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, Ακαδημαϊκό Έτος 2015-2016

Επαναληπτική Εξέταση – Λύσεις

Το παρόν περιγράφει πλήρη λύση των θεμάτων. Για να βοηθήσει στην καλύτερη κατανόηση των απαντήσεων, προσφέρει αναλυτική επεξήγησή τους, η οποία δεν ήταν απαραίτητη για να θεωρείται τέλεια η λύση.

Θέμα 1 (25%)

Δίνεται το πρόγραμμα C σε περιβάλλον UNIX που ακολουθεί μετά τα ζητούμενα. Θεωρήστε ότι οι κλήσεις συστήματος δεν αποτυγχάνουν και ότι κάθε νέα διεργασία κληρονομεί ακριβώς τον τρόπο χειρισμού σημάτων του πατέρα.

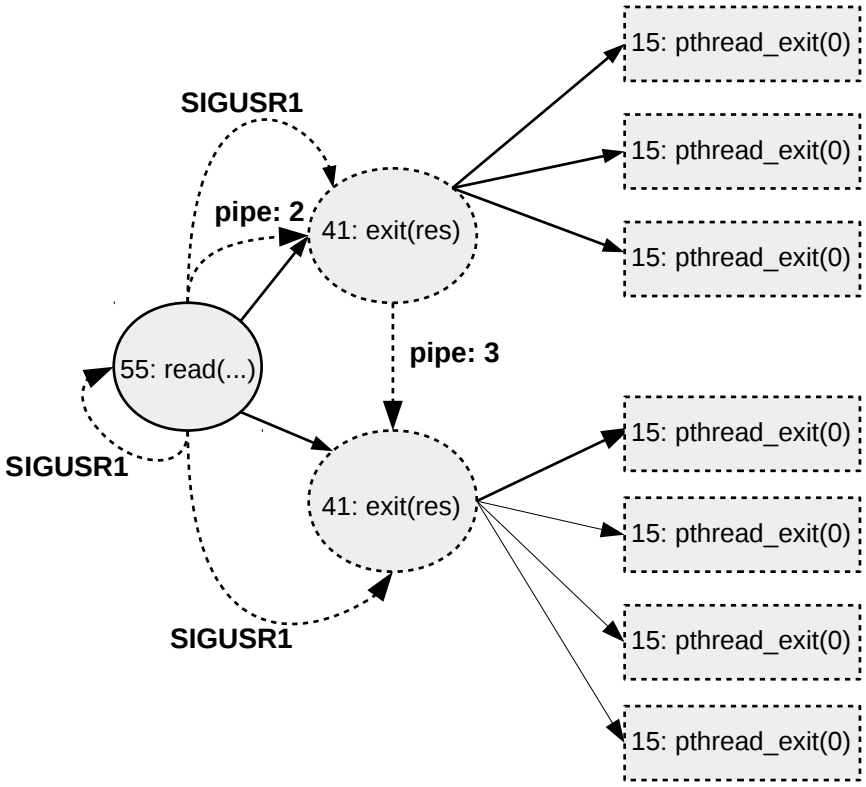
i. Σχεδιάστε το γράφο διεργασιών στην τελική του μορφή, όταν δηλαδή όλες οι διεργασίες και τα νήματα έχουν φτάσει σε μόνιμη κατάσταση. Επισημάνετε τις διεργασίες με κυκλικό κόμβο και τα νήματα με τετράγωνο κόμβο στο γράφο. Για κάθε κόμβο φροντίστε να φαίνεται η γονική διεργασία με βέλος. Αν υπάρχουν, δείξτε και τις διεργασίες/νήματα που έζησαν προσωρινά, αλλά δεν υπάρχουν στη μόνιμη κατάσταση, με διακεκομμένο περίγραμμα. (10%)
ii. Για κάθε κομβο του γράφου επισημάνετε: (i) την κλήση συστήματος ή συνάρτηση μέσα στην οποία βρίσκεται ο Program Counter της αντίστοιχης διεργασίας/νήματος, (ii) τα ορίσματα με τα οποία αυτή έχει κληθεί και (iii) τη γραμμή του προγράμματος από όπου έγινε η κλήση της. Για τις διεργασίες/νήματα που έζησαν προσωρινά επισημάνετε την τελευταία κλήση συστήματος ή συνάρτηση στην οποία βρέθηκε ο Program Counter. (5%)
iii. Συμπληρώστε το σχήμα σας ώστε να φαίνεται η διαδιεργασιακή επικοινωνία: για κάθε μεταφορά δεδομένων ή αποστολή σήματος που συνέβη, σχεδιάστε ένα διακεκομμένο βέλος από τη διεργασία-αποστολέα στη διεργασία-παραλήπτη. Πάνω από το βέλος γράψτε την τιμή που μεταφέρεται κάθε φορά. (4%)
iv. Τί θα τυπωθεί στις γραμμές 54 και 56 του προγράμματος; (6%)

