Ασκήσεις -Ερωτήσεις
2ου Κεφαλαίου

Άσκηση 2.1 (Πρ. 2-4)

Όταν μεταφέρεται ο έλεγχος στο Λ.Σ. (από μια διακοπή ή μια κλήση συστήματος), χρησιμοποιείται πάντα μια περιοχή στοίβας στον πυρήνα, ***έξω από τη στοίβα της διεργασίας που διακόπηκε***. Γιατί;

Απάντηση 2.1

Ο πυρήνας χρησιμοποιεί ξεχωριστή, δική του στοίβα για δύο βασικούς λόγους:

***1.Αξιοπιστία*:** Εάν ένα εσφαλμένο πρόγραμμα χρήστη δεν έχει διατηρήσει επαρκή χώρο για τη στοίβα και ο πυρήνας χρησιμοποιούσε αυτή τη στοίβα, τότε *όλο το σύστημα θα κατέρρεε από ένα εσφαλμένο πρόγραμμα*.

***2.Ασφάλεια*:** Εάν μετά την επιστροφή από μία κλήση συστήματος, ο πυρήνας αφήσει δεδομένα στη στοίβα του προγράμματος χρήστη, *ένα κακόβουλο πρόγραμμα θα μπορούσε να χρησιμοποιήσει αυτά τα δεδομένα* για να μάθει πληροφορίες που αφορούν άλλες διεργασίες.

Άσκηση 2.2 (Πρ. 2-7)

Όταν μια πολυνηματική διεργασία ενεργοποιεί την κλήση ***fork( )***, ενδέχεται να παρουσιαστεί το εξής πρόβλημα.

Υποθέστε ότι ένα από τα αρχικά νήματα περίμενε για είσοδο από το πληκτρολόγιο. Η θυγατρική διεργασία λαμβάνει αντίγραφα όλων των νημάτων της γονικής. Τώρα περιμένουν *δύο νήματα για είσοδο από το πληκτρολόγιο*, ένα σε κάθε διεργασία.

Παρουσιάζεται το πρόβλημα αυτό σε μονο-νηματικές διεργασίες;

Απάντηση 2.2

Δεν δημιουργείται ποτέ αυτό το πρόβλημα στις μονονηματικές διεργασίες για τον εξής λόγο.

Αν μία μονονηματική διεργασία περιμένει Ε/Ε από το πληκτρολόγιο (άρα έχει ***μπλοκαριστεί***), ***δεν μπορεί να καλέσει την fork***.

Άσκηση 2.3 (Πρ. 2-8)

Στην εικόνα 2-8, φαίνεται ένας πολυνηματικός διακομιστής Ιστού.

Αν ο μοναδικός τρόπος να διαβάσουμε από ένα αρχείο είναι η κανονική κλήση συστήματος **read**, νομίζετε ότι στο διακομιστή Ιστού χρησιμοποιούνται νήματα επιπέδου χρήστη ή νήματα επιπέδου πυρήνα; Γιατί;

Απάντηση 2.3

Η κλήση read είναι ***blocking call*** (μπλοκάρεται η διεργασία όταν την καλέσει, μέχρι να είναι έτοιμη η Ε/Ε). Συνεπώς, ένα νήμα-εργάτης (worker thread) θα μπλοκάρει όταν έχει να διαβάσει μία σελίδα από το δίσκο.

Εάν χρησιμοποιηθούν νήματα επιπέδου χρήστη (user-level threads), η ανάγνωση που πραγματοποιεί από ένα νήμα ***θα μπλοκάρει ολόκληρη τη διεργασία***, και σε αυτή την περίπτωση ***δεν θα είχε νόημα η πολυνημάτωση***!

Στην παραπάνω περίπτωση, η χρήση των νημάτων επιπέδου πυρήνα (kernel threads) είναι απαραίτητη για να επιτραπεί σε μερικά νήματα να αναμένουν είσοδο χωρίς να μπλοκάρουν όλη τη διεργασία και τα υπόλοιπα νήματά της.

Άσκηση 2.4 (Πρ. 2-11)

Γιατί θα ήθελε ένα νήμα να επιστρέψει εκούσια τον έλεγχο της CPU καλώντας τη *thread_yield*; Σε τελική ανάλυση, από τη στιγμή που δεν υπάρχουν περιοδικές διακοπές ρολογιού, υπάρχει πιθανότητα να μην ξαναπάρει ποτέ τον έλεγχο της CPU.

