Processes & Threads

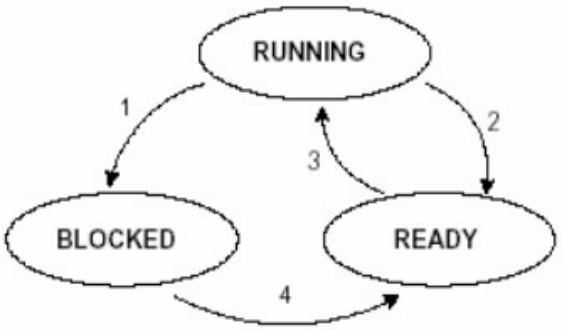
a. Ένας τρόπος συγχρονισμού διεργασιών είναι μέσω απενεργοποίησης των διακοπών. Περιγράψτε πώς ακριβώς χρησιμοποιείται. Σε συστήματα με πολλούς επεξεργαστές πώς θα απέδιδε αυτή η μέθοδος;

Κάθε διεργασία που μπαίνει στη κρίσιμη περιοχή της, απενεργοποιεί όλες τις διακοπές και τις επαναφέρει αμέσως πριν βγει από αυτή. Με την απενεργοποίηση των διακοπών δε προκύπτουν διακοπές ρολογιού, συνεπώς η cpu δε πρόκειται να ασχοληθεί με άλλη διεργασία και η κοινή περιοχή μνήμης δε κινδυνεύει. Σε συστήματα με πολλούς επεξεργαστές, η απενεργοποίηση θα επηρέαζε μόνο τη cpu που έκανε disable, ενώ οι υπόλοιπες θα μπορούσαν να συνεχίσουν με το κίνδυνο να προσπελάσουν τη κοινή μνήμη

b. Αναφέρατε τις 3 καταστάσεις των διεργασιών και περιγράψτε πότε γίνεται μετάβαση από μία κατάσταση σε άλλη

Οι καταστάσεις είναι οι εξής:

- τρέχει: έχει το cpu
- έτοιμη: μπορεί να τρέξει αλλά το cpu είναι απασχολημένο
- μπλοκαρισμένη: περιμένει εξωτερικό γεγονός



Για τις μεταβάσεις έχουμε:
Μετάβαση 1: όταν περιμένει I/O
Μετάβαση 2: όταν cpu δίνεται σε άλλη διεργασία
Μετάβαση 3: όταν cpu δίνεται σε αυτή τη διεργασία
Μετάβαση 4: όταν πραγματοποιηθεί το I/O

Για τη μετάβαση 1 υπάρχει ένα block() system call. Οι μεταβάσεις 2 και 3 προκύπτουν από τον scheduler του λειτουργικού

c. Να γράψετε με κώδικα και να εξηγήσετε τη λύση του Peterson

```
#define FALSE  0
#define TRUE   1
#define N      2                    /* number of processes */

int turn;                           /* whose turn is it? */
int interested[N];                  /* all values initially 0 (FALSE) */

void enter_region(int process);     /* process is 0 or 1 */
{
     int other;                     /* number of the other process */

     other = 1 – process;           /* the opposite of process */
     interested[process] = TRUE;    /* show that you are interested */
     turn = process;                /* set flag */
     while (turn == process && interested[other] == TRUE) /* null statement */ ;
}

void leave_region(int process)      /* process: who is leaving */
{
     interested[process] = FALSE;   /* indicate departure from critical region */
}
```

Αν η P1 εκτελέσει enter_region() πρώτη, μιας και το interested [other] ≠ TRUE δεν θα περιμένει (με busy waiting) και θα συνεχίσει. Οταν μετά ενδιαφερθεί η P2 το interested[P1] = TRUE και το P2 θα συνεχίζει να εκτελεί το while loop.

Αν και οι δύο processes καλέσουν enter_region() (σχεδόν) ταυτόχρονα, τότε, μιας και έχουμε ένα CPU, κάποια από τις 2 processes θ' εκτελέσει τελευταία την εντολή turn=process. Αυτή η process δεν θα μπορέσει να μπεί στο critical section και θα κάνει busy wait στο while loop. Ετσι, μόνο η άλλη process θα μπεί στο critical section της.

