτή μέχρι την ολοκλήρωση της εντολής. Όταν η CPU εκτελεί αυτή την εντολή, κλειδώνει το δίαυλο μνήμης (memory bus), ώστε να αποτρέψει την προσπέλαση της λέξης μνήμης από άλλες CPU μέχρι να τελειώσει την εργασία της.

Το μειονέκτημά τους είναι ότι χρησιμοποιούν ενεργό αναμονή (busy waiting), γεγονός που συνεπάγεται σπατάλη χρόνου και πόρων της CPU και άρα δεν ενδείκνυται σε περιπτώσεις όπου η αναμονή είναι μεγάλη.

6. Τι είναι threads (ίνες); Γιατί χρησιμοποιούνται στην κατασκευή servers;

Τα νήματα (threads) είναι διαφορετικά από τις διεργασίες με την έννοια ότι ένα νήμα δεν αρκεί για να εκτελεστεί πλήρως ένα ολόκληρο πρόγραμμα. Ένα νήμα (ή ελαφριά διεργασία όπως αποκαλείται συχνά) είναι απλά ένα ενεργό μέρος σε ένα πρόγραμμα – κάτι που κάνει η CPU εκ μέρους ενός προγράμματος, αλλά όχι αρκετό για να αποτελεί ολόκληρη διεργασία. Παρά το γεγονός όμως ότι ένα νήμα πρέπει υποχρεωτικά να εκτελείται μέσα σε μια διεργασία, αποτελεί ανεξάρτητη οντότητα και μπορεί να επιδέχεται χειρισμό ξεχωριστά από αυτή. Τα νήματα θυμούνται τι έχει κάνει το καθένα ξεχωριστά αλλά μοιράζονται και την πληροφορία σχετικά με τους πόρους που χρησιμοποιεί η διεργασία μέσα στην οποία τρέχουν καθώς και σε ποια κατάσταση αυτή βρίσκεται. Ένα νήμα είναι μονάχα μία *ανάθεση της CPU*. Πολλά νήματα μπορούν να συνεργάζονται συνεισφέροντας σε μία μόνο διεργασία (*πολυνημάτωση*) ενώ κάθε διεργασία πρέπει να έχει τουλάχιστον ένα νήμα.

Τα νήματα αλλάζουν απλά τον χρονισμό (timing) των λειτουργιών κι επομένως χρησιμοποιούνται κυρίως ως μια «κομψή» λύση σε προβλήματα επιδόσεων.

a) background processing: κάποιες λειτουργίες μπορεί να μην είναι time critical αλλά να είναι απαραίτητη η συνεχής εκτέλεσή τους
b) λειτουργίες I/O: I/O στο δίσκο ή στο δίκτυο μπορεί να έχει απρόβλεπτες καθυστερήσεις → τα νήματα μας εξασφαλίζουν πως αυτή η καθυστέρηση δεν θα επηρεάζει μέρη της εφαρμογής που δεν σχετίζονται άμεσα

Ο κοινός παρανομαστής είναι επομένως πως στον server κάποιες λειτουργίες μπορεί να προκαλέσουν μεγάλη καθυστέρηση ή να καταναλώνουν μεγάλο μέρος της CPU, αλλά αυτή η καθυστέρηση ή η εκτεταμένη χρήση της CPU να μην γίνεται αποδεκτή από τις άλλες λειτουργίες: χρειάζεται να εξυπηρετηθούν *άμεσα*.

Η χρήση των threads κάνει τη διαφορά καθώς απλοποιεί πολλά ζητήματα επιδόσεων και χρηστικότητας κάτι που οφείλεται στο γεγονός ότι μπορούν να υλοποιηθούν (και να συγχρονιστούν) *στο επίπεδο του χρήστη* και χωρίς να απαιτείται η αντιγραφή όλης της διεργασίας.
Αντίθετα η χρήση νέων διεργασιών:

- χρονοβόρα η δημιουργία (χρειάζεται κλήση στον πυρήνα),
- μπορεί να οδηγήσει σε context switch
- σπαταλάει πόρους από τη μνήμη (αναδημιουργείται όλη η διεργασία)
- κόστος επικοινωνίας και συγχρονισμού (χρειάζεται κλήση στον πυρήνα)

Επίσης σε πολυεπεξεργαστικά συστήματα (τα οποία συνήθως χρησιμοποιούνται για την κατασκευή servers) μια εφαρμογή με ένα μόνο νήμα δεν μπορεί να κάνει αποδοτική χρήση πολλών επεξεργαστών.

7. Τα threads μπορούν να υλοποιηθούν σε επίπεδο χρήστη χωρίς την παρέμβαση του Λειτουργικού Συστήματος. Τι μειονέκτημα έχουν αυτά τα threads και πώς αντιμετωπίζεται;

Δημιουργία threads:

- Στον πυρήνα (kernel-level threads): τόσο τα νήματα όσο και οι διεργασίες χειρίζονται και χρονοπρογραμματίζονται στον πυρήνα.
- Στο επίπεδο χρήστη (user-level threads): ο πυρήνας χειρίζεται τις διεργασίες αλλά ο χρήστης είναι υπεύθυνος για τη δημιουργία, την καταστροφή και τον χρονοπρογραμματισμό των νημάτων.

