2η Εργαστηριακή Άσκηση

«Λειτουργικά Συστήματα» (Ε' ΕΞΑΜΗΝΟ)

Ον/μο: **ΠΡΟΚΟΠΙΟΣ ΚΑΜΑΤΣΟΣ**

ΑΜ: **1072586**

Ον/μο: **ΧΡΥΣΟΣΤΟΜΟΣ-ΑΘΑΝΑΣΙΟΣ ΚΑΤΣΙΓΙΑΝΝΗΣ**

ΑΜ: **1072490**

Ον/μο: **ΚΡΙΣΤΙΑΝ ΛΟΥΚΑ**

ΑΜ: **1072625**

Ον/μο: **ΑΡΓΥΡΗΣ ΣΟΦΟΤΑΣΙΟΣ**

ΑΜ: **1079616**

Μέρος 1

[Οι εκτελέσεις όλων των προγραμμάτων έγιναν στο diogenis.]

Ερώτημα Α

(Α)

Χρησιμοποιούμε τις shared μεταβλητές:

```
int *p;  /* shared var for the sum */
int *root;  /* shared min Heap */
int *last;   /* index of the last element in shared min heap to be summed up */
```

και ένα δυαδικό σημαφόρο s για το συγχρονισμό των διεργασιών.

Οι shared μεταβλητές προσαρτώνται μέσω των συναρτήσεων shmget() και shmat() στο χώρο διευθύνσεων των διεργασιών που δημιουργούνται.

Η heap δομή υλοποιείται ως ένα array που αποθηκεύει 10*N διακριτούς ακέραιους από 1 έως 10*N προσπελάσιμο από το δείκτη root. Στη ρίζα του heap αποθηκεύεται κάθε φορά το min στοιχείο (έχουμε min heap δομή).

Οι τιμές 1, 2, …, 10*N αποθηκεύονται αρχικά στη heap δομή με αυτήν τη σειρά και έχουμε έτοιμο το min heap.

Κάθε φορά που κάποια από τις N θυγατρικές διεργασίες αποκτά (αμοιβαία αποκλειόμενη) πρόσβαση στη ρίζα root του heap:

1. αθροίζει την τιμή που είναι αποθηκευμένη στη ρίζα στη shared μεταβλητή p,
2. αντιμεταθέτει το περιεχόμενο της ρίζας με το τελευταίο στοιχείο του heap που δεν έχει ακόμη αθροιστεί (το οποίο βρίσκεται στη θέση του heap array που δείχνει ο δείκτης last - 1) στη μεταβλητή p,
3. μειώνει κατά 1 το περιεχόμενο του δείκτη last και
4. καλεί τη συνάρτηση shift_down() η οποία 'βυθίζει' το νέο περιεχόμενο της ρίζας στο heap tree ανεβάζοντας προς τα πάνω τα στοιχεία του μονοπατιού βύθισης ώστε να ικανοποιείται και πάλι η min heap ιδιότητα (δηλ. να έχουμε και πάλι μια min heap δομή).

Ο σημαφόρος s αρχικοποιείται στο 1 και συγχρονίζει την πρόσβαση στη heap δομή: μέσω των σημάτων wait() και post() μόνο μία διεργασία αποκτά ανά πάσα στιγμή πρόσβαση στη heap δομή (αποκλειστική πρόσβαση) και εκτελεί τα παραπάνω βήματα 1. έως 4.

Όλες οι N θυγατρικές διεργασίες τερματίζουν την εκτέλεσή τους όταν η τιμή του δείκτη last γίνει 0 που σημαίνει ότι στη μεταβλητή p έχουν αθροιστεί όλες οι τιμές της min heap δομής με τη σειρά 1, 2, …, 10*N. Η τελική τιμή της p είναι ίση με 10*N*(10*N+1)/2 που για N = 10 δίνει 5.050.

Το συγκεκριμένο πρόγραμμα είναι το **1.A.A.c** και για N = 10 η έξοδός του απεικονίζεται στο παρακάτω screenshot:

```
Initial value of shared variable p: 0

Process 10 adds 15: p = 120, Process 10 adds 23: p = 276, Process 10 adds 32: p = 528, Process 10 ad
ds 43: p = 946, Process 10 adds 52: p = 1378, Process 10 adds 75: p = 2850, Process 10 adds 82: p =
3403, Process 10 adds 88: p = 3916, Process 10 adds 95: p = 4560, Process 4 adds 3: p = 6, Process 4
 adds 12: p = 78, Process 4 adds 21: p = 231, Process 4 adds 31: p = 496, Process 4 adds 37: p = 703
, Process 4 adds 44: p = 990, Process 4 adds 51: p = 1326, Process 4 adds 59: p = 1770, Process 4 ad
ds 62: p = 1953, Process 4 adds 72: p = 2628, Process 4 adds 79: p = 3160, Process 3 adds 16: p = 13
6, Process 3 adds 47: p = 1128, Process 3 adds 55: p = 1540, Process 3 adds 66: p = 2211, Process 3
adds 93: p = 4371, Process 3 adds 96: p = 4656, Process 7 adds 8: p = 36, Process 7 adds 19: p = 190
, Process 7 adds 25: p = 325, Process 7 adds 36: p = 666, Process 7 adds 41: p = 861, Process 7 adds
 50: p = 1275, Process 7 adds 74: p = 2775, Process 7 adds 85: p = 3655, Process 7 adds 89: p = 4005
, Process 7 adds 97: p = 4753, Process 6 adds 6: p = 21, Process 6 adds 13: p = 91, Process 6 adds 1
7: p = 153, Process 6 adds 28: p = 406, Process 6 adds 33: p = 561, Process 6 adds 38: p = 741, Proc
ess 6 adds 45: p = 1035, Process 6 adds 56: p = 1596, Process 6 adds 60: p = 1830, Process 6 adds 68
: p = 2346, Process 6 adds 73: p = 2701, Process 6 adds 80: p = 3240, Process 9 adds 24: p = 300, Pr
ocess 9 adds 58: p = 1711, Process 9 adds 65: p = 2145, Process 9 adds 77: p = 3003, Process 9 adds
86: p = 3741, Process 9 adds 91: p = 4186, Process 5 adds 5: p = 15, Process 5 adds 9: p = 45, Proce
ss 5 adds 18: p = 171, Process 5 adds 27: p = 378, Process 5 adds 35: p = 630, Process 5 adds 42: p
= 903, Process 5 adds 49: p = 1225, Process 5 adds 76: p = 2926, Process 5 adds 84: p = 3570, Proces
s 5 adds 87: p = 3828, Process 5 adds 94: p = 4465, Process 5 adds 98: p = 4851, Process 1 adds 1: p
 = 1, Process 1 adds 4: p = 10, Process 1 adds 7: p = 28, Process 1 adds 14: p = 105, Process 1 adds
 20: p = 210, Process 1 adds 26: p = 351, Process 1 adds 34: p = 595, Process 1 adds 46: p = 1081, P
rocess 1 adds 53: p = 1431, Process 1 adds 61: p = 1891, Process 1 adds 63: p = 2016, Process 1 adds
 69: p = 2415, Process 1 adds 78: p = 3081, Process 1 adds 100: p = 5050, Process 2 adds 2: p = 3, P
rocess 2 adds 10: p = 55, Process 2 adds 30: p = 465, Process 2 adds 40: p = 820, Process 2 adds 67:
 p = 2278, Process 2 adds 71: p = 2556, Process 2 adds 83: p = 3486, Process 2 adds 90: p = 4095, Pr
ocess 2 adds 92: p = 4278, Process 8 adds 11: p = 66, Process 8 adds 22: p = 253, Process 8 adds 29:
 p = 435, Process 8 adds 39: p = 780, Process 8 adds 48: p = 1176, Process 8 adds 54: p = 1485, Proc
ess 8 adds 57: p = 1653, Process 8 adds 64: p = 2080, Process 8 adds 70: p = 2485, Process 8 adds 81
: p = 3321, Process 8 adds 99: p = 4950,

Final value of shared variable p = 5050

Total cycles taken by CPU: 6.9527e-310
Total time taken by CPU: 0 secs
```

Μεταξύ δύο διαδοχικών προσπελάσεων της ρίζας του heap κάθε διεργασία περιμένει για ένα τυχαίο χρονικό διάστημα 0 – 100 msecs.

Ο χρόνος υπολογισμού που τυπώνεται σε κάθε διαφορετική εκτέλεση του προγράμματος είναι συνήθως 0 secs δεδομένου ότι CLOCKS_PER_SEC = 1.000.000 (σταθερά που ορίζεται στο αρχείο time.h και δεν αντιστοιχεί στην πραγματική ακρίβεια του συστήματος) και για N = 10 ο συνολικός πραγματικός χρόνος επεξεργασίας (CPU time) είναι πολύ μικρός και της τάξης των msecs ή των μsecs.