Σημείωση: Από το man page της kill δίνεται: `int kill(pid_t pid, int sig);` *If pid equals 0, then sig is sent to every process in the process group of the calling process.*

```c
int res = 0, num = 2, pd[2], arg[32];
pthread_t thread[32];
sem_t sem;

void h(int s)
{
    read(pd[0], &num, sizeof(num));
    sem_post(&sem);
}

void *f(void *var)
{
    int tmp = *(int *)var;
    res += tmp;
    pthread_exit((void *)0);
}

int main()
{
    int i, k, p, pstatus;
    void *tstatus;

    sem_init(&sem, 0, 0);
    signal(SIGUSR1, h);
    pipe(pd);

    for (i = 0; i < num; i++) {
        p = fork();

        if (p == 0) {
            sem_wait(&sem);
            num++;
            for (k = 0; k < num; k++) {
                arg[k] = k;
                pthread_create(&thread[k], NULL, f, (void *)&arg[k]);
            }

            for (k = 0; k < num; k++) pthread_join(thread[k], &tstatus);
            if (num < 4) write(pd[1], &num, sizeof(num));

            exit(res);
        }
    }

    write(pd[1], &num, sizeof(num));
    signal(SIGUSR1, SIG_IGN); /* Το SIGUSR1 αγνοείται από την καλούσα διεργασία */
    kill(0, SIGUSR1);   /* Δείτε επισήμανση στην εκφώνηση */

    for (i = 0; i < num; i++) {
        wait(&pstatus);
        if (WIFEXITED(pstatus)) res += WEXITSTATUS(pstatus);
    }

    printf("Final result1: %d \n", res);
    read(pd[0], &num, sizeof(num));
    printf("Final result2: %d \n", num);

    return 0;
}
```

Λύση Θέματος 1

(i), (ii), (iii)



(iv) Στη γραμμή 54 δεν γνωρίζουμε ποια τιμή θα τυπωθεί, καθώς η τιμή του res που επιστρέφεται από κάθε διεργασία-παιδί στη γραμμή 41 υπόκειται σε race condition κατά τη διάρκεια εκτέλεσης των νημάτων (γραμμή 14). Στη γραμμή 56 η διεργασία-πατέρας δεν θα φτάσει ποτέ καθώς έχει κολλήσει στη γραμμή 55, όπου περιμένει να διαβάσει κάποια τιμή από το pipe.

Θέμα 2 (60%)

Δίνεται το ημιτελές πρόγραμμα που ακολουθεί μετά τα ερωτήματα, το οποίο εκτελείται σε σύστημα με τα παρακάτω χαρακτηριστικά:

- Το ΛΣ υποστηρίζει εικονική μνήμη με σελιδοποίηση και ακολουθεί το μοντέλο `fork()/exec()` με Copy-on-Write (CoW) για τη δημιουργία νέων διεργασιών.
- Το μέγεθος σελίδας/πλαισίου είναι 4096 bytes (4KiB). Η φυσική μνήμη περιλαμβάνει 10 πλαίσια.
- Σε κάθε πρόγραμμα που εκτελείται στο εν λόγω σύστημα, το ΛΣ παραχωρεί ξεχωριστή(ές) σελίδα(ες) μνήμης για τον κώδικα του πρόγράμματος, τις global μεταβλητές (αν υπάρχουν), τις σταθερές (αν υπάρχουν) και τη στοίβα, ενώ υποστηρίζει δυναμική παραχώρηση μνήμης από το σωρό. Η στοίβα κάθε προγράμματος είναι 4KiB.
- Το ΛΣ παραχωρεί φυσική μνήμη στο σωρό της διεργασίας όταν πραγματοποιείται εγγραφή στη μνήμη αυτή.

Ο κώδικας του παρακάτω προγράμματος είναι 4KiB, η βιβλιοθήκη της C (συναρτήσεις `fopen(), malloc(), fprintf(), scanf(), fscanf()`) καταλαμβάνει 8KiB και είναι στατικά συνδεδεμένη με το πρόγραμμα, ενώ η μαθηματική βιβλιοθήκη (συνάρτηση `sqrt()`) καταλαμβάνει 4KiB και συνδέεται με το πρόγραμμα δυναμικά. Ο κώδικας και τα δεδομένα του ΛΣ καταλαμβάνουν επίσης από 4KiB στη μνήμη το καθένα. Υπενθυμίζεται ότι `sizeof(int) = 4`.

i. Στους Πίνακες 1-3 σας δίνεται η εικόνα της φυσικής μνήμης ακριβώς πριν από την εκτέλεση του προγράμματος, καθώς και η μορφή του πίνακα σελίδων και του χάρτη μνήμης που το ΛΣ τηρεί για κάθε διεργασία. Δώστε την εικόνα της φυσικής μνήμης, καθώς και τον πίνακα σελίδων και το χάρτη μνήμης για κάθε διεργασία που βρίσκεται σε εκτέλεση στις γραμμές του κώδικα 9, 11, 18, 19 (αμέσως μετά τη fork), 23 και 28 (θεωρήστε ότι η γραμμή 22 εκτελείται πριν την 27). (30 %)

ii. Περιγράψτε αν και υπό ποιες συνθήκες στις γραμμές κώδικα 8, 10, 12, 13, 16, 19, 22 και 27 είναι δυνατόν να εμπλακεί ο χρονοδρομολογητής στην εκτέλεση του προγράμματος. Θεωρήστε ότι όλες οι συναρτήσεις και οι κλήσεις συστήματος επιτυγχάνουν. Αγνοήστε τις περιπτώσεις που ο χρονοδρομολογητής καλείται λόγω διακοπής από το χρονιστή. (10%)

iii. Συμπληρώστε το πρόγραμμα οπουδήποτε κρίνετε σκόπιμο, ώστε να πραγματοποιείται σωστά και παράλληλα (στις 2 διεργασίες που έχουν δημιουργηθεί) ο υπολογισμός. Το αποτέλεσμα της πράξης θα πρέπει να είναι ορατό στη διεργασία πατέρα. (10%)

iv. Δώστε έκδοση του παραπάνω προγράμματος χρησιμοποιώντας νήματα POSIX. (10%)

πλαίσιο	**περιγραφή**
0	κώδικας ΛΣ
1	δεδομένα ΛΣ
2	free
3	free
4	math lib
5	free
6	free
7	free
8	free
9	free

Πίνακας 1: Πλαίσια στη φυσική μνήμη

page *page number*	**permissions** *π.χ. r, rw, x, κλπ*	**frame** *αριθμός πλαισίου στη φυσική μνήμη*
0 ...	... ...	... ...

Πίνακας 2: Πίνακας Σελίδων

page *page number*	**permisssions** *π.χ. r, rw, x, κλπ*	**block** *αριθμός block στο δίσκο - ενδεικτικά*
0 ...	... ...	... ...

