Ερώτημα 1.Α

Το πρόγραμμα δημιουργεί 30 διεργασίες στις οποίες η κάθε μια περιμένει 60-2*i δευτερόλεπτα και τερματίζει με exit status σε 100+i δευτερόλεπτα. (!!!ΤΡΟΠΟΟΙΗΣΗ ΟΣΟ ΓΙΝΕΤΑΙ!!!)

Το πρόγραμμα ξεκινάει με την κλήση των βιβλιοθηκών #include <unistd.h>, η οποία καλείται για να χρησιμοποιήσουμε τις συναρτήσεις exrcl(), wait() και fork(). Η βιβλιοθήκη #include <stdio.h> η οποία καλείται έτσι ώστε αν εισάγουμε και να εξάγουμε τιμές και η βιβλιοθήκη #include <sys/wait.h> η οποία καλείται για την χρήση της συνάρτησης waitpid(). Επίσης θέτουμε μια σταθερά N η οποία είναι ίση με 30

Στην main το πρόγραμμα ξεκινάει θέτοντας έναν πίνακα με 30 τιμές. Για κάθε κελί του πίνακα θέτουμε και τον αριθμό ταυτότητας του. Θέτουμε δύο ακέραιους i και child_status.

Ξεκινάει ένας βρόγχος ο οποίος θα επαναληφθεί 30 φόρες. Συγκεκριμένα για κάθε κελί του πίνακα δημιουργούμε μια διαργασία "πατέρας". Οπότε έχουμε 30 διεργασίες "παιδί" και 30 διεργασίες "πατέρας".

Αν το η τιμή του pid[i] είναι ίση με 0 τότε το σύστημα θα περιμένει για 60-2*i δευτερόλεπτα και θα τερματίσει επιστρέφοντας την τιμή 100+i.

Ξεκινάει ένας δεύτερος βρόγχος που θα επαναληφθεί 30 φορές.

Σε αυτό το βρόγχο προκαλούμε την αναμονή μιας διεργασίας μέχρι το παιδί pid[i] να τερματίσει.

Εάν το παιδί τερματίσει φυσιολογικά τότε το πρόγραμμα τυπώνει "Το παιδί με την τιμή wpid τερματίστηκε με το exit status της τιμής wpid".

Επιστρέφει τον κωδικό εξόδου child status της διαδικασίας του παιδιού. Αυτό γίνεται μόνο αν η WIFEXITED δεν επιστρέψει 0.

Εάν το παιδί δεν τερματίσει φυσιολογικά τότε το πρόγραμμα τυπώνει "Το παιδί με τιμή wpid τερματίστηκε μη φυσιολογικά".

Τελειώνει ο βρόγχος και το πρόγραμμα τερματίζει.

Ερώτημα 1.Β

Στο Ερώτημα αυτό θα γίνει η χρήση ενός συγκεκριμένου εργαλείου, του **αλγόριθμου bakery**. Ο αλγόριθμος αυτός αρχικά είναι μία από τις απλούστερες λύσεις για τα προβλήματα που αφορούν τον αμοιβαίο αποκλεισμό για την γενική περίπτωση των N διεργασιών (με N να συμβολίζει έναν τυχαίο θετικό ακέραιο αριθμό). Είναι η λύση για το κρίσιμο (πιο συγκεκριμένα) σημείο των διεργασιών.

Λειτουργία Αλγορίθμου: Όταν μία διεργασία πάει να μπει στο κρίσιμο της σημείο (critical section), πριν μπει, ο αλγόριθμος θα της δώσει έναν αριθμό – εισιτήριο.

Με απλά λόγια ο αλγόριθμος κάνει την συγκεκριμένη δουλειά:

```
if i < j
    Pi is served first;
else
    Pj is served first;
```

Δηλαδή η διεργασία με τον μικρότερο αριθμό, θα μπει νωρίτερα στο κρίσιμο της σημείο. Επίσης ο αλγόριθμος λειτουργεί με το σκεπτικό του First Come First Serve.