(Β)

Όταν η heap δομή δεν είναι shared αλλά τοπική στις θυγατρικές διεργασίες, τότε κάθε διεργασία αθροίζει στη shared μεταβλητή p όλες τις 10*N διαφορετικές τιμές του τοπικού heap. Άρα, μετά τον τερματισμό όλων των N θυγατρικών διεργασιών, η τιμή της p θα είναι N φορές μεγαλύτερη από την τιμή που υπολογίστηκε με το προηγούμενο πρόγραμμα. Για N = 10 η τιμή αυτή θα είναι 50.050.

Για να επιβεβαιώσουμε το παραπάνω αποτέλεσμα δημιουργήσαμε το πρόγραμμα **1.A.B.c** όπου πλέον οι μεταβλητές root και last δεν είναι shared και κληρονομούνται στις θυγατρικές διεργασίες ως τοπικές μεταβλητές.

Στη συνέχεια απεικονίζονται δύο screenshots με την έξοδο του προγράμματος στην οποία εμφανίζεται, λόγω όγκου, μόνο το πρώτο και το τελευταίο τμήμα της εξόδου:

```
Initial value of shared variable p: 0

Process 2 adds 1: p = 2, Process 2 adds 2: p = 6, Process 2 adds 3: p = 16, Process 2 adds 4: p = 32
, Process 2 adds 5: p = 44, Process 2 adds 6: p = 83, Process 2 adds 7: p = 177, Process 2 adds 8: p
 = 217, Process 2 adds 9: p = 253, Process 2 adds 10: p = 315, Process 2 adds 11: p = 335, Process 2
 adds 12: p = 446, Process 2 adds 13: p = 506, Process 2 adds 14: p = 568, Process 2 adds 15: p = 80
5, Process 2 adds 16: p = 861, Process 2 adds 17: p = 945, Process 2 adds 18: p = 1007, Process 2 ad
ds 19: p = 1160, Process 2 adds 20: p = 1255, Process 2 adds 21: p = 1325, Process 2 adds 22: p = 14
30, Process 2 adds 23: p = 1800, Process 2 adds 24: p = 2044, Process 2 adds 25: p = 2131, Process 2
 adds 26: p = 2199, Process 2 adds 27: p = 2695, Process 2 adds 28: p = 2747, Process 2 adds 29: p =
 2969, Process 2 adds 30: p = 3146, Process 2 adds 31: p = 3230, Process 2 adds 32: p = 3517, Proces
s 2 adds 33: p = 3705, Process 2 adds 34: p = 3872, Process 2 adds 35: p = 4486, Process 2 adds 36:
p = 4706, Process 2 addsProcess 1 adds 1: p = 1, Process 1 adds 2: p = 9, Process 1 adds 3: p = 13,
Process 1 adds 4: p = 49, Process 1 adds 5: p = 61, Process 1 adds 6: p = 129, Process 1 adds 7: p =
 184, Process 1 adds 8: p = 238, Process 1 adds 9: p = 283, Process 1 adds 10: p = 364, Process 1 ad
ds 11: p = 396, Process 1 adds 12: p = 471, Process 1 adds 13: p = 709, Process 1 adds 14: p = 832,
Process 1 adds 15: p = 960, Process 1 adds 16: p = 1034, Process 1 adds 17: p = 1111, Process 1 adds
 18: p = 1224, Process 1 adds 19: p = 1362, Process 1 adds 20: p = 1636, Process 1 adds 21: p = 1759
, Process 1 adds 22: p = 1958, Process 1 adds 23: p = 2067, Process 1 adds 24: p = 2240, Process 1 a
dds 25: p = 2319, Process 1 adds 26: p = 2361, Process 1 adds 27: p = 2488, Process 1 adds 28: p = 2
```

...

```
17, Process 3 adds 81: p = 43564, Process 3 adds 82: p = 44361, Process 3 adds 83: p = 44540, Proces
s 3 adds 84: p = 45106, Process 3 adds 85: p = 45483, Process 3 adds 86: p = 46872, Process 3 adds 8
7: p = 47055, Process 3 adds 88: p = 47438, Process 3 adds 89: p = 48060, Process 3 adds 90: p = 482
36, Process 3 adds 91: p = 48414, Process 3 adds 92: p = 48506, Process 3 adds 93: p = 48776, Proces
s 3 adds 94: p = 48960, Process 3 adds 95: p = 49146, Process 3 adds 96: p = 49242, Process 3 adds 9
7: p = 49339, Process 3 adds 98: p = 49622, Process 3 adds 99: p = 49815, Process 3 adds 100: p = 50
010, cess 8 adds 71: p = 39505, Process 8 adds 72: p = 39742, Process 8 adds 73: p = 42405, Process
8 adds 74: p = 42873, Process 8 adds 75: p = 43037, Process 8 adds 76: p = 43483, Process 8 adds 77:
 p = 43830, Process 8 adds 78: p = 44091, Process 8 adds 79: p = 46142, Process 8 adds 80: p = 46508
, Process 8 adds 81: p = 46687, Process 8 adds 82: p = 47719, Process 8 adds 83: p = 47802, Process
8 adds 84: p = 47886, Process 8 adds 85: p = 47971, Process 8 adds 86: p = 48146, Process 8 adds 87:
 p = 48323, Process 8 adds 88: p = 48594, Process 8 adds 89: p = 48683, Process 8 adds 90: p = 48866
, Process 8 adds 91: p = 49051, Process 8 adds 92: p = 49431, Process 8 adds 93: p = 49524, Process
8 adds 94: p = 49716, Process 8 adds 95: p = 49910, Process 8 adds 96: p = 50106, Process 8 adds 97:
 p = 50203, Process 8 adds 98: p = 50301, Process 8 adds 99: p = 50400, Process 8 adds 100: p = 5050
0,

Final value of shared variable p = 50500

Total cycles taken by CPU: 6.95304e-310
Total time taken by CPU: 0 secs
```

Ο χρόνος επεξεργασίας (CPU time) αυτού του προγράμματος περιμένουμε να είναι θεωρητικά N = 10 φορές μεγαλύτερος από τον αντίστοιχο χρόνο επεξεργασίας του πρώτου προγράμματος. Για τους λόγους όμως που αναφέρθηκαν νωρίτερα, στην έξοδο τυπώνεται 0 secs.

Ερώτημα Β

Για το συγκεκριμένο ερώτημα υλοποιήσαμε τον αλγόριθμο που δόθηκε στις διαλέξεις.

Η υλοποίηση έγινε με **δύο τρόπους**. Στον πρώτο η γονική διεργασία δημιουργεί με τη fork() θυγατρικές διεργασίες που έχουν το ρόλο reader / writer. Στον δεύτερο οι readers / writers δημιουργούνται ως threads της αρχικής (κύριας) διεργασίας.

Οι shared μεταβλητές rc και wc αρχικοποιούνται στην τιμή 0 και μετρούν το πλήθος των readers writers αντίστοιχα που επιθυμούν να διαβάσουν την / γράψουν στην (ενημερώσουν την τιμή της) shared μεταβλητή data (database). Η αρχική τιμή της μεταβλητής data είναι 0 και αυξάνεται κάθε φορά από το writer κατά 1.

Και στους δύο τρόπους χρησιμοποιούνται οι σημαφόροι cSem και dataSem οι οποίοι αρχικοποιούνται στην τιμή 1 και εξασφαλίζουν τον αμοιβαίο αποκλεισμό κατά την πρόσβαση στις μεταβλητές rc, wc και στη μεταβλητή data αντίστοιχα.

Πολλοί readers μπορούν ταυτόχρονα να διαβάσουν την τιμή της data αλλά μόνο ένας writer μπορεί να ενημερώσει την τιμή αυτή και κατά τη διάρκεια της ενημέρωσης κανένας reader δεν επιτρέπεται να τη διαβάσει. Όταν πολλοί readers και writers θέλουν να προσπελάσουν τη μεταβλητή data ταυτόχρονα, την προσπελαύνουν εναλλάξ ένας προς έναν, πχ. πρώτα ένας reader, στη συνέχεια ένας writer, στη συνέχεια και πάλι ένας reader, κ.ο.κ. μέσω κατάλληλου χειρισμού της boolean μεταβλητής priority:

- όταν priority = 1 ο writer ενημερώνει την τιμή της data, όταν priority = 0 ο reader διαβάζει την τιμή της data,
- ο reader, αφού διαβάσει την τιμή της data, θέτει το priority = 1 όταν υπάρχει writer που περιμένει να γράψει (διάβασμα που ακολουθείται από εγγραφή),
- ο writer, αφού ενημερώσει την τιμή της data, θέτει το priority = 0 όταν υπάρχει reader που περιμένει να διαβάσει (εγγραφή που ακολουθείται από διάβασμα),
- όταν priority = 1 και δεν υπάρχουν writers που περιμένουν να γράψουν, τότε ο reader θέτει το priority = 0 ώστε να μπορέσει ο επόμενος reader που περιμένει να διαβάσει την τιμή της data,
- όταν priority = 0 και δεν υπάρχουν readers που περιμένουν να διαβάσουν, τότε ο writer θέτει το priority = 1 ώστε να μπορέσει ο επόμενος writer που περιμένει να ενημερώσει την τιμή της data.