Πίνακας 3: Χάρτης Μνήμης

```
1  #define SIZE 1024 // constant
2
3  int *a, i, t, ITER;
4
5  int main(void)
6  {
7      FILE *inFile;
8      inFile = fopen("1.txt", "r");
9
10     a = malloc(SIZE*sizeof(int));
11
12     fprintf(stdout, "Give me the number of iterations\n");
13     scanf("%d", &ITER);
14
15     for (i = 0; i < SIZE; i++) {
16         fscanf(inFile, "%d\n", &a[i]);
17     }
18
19     if (fork() == 0) {
20         for (t = 0; t < ITER; t++)
21             for (i = 1; i < SIZE/2; i++)
22                 a[i] = sqrt((a[i-1] + a[i+1]) / 2);
23     }
24     else {
25         for (t = 0; t < ITER; t++)
26             for (i = SIZE/2; i < SIZE - 1; i++)
27                 a[i] = sqrt((a[i-1] + a[i+1]) / 2);
28     }
29
30     return 0;
31 }
```

Λύση Θέματος 2

i. Οι πίνακες που ακολουθούν είναι ενδεικτικοί (υπάρχουν πολλοί εναλλακτικοί έγκυροι πίνακες). Στις γραμμή 9 του κώδικα η φυσική μνήμη έχει ως εξής:

πλαίσιο	**περιγραφή**
0	κώδικας ΛΣ
1	δεδομένα ΛΣ
2	κώδικας P1
3	σταθερές P1
4	math lib
5	μεταβλητές P1
6	βιβλιοθήκη C
7	βιβλιοθήκη C
8	στοίβα P1
9	free

Πίνακας 4: Πλαίσια στη φυσική μνήμη

Αντίστοιχα ο πίνακας σελίδων για τη διεργασία (έστω P1):

page	permissions	frame
0	rx	2
1	r	3
2	rw	5
3	rx	6
4	rx	7
5	rw	8

Πίνακας 5: Πίνακας Σελίδων P1

και ο χάρτης μνήμης:

page	permisssions	block
0	rx	-
1	r	-
2	rw	-
3	rx	-
4	rx	-
5	rw	-

Πίνακας 6: Χάρτης Μνήμης P1

Στη γραμμή 11 η εικόνα της φυσικής μνήμης καθώς και ο πίνακας σελίδων παραμένουν ίδιοι, λόγω on demand paging, αλλά ο χάρτης μνήμης της P1 διαμορφώνεται ως εξής:

page	permisssions	block
0	rx	-
1	r	-
2	rw	-
3	rx	-
4	rx	-
5	rw	-
6	rw	-

Πίνακας 7: Χάρτης Μνήμης P1

Στη γραμμή 18 δεσμεύεται μια σελίδα, καθώς πραγματοποιείται εγγραφή στο δεσμευμένο buffer στο σωρό. Ο χάρτης μνήμης δεν αλλάζει, αλλά η φυσική μνήμη και ο πίνακας σελίδων της P1 γίνονται αντίστοιχα:

πλαίσιο	περιγραφή
0	κώδικας ΛΣ
1	δεδομένα ΛΣ
2	κώδικας P1
3	σταθερές P1
4	math lib
5	μεταβλητές P1
6	βιβλιοθήκη C
7	βιβλιοθήκη C
8	στοίβα P1
9	σωρός P1

Πίνακας 8: Πλαίσια στη φυσική μνήμη

page	permissions	frame
0	rx	2
1	r	3
2	rw	5
3	rx	6
4	rx	7
5	rw	8
6	rw	9

Πίνακας 9: Πίνακας Σελίδων P1

Στη γραμμή 19 (αμέσως μετά τη `fork()`) δημιουργείται ένα αντίγραφο του πίνακα σελίδων για την P2 (έχοντας χάσει τα write permissions) και αντίστοιχα ένα αντίγραφο του χάρτη μνήμης για την P2. Η εικόνα της φυσικής μνήμης παραμένει ίδια. Οπότε έχουμε:

page	permissions	frame
0	rx	2
1	r	3
2	r	5
3	rx	6
4	rx	7
5	r	8
6	r	9

Πίνακας 10: Πίνακας Σελίδων P1

page	permissions	frame
0	rx	2
1	r	3
2	r	5
3	rx	6
4	rx	7
5	r	8
6	r	9

Πίνακας 11: Πίνακας Σελίδων P2

page	permisssions	block
0	rx	-
1	r	-
2	rw	-
3	rx	-
4	rx	-
5	rw	-
6	rw	-

Πίνακας 12: Χάρτης Μνήμης P1

page	permisssions	block
0	rx	-
1	r	-
2	rw	-
3	rx	-
4	rx	-
5	rw	-
6	rw	-

Πίνακας 13: Χάρτης Μνήμης P2

Στη γραμμή 22 έχουμε πρόσβαση στη math lib, γεγονός που προσθέτει μία εγγραφή στον αντίστοιχο πίνακα σελίδων. Επίσης, αλλάζει η στοίβα της P2. Σε αυτό το σημείο, όμως, δεν υπάρχει ελεύθερο πλαίσιο στη φυσική μνήμη. Συνεπώς, πρέπει να αντικαταστήσουμε ένα πλαίσιο και να το αντιγράψουμε στο δίσκο (έστω block 515). Επιλέγουμε αυτό να είναι το πλαίσιο 5 (με κάποιον υποθετικό αλγόριθμο αντικατάστασης). Τέλος, η P2 πραγματοποιεί εγγραφή στη μεταβλητή a και στο δεσμευμένο χώρο στο σωρό, οπότε έχουμε επίσης COW. Ομοίως, αντιγράφουμε το πλαίσια 8 (block 516) και το πλαίσιο 9 (block 517). Οι πίνακες στη γραμμή 23 έχουν ως εξής (σχετικά με τον πίνακα σελίδων P1 οι εγγραφές που δεν υπάρχουν θεωρούνται ισοδύναμα ως invalid):

πλαίσιο	περιγραφή
0	κώδικας ΛΣ
1	δεδομένα ΛΣ
2	κώδικας P1
3	σταθερές P1
4	math lib
5	μεταβλητές P2
6	βιβλιοθήκη C
7	βιβλιοθήκη C
8	στοίβα P2
9	σωρός P2

Πίνακας 14: Πλαίσια στη φυσική μνήμη

page	permissions	frame
0	rx	2
1	r	3
2	rw	5
3	rx	6
4	rx	7
5	rw	8
6	rw	9
7	rx	4

Πίνακας 15: Πίνακας Σελίδων P2

page	permisssions	block
0	rx	-
1	r	-
2	rw	-
3	rx	-
4	rx	-
5	rw	-
6	rw	-
7	rx	-

Πίνακας 16: Χάρτης Μνήμης P2

page	permissions	frame
0	rx	2
1	r	3
3	rx	6
4	rx	7

Πίνακας 17: Πίνακας Σελίδων P1

page	permisssions	block
0	rx	-
1	r	-
2	rw	515
3	rx	-
4	rx	-
5	rw	516
6	rw	517

Πίνακας 18: Χάρτης Μνήμης P1

Τέλος, στη γραμμή 26 (θεωρούμε ότι εκτελείται μετά τη γραμμή 22) κάποια από τα πλαίσια που ανήκουν στην P2 θα αντικατασταθούν από αντίστοιχα στης P1 λόγω έλλειψης χώρου. Συγκεκριμένα, τα πλαίσια που αφορούν στη στοίβα, το σωρό και τις μεταβλητές της P2 θα αντιγραφούν στο δίσκο (έστω block 532, 533, 534) και αντιστοίχως θα αντιγραφούν από το δίσκο (block 515, 516, 517) τα σχετικά πλαίσια της P1. Επίσης, θα προστεθεί η σχετική εγγραφή για τη math lib στον πίνακα σελίδων της P1. Έτσι οι πίνακες ενημερώνονται ακολούθως (σχετικά με τον πίνακα σελίδων P2 οι εγγραφές που δεν υπάρχουν θεωρούνται ισοδύναμα ως invalid):