K/L threads vs. U/L threads

	K/L threads	U/L threads
(+)	Scheduling από το ΛΣ → αν ένα νήμα μπλοκαριστεί το ΛΣ μπορεί να το χειριστεί (να δώσει τον έλεγχο σε κάποιο άλλο)	• Το νήμα χρειάζεται κλήση συνάρτησης → γρήγορο • Δεν υπάρχει ανάγκη να έχουν γενική μορφή καθώς υλοποιούνται από βιβλιοθήκη που βρίσκεται σε επίπεδο χρήστη
(-)	• Το νήμα χρειάζεται κλήση συστήματος → αργό • Τα νήματα πρέπει να έχουν γενική μορφή για όλες τις εφαρμογές • Ο πυρήνας δεν γνωρίζει αν υπάρχει παραλληλία σε εφαρμογές σε επίπεδο χρήστη → όχι πλήρης εκμετάλλευση πολλών επεξεργαστών	Scheduling από χρήστη → μπορεί να μπλοκαριστεί ολόκληρη η διεργασία χωρίς το ΛΣ να μπορεί να παρέμβει

Τα παραπάνω μειονεκτήματα μπορούν να αντιμετωπιστούν με την εισαγωγή ενός thread manager σε επίπεδο χρήστη ο οποίος θα προγραμματίζει τα U/L threads χρησιμοποιώντας εικονικούς επεξεργαστές που θα αντιστοιχούν στους πραγματικούς που χειρίζεται ο πυρήνας. Με αυτόν τον τρόπο οι εφαρμογές σε επίπεδο χρήστη μπορούν να ζητούν ή να παραδίδουν ένα ή περισσότερους εικονικούς επεξεργαστές, όπως απαιτείται προκειμένου να ανταποκριθούν στις απαιτήσεις μιας παραλληλίας επιπέδου χρήστη. Ο πυρήνας με τη σειρά του ενημερώνει τον manager για σημαντικά γεγονότα όπως το μπλοκάρισμα ενός thread ή η διαθεσιμότητα φυσικών επεξεργαστών με τη χρήση upcalls και scheduler activations.

8. Κάτω από ποιες συνθήκες ένα thread μπορεί να κληρονομήσει το stack του thread που το καλεί και να μην έχει το δικό του ξεχωριστό stack;

Ένα stack περιέχει μεταβλητές, κλήσεις συναρτήσεων κλπ. Ένα thread επομένως που δεν έχει τοπικές μεταβλητές, δεν καλεί συναρτήσεις και γενικά δεν χρειάζεται να αποθηκεύσει κάτι τοπικά και μπορεί (έχει τη δυνατότητα) να κληρονομήσει το stack του thread που το καλεί;
Δηλαδή όλες οι μεταβλητές βρίσκονται σε κοινόχρηστη μνήμη και το thread που καλείται δεν κάνει κλήσεις συναρτήσεων (μπορεί π.χ. απλά να κάνει αναθέσεις τιμών και πράξεις με τις μεταβλητές της shared memory)

9. Ποιες μέθοδοι συγχρονισμού χρησιμοποιούν ενεργό αναμονή; Ποιες μέθοδοι συγχρονισμού δεν χρησιμοποιούν ενεργό αναμονή; Αναφέρατε τις περιπτώσεις όπου υπερτερεί κάθε μία από τις παραπάνω κατηγορίες. Σχεδιάστε μια μέθοδο συγχρονισμού που συνδυάζει τα πλεονεκτήματα και των δύο κατηγοριών.

- TSL, Peterson
- semaphores, mutex, messages, monitors

- Peterson: απλή λύση, εύκολη στην υλοποίηση (δεν χρειάζεται υποστήριξη από το υλικό, βιβλιοθήκη από το ΛΣ)
- TSL: εφαρμόζεται σε πολύ-επεξεργαστικά συστήματα, είναι γρήγορη (λόγω του ότι βασίζεται σε hardware)
- Δεύτερη κατηγορία: δεν χρησιμοποιούν ενεργό αναμονή → λιγότερο κόστος σε CPU

Μία από τις μεθόδους της δεύτερης κατηγορίας που να υλοποιείται με τη συνδρομή του hardware, δηλαδή με ατομικές εντολές (TSL) με αποτέλεσμα να είναι γρήγορη (λόγω υλικού), εύκολη στην υλοποίηση (γίνεται στο επίπεδο του πυρήνα κι άρα δεν απαιτούνται ξεχωριστές βιβλιοθήκες) και δεν χρησιμοποιεί ενεργό αναμονή (λιγότερο κόστος στη CPU)

10. Οι νέοι επεξεργαστές έχουν όλο και μεγαλύτερο αριθμό καταχωρητών. Τι αποτέλεσμα έχει αυτό στο context switch. Πώς μπορεί αυτή η τάση της αρχιτεκτονικής να οδηγήσει σε μικρότερους χρόνους εναλλαγής διεργασιών;

Σε κάθε context switch χρειάζεται να μεταφερθούν στην μνήμη τα περιεχόμενα όλων των καταχωρητών. Αν αυτοί είναι περισσότεροι, αυξάνονται και τα δεδομένα που πρέπει να μεταφερθούν, με αποτέλεσμα το context switch να παίρνει περισσότερο χρόνο.

Αν όμως κάθε διεργασία μπορεί να δει μόνο ένα μέρος των καταχωρητών τότε θα ήταν δυνατή η αποθήκευση στην μνήμη πολλών διεργασιών ταυτόχρονα. Το context switch μεταξύ τους θα είναι σημαντικά ταχύτερο καθώς δεν θα χρειάζεται μεταφορά των καταχωρητών στην μνήμη και πάλι πίσω στην CPU.

Οι διεργασίες δεν θα υποφέρουν από σημαντική μείωση επιδόσεων καθώς ελάχιστες διεργασίες έχουν σημαντική ανάγκη από τόσους πολλούς καταταχωρητές όσους έχουν οι σύγχρονοι επεξεργαστές.

11. Διαφορά interrupts (διακοπές) – traps (παγίδες)

Το trap προκαλείται από το λογισμικό ενώ το interrupt απευθείας από το υλικό. Στο trap γνωρίζουμε ποια διεργασία το προκάλεσε, ενώ στο interrupt καλείται ο interrupt handler για να διαπιστώσει ποια διεργασία και γιατί το προκάλεσε.