Στα προγράμματα που δημιουργήσαμε κάθε reader / writer ξεκινά την εκτέλεσή του μετά ένα τυχαίο χρονικό διάστημα ίσο με rand()%100 msecs και επιχειρεί 5 φορές να διαβάσει / γράψει. Μεταξύ δυο διαδοχικών προσπαθειών ο reader / writer περιμένει πάλι για τυχαίο χρονικό διάστημα rand()%100 msecs.

1ος τρόπος

Χρησιμοποιούμε τις shared μεταβλητές:

```
int *rc, *wc, *priority;  /* shared vars */
int *data;   /* shared data value accessed by readers and updated by writers */
```

οι οποίες προσαρτώνται μέσω των συναρτήσεων shmget() και shmat() στο χώρο διευθύνσεων των διεργασιών που δημιουργούνται.

Οι διεργασίες reader / writer δημιουργούνται ως θυγατρικές της γονικής: στο i-for loop όπου καλείται η fork() η θυγατρική διεργασία με άρτιο i γίνεται reader και αυτή με περιττό i γίνεται writer, δηλ. έχουμε ίδιο πλήθος readers και writers. Κάθε reader / writer επιχειρεί 5 φορές να διαβάσει / γράψει.

Το πρόγραμμα είναι το **1.B1.c** και παρακάτω απεικονίζεται η έξοδός του για 5 readers και 5 writers.

```
Writer 1 writes 1 at try 0
Reader 0 reads 1 at try 0
Writer 5 writes 2 at try 0
Reader 2 reads 2 at try 1
Reader 0 reads 2 at try 1
Reader 6 reads 2 at try 0
Writer 1 writes 3 at try 3
Reader 8 reads 3 at try 0
Writer 5 writes 4 at try 1
Reader 4 reads 4 at try 2
Writer 9 writes 5 at try 0
Reader 8 reads 5 at try 1
Writer 5 writes 6 at try 2
Reader 4 reads 6 at try 3
Writer 3 writes 7 at try 2
Reader 4 reads 7 at try 4
Writer 5 writes 8 at try 3
Reader 0 reads 8 at try 4
Writer 7 writes 9 at try 0
Writer 9 writes 10 at try 1
Writer 3 writes 11 at try 3
Writer 3 writes 12 at try 4
Writer 7 writes 13 at try 1
Writer 9 writes 14 at try 2
Writer 1 writes 15 at try 4
Writer 9 writes 16 at try 3
Writer 9 writes 17 at try 4
Writer 5 writes 18 at try 4
Writer 7 writes 19 at try 2
Writer 7 writes 20 at try 3
Writer 7 writes 21 at try 4
```

Όπως φαίνεται στο screenshot, και οι 10 συνολικά διεργασίες αποκτούν τελικά πρόσβαση στη μεταβλητή για διάβασμα ή γράψιμο αντίστοιχα, οι writers όμως καταφέρνουν να γράψουν περισσότερες φορές απ' όσες διαβάζουν οι readers. Όλες επιχειρούν 5 φορές αλλά δεν καταφέρνουν να διαβάσουν ή να γράψουν αντίστοιχα. Πχ. οι readers 0 και 4 διαβάζουν τις 3, οι readers 2 και 6 μόνο 1, ο reader 8 2. Οι writers 5, 7 και 9 γράφουν και τις 5 φορές που επιχειρούν και οι writers 1 και 3 τις 3.

Όταν πολλοί readers και writers περιμένουν να διαβάσουν ή να γράψουν αντίστοιχα τότε η σειρά με την οποία επιτυγχάνεται αυτό είναι εναλλάξ, πχ. ο writer 1 γράφει το 3, ο reader 8 διαβάζει το 3, ο writer 5 γράφει το 4, ο reader 4 διαβάζει το 4, ο writer 9 γράφει το 5, ο reader 8 διαβάζει το 5, κ.ο.κ.

Στη συνέχεια παρατίθεται το screenshot από μία άλλη εκτέλεση του προγράμματος όπου φαίνεται ότι οι readers / writers καταφέρουν να διαβάσουν / γράψουν περίπου τις ίδιες φορές:

```
Reader 0 reads 0 at try 1
Writer 3 writes 1 at try 0
Writer 1 writes 2 at try 2
Reader 4 reads 2 at try 0
Writer 5 writes 3 at try 0
Reader 8 reads 3 at try 0
Reader 2 reads 3 at try 2
Reader 0 reads 3 at try 2
Writer 3 writes 4 at try 4
Reader 4 reads 4 at try 1
Writer 5 writes 5 at try 2
Writer 9 writes 6 at try 0
Reader 4 reads 6 at try 2
Reader 8 reads 6 at try 1
Reader 2 reads 6 at try 3
Reader 0 reads 6 at try 3
Reader 8 reads 6 at try 2
Reader 6 reads 6 at try 1
Writer 5 writes 7 at try 3
Writer 9 writes 8 at try 1
Reader 2 reads 8 at try 4
Reader 6 reads 8 at try 2
Reader 8 reads 8 at try 3
Writer 5 writes 9 at try 4
Writer 9 writes 10 at try 3
Reader 4 reads 10 at try 4
Writer 9 writes 11 at try 4
Writer 7 writes 12 at try 2
Writer 7 writes 13 at try 4
```

Τέλος, σε μια άλλη εκτέλεση το πρόγραμμα παράγει την παρακάτω έξοδο όπου φαίνεται ότι readers καταφέρνουν να διαβάσουν περισσότερες φορές από τους writers:

```
Reader 0 reads 0 at try 1
Reader 2 reads 0 at try 0
Writer 3 writes 1 at try 0
Reader 0 reads 1 at try 2
Writer 1 writes 2 at try 0
Reader 4 reads 2 at try 0
Reader 2 reads 2 at try 1
Reader 0 reads 2 at try 3
Reader 2 reads 2 at try 2
Writer 3 writes 3 at try 2
Reader 4 reads 3 at try 1
Writer 7 writes 4 at try 0
Reader 4 reads 4 at try 2
Writer 7 writes 5 at try 1
Writer 1 writes 6 at try 4
Reader 4 reads 6 at try 3
Writer 3 writes 7 at try 3
Reader 2 reads 7 at try 4
Reader 6 reads 7 at try 2
Writer 5 writes 8 at try 1
Reader 6 reads 8 at try 3
Reader 8 reads 8 at try 2
Reader 6 reads 8 at try 4
Reader 8 reads 8 at try 3
Reader 4 reads 8 at try 4
Reader 8 reads 8 at try 4
Writer 7 writes 9 at try 4
Writer 9 writes 10 at try 2
```

2ος τρόπος

Χρησιμοποιούμε τις global μεταβλητές:

```
int rc, wc, priority;  /* globally accessible (shared) vars */
int data;  /* shared data value accessed by readers and updated by writers */
```

Οι συγκεκριμένες μεταβλητές είναι προσπελάσιμες από όλα τα threads που δημιουργούνται και άρα shared και επομένως δεν απαιτείται η δημιουργία shared μνήμης μέσω των συναρτήσεων shmget() και shmat().

Κάθε thread έχει ρόλο reader ή writer: στο i-for loop για άρτιο i δημιουργείται ένα thread που εκτελεί τη διαδικασία reader() και για περιττό i δημιουργείται ένα thread που εκτελεί τη διαδικασία reader(), δηλ. έχουμε κι εδώ το ίδιο πλήθος readers και writers.