Απάντηση 2.4

Τα νήματα μίας διεργασίας θα πρέπει να συνεργάζονται, εφόσον λειτουργούν στο πλαίσιο του ίδιου προγράμματος το οποίο συνήθως έχει γραφτεί από ένα προγραμματιστή (ή μία ομάδα προγραμματιστών) για ένα συγκεκριμένο λόγο.

Συνεπώς, εάν ο προγραμματιστής κρίνει ότι ένα νήμα θα πρέπει να έχει προτεραιότητα, τα άλλα νήματα θα είναι προγραμματισμένα να καλούν την yield.

Άσκηση 2.5 (Πρ. 2-12)

Είναι δυνατόν να προεκτοπιστεί ένα νήμα από διακοπή ρολογιού; Αν ναι, κάτω από ποιές προϋποθέσεις; Αν όχι, γιατί;

Απάντηση 2.5

Τα νήματα επιπέδου χρήστη δεν προεκτοπίζονται από τις διακοπές ρολογιού. Ο χρονοπρογραμματισμός τους γίνεται σε ***επίπεδο προγράμματος χρήστη***, ενώ για ολόκληρη τη διεργασία ο χρονοπρογραμματισμός γίνεται φυσικά σε επίπεδο πυρήνα. Η διεργασία λαμβάνει ένα κβάντο χρόνου στη CPU και αποφασίζει μόνη της πώς θα το μοιράσει στα νήματά της.

Η διακοπή των νημάτων από διακοπή ρολογιού *είναι δυνατή μόνο για τα νήματα επιπέδου πυρήνα.* Σε αυτή την περίπτωση, κάθε νήμα χρονοπρογραμματίζεται όπως μία διεργασία με ένα μόνο νήμα.

Άσκηση 2.6 (Πρ. 2-13)

Να συγκρίνετε την ανάγνωση ενός αρχείου με τη χρήση **μονονηματικού διακομιστή** αρχείων (single-threaded server) και με τη χρήση **πολυνηματικού διακομιστή** (multi-threaded-server).

- Εάν τα δεδομένα που απαιτούνται βρίσκονται στην κρυφή μνήμη (cache hit), χρειάζονται 15 msec για να ληφθεί μια αίτηση εργασίας, να διεκπεραιωθεί, και να γίνει η υπόλοιπη επεξεργασία.
- Στο 1/3 των περιπτώσεων χρειάζεται λειτουργία δίσκου, και απαιτούνται επιπλέον 75 msec, στη διάρκεια των οποίων το νήμα είναι σε λήθαργο.

Ποια η διεκπεραιωτική ικανότητα (αιτήσεις ανά sec) για έναν μονονηματικό και ποια για έναν πολυνηματικό διακομιστή;

Απάντηση 2.6

- Μονονηματικός εξυπηρετητής:
  - Όταν συμβαίνει ευστοχία κρυφής μνήμης, η διεκπεραίωση μίας αίτησης απαιτεί 15 msec.
  - Η αστοχία κρυφής μνήμης και η εξυπηρέτηση μέσω του δίσκου, απαιτεί 75 msec.
  - Ο μέσος χρόνος εξυπηρέτησης είναι:

    (2/3×15 msec) + (1/3×(15+75) msec) = 40 msec, άρα **25 αιτήσεις /sec**.
- Πολυνηματικός εξυπηρετητής:
  - Σε αυτή την περίπτωση, όταν ένα νήμα αναμένει Ε/Ε από το δίσκο, κάποιο άλλο νήμα μπορεί να εξυπηρετεί ένα άλλο αίτημα.
  - Συνεπώς η αναμονή για Ε/Ε από το δίσκο επικαλύπτεται.
  - Κάθε αίτηση διεκπαιρεώνεται σε 15 msec, και ο server εξυπηρετεί **66 2/3 αιτήσεις/sec.**

Άσκηση 2.7 (Πρ. 2-21)

Σε κάποιο σύστημα που χρησιμοποιεί νήματα, υπάρχει μία στοίβα ανά νήμα ή μία στοίβα ανά διεργασία, όταν χρησιμοποιούνται νήματα επιπέδου χρήστη; Τι συμβαίνει όταν χρησιμοποιούνται νήματα επιπέδου πυρήνα; Εξηγήστε την απάντησή σας.