πλαίσιο	περιγραφή
0	κώδικας ΛΣ
1	δεδομένα ΛΣ
2	κώδικας P1
3	σταθερές P1
4	math lib
5	μεταβλητές P1
6	βιβλιοθήκη C
7	βιβλιοθήκη C
8	στοίβα P1
9	σωρός P1

Πίνακας 19: Πλαίσια στη φυσική μνήμη

page	permissions	frame
0	rx	2
1	r	3
2	rw	5
3	rx	6
4	rx	7
5	rw	8
6	rw	9
7	rx	4

Πίνακας 20: Πίνακας Σελίδων P1

page	permisssions	block
0	rx	-
1	r	-
2	rw	-
3	rx	-
4	rx	-
5	rw	-
6	rw	-
7	rx	-

Πίνακας 21: Χάρτης Μνήμης P1

page	permissions	frame
0	rx	2
1	r	3
3	rx	6
4	rx	7
7	rx	4

Πίνακας 22: Πίνακας Σελίδων P2

page	permisssions	block
0	rx	-
1	r	-
2	rw	532
3	rx	-
4	rx	-
5	rw	533
6	rw	534
7	rx	-

Πίνακας 23: Χάρτης Μνήμης P2

ii. Θεωρούμε διακοπτή στρατηγική χρονοδρομολόγησης. Ο χρονοδρομολογητής σε αυτή την περίπτωση εμπλέκεται (εκτός από τις περιπτώσεις του χρονιστή) όταν μια διεργασία κάνει Ε/Ε ή όταν μια διεργασία μεταβεί σε κατάσταση εκτέλεσης (ready). Επομένως για τις συγκεκριμένες γραμμές κώδικα:

- Γραμμή 8: Η διεργασία κάνει αίτημα στο ΛΣ για να ανοίξει το αρχείο. Δεν πραγματοποιεί κάποια Ε/Ε, οπότε δεν εμπλέκεται ο χρονοδρομολογητής.
- Γραμμή 10: Η διεργασία κάνει αίτημα στο ΛΣ για παραχώρηση δυναμικής μνήμης. Δεν πραγματοποιεί κάποια Ε/Ε, οπότε δεν εμπλέκεται ο χρονοδρομολογητής.
- Γραμμή 12: Η διεργασία πραγματοποιεί Ε/Ε οπότε εμπλέκεται ο χρονοδρομολογητής. Σημείωση: Ανάλογα με την υλοποίηση η εγγραφή στο τερματικό μπορεί να γίνει ασύγχρονα χωρίς ανάγκη αναμονής της διεργασίας. Σε αυτή την περίπτωση ο χρονοδρομολογητής δεν εμπλέκεται.
- Γραμμή 13: Η διεργασία πραγματοποιεί Ε/Ε και αναμονή εισόδου από το χρήστη, οπότε εμπλέκεται ο χρονοδρομολογητής.
- Γραμμή 16: Η διεργασία πραγματοποιεί Ε/Ε (ανάγνωση αρχείου από το δίσκο), άρα εμπλέκεται ο χρονοδρομολογητής.
- Γραμμή 19: Δημιουργείται νέα διεργασία και μεταβαίνει σε κατάσταση εκτέλεσης. Εμπλέκεται ο χρονοδρομολογητής.
- Γραμμή 22: Αν τα πλαίσια που απαιτούνται για τις συγκεκριμένες πράξεις βρίσκονται στη φυσική μνήμη, δεν δημιουργείται αίτημα Ε/Ε, οπότε δεν εμπλέκεται ο χρονοδρομολογητής. Στη συγκεκριμένη περίπτωση (βλέπε λύση ερωτήματος i) η φυσική μνήμη δεν επαρκεί και γίνεται μεταφορά πλαισίων στο δίσκο. Εφόσον πραγματοποιείται Ε/Ε, εμπλέκεται ο χρονοδρομολογητής.
- Γραμμή 27: Ομοίως με γραμμή 22.