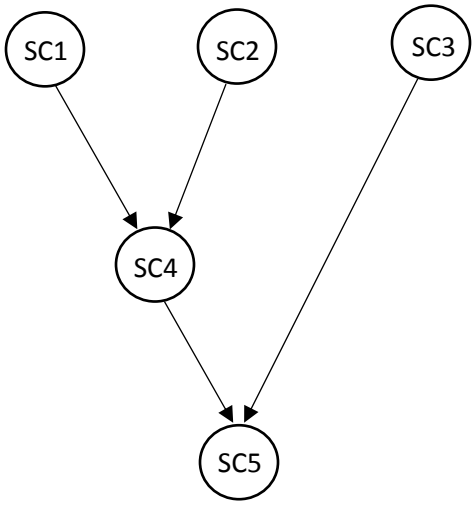
Το 2ο πρόγραμμα είναι το **1.B2.c** και στη συνέχεια παρατίθεται ένα screenshot με τα αποτελέσματα μιας εκτέλεσής του:

```
Reader 0 reads 0 at try 1
Writer 1 writes 1 at try 0
Writer 1 writes 2 at try 1
Reader 0 reads 2 at try 3
Reader 0 reads 2 at try 4
Reader 2 reads 2 at try 0
Writer 3 writes 3 at try 0
Writer 1 writes 4 at try 3
Writer 3 writes 5 at try 1
Writer 3 writes 6 at try 2
Reader 2 reads 6 at try 1
Writer 3 writes 7 at try 4
Reader 2 reads 7 at try 2
Reader 4 reads 7 at try 4
Writer 5 writes 8 at try 3
Reader 6 reads 8 at try 1
Writer 5 writes 9 at try 4
Reader 2 reads 9 at try 4
Writer 7 writes 10 at try 1
Reader 6 reads 10 at try 3
Writer 7 writes 11 at try 3
Reader 8 reads 11 at try 2
Writer 7 writes 12 at try 4
Writer 9 writes 13 at try 4
```

Στη συγκεκριμένη εκτέλεση οι readers / writers καταφέρουν να διαβάσουν / γράψουν περίπου τις ίδιες φορές και όλοι οι readers / writers διαβάζουν / γράφουν τουλάχιστον μία φορά.

Ερώτημα Γ

Ο γράφος προτεραιοτήτων για την εκτέλεση των 5 κλήσεων συστήματος είναι ο εξής:



Το αντίστοιχο πρόγραμμα εκτέλεσης είναι:

```
cobegin
   begin
      cobegin
         SC1;
         SC2;
      coend
      SC4;
   end
   SC3;
coend
SC5;
```

Για το συγχρονισμό της εκτέλεσης των system calls μπορούμε να χρησιμοποιήσουμε 4 δυαδικούς σημαφόρους s14, s24, s35, s45 (έναν σημαφόρο για κάθε ακμή του γράφου προτεραιοτήτων), οι οποίοι αρχικοποιούνται στο 0. Το παράλληλο πρόγραμμα είναι το εξής:

```
var s14, s24, s35, s45: binary semaphores;
s14 := s24 := s35 := s45 := 0;
cobegin
   begin SC1; signal(s14); end
   begin SC2; signal(s24); end
   begin SC3; signal(s35); end
   begin wait(S14); wait(s24); SC4; signal(s45); end
   begin wait(s35); wait(s45); SC5; end
coend
```

Εφόσον επιτρέπεται η χρήση γενικών σημαφόρων που μπορούν να αρχικοποιηθούν σε αρνητικές τιμές, τότε μπορούμε να χρησιμοποιήσουμε 2 μόνο σημαφόρους, τους s4 και s5 που αρχικοποιούνται στην τιμή -1. Τότε, το παράλληλο πρόγραμμα είναι το εξής:

```
var s4, s5: semaphores;
s4 := s5 := -1;
cobegin
   begin SC1; signal(s4); end
   begin SC2; signal(s4); end
   begin SC3; signal(s5); end
   begin wait(s4); SC4; signal(s5); end
   begin wait(s5); SC5; end
coend
```

Τα system calls που εκτελούμε είναι τα εξής:

SC1: system ("echo Hi there! >> output.txt")

SC2: system ("echo Have a nice day. >> output.txt");

SC3: system ("ls -l");

SC4: system ("cat output.txt");

SC5: system ("rm output.txt");

Στη συνέχεια παραθέτουμε δύο διαφορετικές υλοποιήσεις του παράλληλου προγράμματος. Και στις δύο υλοποιήσεις δημιουργούμε 5 θυγατρικές διεργασίες της ίδιας γονικής καθεμία εκ των οποίων εκτελεί ένα διαφορετικό system call.

Υλοποίηση 1

Υλοποιούμε τη λύση με τους 4 δυαδικούς σημαφόρους δεδομένου ότι η λύση των 2 γενικών σημαφόρων που αρχικοποιούνται στο -1 δεν

δουλεύει καθώς στις συναρτήσεις sem_open() και sem_init() η αρχική τιμή για το σημαφόρο είναι τύπου unsigned int και άρα δεν μπορεί να είναι αρνητική).

Το πρόγραμμα είναι το **1.C1.c** και στη συνέχεια παρατίθεται η έξοδος από την εκτέλεσή του:

```
total 172
-rwxrwxr-x. 1 st1079616 st1079616 11256 2022-01-20 11:45 1.A.A
-rw-rw-r--. 1 st1079616 st1079616  5937 2022-01-20 11:45 1.A.A.c
-rwxrwxr-x. 1 st1079616 st1079616 11102 2022-01-19 20:22 1.A.B
-rw-rw-r--. 1 st1079616 st1079616  5370 2022-01-19 20:22 1.A.B.c
-rwxrwxr-x. 1 st1079616 st1079616 11222 2022-01-19 21:10 1.B1
-rw-rw-r--. 1 st1079616 st1079616  5383 2022-01-19 20:34 1.B1.c
-rwxrwxr-x. 1 st1079616 st1079616 10336 2022-01-19 23:53 1.B2
-rw-rw-r--. 1 st1079616 st1079616  3993 2022-01-15 21:59 1.B2.c
-rwxrwxr-x. 1 st1079616 st1079616  8541 2022-01-19 21:13 1.C
-rwxrwxr-x. 1 st1079616 st1079616  8565 2021-12-31 20:40 1.C2
-rw-rw-r--. 1 st1079616 st1079616  2792 2021-12-31 20:40 1.C2.c
-rw-rw-r--. 1 st1079616 st1079616  2775 2022-01-19 21:13 1.C.c
-rwxrwxr-x. 1 st1079616 st1079616 11006 2022-01-21 18:52 1.D1.1
-rw-rw-r--. 1 st1079616 st1079616  5531 2022-01-21 18:52 1.D1.1.c
-rwxrwxr-x. 1 st1079616 st1079616 12150 2022-01-21 18:46 1.D1.2
-rw-rw-r--. 1 st1079616 st1079616  6529 2022-01-21 10:55 1.D1.2.c
-rwxrwxr-x. 1 st1079616 st1079616 10718 2022-01-21 18:47 1.D2
-rw-rw-r--. 1 st1079616 st1079616  5590 2022-01-21 18:42 1.D2.c
-rw-rw-r--. 1 st1079616 st1079616    27 2022-01-21 22:00 output.txt
Hi there!
Have a nice day.
```

Υλοποίηση 2

Στην υλοποίηση αυτή χρησιμοποιούμε μόνο ένα δυαδικό σημαφόρο s και δύο shared μεταβλητές s4 και s4. Ο σημαφόρος s εξασφαλίζει την αποκλειστική πρόσβαση στις shared μεταβλητές και αρχικοποιείται στην τιμή 1. Οι shared μεταβλητές αρχικοποιούνται στην τιμή -1 και:

- οι διεργασίες που εκτελούν τα SC1 και SC2 μετά την εκτέλεση των αντίστοιχων εντολών συστήματος αυξάνουν την τιμή της s4 κατά 1,
- η διεργασία που εκτελεί το SC4 περιμένει έως ότου η τιμή της s4 γίνει 1 οπότε και εκτελεί την αντίστοιχη εντολή συστήματος.
- οι διεργασίες που εκτελούν τα SC3 και SC4 μετά την εκτέλεση των αντίστοιχων εντολών συστήματος αυξάνουν την τιμή της s5 κατά 1,
- η διεργασία που εκτελεί το SC5 περιμένει έως ότου η τιμή της s5 γίνει 1 οπότε και εκτελεί την αντίστοιχη εντολή συστήματος,

δηλ. μέσω του αυτού του χειρισμού των s4, s5 προσπαθούμε να προσομοιώσουμε τη λειτουργία των 2 δυαδικών σημαφόρων που αναφέρθηκε νωρίτερα οι οποίοι, εφόσον μπορούν να αρχικοποιηθούν στο -1, επιλύουν το πρόβλημα συγχρονισμού μεταξύ των 5 διεργασιών που εκτελούν τα system calls.