12. Περίπτωση που αποτυγχάνει η κλίση

fork
έχει γεμίσει ο process table → δεν μπορεί να δημιουργηθεί νέα διεργασία
exec
αν δεν έχουμε άδεια εκτέλεσης της διεργασίας
αν το stack του καινούριου process που θέλουμε να δημιουργήσουμε είναι αρκετά μεγάλο και δεν χωράει στη μνήμη
unlink
αν δεν υπάρχει το αρχείο ζητηθεί αρχείο το όποιο το προσπελάζει και κάποια άλλη διεργασία

13. Τι είναι το context switch;

In your average, memory-protected environment, a "context" is a virtual address space, the executable contained in it, its data etc.
A "context switch" occurs for a variety of reasons - because a kernel function has been called, the application has been preempted, or because it had yielded its time slice.
A context switch involves storing the old state and retrieving the new state
Because a context switch can involve changing a large amount data it can be the one most costly operation in an operating system.

Αδιέξοδα

14. Αναφέρατε τις 4 συνθήκες αδιεξόδου. Δώστε ένα παράδειγμα που ικανοποιούνται.

1. αμοιβαίου αποκλεισμού
2. δέσμευσης και αναμονής
3. μη προεκτόπισης
4. κυκλική αναμονή

- 3 διεργασίες
- πόροι: 2 CD-RW 1 εκτυπωτής
- η πρώτη εχει CD ζητάει εκτυπωτή, $2^{η}$ έχει εκτυπωτή ζητάει $2^{ο}$ CD, $3^{η}$ έχει $2^{ο}$ CD και ζητάει το $1^{ο}$ → αδιέξοδο διότι:
    - αμοιβαίος αποκλεισμός (κάθε πόρος δεσμευμένος από μία μόνο διεργασία)
    - δέσμευσης και αναμονής (κάθε διεργασία έχει ήδη δεσμεύσει ένα πόρο και ζητάει και άλλους)
    - μη προεκτόπισης (δεν μπορούν οι πόροι αυτοί να αφαιρεθούν με ασφάλεια)
    - κυκλική αναμονή (έχουμε κυκλικό γράφο)

15. Έστω ο ακόλουθος τρόπος αποφυγής αδιεξόδου: όλοι οι πόροι αριθμημένοι, οι διεργασίες είναι υποχρεωμένες να ζητούν πόρους με αύξουσα αριθμητική σειρά. Αποδείξτε ότι αποφεύγει το αδιέξοδο.

Ζητάμε <u>σειριακά</u> πόρους και ποτέ μικρότερους από τον πρώτο που ζητήσαμε.
→ δεν ικανοποιούνται οι συνθήκες i και iv (δεν έχουμε κύκλο)

16. Αλγόριθμος του τραπεζίτη.

<σελ. 229>

Μνήμη

17. Τι είναι ο ανεστραμμένος πίνακας σελίδων; Ποιο είναι το βασικό μειονέκτημά του; Αναφέρατε (λεπτομερώς) πως γίνεται η προσπέλασή του;

Οι ανεστραμμένοι πίνακες είναι ένας εναλλακτικός τρόπος απεικόνισης/ αποθήκευσης του πίνακα σελίδων. Η ανάγκη για εξοικονόμηση του χώρου που καταλαμβάνει ένας πίνακας σελίδων, ιδιαίτερα όταν ο χώρος εικονικών διευθύνσεων είναι πολύ μεγαλύτερος από τη φυσική μνήμη οδήγησε στους ανεστραμμένους πίνακες σελίδων.
Είναι ένας πίνακας σελίδων όπου υπάρχει μια καταχώρηση ανά πλαίσιο σελίδας στην πραγματική μνήμη, αντί να υπάρχει μια καταχώρηση ανά σελίδα του εικονικού χώρου διευθύνσεων.

Ωστόσο , το κύριο μειονέκτημά τους – η δύσκολη μετάφραση των εικονικών διευθύνσεων σε φυσικές- τους κάνει να καθυστερούν αρκετά τη λειτουργία της μηχανής, εκτός αν χρησιμοποιηθεί η λύση του κατακερματισμού για τις εικονικές διευθύνσεις.

Η προσπέλαση γίνεται χρησιμοποιώντας την TLB με μια μικρή μετατροπή:

1. Αν η TLB διατηρεί όλες τις σελίδες που χρησιμοποιούνται συχνά, η μετάφραση είναι δυνατό να γίνεται όσο γρήγορα γινόταν και με τους κανονικούς πίνακες σελίδων ενώ όταν έχουμε αποτυχία στην TLB χρησιμοποιούμε ένα πίνακα κατακερματισμού για τις εικονικές διευθύνσεις
2. όλες οι εικονικές σελίδες που βρίσκονται την τρέχουσα χρονική στιγμή στη μνήμη και έχουν την ίδια τιμή κατακερματισμού σχηματίζουν αλυσίδα
3. αν ο πίνακας κατακερματισμού διαθέτει τον ίδιο αριθμό θέσεων με τον αριθμό των φυσικών σελίδων που έχει η μηχανή, η μέση αλυσίδα θα έχει μήκος μία μόνο καταχώρηση, γεγονός που επιταχύνει σημαντικά την χαρτογράφηση
4. από τη στιγμή που θα βρεθεί ο αριθμός πλαισίου σελίδας το νέο ζευγάρι (εικονική – φυσική σελίδα) εισάγεται στην TLB

18. Αν η αρχιτεκτονική μας δεν υποστηρίζει bits αναφοράς και μεταβολής πώς μπορούμε να τα προσομοιώσουμε σε λογισμικό; Ποιο είναι το κόστος προσομοίωσης;

Το bit αναφοράς αποθηκεύεται στον header κάθε σελίδας. Έχει αρχική τιμή 0 η οποία αλλάζει όταν γίνεται αναφορά σε αυτή τη σελίδα (είτε για ανάγνωση είτε για εγγραφή). Χρησιμοποιείται για να βοηθά το ΛΣ στο να αποφασίσει ποια σελίδα θα αντικαταστήσει, ανάλογα με τον αλγόριθμο σελιδοποίησης που εφαρμόζεται.