iii. Η λύση δίνεται στη συνέχεια.

```
1
2   #define SIZE 1024 // constant
3
4   int main(void)
5   {
6       int *a;
7       int i, t, ITER;
8
9       int pd[2][2];
10
11      int *status;
12
13      FILE *inFile;
14      inFile = fopen("1.txt", "r");
15
16      a = malloc(SIZE*sizeof(int));
17
18      fprintf(stdout, "Give me the number of iterations\n");
19      scanf("%d", &ITER);
20
21      for (i = 0; i < SIZE; i++) {
22          fscanf(inFile, "%d\n", &a[i]);
23      }
24
25      pipe(pd[0]);
26      pipe(pd[1]);
27
28      if (fork() == 0) {
29
30          for (t = 0; t < ITER; t++) {
31
32              for (i = 1; i < SIZE/2; i++)
33                  a[i] = (a[i-1] + a[i+1]) / 2;
34
35              /* exchange boundary values */
36              write(pd[0][1],&a[SIZE/2-1], sizeof(int));
37              read(pd[1][0],&a[SIZE/2],sizeof(int));
38          }
39          read(pd[1][0],&a[SIZE/2],SIZE/2*sizeof(int));
40
41          /* wait for child to finish its part */
42          wait(status);
43      }
44      else {
45
46          for (t = 0; t < ITER; t++) {
47
48              for (i = SIZE/2; i < SIZE - 1; i++)
49                  a[i] = (a[i-1] + a[i+1]) / 2;
50
51              /* exchange boundary values */
52              write(pd[1][1],&a[SIZE/2], sizeof(int));
53              read(pd[0][0],&a[SIZE/2-1],sizeof(int));
54          }
55
56          /* send result to parent */
57          write(pd[1][1],&a[SIZE/2],SIZE/2*sizeof(int));
```

```
58          exit(0);
59      }
60
61      /* father prints the result */
62      for (i = 0; i < SIZE; i++) fprintf(stdout,"%d\n", a[i]);
63
64      return 0;
65  }
```

iv. Η λύση δίνεται στη συνέχεια.

```
1
2   #define SIZE 1024 // constant
3
4   int *a;
5
6   sem_t sem[2];
7
8   typedef struct thread_info_t{
9           int mystart;
10          int myend;
11          int iterations;
12          sem_t *wsem;
13          sem_t *psem;
14  } thread_info;
15
16  void *calc (void *args);
17
18  int main(void)
19  {
20      int i, t, ITER, rc;
21
22      pthread_t threads[2];
23      thread_info thread_info_array[2];
24
25      FILE *inFile;
26      inFile = fopen("1.txt", "r");
27
28      a = malloc(SIZE*sizeof(int));
29
30      fprintf(stdout, "Give me the number of iterations\n");
31      scanf("%d", &ITER);
32
33      for (i = 0; i < SIZE; i++) {
34          fscanf(inFile, "%d\n", &a[i]);
35      }
36
37      sem_init(&sem[0], 0, 0);
38      sem_init(&sem[1], 0, 0);
39
40      thread_info_array[0].mystart = 0;
41      thread_info_array[0].myend = SIZE/2;
42      thread_info_array[0].iterations = ITER;
43      thread_info_array[0].wsem = &sem[0];
44      thread_info_array[0].psem = &sem[1];
45
46      thread_info_array[1].mystart = SIZE/2;
47      thread_info_array[1].myend = SIZE - 1;
48      thread_info_array[1].iterations = ITER;
49      thread_info_array[1].wsem = &sem[1];
50      thread_info_array[1].psem = &sem[0];
51
52      for(t = 0; t < 2; t++) {
53          rc = pthread_create(&threads[t], NULL, calc, (void *)&thread_info_array[t]);
54      }
55      for(t = 0; t < 2; t++) {
56          pthread_join(threads[t], NULL);
57      }
```

```
58
59      for (i = 0; i < SIZE; i++) printf("%d\n", a[i]);
60
61      return 0;
62  }
63
64  void *calc (void *args)
65  {
66      thread_info *myinfo;
67      myinfo=(thread_info *)args;
68
69      int t, i;
70
71      int start = myinfo->mystart;
72      int end = myinfo->myend;
73      int iterations = myinfo->iterations;
74      sem_t *wsem = myinfo->wsem;
75      sem_t *psem = myinfo->psem;
76
77      for (t = 0; t < iterations; t++) {
78          for (i = start + 1; i < end; i++)
79              a[i] = (a[i-1] + a[i+1]) / 2;
80
81          /* notify sibling thread that boundary value has been written for this
82             iteration */
83          sem_post(psem);
84
85          /* wait sibling thread to write its boundary value for this iteration */
86          sem_wait(wsem);
87      }
88
89  }
```