Η αναμονή των διεργασιών SC4 και SC5 μέχρις ότου μπορέσουν να εκτελέσουν την εντολή συστήματος υλοποιείται με τον εξής κώδικα (δίνεται ο κώδικας για τη διεργασία SC4, αντίστοιχος είναι ο κώδικας και για τη διεργασία SC5):

```
/* acquire a lock to shared var s4 */
while (1)
{
     sem_wait (s);
     if ((*s4) <= 0)
          /* release lock to s4 */
          sem_post (s);
     else
          break;
}
```

Το πρόγραμμα αυτής της υλοποίησης είναι το **1.C2.c** και στη συνέχεια παρατίθεται η έξοδος από την εκτέλεσή του:

```
total 192
-rwxrwxr-x. 1 st1079616 st1079616 11256 2022-01-20 11:45 1.A.A
-rw-rw-r--. 1 st1079616 st1079616  5937 2022-01-20 11:45 1.A.A.c
-rwxrwxr-x. 1 st1079616 st1079616 11102 2022-01-19 20:22 1.A.B
-rw-rw-r--. 1 st1079616 st1079616  5370 2022-01-19 20:22 1.A.B.c
-rwxrwxr-x. 1 st1079616 st1079616 11222 2022-01-22 14:20 1.B1
-rw-rw-r--. 1 st1079616 st1079616  5383 2022-01-22 14:22 1.B1.c
-rwxrwxr-x. 1 st1079616 st1079616 10336 2022-01-19 23:53 1.B2
-rw-rw-r--. 1 st1079616 st1079616  3993 2022-01-15 21:59 1.B2.c
-rwxrwxr-x. 1 st1079616 st1079616  8542 2022-01-22 22:04 1.C1
-rw-rw-r--. 1 st1079616 st1079616  2775 2022-01-19 21:13 1.C1.c
-rwxrwxr-x. 1 st1079616 st1079616  9814 2022-01-22 22:03 1.C2
-rw-rw-r--. 1 st1079616 st1079616  4073 2022-01-22 22:03 1.C2.c
-rwxrwxr-x. 1 st1079616 st1079616 11006 2022-01-21 22:26 1.D1.1
-rw-rw-r--. 1 st1079616 st1079616  5567 2022-01-21 22:26 1.D1.1.c
-rwxrwxr-x. 1 st1079616 st1079616 12166 2022-01-21 22:37 1.D1.2
-rw-rw-r--. 1 st1079616 st1079616  6746 2022-01-21 22:37 1.D1.2.c
-rwxrwxr-x. 1 st1079616 st1079616 11008 2022-01-21 22:47 1.D1dead
-rw-rw-r--. 1 st1079616 st1079616  5791 2022-01-21 22:21 1.D1dead.c
-rwxrwxr-x. 1 st1079616 st1079616 10734 2022-01-21 22:40 1.D2
-rw-rw-r--. 1 st1079616 st1079616  5626 2022-01-21 22:40 1.D2.c
-rw-rw-r--. 1 st1079616 st1079616    27 2022-01-22 22:04 output.txt
Have a nice day.
Hi there!
```

Ερώτημα Δ

Το πρόβλημα παρουσιάζεται εξαιτίας του τρόπου με τον οποίο γίνεται η διαχείριση της shared μεταβλητής free_s και του shared πίνακα free_a στη διαδικασία Leave_p().

Έστω ότι σε μια δεδομένη χρονική στιγμή δεν υπάρχουν ελεύθερες θέσεις στο παρκινγκ και ένα αυτοκίνητο εισέρχεται για έκδοση εισιτηρίου στάθμευσης. Η αντίστοιχη διεργασία (έστω Χ) θα εκτελέσει τη διαδικασία Enter_p() και θα μπλοκαριστεί στην κρίσιμη περιοχή υπό συνθήκη await (free_s > 0) αφού ο μετρητής ελεύθερων θέσεων free_s έγινε 0 από την τελευταία διεργασία που κατάφερε και βρήκε θέση στάθμευσης (το πάρκινγκ γέμισε με την τελευταία διεργασία). Αν αργότερα κάποιο σταθμευμένο αυτοκίνητο αναχωρήσει από το πάρκινγκ, τότε η αντίστοιχη διεργασία (έστω Υ) θα εκτελέσει τη διαδικασία Leave_p() και ο μετρητής ελεύθερων θέσεων free_s θα γίνει 1.

Έστω τώρα η διεργασία Υ χάνει τον έλεγχο της CPU πριν εκτελεστεί η δεύτερη κρίσιμη περιοχή της διαδικασίας Leave_p(), δηλαδή πριν χαρακτηριστεί η συγκεκριμένη θέση στάθμευσης ως ελεύθερη. Στη συνέχεια, η διεργασία Χ θα αφυπνιστεί, θα κάνει τον μετρητή ελεύθερων θέσεων free_s ίσο με 0 και θα εκτελεστεί η δεύτερη κρίσιμη περιοχή της διαδικασίας Enter_p(). Όμως τώρα η συνάρτηση Επιλογή_Θέσης() δεν θα επιστρέψει μια έγκυρη θέση αφού

προηγουμένως η διεργασία Υ δεν πρόλαβε να ενημερώσει τον πίνακα free_a (free_a[i] = FALSE για όλα τα $1 \leq i \leq N$). Στη συνέχεια ο έλεγχος περνά στη διεργασία Υ η οποία θα ολοκληρώσει την εκτέλεση της διαδικασίας Leave_p() εκτελώντας τη δεύτερη κρίσιμη περιοχή της. Σύμφωνα με τα παραπάνω, παρόλο που έχει ελευθερωθεί μια θέση, αυτή δεν κατέστη δυνατόν να διατεθεί στο αυτοκίνητο που περίμενε. Ο δε οδηγός του θα περιμένει χωρίς αποτέλεσμα από τη μηχανή την έκδοση του εισιτηρίου του, ενώ βλέπει στο πάρκινγκ μια θέση ελεύθερη.

Για την επίλυση του προβλήματος αντιστρέφουμε τη σειρά εκτέλεσης των δυο κρίσιμων περιοχών στη διαδικασία Leave_p (free_p) (δηλαδή πρώτα εκτελείται η region free_a do … και στη συνέχεια η region free_s do …).

Η υλοποίηση της λύσης έγινε με **δύο τρόπους** και στον $1^{ο}$ τρόπο έχουμε δύο διαφορετικές υλοποιήσεις, άρα συνολικά τρία προγράμματα τα οποία αναλύονται στη συνέχεια.

$1^{ος}$ τρόπος

Υλοποίηση 1

Δημιουργούμε θυγατρικές διεργασίες της ίδιας γονικής οι οποίες προσομοιώνουν τη λειτουργία ενός αυτοκινήτου. Κάθε αυτοκίνητο επισκέπτεται το πάρκινγκ μετά από κάποιο τυχαίο χρονικό διάστημα από την τελευταία του επίσκεψη, σταθμεύει για κάποιο επίσης τυχαίο χρονικό διάστημα και στη συνέχεια αποχωρεί κι αυτό επαναλαμβάνεται για ένα συγκεκριμένο αριθμό επισκέψεων, ίδιο για όλα τα αυτοκίνητα.

Στο πρόγραμμά μας κάθε θυγατρική διεργασία καλεί τη διαδικασία car() η οποία εκτελεί σε ένα for loop τις εντολές:

```
usleep(rand()%100);
ticket = enter_p();
if (ticket != -1)
{
     usleep(rand()%100);
     leave_p(ticket);
}
```

Οι δυαδικοί σημαφόροι free_aSem και free_sSem συγχρονίζουν την αποκλειστική πρόσβαση στις shared μεταβλητές free_a και free_s αντίστοιχα και αρχικοποιούνται στην τιμή 1.

Η υπό συνθήκη κρίσιμη περιοχή στη διεργασία enter_p() υλοποιείται με τον παρακάτω κώδικα:

```
while (1)
{
     /* acquire a lock to shared var free_s */
     sem_wait (free_sSem);
     if ((*free_s) <= 0)
          /* release lock to free_s */
          sem_post (free_sSem);
     else
          break;
}

(*free_s)--;

/* release lock to free_s */
sem_post (free_sSem);
```

ενώ η κρίσιμη περιοχή free_s στη διαδικασία leave_p() υλοποιείται με τον εξής κώδικα:

```
/* acquire a lock to shared var free_s */
sem_wait (free_sSem);

(*free_s)++;

/* release lock to free_s */
sem_post (free_sSem);
```

Τέλος, οι έγκυρες θέσεις στο πάρκινγκ μετράνε από το 0 έως το N-1 αντί του 1 έως N που αναφέρεται στην εκφώνηση, χωρίς φυσικά να αλλάζει τις προδιαγραφές του προβλήματος που επιλύουμε.