Το bits τροποποίησης (dirty bit) επίσης βρίσκεται στο header κάθε σελίδας και η τιμή του μεταβάλλεται όταν γίνεται αναφορά σε αυτή τη σελίδα για εγγραφή. Χρησιμοποιείται για να ξέρει το ΛΣ αν όταν θα αντικαταστήσει τη σελίδα χρειάζεται να τη γράψει στο δίσκο.

Η εξομοίωση γίνεται από το λογισμικό με τη χρήση των valid και protection bit

Πώς εξομοιώνουμε το reference bit:

- Αρχικά το valid bit έχει την τιμή 0 → δεν επιτρέπεται η πρόσβαση στη σελίδα (σημαίνει ότι είναι σκουπίδια – non valid) αλλά η σελίδα εξακολουθεί να βρίσκεται στη μνήμη και το ΛΣ γνωρίζει εσωτερικά (μέσω ενός πίνακα που κρατάει αυτές τις τιμές για κάθε σελίδα) αν είναι έγκυρη (και άρα το περιεχόμενό της είναι ίδιο με αυτό που βρίσκεται στο δίσκο) ή όχι
- Γίνεται μία προσπάθεια αναφοράς στη σελίδα → page fault λόγω μη έγκυρης σελίδας → το ΛΣ διαπιστώνει αν ισχύει (και θέτει valid bit = 1 χωρίς να διαβάζει τη σελίδα από το δίσκο) ή όχι (οπότε και επαναφέρει τη σελίδα από το δίσκο)
- Έτσι το valid bit της σελίδας χρησιμοποιείται και ως reference bit

Πώς εξομοιώνουμε το modified (dirty) bit:

- Ομοίως με τα παραπάνω το protection bit έχει αρχικά τιμή r (read only) → δεν επιτρέπεται η εγγραφή στη σελίδα αλλά το ΛΣ γνωρίζει εσωτερικά (μέσω ενός πίνακα που κρατάει αυτές τις τιμές για κάθε σελίδα) αν η σελίδα μπορεί να εγγραφεί
- Γίνεται μία προσπάθεια εγγραφής στη σελίδα → page fault λόγω write protected σελίδας → το ΛΣ διαπιστώνει αν μπορεί να εγγραφεί (και θέτει protection bit = rw) ή όχι (οπότε και έχουμε memory access violation error)
- Έτσι το protection bit της σελίδας χρησιμοποιείται και ως modified bit

Η εξομοίωση είναι πολύ πιο αργή από την υλοποίηση του υλικού (διότι προκαλούνται πολύ περισσότερα page faults) και επίσης το λειτουργικό πρέπει

να ξέρει εσωτερικά για κάθε σελίδα αν είναι valid και αν μπορεί να εγγραφεί (περισσότερη μνήμη).

19. Δώστε ένα παράδειγμα (πρόγραμμα, δεδομένα και μέγεθος μνήμης) στην οποία η LRU δεν αποδίδει καλά και αναφέρατε ποια μέθοδος αντικατάστασης σελίδων αποδίδει καλύτερα.

Έχουμε real time video (μεγέθους n+1 σελίδων) και μνήμη μεγέθους n σελίδων.

- Οι πρώτες n σελίδες τοποθετούνται στη μνήμη
- Όταν ζητηθεί η επιπλέον σελίδα του τελευταίου κομματιού του video ο LRU αντικαθιστά την πρώτη.
- Αν ζητηθεί ξανά η αναπαραγωγή του video (δηλαδή ζητείται ξανά η πρώτη σελίδα) τότε ο LRU θα την τοποθετήσει στη θέση της δεύτερης (ως η πιο παλιά) κ.ο.κ. για τις επόμενες → αυτό οδηγεί σε 2n+2 page faults (στο σύνολο του video)

Αν χρησιμοποιήσουμε ένα αλγόριθμο ο οποίος αντικαθιστά κάποια από τις πιο πρόσφατα χρησιμοποιημένες σελίδες τότε δεν θα έχουμε την σειριακή αντικατάσταση που είδαμε παραπάνω με αποτέλεσμα να προκύψουν συνολικά λιγότερα page faults

20. Έστω ένα υπολογιστικό σύστημα με επεξεργαστή 1 GHz. Έστω ότι ο χρόνος προσπέλασης της μνήμης είναι 10 κύκλοι, ενώ ο μέσος χρόνος προσπέλασης του δίσκου είναι 10 msec. Ποιο θα πρέπει να είναι το page fault rate για ένα τέτοιο σύστημα;

- Κάθε κύκλος διαρκεί 1/CPU clock = 1/10^9 = 10^-9 sec
- Η προσπέλαση της μνήμης είναι 10 κύκλοι
- Επομένως έχουμε 10/10^9 secs για την εμφάνιση page fault
- Από την εμφάνιση του page fault μέχρι να έρθει η ζητούμενη σελίδα στη μνήμη έχουμε 10 msec
- Επομένως ο συνολικός χρόνος είναι t = 10^-8 + 10^-6 secs
- Άρα το maximun page fault rate είναι 1/t=990099 page faults / sec
- Όπότε το page fault rate του συστήματος πρέπει να είναι κάθε φορά μικρότερο ή ίσο του maximum

21. Τι είναι σελιδοποίηση (paging); Τι είναι τμηματοποίηση (segmentation); Σύγκριση.

Είναι μέθοδοι αντιστοίχισης της φυσικής μνήμης με τη μνήμη που βλέπει ο χρήστης.