Απάντηση 2.7

Κάθε νήμα έχει τη δυνατότητα να καλέσει διαφορετικές διαδικασίες για να τις εκτελέσει (*έχει τη δική του* ***κατάσταση εκτέλεσης***). Συνεπώς, πρέπει να έχει τη δική του στοίβα για τις τοπικές μεταβλητές, διευθύνσεις επιστροφής συναρτήσεων κτλ.

Δεν έχει σημασία εάν τα νήματα είναι επιπέδου χρήστη ή επιπέδου πυρήνα. ***Όλα τα νήματα έχουν τη δική τους στοίβα.***

Άσκηση 2.8 (Πρ. 2-24)

Ισχύει η λύση του Peterson στο πρόβλημα του αμοιβαίου αποκλεισμού που παρουσιάζεται στην Εικόνα 2-24, όταν ο χρονοπρογραμματισμός των διεργασιών είναι προεκτοπιστικός; Τι συμβαίνει όταν είναι μη προεκτοπιστικός;

Απάντηση 2.8

Με έναν προεκτοπιστικό αλγόριθμο χρονοπρογραμματισμού, σε κάθε διακοπή ρολογιού επιλέγεται μία διεργασία, η οποία θα εκτελεστεί για ένα μέγιστο χρονικό διάστημα. Εάν δεν έχει ολοκληρωθεί σε αυτό το διάστημα, αναστέλλεται για να συνεχίσει την εκτέλεσή της κάποια άλλη στιγμή. Η λύση του Peterson ισχύει (*και σχεδιάστηκε*) για αυτή την περίπτωση.

Με έναν μη-προεκτοπιστικό αλγόριθμο, η λύση του Peterson ***μπορεί και να αποτύχει***. Π.χ.: έστω ότι η *turn* είναι αρχικά 0. Έστω επίσης ότι εκτελείται πρώτη η διεργασία 1. Θα βρίσκεται σε ένα ***ατέρμονα βρόγχο*** και δεν θα απελευθερώσει τη CPU (στους μη προεκτοπιστικούς αλγορίθμους μία διεργασία που εκτελείται δεν μπορεί να διακοπεί, παρά μόνο όταν θα περιμένει για Ε/Ε).

Άσκηση 2.9 (Πρ. 2-33)

Μπορεί να μετρηθεί αν μια διεργασία είναι ***εξαρτημένη από τη CPU*** ή ***εξαρτημένη από Ε/Ε***, αν αναλυθεί ο πηγαίος κώδικάς της;

Μπορεί αυτό να προσδιοριστεί κατά το χρόνο εκτέλεσης του προγράμματος;

Απάντηση 2.9

Σε απλές περιπτώσεις μπορούμε να δούμε εάν ένα πρόγραμμα είναι εξαρτημένο από Ε/Ε. Π.χ. ***ένα πρόγραμμα το οποίο διαβάζει στην αρχή όλα τα αρχεία εισόδου*** και τοποθετεί τα δεδομένα που χρειάζεται σε buffers ***πιθανότατα δεν θα είναι εξαρτημένο από Ε/Ε.***

Ένα πρόγραμμα που διαβάζει και διαβάζει συνέχεια διαφορετικά αρχεία (π.χ. ένας μεταγλωττιστής) ***πιθανότατα θα είναι εξαρτημένο από Ε/Ε.***

Κατά τη διάρκεια του χρόνου εκτέλεσης μπορούμε (κατά προσέγγιση) να δούμε ***εάν το πρόγραμμα είναι εξαρτημένο από τη CPU*** με τη ***χρήση π.χ. της εντολής ps*** *(βλέποντας τι ποσοστό της CPU έχει χρησιμοποιήσει ένα πρόγραμμα, σε σύγκριση με το συνολικό χρόνο εκτέλεσης του προγράμματος).*

Άσκηση 2.10 (Πρ. 2-35)

Μετρήσεις σε συγκεκριμένο σύστημα έχουν δείξει ότι ο μέσος όρος του χρόνου εκτέλεσης (CPU time) για μια διεργασία είναι ***T***, πριν αυτή μπλοκαριστεί από είσοδο/έξοδο. Η εναλλαγή διεργασίας (context switch time) απαιτεί χρόνο ***S***, ο οποίος λογίζεται ως επιβάρυνση (χαμένος χρόνος). Εάν έχουμε χρονοπρογραμματισμό εκ περιτροπής (*Round-Robin*) με κβάντο χρόνου ***Q***, δώστε το μαθηματικό τύπο για την ***αποδοτικότητα (efficiency) της CPU*** σε κάθε μία από τις εξής περιπτώσεις.