Το συνολικό πρόγραμμα για τη συγκεκριμένη υλοποίηση υπάρχει στο αρχείο **1.D1.1.c.** Στη συνέχεια παρατίθενται τα αποτελέσματα από την εκτέλεση του προγράμματος όπου έχουμε 5 αυτοκίνητα και 3 συνολικά θέσεις στο πάρκινγκ ενώ κάθε αυτοκίνητο επισκέπτεται το πάρκινγκ 3 φορές:

```
Car 0 enters parking area with ticket 0 after visit 0
Car 3 enters parking area with ticket 1 after visit 0
Car 1 enters parking area with ticket 2 after visit 0
Car 3 with ticket 1 leaves parking area after visit 0
Car 4 enters parking area with ticket 1 after visit 0
Car 0 with ticket 0 leaves parking area after visit 0
Car 2 enters parking area with ticket 0 after visit 0
Car 1 with ticket 2 leaves parking area after visit 0
Car 3 enters parking area with ticket 2 after visit 1
Car 4 with ticket 1 leaves parking area after visit 0
Car 0 enters parking area with ticket 1 after visit 1
Car 3 with ticket 2 leaves parking area after visit 1
Car 0 with ticket 1 leaves parking area after visit 1
Car 2 with ticket 0 leaves parking area after visit 0
Car 4 enters parking area with ticket 0 after visit 1
Car 3 enters parking area with ticket 1 after visit 2
Car 4 with ticket 0 leaves parking area after visit 1
Car 0 enters parking area with ticket 0 after visit 2
Car 2 enters parking area with ticket 2 after visit 1
Car 0 with ticket 0 leaves parking area after visit 2
Car 4 enters parking area with ticket 0 after visit 2
Car 4 with ticket 0 leaves parking area after visit 2
Car 2 with ticket 2 leaves parking area after visit 1
Car 1 enters parking area with ticket 0 after visit 1
Car 2 enters parking area with ticket 2 after visit 2
Car 1 with ticket 0 leaves parking area after visit 1
Car 2 with ticket 2 leaves parking area after visit 2
Car 3 with ticket 1 leaves parking area after visit 2
Car 1 enters parking area with ticket 0 after visit 2
Car 1 with ticket 0 leaves parking area after visit 2
```

Όπως φαίνεται στο screenshot, κάθε αυτοκίνητο στην κάθε του επίσκεψη στο πάρκινγκ εισέρχεται στο πάρκινγκ και παρκάρει.

Στο επόμενο screenshot φαίνεται η έξοδος από την εκτέλεση του προγράμματος για 3 αυτοκίνητα που κάνουν 3 επισκέψεις το καθένα και το πάρκινγκ διαθέτει μόνο 1 θέση (κάθε αυτοκίνητο εισέρχεται στο πάρκινγκ και μόνο όταν εξέλθει εισέρχεται το επόμενο):

```
Car 1 enters parking area with ticket 0 after visit 0
Car 1 with ticket 0 leaves parking area after visit 0
Car 0 enters parking area with ticket 0 after visit 0
Car 0 with ticket 0 leaves parking area after visit 0
Car 1 enters parking area with ticket 0 after visit 1
Car 1 with ticket 0 leaves parking area after visit 1
Car 2 enters parking area with ticket 0 after visit 0
Car 2 with ticket 0 leaves parking area after visit 0
Car 1 enters parking area with ticket 0 after visit 2
Car 1 with ticket 0 leaves parking area after visit 2
Car 0 enters parking area with ticket 0 after visit 1
Car 0 with ticket 0 leaves parking area after visit 1
Car 2 enters parking area with ticket 0 after visit 1
Car 2 with ticket 0 leaves parking area after visit 1
Car 0 enters parking area with ticket 0 after visit 2
Car 0 with ticket 0 leaves parking area after visit 2
Car 2 enters parking area with ticket 0 after visit 2
Car 2 with ticket 0 leaves parking area after visit 2
```

Υλοποίηση 2

Δημιουργούμε και εδώ τις ίδιες θυγατρικές διεργασίες οι οποίες καλούν και πάλι τη διαδικασία car(). Η υπό συνθήκη κρίσιμη free_s τώρα υλοποιείται με την **conditional var free_sCond.** Για την αποκλειστική πρόσβαση στις shared μεταβλητές free_s και free_a χρησιμοποιούνται δύο **mutexes,** οι free_sLock και free_aLock αντίστοιχα (ο free_sLock χρησιμοποιείται μαζί με την conditional var).

Η conditional var και οι δύο mutexes δηλώνονται ως πεδία ενός struct το οποίο επισυνάπτεται στη shared memory των διεργασιών μαζί με τη μεταβλητή free_s και το array free_a.

Η υπό συνθήκη κρίσιμη περιοχή στη διεργασία enter_p() υλοποιείται τώρα με τον παρακάτω κώδικα:

```
/* acquire a lock to shared var free_s */
pthread_mutex_lock (&cm->free_sLock);

while ((*free_s) <= 0)
     pthread_cond_wait (&cm->free_sCond, &cm->free_sLock);

(*free_s)--;

/* release lock to free_s */
pthread_mutex_unlock (&cm->free_sLock);
```

ενώ η κρίσιμη περιοχή free_s στη διαδικασία leave_p() υλοποιείται με τον εξής κώδικα:

```
/* acquire a lock to shared var free_s */
pthread_mutex_lock (&cm->free_sLock);

(*free_s)++;

/* release lock to free_s */
pthread_mutex_unlock (&cm->free_sLock);

/* unblock enter_p() waiting on condition var */
pthread_cond_signal (&cm->free_sCond);
```

Το πλήρες πρόγραμμα της συγκεκριμένης υλοποίησης υπάρχει στο αρχείο **1.D1.2.c.** Στη συνέχεια παρατίθενται τα αποτελέσματα από την εκτέλεση του προγράμματος όπου έχουμε και πάλι 5 αυτοκίνητα και 3 συνολικά θέσεις στο πάρκινγκ ενώ κάθε αυτοκίνητο χρησιμοποιεί το πάρκινγκ 3 φορές:

```
Car 0 enters parking area with ticket 0 after visit 0
Car 1 enters parking area with ticket 1 after visit 0
Car 2 enters parking area with ticket 2 after visit 0
Car 0 with ticket 0 leaves parking area after visit 0
Car 3 enters parking area with ticket 0 after visit 0
Car 1 with ticket 1 leaves parking area after visit 0
Car 0 enters parking area with ticket 1 after visit 1
Car 2 with ticket 2 leaves parking area after visit 0
Car 1 enters parking area with ticket 2 after visit 1
Car 0 with ticket 1 leaves parking area after visit 1
Car 3 with ticket 0 leaves parking area after visit 0
Car 2 enters parking area with ticket 0 after visit 1
Car 0 enters parking area with ticket 1 after visit 2
Car 1 with ticket 2 leaves parking area after visit 1
Car 3 enters parking area with ticket 2 after visit 1
Car 0 with ticket 1 leaves parking area after visit 2
Car 1 enters parking area with ticket 1 after visit 2
Car 2 with ticket 0 leaves parking area after visit 1
Car 4 enters parking area with ticket 0 after visit 0
Car 1 with ticket 1 leaves parking area after visit 2
Car 3 with ticket 2 leaves parking area after visit 1
Car 3 enters parking area with ticket 1 after visit 2
Car 4 with ticket 0 leaves parking area after visit 0
Car 2 enters parking area with ticket 0 after visit 2
Car 3 with ticket 1 leaves parking area after visit 2
Car 4 enters parking area with ticket 1 after visit 1
Car 2 with ticket 0 leaves parking area after visit 2
Car 4 with ticket 1 leaves parking area after visit 1
Car 4 enters parking area with ticket 0 after visit 2
Car 4 with ticket 0 leaves parking area after visit 2
```

Παρακάτω απεικονίζονται τα αποτελέσματα του προγράμματος για 3 αυτοκίνητα που κάνουν 3 επισκέψεις στο πάρκινγκ το καθένα και το πάρκινγκ διαθέτει μόνο 1 θέση:

```
Car 0 enters parking area with ticket 0 after visit 0
Car 0 with ticket 0 leaves parking area after visit 0
Car 2 enters parking area with ticket 0 after visit 0
Car 2 with ticket 0 leaves parking area after visit 0
Car 0 enters parking area with ticket 0 after visit 1
Car 0 with ticket 0 leaves parking area after visit 1
Car 1 enters parking area with ticket 0 after visit 0
Car 1 with ticket 0 leaves parking area after visit 0
Car 0 enters parking area with ticket 0 after visit 2
Car 0 with ticket 0 leaves parking area after visit 2
Car 2 enters parking area with ticket 0 after visit 1
Car 2 with ticket 0 leaves parking area after visit 1
Car 1 enters parking area with ticket 0 after visit 1
Car 1 with ticket 0 leaves parking area after visit 1
Car 2 enters parking area with ticket 0 after visit 2
Car 2 with ticket 0 leaves parking area after visit 2
Car 1 enters parking area with ticket 0 after visit 2
Car 1 with ticket 0 leaves parking area after visit 2
```

2ος τρόπος

Το πρόβλημα μπορεί να θεωρηθεί ως μια ειδική περίπτωση του προβλήματος Παραγωγού – Καταναλωτή:

- Η enter_p() είναι η διεργασία -παραγωγός: 'παράγει' αυτοκίνητα τα οποία τοποθετούνται στις άδειες θέσεις του πάρκινγκ.
- Η leave_p() είναι η διεργασία-καταναλωτής: 'καταναλώνει' αυτοκίνητα αδειάζοντας τις κατειλημμένες θέσεις του πάρκινγκ.
- Ο χώρος του πάρκινγκ με τις N θέσεις αντιστοιχεί στη δομή buffer[N] του κλασικού προβλήματος Παραγωγού – Καταναλωτή.
- Η διαφοροποίηση σε σχέση με το κλασικό πρόβλημα Παραγωγού – Καταναλωτή συνίσταται στο ότι η leave_p() για κάποιο αυτοκίνητο καλείται πάντα μετά από μια enter_p() για το ίδιο αυτοκίνητο (δεν είναι δυνατό να φύγει από το πάρκινγκ ένα αυτοκίνητο αν νωρίτερα δεν ήταν σταθμευμένο εκεί) και επομένως δεν είναι δυνατόν να φύγει αυτοκίνητο από ένα άδειο πάρκινγκ.