Paging

- Ένα πρόγραμμα του χρήστη βλέπει τη μνήμη ως ένα μεγάλο συνεχόμενο χώρο που περιέχει μόνο το πρόγραμμα αυτό
- Επιτρέπεται στο φυσικό χώρο διευθύνσεων μιας διεργασίας να μην είναι συνεχόμενος
- Η φυσική μνήμη είναι χωρισμένη σε σταθερού (fixed) μεγέθους πλαίσια
- Η λογική μνήμη είναι χωρισμένη σε σταθερού μεγέθους σελίδες
- Υπάρχει πίνακας σελίδων
- Μία λογική διεύθυνση αποτελείται από ένα αριθμό σελίδας και ένα page offset
- Δεν υπάρχει εξωτερικό fragmentation
- Υπάρχει κάποιο περιορισμένο εσωτερικό fragmentation
- Μικρό μέγεθος σελίδας → λιγότερο εσωτερικό fragmentation
- Μεγάλο μέγεθος σελίδας → μικρότερος πίνακας σελίδων και πιο αποτελεσματική I/O δίσκου
- Πίνακας πλαισίων
  - Διαχειρίζεται από το ΛΣ
  - Δείχνει ποια πλαίσια είναι ελεύθερα ή δεσμευμένα (και από ποια διεργασία)
- Το ΛΣ κρατά αντίγραφο του πίνακα σελίδων για κάθε διεργασία ώστε να μεταφράζει την εικονική διεύθυνση σε φυσική → μπορεί να έχουμε αυξημένο χρόνο context switch

```
* HW support for paging
  - page table base register (PTBR)
      - keep page tables in memory
      - just change the PTBR value
      - reduce context switch time
  - translation look-aside buffer (TLB)
      - HW cache
      - speed up memory access by reducing #page table references
      - Ex.
      (1) No TLB
            memory access time = 100ns
            effective access time = 200ns
      (2) TLB
            hit ratio = 98%
            TLB access time = 20ns
            effective access time = 0.98 * 120 + 0.02 * 220
```

```
= 122ns
```

Segmentation:

- Ο χρήστης βλέπει τη μνήμη ως μια συλλογή από μεταβλητού μεγέθους κομμάτια.
- Εικονικός χώρος διευθύνσεων
  - συλλογή τμημάτων
  - κάθε τμήμα έχει όνομα και μέγεθος
  - διεύθυνση = όνομα τμήματος και offset
    - υλοποίηση <segment#, offset>
- Δεν υπάρχει εσωτερικό fragmentation
- Υπάρχει εξωτερικό fragmentation (σκακιεροποίηση – διορθώνεται με σύμπτυξη)

HW

- segmentation table
- segment table base register
- segment table length register

Σύγκριση : <σελίδα 314>

Συνδυασμός: ο χρήστης βλέπει segments, αλλά το ΛΣ χωρίζει εσωτερικά τη μνήμη σε σελίδες (η εικονική μνήμη είναι fragmented). Σήμερα προτιμάται αυτή η λύση (π.χ. Pentium, MULTICS)

22. Μερικοί υπολογιστές μας παρέχουν κύρια μνήμη (~1MB) τα περιεχόμενα της οποίας δεν σβήνουν αν κοπεί το ρεύμα. Αναφέρατε πιθανές χρήσεις της μνήμης αυτής.

Χρησιμεύουν για την αποθήκευση του BIOS του Η/Υ (Basic Input Output System) το οποίο παρέχει ένα βασικό πρόγραμμα έναρξης το οποίο αρχικοποιεί το υλικό και παραδίδει τον έλεγχο στο λειτουργικό σύστημα.

Μια άλλη χρήση είναι για τις ευσταθείς εγγραφές:

- τοποθετούν τον αριθμό του μπλοκ που αντιγράφουν ή ενημερώνουν στη μνήμη αυτή πριν την έναρξη της εγγραφής
- αν η εγγραφή ολοκληρωθεί επιτυχώς τότε αντικαθίσταται με έναν άκυρο αριθμό (π.χ. -1)
- σε περίπτωση κατάρρευσης το πρόγραμμα ανάκτησης μπορεί να ελέγξει αυτή τη μνήμη και στη συνέχει να ελέγξει τα δύο αντίγραφα του συγκεκριμένου μπλοκ για να διαπιστώσει αν είναι σωστά και συνεπή

23. Μετά από ένα context switch τα περιεχόμενα της TLB ακυρώνονται. Γιατί; Μετά από ένα context switch τα περιεχόμενα της cache συνήθως δεν ακυρώνονται. Γιατί; Μερικές φορές μετά από ένα context switch και τα περιεχόμενα της cache ακυρώνονται. Γιατί;

Στην TLB υπάρχουν αντιστοιχίες μεταξύ virtual address και physical address --> επειδή όμως ίδια v. addr αντιστοιχούν σε διαφορετικά ph addr επιβάλλεται το flushing του TLB αλλιώς μπορεί η μια να μπει στην πραγματικότητα στο χώρο της άλλης (καθώς υπάρχει η περίπτωση να έχουν ίδια virtual address)
Η cache βλέπει physical addresses κι άρα δεν υπάρχει η παραπάνω ανάγκη για flushing.
Σε πολυεπεξεργαστικά συστήματα όπου κάθε επεξεργαστής έχει τη δικιά του cache και υπάρχει μια ενιαία κύρια μνήμη ίσως να χρειάζεται ακύρωση των περιεχομένων για λόγους cache coherence.

24. Παράδοξο του Belady.

Ο Belady ανακάλυψε ένα αντί-παράδειγμα στο οποίο ο αλγόριθμος FIFO προκαλεί περισσότερα σφάλματα σελίδας με 4 πλαίσια σελίδας αντί με 3, εάν οι αναφορές γίνουν με τη σειρά 0,1,2,3 ,0,1,4,0,1,2,3,4.