(α) $Q = \infty$

(β) $Q > T$

(γ) $S < Q < T$

(δ) $Q = S$

(ε) $Q \approx 0$

Απάντηση 2.10...

$$\textbf{Αποδοτικότητα CPU} = \frac{\textbf{Χρήσιμος χρόνος CPU}}{\textbf{Συνολικός χρόνος CPU}}$$

1. Εάν Q ≥ T, τότε κάθε διεργασία θα τρέξει για **T** και μετά θα γίνει εναλλαγή διεργασίας για χρόνο **S**. Άρα η αποδοτικότητα είναι: $\textbf{Αποδοτικότητα} = \frac{T}{T+S}$

2. Εάν Q<T, κάθε διεργασία θα χρειαστεί **T/Q** εναλλαγές διεργασιών, σπαταλώντας χρόνο **S×(T/Q)**. Άρα η αποδοτικότητα είναι: $\textbf{Αποδοτικότητα} = \frac{T}{T+S\cdot(T/Q)}$

...Απάντηση 2.10

Με βάση την περίπτωση (1), ισχύει για τα (α), (β):

(α) $Q = \infty$

(β) $Q > T$

$$\textbf{Αποδ.} = \frac{T}{T+S}$$

Με βάση την περίπτωση (2) ισχύει για τα (γ), (δ), (ε):

(γ) $S < Q < T$ $$\textbf{Αποδ.} = \frac{T}{T + S\cdot(T/Q)} = \frac{T}{T(1+S/Q)} = \frac{1}{\frac{Q+S}{Q}} = \frac{Q}{Q+S}$$

(δ) $Q = S\ (<T)$ $$\textbf{Αποδ.} = \frac{T}{T+S\cdot(T/Q)} = \frac{T}{T+T} = \frac{1}{2}$$

(ε) $Q \approx 0$. Εφόσον το Q→0, τότε T/Q→∞ και **Αποδ.**→ 0.

Άσκηση 2.11 (Πρ. 2-36)

Πέντε εργασίες περιμένουν να εκτελεστούν. Οι αναμενόμενοι χρόνοι εκτέλεσης τους είναι 9, 6, 3, 5, και Χ.

Με *ποιο αλγόριθμο* και με *ποιά σειρά* πρέπει να εκτελεστούν ώστε να ελαχιστοποιηθεί ο μέσος χρόνος απόκρισης;

(Η απάντηση πρέπει να είναι συνάρτηση του Χ.)

Απάντηση 2.11

Ο ***αλγόριθμος SJF*** (shortest job first) δίνει το ***μικρότερο μέσο χρόνο απόκρισης***. Συνεπώς θα ισχύει:

- Αν $0 < X \leq 3$: **X**, 3, 5, 6, 9.
- Αν $3 < X \leq 5$: 3, **X**, 5, 6, 9.
- Αν $5 < X \leq 6$: 3, 5, **X**, 6, 9.
- Αν $6 < X \leq 9$: 3, 5, 6, **X**, 9.
- Αν $X > 9$: 3, 5, 6, 9, **X**.

Άσκηση 2.12 (Πρ. 2-37)

Εργασίες δέσμης Α, Β, Γ, Δ, Ε, καταφθάνουν σε ένα κέντρο υπολογιστών την ίδια περίπου χρονική στιγμή. Οι χρόνοι εκτέλεσής τους εκτιμώνται σε 10, 6, 2, 4, και 8 λεπτά αντίστοιχα.

Οι προτεραιότητές τους (που καθορίστηκαν εξωτερικά) είναι 3, 5, 2, 1, και 4 αντίστοιχα, (όπου 5 η υψηλότερη).