Συνέπεια της τελευταίας παρατήρησης είναι ότι δεν απαιτείται η χρήση του σημαφόρου full που χρησιμοποιείται στο κλασικό πρόβλημα. Έτσι, η μεταβλητή free_s που στις προηγούμενες υλοποιήσεις ήταν shared μπορεί να αντικατασταθεί από έναν γενικό (counting) σημαφόρο μέσω του οποίου διασφαλίζεται η είσοδος του αυτοκινήτου στο πάρκινγκ όταν υπάρχουν ελεύθερες θέσεις (όταν η τιμή του σημαφόρου είναι > 0) ή η αναμονή του αυτοκινήτου έξω από το πάρκινγκ μέχρι να ελευθερωθεί κάποια θέση (όταν η τιμή του σημαφόρου είναι 0).

Έτσι, στην υλοποίηση χρησιμοποιούμε:

- το δυαδικό σημαφόρο free_aSem που συγχρονίζει την αποκλειστική πρόσβαση στη μοναδική shared μεταβλητή free_a και αρχικοποιείται στην τιμή 1 και
- το γενικό σημαφόρο free_sSem που συγχρονίζει την εκτέλεση της διεργασίας enter_p(), ανάλογα με το αν υπάρχουν άδειες θέσεις στο πάρκινγκ ή όχι και αρχικοποιείται στην τιμή N (όλες οι θέσεις του πάρκινγκ αρχικά είναι ελεύθερες).

Ο κώδικας των δύο διεργασιών είναι ο εξής:

```
/* enter to parking area */
int enter_p (int k, int visit)
{

      /* wait for a free parking place */
      sem_wait (free_sSem);


      /*****************/
      /* region free_a */
      /*****************/
      int free_p;

      /* acquire a lock to shared var free_a */
      sem_wait (free_aSem);

      free_p = select_place (free_a);
      if (free_p != -1)
      {
            printf ("\nCar %d enters parking area with ticket %d
                        after visit %d", k, free_p, visit);
            *(free_a + free_p) = 0;
      }
      else
            printf ("\nNo valid ticket for car %d in visit %d\n", k, visit);

      /* release lock to free_a */
      sem_post(free_aSem);

      return free_p;
}


/* leave parking area */
void leave_p (int k, int free_p, int visit)
{
```

```
        /*****************/
        /* region free_a */
        /*****************/
        /* acquire a lock to shared var free_a */
        sem_wait (free_aSem);

        *(free_a + free_p) = 1;
        printf("\nCar %d with ticket %d leaves parking area after
                  visit %d", k, free_p, visit);

        /* release lock to free_a */
        sem_post (free_aSem);


        /* create a free parking place */
        sem_post (free_sSem);
}
```

Το πλήρες πρόγραμμα του 2ου τρόπου υλοποίησης υπάρχει στο αρχείο **1.D2.c.** Στη συνέχεια παρατίθενται τα αποτελέσματα από την εκτέλεση του προγράμματος όπου έχουμε και πάλι 5 αυτοκίνητα και 3 συνολικά θέσεις στο πάρκινγκ ενώ κάθε αυτοκίνητο χρησιμοποιεί το πάρκινγκ 3 φορές:

```
Car 0 enters parking area with ticket 0 after visit 0
Car 1 enters parking area with ticket 1 after visit 0
Car 2 enters parking area with ticket 2 after visit 0
Car 0 with ticket 0 leaves parking area after visit 0
Car 3 enters parking area with ticket 0 after visit 0
Car 2 with ticket 2 leaves parking area after visit 0
Car 4 enters parking area with ticket 2 after visit 0
Car 4 with ticket 2 leaves parking area after visit 0
Car 0 enters parking area with ticket 2 after visit 1
Car 3 with ticket 0 leaves parking area after visit 0
Car 2 enters parking area with ticket 0 after visit 1
Car 0 with ticket 2 leaves parking area after visit 1
Car 2 with ticket 0 leaves parking area after visit 1
Car 4 enters parking area with ticket 0 after visit 1
Car 3 enters parking area with ticket 2 after visit 1
Car 4 with ticket 0 leaves parking area after visit 1
Car 1 enters parking area with ticket 0 after visit 1
Car 3 with ticket 2 leaves parking area after visit 1
Car 1 with ticket 0 leaves parking area after visit 1
Car 1 enters parking area with ticket 0 after visit 2
Car 0 enters parking area with ticket 2 after visit 2
Car 0 with ticket 2 leaves parking area after visit 2
Car 1 with ticket 0 leaves parking area after visit 2
Car 4 enters parking area with ticket 0 after visit 2
Car 2 enters parking area with ticket 2 after visit 2
Car 4 with ticket 0 leaves parking area after visit 2
Car 3 enters parking area with ticket 0 after visit 2
Car 2 with ticket 2 leaves parking area after visit 2
Car 3 with ticket 0 leaves parking area after visit 2
```

Ακολουθούν τα αποτελέσματα από την εκτέλεση του προγράμματος για 3 αυτοκίνητα που επισκέπτονται το πάρκινγκ 3 φορές το καθένα και πάρκινγκ με 1 μόνο θέση:

```
Car 1 enters parking area with ticket 0 after visit 0
Car 1 with ticket 0 leaves parking area after visit 0
Car 0 enters parking area with ticket 0 after visit 0
Car 0 with ticket 0 leaves parking area after visit 0
Car 2 enters parking area with ticket 0 after visit 0
Car 2 with ticket 0 leaves parking area after visit 0
Car 1 enters parking area with ticket 0 after visit 1
Car 1 with ticket 0 leaves parking area after visit 1
Car 0 enters parking area with ticket 0 after visit 1
Car 0 with ticket 0 leaves parking area after visit 1
Car 2 enters parking area with ticket 0 after visit 1
Car 2 with ticket 0 leaves parking area after visit 1
Car 1 enters parking area with ticket 0 after visit 2
Car 1 with ticket 0 leaves parking area after visit 2
Car 0 enters parking area with ticket 0 after visit 2
Car 0 with ticket 0 leaves parking area after visit 2
Car 2 enters parking area with ticket 0 after visit 2
Car 2 with ticket 0 leaves parking area after visit 2
```

Μέρος 2

Ερώτημα Α

Παραδοχές:

1. Στην εκφώνηση της άσκησης δεν υποδεικνύεται συγκεκριμένος αλγόριθμος που θα πρέπει να χρησιμοποιηθεί για την επιλογή της περιοχής μνήμης που θα τοποθετηθεί η εκάστοτε διεργασία. Κάνουμε την παραδοχή και επιλύουμε την άσκηση χρησιμοποιώντας τον αλγόριθμο First Fit.

2. Θεωρούμε ότι από την στιγμή που μία διεργασία εκχωρηθεί στην μνήμη, την επόμενη χρονική στιγμή μπορεί να γίνει κάτοχος της CPU, εάν ο αλγόριθμος SRTF (Shortest Remaining Time First) την επιλέξει, χωρίς να πρέπει πρώτα να μπει στην ουρά της CPU.

   Η παραδοχή αυτή βασίζεται στην παρατήρηση, από άλλες αντίστοιχες λυμένες ασκήσεις στις διαφάνειες του μαθήματος, πως η πρώτη διεργασία που εκχωρείται στην μνήμη γίνεται κάτοχος της CPU χωρίς να μπαίνει πρώτα στην ουρά της CPU.