Ο βέλτιστος αλγόριθμος αντικατάστασης δεν υποφέρει από το παράδοξο αυτό γιατί είναι αλγόριθμος στοίβας. Αλγόριθμοι στοίβας είναι αυτοί που έχουν την εξής ιδιότητα:

$$M(m,a) \subseteq M(m+1,a)$$

Όπου μ είναι ο αριθμός των πλαισίων σελίδας που διαθέτει η μνήμη και α είναι ένας δείκτης θέσης στο αλφαριθμητικό αναφορών. Η σχέση αυτή δηλώνει ότι το σύνολο των σελίδων που βρίσκονται στο άνω τμήμα του M (memory) όταν η μνήμη διαθέτει μ πλαίσια σελίδας και έχουν α αναφορές στην μνήμη περιέχονται επίσης στο M όταν η μνήμη διαθέτει μ+1 πλαίσια σελίδας.

25. Τι είναι εξωτερική και τι εσωτερική κατάτμηση;

Όταν χρησιμοποιούμε κατάτμηση, και αφού το σύστημα δουλέψει για λίγο, η μνήμη θα χωριστεί σε κομμάτια, μερικά από τα οποία θα περιέχουν τμήματα (segments) και μερικά θα περιέχουν οπές. Το φαινόμενο αυτό ονομάζεται σκακιεροποίηση ή εξωτερική κατάτμηση

Φορτώνοντας δεδομένα στη μνήμη, η τελευταία σελίδα θα περιέχει κατά πάσα πιθανότητα λιγότερα δεδομένα από το συνολικό μέγεθος σελίδας και η υπόλοιπη θα είναι άδεια. Το φαινόμενο αυτό ονομάζεται εσωτερική κατάτμηση.

26. Τι είναι οι στοιχειοσειρές αναφορών και τι οι στοιχειοσειρές αποστάσεων; Πού χρησιμοποιούνται;

Η πρόσβαση μιας διεργασίας στη μνήμη μπορεί να χαρακτηριστεί με μια διατεταγμένη λίστα από αριθμούς σελίδων. Η λίστα αυτή ονομάζεται αλφαριθμητικό αναφορών και χρησιμοποιείται στην μοντελοποίηση των αλγορίθμων αντικατάστασης σελίδας.

Το αλφαριθμητικό απόστασης είναι μια πιο γενικευμένη μορφή του αλφαριθμητικού αποστάσεων, όπου κάθε αναφορά σε μια σελίδα δηλώνεται από την απόσταση από την κορυφή της στοίβας στην οποία βρέθηκε η σελίδα στην οποία έγινε αναφορά. Χρησιμοποιείται στους αλγόριθμους στοίβας.

Αρχεία

27. Τι είναι RAM disk; Κάποιος θα μπορούσε να υποστηρίξει ότι αντί να χρησιμοποιούμε την κύρια μνήμη σαν RAM disk θα μπορούσαμε να τη χρησιμοποιούμε σαν file cache και να έχουμε την ίδια απόδοση αφού και στις δύο περιπτώσεις τα αρχεία θα βρίσκονται στην κύρια μνήμη. Σχολιάστε.

Είναι ένα κομμάτι της μνήμης το οποίο γίνεται mount από το ΛΣ ως δίσκος ή φάκελος και είναι ορατός από το filesystem του ΛΣ.
Παράδειγμα η boot disk του DOS (βάζαμε συμπιεσμένα περισσότερα προγράμματα στη δισκέτα των 1,44MB και τα αποσυμπιέζαμε για χρήση σε ένα RAM disk που παίρναμε από τη μνήμη των 8MB)
Ο λόγος είναι ότι με τη χρήση ως RAM disk επιτρέπουμε στο χρήστη την προσπέλαση μέσω του filesystem και του δίνουμε τα privileges που έχει και σε ένα κοινό δίσκο, σε αντίθεση με την file cache που δεν δίνει τέτοια δυνατότητα καθώς αποτελεί κρυφή μνήμη.

28. Τι είναι DMA; Πώς αλληλεπιδρά με τη σελιδοποίηση; Γιατί οι συνηθισμένοι χρήστες δεν μπορούν να ξεκινήσουν πράξεις DMA χωρίς την βοήθεια του λειτουργικού;

DMA = Direct Memory Access (Άμεση Προσπέλαση Μνήμης). Είναι ένας ελεγκτής που έχει πρόσβαση στο δίαυλο του συστήματος ανεξάρτητα από τη CPU και περιέχει διάφορους καταχωρητές στους οποίους η CPU μπορεί να διαβάζει και να γράψει δεδομένα. Η χρήση της συνεπάγεται μείωση φόρτου εργασίας για τη CPU αναφορικά με τις I/O λειτουργίες.

Να δίνονται οι φυσικές διευθύνσεις στο DMA ή οι εικονικές. Αν δοθούν εικονικές πρέπει να χρησιμοποιηθεί η MMU για να γίνει η μετάφραση και για να γίνει αυτό πρέπει η μνήμη να έχει τι δική της MMU, πράγμα πολύ σπάνιο.

i) συνήθως οι ελεγκτές DMA χρησιμοποιούν φυσικές διευθύνσεις τις οποίες δεν μπορεί να δει ο χρήστης
ii) οι πράξεις DMA δέχονται εντολές από τη CPU στην οποία ο χρήστης έχει πρόσβαση μόνο μέσω του λειτουργικού

29. Τι είναι τα αρχεία που απεικονίζονται στη μνήμη; Πλεονέκτημα και μειονεκτήματα έναντι των άλλων αρχείων. Πώς επηρεάζουν την απόδοση της TLB. Άρχισαν να χρησιμοποιούνται μόνο πριν από μερικά χρόνια. Θα μπορούσαν να είχαν αρχίσει να χρησιμοποιούνται από τη δεκαετία του 60 όταν άρχισαν να χρησιμοποιούνται τα αρχεία σαν μέσο αποθήκευσης.