Θέμα 3 (15%)

Απαντήστε συνοπτικά στις παρακάτω ερωτήσεις:

i. Αναφέρετε μηχανισμούς με τους οποίους το υλικό υποστηρίζει τις παρακάτω λειτουργίες:
   - Διαχωρισμός χώρου χρήστη και χώρου πυρήνα. (1%)
   - Διακοπτή χρονοδρομολόγηση. (1%)
   - Εικονική μνήμη. (1%)
   - Αποδοτικός συγχρονισμός. (1%)
   - Εικονικοποίηση. (1%)

ii. Ποια ζητήματα πρέπει να λάβει υπόψη του ο σχεδιαστής του χρονοδρομολογητή όταν στο σύστημα υπάρχουν πολλαπλοί επεξεργαστές; (4%)

iii. Τι είναι το πρόβλημα της ενεργού αναμονής; Δώστε ψευδοκώδικα για την υλοποίηση των σημαφόρων με και χωρίς ενεργό αναμονή. (6%)

Λύση Θέματος 3

i.
   - *Διαχωρισμός χώρου χρήστη και χώρου πυρήνα:* Trap
   - *Διακοπτή χρονοδρομολόγηση:* Hardware timer
   - *Εικονική μνήμη:* Memory Management Unit (MMU)
   - *Αποδοτικός συγχρονισμός:* Atomic instructions
   - *Εικονικοποίηση:* Protection rings/Privilege levels

ii.
   - Προσκόλληση σε Επεξεργαστή (processor affinity)
     - Επίδοση κρυφής μνήμης
   - Εξισορρόπηση Φόρτου (load balancing)
     - Απαιτεί μετακίνηση διεργασιών
       * Μετακίνηση ώθησης (push migration)
       * Μετακίνηση έλξης (pull migration)
   - Γνώση φυσικής τοπολογίας συστήματος (πχ. NUMA, SMT)

iii. Το πρόβλημα της ενεργού αναμονής (busy-wait) αναφέρεται στην κατανάλωση υπολογιστικών πόρων από μια διεργασία που εκτελεί επαναληπτικά ένα βρόχο αναμονής περιμένοντας άλλη διεργασία που βρίσκεται στο κρίσιμο τμήμα της. Κατά της διάρκεια της επαναληπτικής εκτέλεσης η ΚΜΕ δεν μπορεί να χρησιμοποιηθεί για άλλες εργασίες.

   Ψευδοκώδικας:

   Υλοποίηση με ενεργό αναμονή

```
1  wait (S) {
2    while (1) {
3      lock(S_lock);
4      if (S > 0) {
5        S--;
6        unlock(S_lock);
7        break;
8      }
9      unlock(S_lock);
10   }
11 }
12
```

```
1  signal (S) {
2    lock(S_lock);
3    S++;
4    unlock(S_lock);
5  }
6
```

Υλοποίηση χωρίς ενεργό αναμονή

```
1   wait (S) {
2      lock(S_lock);
3      S--;
4      if (S < 0) {
5         add proc to S_list;
6         unlock(S_lock);
7         block();
8      }
9      unlock(S_lock);
10  }
11
```

```
1   signal (S) {
2      lock(S_lock);
3      S++;
4      if (S <= 0) {
5         remove P from S_list;
6         wakeup(P);
7      }
8      unlock(S_lock);
9   }
10
```