Για τον καθέναν από τους επόμενους αλγόριθμους χρονοπρογραμματισμού, υπολογίστε το μέσο χρόνο διεκπεραίωσης των διεργασιών.

Αγνοήστε την επιβάρυνση λόγω εναλλαγής των διεργασιών.

...

...Άσκηση 2.12 (Πρ. 2-37)

... (α) Εξυπηρέτηση εκ περιτροπής.

(β) Χρονοπρογραμματισμός προτεραιοτήτων.

(γ) Εξυπηρέτηση με βάση τη σειρά άφιξης (εκτέλεση με τη σειρά 10, 6, 2, 4, 8).

(δ) Εξυπηρέτηση με βάση τη μικρότερη διάρκεια.

Για την περίπτωση (α), υποθέστε ότι το σύστημα είναι πολυπρογραμματιζόμενο και ότι κάθε εργασία παίρνει **δίκαιο μερίδιο του χρόνου** της CPU.

Για τις περιπτώσεις (β) έως (δ), υποθέστε ότι ο αλγόριθμος είναι **μη προεκτοπιστικός**.

Όλες οι εργασίες είναι τελείως εξαρτημένες από τη CPU.

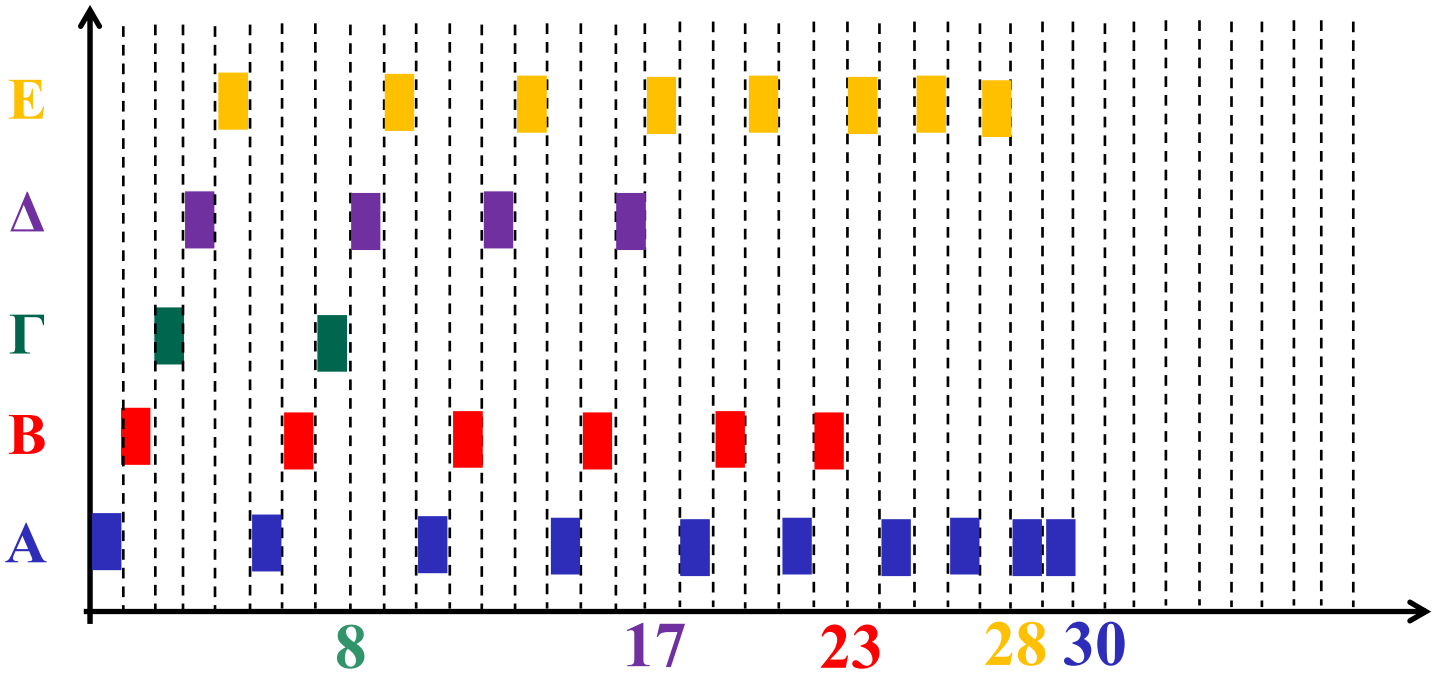
Απάντηση 2.12 (α)

Α: 10, Β: 6, Γ: 2, Δ: 4, και Ε: 8.