Αρχεία τα οποία χαρτογραφούνται στο χώρο διευθύνσεων κάποιας διεργασίας (βρίσκονται και στο σκληρό και στη μνήμη).
(+) δεν χρειάζεται I/O από το δίσκο → ταχύτητα, ευκολία προγραμματισμού
(-) Το ΛΣ δεν μπορεί να ξέρει το ακριβές μήκος του αρχείου εξόδου → δεν γνωρίζει αν έχει χρησιμοποιηθεί όλη η σελίδα
(-) Αν ένα αρχείο χαρτογραφηθεί στη μνήμη από μία διεργασία και μια άλλη διεργασία το ανοίξει για ανάγνωση, συμβατικά από το δίσκο → αν αλλάξει το περιεχόμενο στη μνήμη η δεύτερη διεργασία δεν βλέπει αυτή την αλλαγή.
(-) Κάποιο αρχείο μπορεί να είναι μεγαλύτερο ακόμη και από ολόκληρο το χώρο των διευθύνσεων.

Αν κάνουμε πολλές σελίδες MAP μαζί η TLB θα περιέχει τις αντιστοιχίσεις αυτών --> όταν θα προσπαθήσει η CPU να βρει κάποια άλλη σελίδα, η TLB θα περιέχει τις αντιστοιχίσεις του αρχείου και άρα υπάρχει πιθανότητα να καθυστερήσει η διαδικασία λόγω miss στην TLB.

Όχι γιατί τότε 1ον το μέγεθος της μνήμης ήταν πολύ μικρό σε σχέση με το μέγεθος των αρχείων
2ον δεν υπήρχε ούτε καν η έννοια του paging

30. Μερικά συστήματα αρχείων χωρίς να ειδοποιούν τους χρήστες αποθηκεύουν τα δεδομένα σε συμπιεσμένη μορφή. Πλεονεκτήματα – μειονεκτήματα και πώς αντιμετωπίζονται.

Ποτέ δεν ξέρουμε πόσο πραγματικά χώρο έχουμε (π.χ. απόπειρα εγγραφής ήδη συμπιεσμένου αρχείου στο δίσκο). Το πρόβλημα αυτό αντιμετωπίζεται εν μέρη με την πραγματική απεικόνιση του ελεύθερου χώρου (ακόμη και αν λόγω της συμπίεσης μπορούν τελικά να χωρέσουν περισσότερα δεδομένα)

Η διαδικασία της αποσυμπίεσης για την επεξεργασία σπαταλάει πόρους του συστήματος (μνήμη, CPU). Η εισαγωγή υλικού που αναλαμβάνει αποκλειστικά την εργασία αυτή αμβλύνει κάπως το πρόβλημα.

31. Περιγράψτε ένα κόμβο – δ του UNIX. Πλεονεκτήματα και μειονεκτήματα της δομής που δείχνει στις διευθύνσεις τον μπλοκ του δίσκου.

Μια μέθοδος που χρησιμοποιείται για την παρακολούθηση των block που ανήκουν σε κάθε αρχείο είναι να αντιστοιχίζουμε σε κάθε αρχείο μια δομή δεδομένων. Αυτή η δομή ονομάζεται κόμβος i.Κάθε ένα κρατάει τις ιδιότητες του αρχείου, δείκτες στα 7 πρώτα blocks του αρχείου στο δίσκο καθώς και ένα δείκτη σε ένα άλλο μπλοκ δίσκου που περιέχει πρόσθετες διευθύνσεις δίσκου, αν χρειάζονται (το οποίο πάλι με τη σειρά του μπορεί να δείχνει σε άλλο μπλοκ κ.ο.κ.)

Κατανεμημένα - Πολυεπεξεργαστικά Συστήματα

32. Τι είναι process migration; Διαφορές από remote procedure call. Κάτω από ποιες συνθήκες είναι επιθυμητή η κάθε μια τεχνική;

Έχουμε διαδικασίες που ταξιδεύουν δια μέσου ενός δικτύου και εκτελούν εργασίες σε απομακρυσμένες μηχανές που τους παρέχουν αυτή τη δυνατότητα. Αυτό επιτρέπει σε διεργασίες τη μετανάστευση (migration) από Η/Υ σε Η/Υ, να χωρίζονται σε πολλαπλά στιγμιότυπα τα οποία τρέχουν σε διαφορετικά μηχανήματα και επιστρέφουν. Σε αντίθεση με το remote procedure call, όπου μια διεργασία απλά προκαλεί (invokes) διαδικασίες σε ένα remote host, η τεχνική του process migration επιτρέπει σε εκτελέσιμο κώδικα να ταξιδέψει και να αλληλεπιδράσει με βάσεις, file systems, πληροφοριακά συστήματα και άλλους agents.
π.χ. οι mobile agents που ταξιδεύουν στο Internet θα μπορούν να ψάχνουν πληροφορίες για προϊόντα και υπηρεσίες και να αλληλεπιδρούν με άλλους agents και να επιστρέφουν με τα αποτελέσματα.

Γενικά για να μπορούμε να έχουμε process migration σε ένα δίκτυο πρέπει να εξασφαλίσουμε τα εξής:

1. κοινή γλώσσα εκτέλεσης (execution language)
2. process persistence
3. Μηχανισμοί επικοινωνίας μεταξύ των agent hosts
4. Μηχανισμοί ασφαλείας για την προστασία των agents και των agent hosts.