Υλοποιώντας λίγο πιο αναλυτικά, ο ψευδοκώδικας του αλγορίθμου αυτού αναγράφεται:

```
    repeat
    choosing[i] := true;
    number[i] := max(number[0], number[1], ..., number[n - 1])+1;
    choosing[i] := false;
    for j := 0 to n - 1
        do begin
            while choosing[j] do no-op;
            while number[j] != 0
                and (number[j], j) < (number[i], i) do no-op;
        end;

        critical section

    number[i] := 0;

        remainder section

until false;
```

Ερώτημα 2.Α

Στο ερώτημα αυτό έχουμε να κάνουμε με έναν «παράλληλο» κώδικά και αρχικοποιημένες τιμές ακεραίων μεταβλητών.

Στον κώδικα:
cobegin
X := X +1;
X := Y+1;
coend

Τα βήματα που εκτελούνται ώστε να αποθηκευτούν κάποιες τιμές στην μεταβλητή Χ, εκτελούνται παράλληλα, με άλλα λόγια κάθε φορά που θα τρέξει ο κώδικας αυτός θα έχουμε διαφορετική (ή και ίδια) τιμή, η οποία είναι ανάλογη της σειράς των βημάτων του κώδικα.

Δηλαδή επειδή θα τρέξει παράλληλα, μπορεί καθώς έχει εκτελέσει τα 2 πρώτα βήματα για την εντολή X:= X+1; Να προλάβει στην πορεία να εκτελέσει και τα 3 βήματα της εντολής X:=Y+1;

Ο κώδικας αναλυτικά με τα βήματα σχηματίζεται:

Cobegin

TX:= X;	*TY:= Y;*
TX:= TX + 1;	*TY:= TY + 1;*
X:= TX;	*X:= TY;*

Coend

Λαμβάνοντας υπόψιν όλες τις πιθανές πορείες που μπορεί να ακολουθήσει ο κώδικας αυτός, τα έχουμε τα εξής αποτελέσματα.

Σειρά	1η	2η	3η	4η	5η	6η	Τιμή του X
1	TX:=X;	TX:=TX+1;	X:=TX;	TY:=Y;	TY:=TY+1;	X:=TY;	11
2	TX:=X;	TX:=TX+1;	TY:=Y;	X:=TX;	TY:=TY+1;	X:=TY;	11
3	TX:=X;	TX:=TX+1;	TY:=Y;	TY:=Y+1;	X:=TX;	X:=TY;	11
4	TX:=X;	TX:=TX+1;	TY:=Y;	TY:=Y+1;	X:=TY;	X:=TX;	1
5	TX:=X;	TY:=Y;	TX:=TX+1;	X:=TX;	TY:=TY+1;	X:=TY;	11
6	TX:=X;	TY:=Y;	TX:=TX+1;	TY:=Y+1;	X:=TX;	X:=TY;	11
7	TX:=X;	TY:=Y;	TX:=TX+1;	TY:=Y+1;	X:=TY;	X:=TX;	1
8	TX:=X;	TY:=Y;	TY:=Y+1;	TX:=TX+1;	X:=TX;	X:=TY;	11
9	TX:=X;	TY:=Y;	TY:=Y+1;	TX:=TX+1;	X:=TY;	X:=TX;	1
10	TX:=X;	TY:=Y;	TY:=Y+1;	X:=TY;	TX:=TX+1;	X:=TX;	1
11	TY:=Y;	TX:=X;	TX:=TX+1;	X:=TX;	TY:=TY+1;	X:=TY;	11
12	TY:=Y;	TX:=X;	TX:=TX+1;	TY:=Y+1;	X:=TX;	X:=TY;	11
13	TY:=Y;	TX:=X;	TX:=TX+1;	TY:=Y+1;	X:=TY;	X:=TX;	1
14	TY:=Y;	TX:=X;	TY:=Y+1;	TX:=TX+1;	X:=TX;	X:=TY;	11
15	TY:=Y;	TX:=X;	TY:=Y+1;	TX:=TX+1;	X:=TY;	X:=TX;	1
16	TY:=Y;	TX:=X;	TY:=Y+1;	X:=TY;	TX:=TX+1;	X:=TX;	1
17	TY:=Y;	TY:=Y+1;	TX:=X;	TX:=TX+1;	X:=TX;	X:=TY;	11
18	TY:=Y;	TY:=Y+1;	TX:=X;	TX:=TX+1;	X:=TY;	X:=TX;	1
19	TY:=Y;	TY:=Y+1;	TX:=X;	X:=TY;	TX:=TX+1;	X:=TX;	1
20	TY:=Y;	TY:=Y+1;	X:=TY;	TX:=X;	TX:=TX+1;	X:=TX;	12

Στον πίνακα διαγράφονται όλα τα πιθανά μονοπάτια που μπορεί να ακολουθήσει ο κώδικας αυτός.

