ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Αδιέξοδα
Εργαστηριακές Ασκήσεις

Υλικό από:
Κ. Διαμαντάρας, Λειτουργικά Συστήματα, Τμήμα Πληροφορικής ΤΕΙΘ

Σύνθεση
Κ.Γ. Μαργαρίτης, Τμήμα Εφαρμοσμένης Πληροφορικής, Πανεπιστήμιο Μακεδονίας

Άσκηση 1

Να σχεδιάσετε ένα γράφο εκχώρησης πόρων για τα παρακάτω:

- Η διεργασία P1 απαιτεί τον πόρο R1
- Η διεργασία P2 απαιτεί τον πόρο R3
- Ο πόρος R1 εκχωρείται στη διεργασία P2
- Ο πόρος R2 εκχωρείται στη διεργασία P1
- Ο πόρος R3 εκχωρείται στη διεργασία P3

Υπάρχει αδιέξοδο;

Άσκηση 2-4

Ένα σύστημα διαθέτει 6 όμοια tape drives και ***n*** σε πλήθος διεργασίες που ανταγωνίζονται για τη χρήση τους. Κάθε διεργασία μπορεί να απαιτήσει 2 tape drives. Για ποια τιμή του n το σύστημα είναι απαλλαγμένο από αδιέξοδα;

Υποθέστε ένα σύστημα με P διεργασίες και R όμοιους επαναχρησιμοποιήσιμους πόρους. Αν κάθε διεργασία μπορεί να απαιτήσει κατά μέγιστο 2 μονάδες πόρων, να αποδείξετε ότι δεν μπορεί να υπάρξει αδιέξοδο μόνον όταν ισχύει η συνθήκη P<=R-1.

Σε ένα σύστημα υπάρχουν N σε πλήθος ενεργές διεργασίες που διαμοιράζονται M μονάδες ενός επαναχρησιμοποιήσιμου πόρου R. Κάθε διεργασία μπορεί να απαιτήσει κατά μέγιστο 3 μονάδες του πόρου R. Να βρείτε ποια σχέση πρέπει να έχουν οι παράμετροι N και M ώστε να μην υπάρχει κίνδυνος αδιεξόδου.

Άσκηση 5

Ένα σύστημα αποτελείται από 4 διεργασίες, (p1, p2, p3, p4), και 3 τύπους από πόρους, (R1, R2, R3). Ο αριθμός των μονάδων των πόρων είναι E=[3,2,2].

- Η διεργασία p1 κρατά 1 μονάδα του R1 και απαιτεί 1 μονάδα του R2
- Η διεργασία p2 κρατά 2 μονάδες του R2 και απαιτεί 1 μονάδα καθενός από τους R1 και R3.
- Η διεργασία p3 κρατά 1 μονάδα του R1 και απαιτεί 1 μονάδα του R2
- Η διεργασία p4 κρατά 2 μονάδες του R3 και απαιτεί 1 μονάδα του R1

Να σχεδιάσετε το γράφημα εκχώρησης πόρων που αναπαριστά την κατάσταση του συστήματος. Υπάρχουν διεργασίες που βρίσκονται σε αδιέξοδο στην κατάσταση αυτή; Εκτελέστε τον αλγόριθμο ανίχνευσης αδιεξόδου με αυτή την αρχική κατάσταση.

Άσκηση 6 (1)

Υπάρχουν 3 τύποι πόρων με πλήθος :

$$R(1) = 9,\ R(2) = 3,\ R(3) = 6$$

Είναι η παρακάτω κατάσταση ασφαλής;

Αλγόριθμος τραπεζίτη.

Claim

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Allocated

R1	R2	R3
1	0	0
6	1	2
2	1	1
0	0	2

Total

R1	R2	R3
9	3	6

Available

R1	R2	R3
0	1	1

Άσκηση 6 (2)

	Claim			Allocated		
	R1	R2	R3	R1	R2	R3
P1	3	2	2	1	0	0
P2	0	0	0	0	0	0
P3	3	1	4	2	1	1
P4	4	2	2	0	0	2

Total		
R1	R2	R3
9	3	6

Available		
6	2	3

	Claim			Allocated		
	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0
P2	0	0	0	0	0	0
P3	3	1	4	2	1	1
P4	4	2	2	0	0	2

Total		
R1	R2	R3
9	3	6

Available		
7	2	3

Άσκηση 6 (3)

	Claim			Allocated			Total		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	9	3	6
P2	0	0	0	0	0	0			
P3	0	0	0	0	0	0	Available		
P4	4	2	2	0	0	2	9	3	4