33. Andrew File System. Πώς βοηθάει στη μείωση της κυκλοφορίας μεταξύ των clients και των servers.

Το AFS(Andrew File System) είναι ένα σύστημα αρχείων το οποίο επιτρέπει σε πολλά μηχανήματα να προσπελαύνουν καταλόγους και αρχεία που βρίσκονται σε ένα κοινόχρηστο σύστημα αρχείων. Υλοποιείται με την προσθήκη στον πυρήνα ενός τμήματος κώδικα που ονομάζεται venus και με την εκτέλεση στον χώρο χρήστη ενός διακομιστή αρχείων στον χώρο χρήστη(vice). Οι σταθμοί εργασίας οργανώνονται σε κελιά.

Προστίθενται δύο κατάλογοι στο (τοπικό) σύστημα αρχείων, οι cmu και cache.Ο cache περιέχει απομακρυσμένα αρχεία που έχουν αποθηκευτεί στην κεντρική μνήμη ενώ ο cmu τα ονόματα των απομακρυσμένων κελιών, κάτω από τα οποία βρίσκονται τα περιεχόμενα των αντίστοιχων καταλόγων cmu του κάθε απομακρυσμένου υπολογιστή.

Όταν γίνει κλήση της κλήσης συστήματος open ο venus κατεβάζει το απομακρυσμένο αρχείο στην cache (ή ένα τμήμα του αν είναι πολύ μεγάλο) και ενημερώνει τον file descriptor να δείχνει σε αυτό. Με το κλείσιμο του αρχείου το απομακρυσμένο αντίγραφό του ενημερώνεται.

Κάθε φορά που ο venus κατεβάζει ένα αρχείο ενημερώνει τον vice για το αν ενδιαφέρεται για την χρήση του αρχείου από διεργασίες που το χρησιμοποιούν σε κάποιον άλλο υπολογιστή. Αν ναι, ο vice καταγράφει την τοποθέτηση του αρχείου στην cache και όποτε κάποια άλλη διεργασία ανοίξει το αρχείο ο vice στέλνει μήνυμα στον venus ειδοποιώντας τον να διαγράψει την καταχώρηση που έχει στην κρυφή του μνήμη και να πάρει ένα καινούριο έγκυρο αντίγραφο.

Με αυτήν την πολιτική το AFS ελαχιστοποιεί την κίνηση ανάμεσα στους υπολογιστές καθώς δεν στέλνονται συνεχώς οι αλλαγές ανάμεσα στον υπολογιστή που επεξεργάζεται το αρχείο και σε αυτόν που βρίσκεται το πρωτότυπο αρχείο παρά μόνο μια φορά όταν ολοκληρωθεί η μετατροπή του.

34. Τι δρομολογήσεις γίνονται όταν ακολουθείται ένα URL;

<σελ 660>

35. Εξηγήστε τι πρόβλημα υπάρχει σε ένα πολύ – επεξεργαστικό σύστημα αν έχουμε context switch ενώ μια διεργασία βρίσκεται στην κρίσιμη περιοχή. Περιγράψτε δύο τουλάχιστον λύσεις που αποφεύγουν ή διορθώνουν αυτό το πρόβλημα.

Όταν συμβαίνει αυτό, η διεργασία έχει κλειδώσει το δίαυλο μνήμης. Όλες οι υπόλοιπες διεργασίες επομένως θα πρέπει να εκτελούν ενεργό αναμονή (αφού ο αμοιβαίος αποκλεισμός σε πολυεπεξεργαστικά συστήματα επιτυγχάνεται με την εντολή TSL) μέχρι να ξαναδρομολογηθεί η αρχική διεργασία από τον χρονοπρογραμματιστή και να βγει από την κρίσιμη περιοχή της.
Λύσεις:

- Τροποποίηση του χρονοπρογραμματιστή έτσι ώστε να μην διακόπτει μια διεργασία, όταν αυτή βρίσκεται στην κρίσιμη περιοχή της.
- Το ΛΣ μπορεί να κρατάει ανά τακτά χρονικά διαστήματα check points για κάθε διεργασία κι έτσι αν συμβεί context switch ενώ βρίσκεται σε κρίσιμη περιοχή να την επαναφέρει στο check point που βρισκόταν πριν την είσοδο, ελευθερώνοντας παράλληλα το δίαυλο μνήμης

Ασφάλεια

36. Τι είναι το σκουλήκι του Internet; Πώς πολλαπλασιάζεται;

Ένα πρόγραμμα που αντιγράφει τον εαυτό του που είχε τη δυνατότητα να αξιοποιήσει δύο σφάλματα του UNIX που επέτρεπαν την μη εξουσιοδοτημένη πρόσβαση σε υπολογιστές ολόκληρου του διαδικτύου και προσπαθούσε να κρύψει τα ίχνη του.

Αρχικά προσπαθούσε να εκτελέσει ένα απομακρυσμένο κέλυφος rsh στους Η/Υ με τους οποίους ήταν συνδεδεμένος ο Η/Υ στον οποίο βρισκόταν (καθώς το rsh δεν απαιτεί πιστοποίηση ταυτότητας στους υπολογιστές που χαρακτηρίζει ως έμπιστους) και αν κατάφερνε να συνδεθεί τους μόλυνε και αυτούς.
Εναλλακτικά χρησιμοποιούσε το bug του δαίμονα finger ο οποίος δεν έκανε έλεγχο υπερχείλισης κι έτσι όταν ο δαίμονας επέστρεφε δεν επέστρεφε στη main αλλά σε μια διαδικασία που είχε οριστεί στον κώδικα του worm και η οποία καλούσε το sh. Όταν γινόταν αυτό είχε στη διάθεσή του ένα κέλυφος όπου έτρεχε τον εαυτό του.
Τέλος μπορούσε να αξιοποιήσει ένα σφάλμα στο σύστημα αλληλογραφίας sendmail το οποίο επέτρεπε στο σκουλήκι να ταχυδρομήσει ένα αντίγραφο του προγράμματος εκκίνησης και στη συνέχεια να εκτελεστεί