(α) RR με δίκαιη κατανομή χρόνου.

Ο μέσος χρόνος εκτέλεσης είναι:

(8+17+23+28+30)/5 = 106/5 = 21,2 λεπτά.



…Απάντηση 2.12 (β)

Α: 10, Β: 6, Γ: 2, Δ: 4, και Ε: 8.

Προτεραιότητες **3, 5, 2, 1, και 4.**

(β) Χρονοπρογραμματισμός προτεραιοτήτων μη-προεκτοπιστικός.

Η **Β**: τερματίζει σε 6 λεπτά.

Η **Ε**: τερματίζει σε 6+8 = 14 λεπτά.

Η **Α**: τερματίζει σε 14+10 = 24 λεπτά.

Η **Γ**: τερματίζει σε 24+2 = 26 λεπτά.

Η **Δ**: τερματίζει σε 26+4 = 30 λεπτά.

Ο μέσος χρόνος εκτέλεσης είναι:

(6+14+24+26+30)/5 = 100/5 = 20 λεπτά.

…Απάντηση 2.12 (γ)

Α: 10, Β: 6, Γ: 2, Δ: 4, και Ε: 8.

(γ) Εξυπηρέτηση με βάση τη σειρά άφιξης .

Η **Α**: τερματίζει σε 10 λεπτά.

Η **Β**: τερματίζει σε 10+6 = 16 λεπτά.

Η **Γ**: τερματίζει σε 16+2 = 18 λεπτά.

Η **Δ**: τερματίζει σε 18+4 = 22 λεπτά.

Η **Ε**: τερματίζει σε 22+8 = 30 λεπτά.

Ο μέσος χρόνος εκτέλεσης είναι:

(10+16+18+22+30)/5 = 96/5 = 19,2 λεπτά.

...Απάντηση 2.12 (δ)

Α: 10, Β: 6, Γ: 2, Δ: 4, και Ε: 8.

(γ) Εξυπηρέτηση με βάση τη μικρότερη διάρκεια (SJF).

Η **Γ**: τερματίζει σε 2 λεπτά.

Η **Δ**: τερματίζει σε 2+4 = 6 λεπτά.

Η **Β**: τερματίζει σε 6+6 = 12 λεπτά.

Η **Ε**: τερματίζει σε 12+8 = 20 λεπτά.

Η **Α**: τερματίζει σε 20+10 = 30 λεπτά.

Ο μέσος χρόνος εκτέλεσης είναι:

(2+6+12+20+30)/5 = 70/5 = 14 λεπτά.

Άσκηση 2.13 (Πρ. 2-40)

Ο *αλγόριθμος γήρανσης* με **α=1/2**, χρησιμοποιείται για την πρόβλεψη των χρόνων εκτέλεσης. Οι τέσσερις προηγούμενες εκτελέσεις, από την παλαιότερη προς την πιο πρόσφατη, έδωσαν χρόνους 40, 20, 40, και 15 msec αντίστοιχα. Ποιά είναι η πρόβλεψη χρόνου για την επόμενη εκτέλεση;

Απάντηση 2.13

Ο ***αλγόριθμος της γήρανσης (aging algorithm)*** για τον υπολογισμό της πρόβλεψης του επόμενου χρόνου εκτέλεσης σε συνάρτηση με τους προηγούμενους γνωστούς χρόνους, δίνεται από τον τύπο: $\mathbf{T_{εκτιμώμενο} = α\, T_0 + (1-α)\, T_1}$

Προηγούμενοι χρόνοι: T_0 =40, T_1 =20, T_2 = 40, T_3 = 15 msec (T_0 η παλαιότερη)

$\mathbf{T_{εκτ.}(1)} = T_0/2 + T_1/2 = (40+20)/2 = 30$ msec

$\mathbf{T_{εκτ.}(2)} = T_0/4 + T_1/4 + T_2/2 = (40+20+80)/4 = 35$ msec

$\mathbf{T_{εκτ.}(3)} = T_0/8 + T_1/8 + T_2/4 + T_3/2 = (40+20+80+60)/8 = 25$

Άσκηση 2.14 (Πρ. 2-41)

Σε ένα ήπιο σύστημα πραγματικού χρόνου υπάρχουν τέσσερα συμβάντα, με περιόδους 50, 100, 200, και 250 msec αντίστοιχα. Υποθέστε ότι τα τέσσερα συμβάντα χρειάζονται 35, 20, 10, και *x* msec χρόνου της CPU αντίστοιχα.