	Claim			Allocated			Total		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	0	0	0	0	0	9	3	6
P2	0	0	0	0	0	0			
P3	0	0	0	0	0	0	Available		
P4	0	0	0	0	0	0	9	3	6

state is safe: P2->P1->P3->P4

Άσκηση 7(1)

$C = \langle 8, 5, 9, 7 \rangle$

Maximum Claim

Process	R_0	R_1	R_2	R_3
p_0	3	2	1	4
p_1	0	2	5	2
p_2	5	1	0	5
p_3	1	5	3	0
p_4	3	0	3	3

Allocated Resources

Process	R_0	R_1	R_2	R_3
p_0	2	0	1	1
p_1	0	1	2	1
p_2	4	0	0	3
p_3	0	2	1	0
p_4	1	0	3	0

Άσκηση 7(2)

Maximum Claim

Process	R_0	R_1	R_2	R_3
p_0	3	2	1	4
p_1	0	2	5	2
p_2	5	1	0	5
p_3	1	5	3	0
p_4	3	0	3	3

Allocated Resources

Process	R_0	R_1	R_2	R_3
p_0	2	0	1	1
p_1	0	1	2	1
p_2	4	0	0	3
p_3	0	2	1	0
p_4	1	0	3	0
Sum	7	3	7	5

C = <8, 5, 9, 7>

- Compute Total Held = <7,3,7,5>

Άσκηση 7(3)

Maximum Claim

Process	R_0	R_1	R_2	R_3
p_0	3	2	1	4
p_1	0	2	5	2
p_2	5	1	0	5
p_3	1	5	3	0
p_4	3	0	3	3

Allocated Resources

Process	R_0	R_1	R_2	R_3
p_0	2	0	1	1
p_1	0	1	2	1
p_2	4	0	0	3
p_3	0	2	1	0
p_4	1	0	3	0
Sum	7	3	7	5

$C = \langle 8, 5, 9, 7 \rangle$

- Compute Total Held = <7,3,7,5>
- Find Available Units (C-Held)

avail = <8-7, 5-3, 9-7, 7-5>
= <1, 2, 2, 2>

Άσκηση 7(4)

Maximum Claim

Process	R_0	R_1	R_2	R_3
p_0	3	2	1	4
p_1	0	2	5	2
p_2	5	1	0	5
p_3	1	5	3	0
p_4	3	0	3	3

Allocated Resources

Process	R_0	R_1	R_2	R_3
p_0	2	0	1	1
p_1	0	1	2	1
p_2	4	0	0	3
p_3	0	2	1	0
p_4	1	0	3	0
Sum	7	3	7	5

$C = \langle 8, 5, 9, 7 \rangle$

- Compute Total Held = <7,3,7,5>
- Find Available Units (C-Held)

avail = <8-7, 5-3, 9-7, 7-5>
= <1, 2, 2, 2>

- Can p0's maxc be met?

maxc[0,0]-alloc'[0,0] = 3-2 = 1≤1 = avail[0]

maxc[0,1]-alloc'[0,1] = 2-0 = 2≤2 = avail[1]

maxc[0,2]-alloc'[0,2] = 1-1 = 0≤2 = avail[2]

maxc[0,3]-alloc'[0,3] = 4-1 = 3≤2 = avail[3]

- Process p0 Fails on avail[3]

Άσκηση 7(5)

Maximum Claim

Process	R_0	R_1	R_2	R_3
p_0	3	2	1	4
p_1	0	2	5	2
p_2	5	1	0	5
p_3	1	5	3	0
p_4	3	0	3	3

Allocated Resources

Process	R_0	R_1	R_2	R_3
p_0	2	0	1	1
p_1	0	1	2	1
p_2	4	0	0	3
p_3	0	2	1	0
p_4	1	0	3	0
Sum	7	3	7	5

$C = \langle 8, 5, 9, 7 \rangle$

- Compute Total Held = <7,3,7,5>
- Find Available Units (C-Held)

avail = <8-7, 5-3, 9-7, 7-5>
= <1, 2, 2, 2>

- Can p1's maxc be met?

maxc[1,0]-alloc'[1,0] = 0-0 = 0≤1 = avail[0]
maxc[1,1]-alloc'[1,1] = 2-1 = 1≤2 = avail[1]
maxc[1,2]-alloc'[1,2] = 5-2 = 3≤2 = avail[2]
maxc[1,3]-alloc'[1,3] = 2-1 = 1≤2 = avail[3]

- Process p1 Fails on avail[2]

Άσκηση 7(6)