d. Να λύσετε το πρόβλημα του παραγωγού – καταναλωτή με πέρασμα μηνυμάτων

Αρχικά, ο καταναλωτής στέλνει Ν άδεια μηνύματα στον παραγωγό. Κάθε φορά που ο παραγωγός παράγει ένα item, παίρνει ένα άδειο μήνυμα, το γεμίζει και το στέλνει πίσω στο καταναλώτη. Ο παραγωγός μπλοκάρει μέχρι να λάβει άδεια μηνύματα από το καταναλωτή. Ο καταναλωτής μπλοκάρει μέχρι να λάβει γεμάτα μηνύματα από τον παραγωγό

f. Εξηγήστε τη διαφορά πολιτικής και μηχανισμού (policy-mechanism separation) στις πολιτικές δρομολόγησης

Η απάντηση βρίσκεται στη σελ. 197

Αδιέξοδα

a. Αναφέρατε και περιγράψτε τις 4 συνθήκες του αδιεξόδου

Οι συνθήκες είναι:

- αμοιβαίου αποκλεισμού – αν οι πόροι δεν εκχωρούνται αποκλειστικά σε μία μόνο διεργασία, τότε δε θα συμβεί ποτέ αδιέξοδο
- δέσμευσης και αναμονής – αν δεν επιτρέπουμε στις διεργασίες που δεσμεύουν πόρους να περιμένουν και άλλους, τότε εξαλείφουμε τα αδιέξοδα. Αυτό μπορεί να γίνει αν η κάθε διεργασία ζητά όλους τους πόρους που θέλει πριν αρχίσει η εκτέλεσή της
- μη προεκτόπισης – αφαίρεση των πόρων για μια διεργασία
- κυκλική αναμονή – κάθε διεργασία μπορεί να δεσμεύει ένα μόνο πόρο ανα πάσα στιγμή, ενώ για να δεσμεύσει έναν δεύτερο πρέπει πρώτα να αποδεσμεύσει τον πρώτο. Ένας άλλος τρόπος είναι οι διεργασίες να μπορούν να ζητούν όσους πόρους θέλουν, αλλά όλες οι αιτήσεις πρέπει να γίνονται με βάση τη σειρά αρίθμησης

b. Περιγράψτε τον αλγόριθμο του τραπεζίτη για (i) ένα πόρο και (ii) πολλούς πόρους

i) Ο αλγόριθμος εξετάζει κάθε αίτηση αμέσως όταν εμφανίζεται και ελέγχει αν η αποδοχή της οδηγεί σε ασφαλή κατάσταση. Αν οδηγεί, τότε η αίτηση γίνεται αποδεκτή, ενώ αν οδηγεί σε ανασφαλή κατάσταση τότε η αποδοχή αναβάλλεται για αργότερα. Ο τραπεζίτης, προκειμένου να εξετάσει αν μια κατάσταση είναι ασφαλής, ελέγχει αν υπάρχουν αρκετοί πόροι για να ικανοποιηθεί κάποιος πελάτης. Αν συμβαίνει αυτό, θεωρεί ότι ο πελάτης θα αποπληρώσει το δάνειο, ελέγχει το πελάτη που είναι πιο κοντά στο πιστωτικό όριο, κλπ. Αν όλα τα δάνεια μπορούν να αποπληρωθούν, η κατάσταση είναι ασφαλής και η αρχική αίτηση γίνεται δεκτή.

ii) Ο αλγόριθμος που ελέγχει αν μία κατάσταση είναι ασφαλής είναι ο εξής:
1. Ψάχνουμε για μία γραμμή Μ, της οποίας οι επιπλέον απαιτήσεις σε πόρους είναι παντού μικρότερες ή ίσεις απο τους διαθέσιμους πόρους Θ. Αν δεν υπάρχει τέτοια

γραμμή, τότε το σύστημα θα οδηγηθεί τελικά σε αδιέξοδο αφού δεν υπάρχει διεργασία που να μπορεί να εκτελεστεί μέχρι τέλους.
2. Υπόθετουμε ότι η διεργασία που αντιστοιχεί στη γραμμή που επιλέχτηκε ζητάει όλους τους πόρους που χρειάζεται και τερματίζει. Σημειώνουμε αυτή τη διεργασία ως ολοκληρωμένη και προσθέτουμε όλους τους πόρους της στο διάνυσμα Θ
3. Επαναλαμβάνουμε τα βήματα 1 και 2 μέχρι να σημειωθούν όλες οι διεργασίες ως ολοκληρωμένες. Σε αυτή τη περίπτωση, η αρχική κατάσταση θεωρείται ασφαλής, ενώ αν εμφανιστεί αδιέξοδο θεωρείται ανασφαλής.

Διαχείριση μνήμης

a. σελίδα 249 του βιβλίου
c. 267-268, 259

I/O – Δίσκοι

a. 369
b. 388

Αρχεία

a. 505
b. 513
c. 471