Ποιά είναι η μεγαλύτερη τιμή του *x*, για την οποία το σύστημα είναι χρονοπρογραμματίσιμο;

Απάντηση 2.14

Το ποσοστό της CPU που χρησιμοποιείται δίνεται από τον τύπο $\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$

Συνεπώς $\sum_{i=1}^{4} \frac{C_i}{P_i} = \frac{35}{50} + \frac{20}{100} + \frac{10}{200} + \frac{x}{250} \leq 1$

...

Προκύπτει ότι το *x πρέπει να είναι μικρότερο ή ίσο με 12.5 msec.*

Άσκηση 2.15 (Πρ. 2-42)

Εξηγήστε γιατί ο χρονοπρογραμματισμός δύο επιπέδων χρησιμοποιείται ευρύτατα.

Απάντηση 2.15

Ο χρονοπρογραμματισμός δύο επιπέδων χρειάζεται *όταν η μνήμη δεν είναι επαρκής ώστε να χωράει όλες τις έτοιμες διεργασίες*.

Ένα υποσύνολο από αυτές τοποθετείται στη μνήμη και επιλέγεται μία από αυτές για εκτέλεση. Σε τακτά διαστήματα το σύνολο των διεργασιών που είναι στη μνήμη *αναπροσαρμόζεται* και αλλάζει, ώστε όλες οι διεργασίες τελικά να επιλεγούν για εκτέλεση.

Ο αλγόριθμος αυτός είναι εύκολος στην υλοποίηση και αρκετά αποδοτικός, πολύ καλύτερος από τον round robin ο οποίος δεν λαμβάνει υπόψη εάν μία διεργασία είναι στη μνήμη ή όχι.

Άσκηση 2.16 (Πρ. 2-45)

Στη λύση του προβλήματος του δείπνου των φιλοσόφων (Εικόνα 2-46), εξηγήστε γιατί στη μεταβλητή κατάστασης ανατίθεται η τιμή *HUNGRY* στη διαδικασία *take_forks*.

Απάντηση 2.16

Εάν ένας φιλόσοφος μπλοκάρει, οι γείτονές του ελέγχουν την κατάστασή του με την *test.* Εάν δουν αργότερα ότι βρίσκεται σε κατάσταση hungry (πεινάει), *είναι δυνατό να τον ξυπνήσουν* όταν τα δύο πιρούνια θα είναι διαθέσιμα.

Άσκηση 2.17 (Πρ. 2-46)

Θεωρήστε τη διαδικασία *put_forks* στην Εικόνα 2-46. Υποθέστε ότι η μεταβλητή *state*[*i*] παίρνει την τιμή *THINKING μετά* από τις δύο κλήσεις στην *test* και όχι *πριν* από αυτές. Πώς θα επηρέαζε αυτή η αλλαγή τη λύση του προβλήματος;

Απάντηση 2.17

Η αλλαγή αυτή θα σήμαινε ότι μόλις ο φιλόσοφος σταματούσε να τρώει, κανείς από τους γείτονές του δεν θα μπορούσε να επιλεγεί ως ο επόμενος. Στην πραγματικότητα, ***δεν θα επιλεγόταν ποτέ***.

Υποθέστε ότι ο 2ος φιλόσοφος σταμάτησε να τρώει. Θα έτρεχε την *test* για τους φιλοσόφους 1 και 3, αλλά κανείς δεν θα μπορούσε να ξεκινήσει, ακόμα και αν και οι δύο πεινούσαν και τα δύο πιρούνια κάποιου από τους δύο ήταν διαθέσιμα.

Ομοίως, εάν ο 4ος φιλόσοφος σταματούσε να τρώει, ο φιλόσοφος 3 δεν θα μπορούσε να ξεκινήσει.