Maximum Claim

Process	R_0	R_1	R_2	R_3
p_0	3	2	1	4
p_1	0	2	5	2
p_2	5	1	0	5
p_3	1	5	3	0
p_4	3	0	3	3

Allocated Resources

Process	R_0	R_1	R_2	R_3
p_0	2	0	1	1
p_1	0	1	2	1
p_2	4	0	0	3
p_3	0	2	1	0
p_4	1	0	3	0
Sum	7	3	7	5

$C = \langle 8, 5, 9, 7 \rangle$

- Compute Total Held = <7,3,7,5>
- Find Available Units (C-Held)

avail = <8-7, 5-3, 9-7, 7-5>
= <1, 2, 2, 2>

- Can p2's maxc be met?

maxc[2,0]-alloc'[2,0] = 5-4 = 1≤1 = avail[0]

maxc[2,1]-alloc'[2,1] = 1-0 = 1≤2 = avail[1]

maxc[2,2]-alloc'[2,2] = 0-0 = 0≤2 = avail[2]

maxc[2,3]-alloc'[2,3] = 5-3 = 2≤2 = avail[3]

Άσκηση 7(7)

Maximum Claim

Process	R_0	R_1	R_2	R_3
p_0	3	2	1	4
p_1	0	2	5	2
p_2	5	1	0	5
p_3	1	5	3	0
p_4	3	0	3	3

Allocated Resources

Process	R_0	R_1	R_2	R_3
p_0	2	0	1	1
p_1	0	1	2	1
p_2	4	0	0	3
p_3	0	2	1	0
p_4	1	0	3	0
Sum	7	3	7	5

$C = \langle 8, 5, 9, 7 \rangle$

- Compute Total Held = <7,3,7,5>
- Find Available Units (C-Held)

avail = <8-7, 5-3, 9-7, 7-5>
= <1, 2, 2, 2>

- Can p2's maxc be met?

maxc[2,0]-alloc'[2,0] = 5-4 = 1≤1 = avail[0]
maxc[2,1]-alloc'[2,1] = 1-0 = 1≤2 = avail[1]
maxc[2,2]-alloc'[2,2] = 0-0 = 0≤2 = avail[2]
maxc[2,3]-alloc'[2,3] = 5-3 = 2≤2 = avail[3]

- Yes! Redo avail/Update alloc'

avail[0] = avail[0]+alloc'[2,0] = 1+4 = 5
avail[1] = avail[1]+alloc'[2,1] = 2+0 = 2
avail[2] = avail[2]+alloc'[2,2] = 2+0 = 2
avail[3] = avail[3]+alloc'[2,3] = 2+3 = 5

Άσκηση 7(8)

Maximum Claim

Process	R_0	R_1	R_2	R_3
p_0	3	2	1	4
p_1	0	2	5	2
p_2	5	1	0	5
p_3	1	5	3	0
p_4	3	0	3	3

Allocated Resources

Process	R_0	R_1	R_2	R_3
p_0	2	0	1	1
p_1	0	1	2	1
p_2	0	0	0	0
p_3	0	2	1	0
p_4	1	0	3	0
Sum	3	3	7	2

$C = \langle 8, 5, 9, 7 \rangle$

- Recompute Total Held=<3,3,7,2>
- Find Available Units (C-Held)

avail = <8-3, 5-3, 9-7, 7-2>
= <5, 2, 2, 5>

- Can anyone's maxc be met?

maxc[4,0]-alloc'[4,0] = 5-1 = 4≤5 = avail[0]
maxc[4,1]-alloc'[4,1] = 0-0 = 0≤2 = avail[1]
maxc[4,2]-alloc'[4,2] = 3-3 = 0≤2 = avail[2]
maxc[4,3]-alloc'[4,3] = 3-0 = 3≤5 = avail[3]

Άσκηση 7(9)

Maximum Claim

Process	R_0	R_1	R_2	R_3
p_0	3	2	1	4
p_1	0	2	5	2
p_2	5	1	0	5
p_3	1	5	3	0
p_4	3	0	3	3

Allocated Resources

Process	R_0	R_1	R_2	R_3
p_0	2	0	1	1
p_1	0	1	2	1
p_2	0	0	0	0
p_3	0	2	1	0
p_4	1	0	3	0
Sum	3	3	7	2

$C = \langle 8, 5, 9, 7 \rangle$

- Recompute Total Held=<3,3,7,2>
- Find Available Units (C-Held)

avail = <8-3, 5-3, 9-7, 7-2>
= <5, 2, 2, 5>

- Can anyone's maxc be met?