ΧΣ: Χρονική Στιγμή

Ο: Οπή

Διαδικασίες	Στιγμή Άφιξης	Διάρκεια	Μνήμη
P_1	0	7	180K
P_2	1	2	100K
P_3	2	5	350K
P_4	3	6	100K
P_5	3	4	90K
P_6	4	2	80K

ΧΣ	Άφιξη	Εικόνα Μνήμης	ΚΜΕ	Ουρά Μνήμης	Ουρά ΚΜΕ	Τέλος
0	P_1	Ο:620K	-	P_1	-	-
1	P_2	P_1:180K Ο:440K	P_1	P_2	-	-
2	P_3	P_1:180K P_2:100K Ο:340K	P_2	P_3	P_1	-

3	P_4,P_5	P_1:180K P_2:100K O:340K	P_2	P_3,P_4,P_5	P_1	P_2
4	P_6	P_1:180K P_4:100K P_5:90K O:250K	P_5	P_3,P_6	P_1,P_4	-
5	-	P_1:180K P_4:100K P_5:90K P_6:80K O:170K	P_6	P_3	P_1,P_4,P_5	-
6	-	P_1:180K P_4:100K P_5:90K P_6:80K O:170K	P_6	P_3	P_1,P_4,P_5	P_6
7	-	P_1:180K P_4:100K P_5:90K O:250K	P_5	P_3	P_1,P_4	-
8	-	P_1:180K P_4:100K P_5:90K O:250K	P_5	P_3	P_1,P_4	-
9	-	P_1:180K P_4:100K P_5:90K O:250K	P_5	P_3	P_1,P_4	P_5
10	-	P_1:180K P_4:100K O:340K	P_1	P_3	P_4	-
11	-	P_1:180K P_4:100K O:340K	P_1	P_3	P_4	-
12	-	P_1:180K P_4:100K O:340K	P_1	P_3	P_4	-
13	-	P_1:180K P_4:100K O:340K	P_1	P_3	P_4	-
14	-	P_1:180K P_4:100K O:340K	P_1	P_3	P_4	-
15	-	P_1:180K P_4:100K O:340K	P_1	P_3	P_4	P_1
16	-	O:180K P_4:100K O:340K	P_4	P_3	-	-
17	-	O:180K P_4:100K O:340K	P_4	P_3	-	-
18	-	O:180K P_4:100K O:340K	P_4	P_3	-	-
19	-	O:180K P_4:100K O:340K	P_4	P_3	-	-
20	-	O:180K P_4:100K O:340K	P_4	P_3	-	-
21	-	O:180K P_4:100K O:340K	P_4	P_3	-	P_4
22	-	O:620K	-	P_3	-	-
23	-	P_3:350K O:270K	P_3	-	-	-
24	-	P_3:350K O:270K	P_3	-	-	-
25	-	P_3:350K O:270K	P_3	-	-	-
26	-	P_3:350K O:270K	P_3	-	-	-

27	-	P_3:350K O:270K	P_3	-	-	P_3

Ερώτημα Β

Μέγεθος σελίδας/πλαισίου: 1K bytes = 2^{10} bytes. Επομένως, η μετατόπιση δεσμεύει 10 bits.

Εύρος λογικών διευθύνσεων: 20 bits. Συνεπώς, το μέρος που δηλώνει τον αριθμό της ιδεατής σελίδας θα δεσμεύει 20 – 10 = 10 bits της λογικής διεύθυνσης.

Μέγεθος φυσικής μνήμης: 4 MB = 2^{22} bytes. Συνεπώς, εύρος φυσικών διευθύνσεων: 22 bits.

Επομένως, το μέρος που δηλώνει τον αριθμό της φυσικής σελίδας (πλαίσιο) θα δεσμεύει 22 – 10 = 12 bits της φυσικής διεύθυνσης.

Άρα, κάθε ιδεατή διεύθυνση U (20 bits) είναι στην μορφή:

Αριθμός Ιδεατής Σελίδας	**Μετατόπιση**
10 bits	10 bits

Και κάθε φυσική διεύθυνση Φ (22 bits) είναι στην μορφή:

Αριθμός Φυσικής Σελίδας	**Μετατόπιση**
12 bits	10 bits

α) Μια διεργασία η οποία αποτελείται από 6.100 bytes εισέρχεται στο σύστημα για εκτέλεση. Η εσωτερική κλασματοποίηση που θα προκαλέσει αν φορτωθεί εξ’ ολοκλήρου στη φυσική μνήμη του συστήματος υπολογίζεται ως εξής:

Αρχικά προσδιορίζουμε τον αριθμό φυσικών σελίδων (πλαισίων) που θα χρειαστεί για την αποθήκευσή της και στη συνέχεια βρίσκουμε την εσωτερική κλασματοποίηση που προκαλεί.

6.100 / 1.024 = 5,957.

Επομένως, η διεργασία θα απαιτήσει 6 φυσικές σελίδες για την αποθήκευσή της στη φυσική μνήμη του συστήματος.

Συνεπώς, η εσωτερική κλασματοποίηση που προκαλεί ισούται με:

6 * 1.024 – 6.100 = 44 bytes.

β) Έστω ότι στον πίνακα σελίδων της διεργασίας υπάρχουν οι ακόλουθες εγγραφές (οι αριθμοί δίνονται σε δεκαδικό σύστημα):

Αριθμός Σελίδας	**Αριθμός Πλαισίου**
0	11
1	12
2	1
3	15
4	-
5	8

i) 00388_{16}

Μετατρέπουμε την δοθείσα λογική διεύθυνση στο δυαδικό σύστημα και διαχωρίζουμε τον αριθμό σελίδας και τη μετατόπιση σύμφωνα με τις παρατηρήσεις μας στο πρώτο μέρος της άσκησης.

00388_{16}: $0000\ 0000\ 0011\ 1000\ 1000_2$

Αριθμός σελίδας: $0000\ 0000\ 00_2 = 0_{16} = 0_{10}$

Μετατόπιση: $11\ 1000\ 1000_2 = 388_{16}$

Με βάση τον πίνακα σελίδων, η ιδεατή σελίδα 0_{10} αντιστοιχεί στη φυσική σελίδα (πλαίσιο) $11_{10} = B_{16}$

Επομένως, η τελική φυσική διεύθυνση είναι:

$00\ 0000\ 0010\ 1111\ 1000\ 1000_2 = 002F88_{16}$

ii) $0125F_{16}$

Μετατρέπουμε την δοθείσα λογική διεύθυνση στο δυαδικό σύστημα και διαχωρίζουμε τον αριθμό σελίδας και τη μετατόπιση.

$0125F_{16}$: $0000\ 0001\ 0010\ 0101\ 1111_2$

Αριθμός σελίδας: $0000\ 0001\ 00_2 = 4_{16} = 4_{10}$

Μετατόπιση: $10\ 0101\ 1111_2 = 25F_{16}$

Με βάση τον πίνακα σελίδων, η ιδεατή (λογική) σελίδα 4_{10} δεν βρίσκεται στη φυσική μνήμη.

Επομένως έχουμε page fault.

iii) $015A4_{16}$

Μετατρέπουμε την δοθείσα λογική διεύθυνση στο δυαδικό σύστημα και διαχωρίζουμε τον αριθμό σελίδας και τη μετατόπιση.

$015A4_{16}$: $0000\ 0001\ 0101\ 1010\ 0100_2$

Αριθμός σελίδας: $0000\ 0001\ 01_2 = 5_{16} = 5_{10}$

Μετατόπιση: $01\ 1010\ 0100_2 = 1A4_{16}$

Με βάση των πίνακα σελίδων, η ιδεατή σελίδα 5_{10} αντιστοιχεί στη φυσική σελίδα (πλαίσιο) $8_{10} = 8_{16}$

Επομένως, η τελική φυσική διεύθυνση είναι:

$00\ 0000\ 0010\ 0001\ 1010\ 0100_2 = 0021A4_{16}$

Ερώτημα Γ

Έστω η παρακάτω ακολουθία αναφοράς μίας διεργασίας:

2 5 8 1 8 7 5 1 8 2 4 2 1 3 6 4 7 5 3 7

Με κόκκινο χρώμα σημειώνουμε τους αριθμούς σελίδων στα σημεία στα οποία συμβαίνουν σφάλματα σελίδας για την πολιτική αντικατάστασης σελίδων LRU (Least Recently Used).

Με **μαύρο-bold** χρώμα σημειώνουμε τις σελίδες που υπάρχουν στον πίνακα σελίδων ανά χρονική στιγμή.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

	2	5	8	1	8	7	5	1	8	2	4	2	1	3	6	4	7	5	3	7
0	2	2	2	2	2	7	7	7	7	2	2	2	2	2	2	4	4	4	4	4
1		5	5	5	5	5	5	5	5	5	5	4	4	4	4	6	6	6	6	3	4
2			8	8	8	8	8	8	8	8	8	8	8	3	3	3	3	5	5	5
3				1	1	1	1	1	1	1	1	1	1	1	1	1	7	7	7	7