maxc[4,0]-alloc'[4,0] = 5-1 = 4≤5 = avail[0]
maxc[4,1]-alloc'[4,1] = 0-0 = 0≤2 = avail[1]
maxc[4,2]-alloc'[4,2] = 3-3 = 0≤2 = avail[2]
maxc[4,3]-alloc'[4,3] = 3-0 = 3≤5 = avail[3]

- P_4 can exercise max claim

avail[0] = avail[0]+alloc'[4,0] = 5+1 = 6
avail[1] = avail[1]+alloc'[4,1] = 2+0 = 2
avail[2] = avail[2]+alloc'[4,2] = 2+3 = 5
avail[3] = avail[3]+alloc'[4,3] = 5+0 = 5

Άσκηση 7(10)

Maximum Claim

Process	R_0	R_1	R_2	R_3
p_0	3	2	1	4
p_1	0	2	5	2
p_2	5	1	0	5
p_3	1	5	3	0
p_4	3	0	3	3

Allocated Resources

Process	R_0	R_1	R_2	R_3
p_0	2	0	1	1
p_1	0	1	2	1
p_2	0	0	0	0
p_3	0	2	1	0
p_4	0	0	0	0
Sum	2	3	4	2

$C = \langle 8, 5, 9, 7 \rangle$

- Recompute Total Held=<2,3,4,2>
- Find Available Units (C-Held)

avail = <8-2, 5-3, 9-4, 7-2>
= <6, 2, 5, 5>

- Can anyone's maxc be met?
- Yes, any of them can!
- Choose p0

Άσκηση 7(11)

Maximum Claim

Process	R_0	R_1	R_2	R_3
p_0	3	2	1	4
p_1	0	2	5	2
p_2	5	1	0	5
p_3	1	5	3	0
p_4	3	0	3	3

Allocated Resources

Process	R_0	R_1	R_2	R_3
p_0	0	0	0	0
p_1	0	1	2	1
p_2	0	0	0	0
p_3	0	2	1	0
p_4	0	0	0	0
Sum	0	3	4	2

$C = \langle 8, 5, 9, 7 \rangle$

- Recompute Total Held=<0,3,4,2>
- Find Available Units (C-Held)

avail = <8-0, 5-3, 9-4, 7-2>
= <8, 2, 5, 5>

- Can anyone's maxc be met?
- Yes, any of them can!
- Choose p1

Άσκηση 7(12)

Maximum Claim

Process	R_0	R_1	R_2	R_3
p_0	3	2	1	4
p_1	0	2	5	2
p_2	5	1	0	5
p_3	1	5	3	0
p_4	3	0	3	3

Allocated Resources

Process	R_0	R_1	R_2	R_3
p_0	0	0	0	0
p_1	0	0	0	0
p_2	0	0	0	0
p_3	0	2	1	0
p_4	0	0	0	0
Sum	0	2	1	0

$C = \langle 8, 5, 9, 7 \rangle$

- Recompute Total Held=<0,2,1,0>
- Find Available Units (C-Held)

avail = <8-0, 5-2, 9-1, 7-0>
= <8, 3, 8, 7>

- Can anyone's maxc be met?
- Yes, Choose p3

Άσκηση 7(13)

Maximum Claim

Process	R_0	R_1	R_2	R_3
p_0	3	2	1	4
p_1	0	2	5	2
p_2	5	1	0	5
p_3	1	5	3	0
p_4	3	0	3	3

Allocated Resources

Process	R_0	R_1	R_2	R_3
p_0	0	0	0	0
p_1	0	0	0	0
p_2	0	0	0	0
p_3	0	0	0	0
p_4	0	0	0	0
Sum	0	0	0	0

- Alloc' Now Zeroed
- Safe State with All Processes Completed
- Halt Algorithm with Success
- Process Order of
  - p2, p4, p0, p1, p3
  - Safe Execution
- All Maximum Claims Satisfied in Some Order
- No Deadlock or Unsafe State

Τα 3 προβλήματα που ακολουθούν είναι κλασσικά στη
Μελέτη της Διαδιεργασιακής Επικοινωνίας και των Αδιεξόδων.

Με βάση το σκελετό κώδικα που σας δίνεται και τα POSIX APIs γράψτε προγράμματα και δοκιμάστε τα ώστε να αντιληφθείτε τη λειτουργία τους.

Κάνετε αλλαγές στο κώδικα για να προκαλέστε αδιέξοδα και μελετήστε τη συμπεριφορά των προγραμμάτων.

Γευματίζοντες Φιλόσοφοι

```
#define N                5                  /* number of philosophers */
#define LEFT             (i+N–1)%N          /* number of i's left neighbor */
#define RIGHT            (i+1)%N            /* number of i's right neighbor */
#define THINKING         0                  /* philosopher is thinking */
#define HUNGRY           1                  /* philosopher is trying to get forks */
#define EATING           2                  /* philosopher is eating */
typedef int semaphore;                      /* semaphores are a special kind of int */
int state[N];                               /* array to keep track of everyone's state */
semaphore mutex = 1;                        /* mutual exclusion for critical regions */
semaphore s[N];                             /* one semaphore per philosopher */

void philosopher(int i)                     /* i: philosopher number, from 0 to N–1 */
{
      while (TRUE) {                        /* repeat forever */
            think( );                       /* philosopher is thinking */
            take_forks(i);                  /* acquire two forks or block */
            eat( );                         /* yum-yum, spaghetti */
            put_forks(i);                   /* put both forks back on table */
      }
}
```

```
void take_forks(int i)                    /* i: philosopher number, from 0 to N–1 */
{
        down(&mutex);                     /* enter critical region */
        state[i] = HUNGRY;                /* record fact that philosopher i is hungry */
        test(i);                          /* try to acquire 2 forks */
        up(&mutex);                       /* exit critical region */
        down(&s[i]);                      /* block if forks were not acquired */
}

void put_forks(i)                         /* i: philosopher number, from 0 to N–1 */
{
        down(&mutex);                     /* enter critical region */
        state[i] = THINKING;              /* philosopher has finished eating */
        test(LEFT);                       /* see if left neighbor can now eat */
        test(RIGHT);                      /* see if right neighbor can now eat */
        up(&mutex);                       /* exit critical region */
}

void test(i) /* i: philosopher number, from 0 to N–1 */
{
        if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) {
                state[i] = EATING;
                up(&s[i]);
        }
}
```

Αναγνώστες και Συγγραφείς

```
typedef int semaphore;                  /* use your imagination */
semaphore mutex = 1;                    /* controls access to 'rc' */
semaphore db = 1;                       /* controls access to the database */
int rc = 0;                             /* # of processes reading or wanting to */

void reader(void)
{
    while (TRUE) {                      /* repeat forever */
        down(&mutex);                   /* get exclusive access to 'rc' */
        rc = rc + 1;                    /* one reader more now */
        if (rc == 1) down(&db);         /* if this is the first reader ... */
        up(&mutex);                     /* release exclusive access to 'rc' */
        read_data_base( );              /* access the data */
        down(&mutex);                   /* get exclusive access to 'rc' */
        rc = rc – 1;                    /* one reader fewer now */
        if (rc == 0) up(&db);           /* if this is the last reader ... */
        up(&mutex);                     /* release exclusive access to 'rc' */
        use_data_read( );               /* noncritical region */
    }
}


void writer(void)
{
    while (TRUE) {                      /* repeat forever */
        think_up_data( );               /* noncritical region */
        down(&db);                      /* get exclusive access */
        write_data_base( );             /* update the data */
        up(&db);                        /* release exclusive access */
    }
}
```

Κοιμώμενος Κουρέας

```
#define CHAIRS 5                    /* # chairs for waiting customers */

typedef int semaphore;              /* use your imagination */

semaphore customers = 0;            /* # of customers waiting for service */
semaphore barbers = 0;              /* # of barbers waiting for customers */
semaphore mutex = 1;                /* for mutual exclusion */
int waiting = 0;                    /* customers are waiting (not being cut) */

void barber(void)
{
    while (TRUE) {
        down(&customers);           /* go to sleep if # of customers is 0 */
        down(&mutex);               /* acquire access to 'waiting' */
        waiting = waiting – 1;      /* decrement count of waiting customers */
        up(&barbers);               /* one barber is now ready to cut hair */
        up(&mutex);                 /* release 'waiting' */
        cut_hair( );                /* cut hair (outside critical region) */
    }
}


void customer(void)
{
    down(&mutex);                   /* enter critical region */
    if (waiting < CHAIRS) {         /* if there are no free chairs, leave */
        waiting = waiting + 1;      /* increment count of waiting customers */
        up(&customers);             /* wake up barber if necessary */
        up(&mutex);                 /* release access to 'waiting' */
        down(&barbers);             /* go to sleep if # of free barbers is 0 */
        get_haircut( );             /* be seated and be serviced */
    } else {
        up(&mutex);                 /* shop is full; do not wait */
    }
}
```