Όπως φαίνεται, οι τιμές που μπορεί να πάρει η μεταβλητή X είναι 1, 11 και 12.

Όλες οι φορές που θα πάρει την τιμή 1, έχει να κάνει αν το βήμα X:= TY δεν έχει γίνει πριν το βήμα TX:= X; Και έχει καταλήξει στο X:= TX΄;

Ανάλογα με την τιμή 11 εάν καταλήξει το X:=TY;

Και θα πάρει την τιμή 12 μονάχα στην περίπτωση που θα τρέξουν πρώτα όλα τα βήματα του X := Y +1 και μετά τα υπόλοιπα. Αυτό είναι σημαντικό ώστε να λάβει το TX την τιμή που υπάρχει πλέον στο ανανεωμένο X η οποία και θα είναι 11 εκείνη την στιγμή.

Ερώτημα 2.Β

(α)

```
cobegin

Process1()      Process2()      Process3()
{               {               {
while(TRUE){    while(TRUE){    while(TRUE){
print("P");     print("Z");     print("A");
print("I");
}               }               }
}               }               }

coend
```

```
semaphores a=0; b=0; c=0;

cobegin

begin Process1; signal(a); end;
begin wait(a); Process2; signal(b); end;
begin wait(b); Process2; signal(c); end;
begin wait(c); Process3; end;

coend;
```

(β)

```
cobegin

Process1()      Process2()      Process3()
{               {               {
while(TRUE){    while(TRUE){    while(TRUE){
print("P");     print("Z");      print("A");
print("I");
forever;        forever;forever;
}               }               }
}               }               }

coend
```

```
semaphores a=0; b=0; c=0;

cobegin

begin Process1; signal(a); end;
begin wait(a); Process2; signal(b); end;
begin wait(b); Process2; signal(c); end;
begin wait(c); Process3; signal(a) end;

coend;
```

Ερώτημα 2.Γ

α)

```
                                Var s1, s2: semaphore;
                                shared var x: integer;
                                s1:=2; s2:=0;
                                cobegin

Διεργασία_A1{          Διεργασία_A2{  Διεργασία_B1{  Διεργασία_A3{
        Διεργασία_B2{
down(s1);       down(s1);       down(s2);       down(s1);       down(s2);
up(s2);                 up(s2);             down(s2);       up(s2);             down(s2);
}               }               up(s1);         }               up(s1);
                                up(s2);                         up(s2);
                                }                               }

                                coend
```

Η λειτουργία down μειώνει την τιμή του σημαφόρου κατά 1.
Η λειτουργία up αυξάνει την τιμή του σημαφόρου κατά 1.

ΑΡΧΙΚΑ	s1=2	s2=0
Διεργασία Α1:	s1=1	s2=1
Διεργασία Α2:	s1=0	s2=2
Διεργασία Β1:	s1=1	s2=1 -> 0 -> 1
Διεργασία Α3:	s1=0	s2=2
Διεργασία Β2:	s1=1	s2=1 -> 0 -> 1

Οι διεργασίες μπορούν να λειτουργήσουν επειδή κανένας σημαφόρος δεν παίρνει αρνητική τιμή.

β)

ΑΡΧΙΚΑ	s1=2	s2=0
Διεργασία Α1:	s1=1	s2=1
Διεργασία Α2:	s1=0	s2=2
Διεργασία Β1:	s1=1	s2=1 -> 0 -> 1
Διεργασία Β2:	s1=2	s2=0 -> -1 ΣΦΑΛΜΑ
Διεργασία Α3:		

Στην διεργασία Β2 ο σημαφόρος δεν μπορεί να πάρει αρνητική τιμή
οπότε το πρόγραμμα αναστέλλεται.

Ερώτημα 2.Δ

(Α) Στην πρώτη περίπτωση θα εκτελέσουμε τον κώδικα χωρίς την χρήση σηματοφόρων. Παρόλο που ο κώδικας είναι υλοποιήσιμος σωστά, η απουσία σηματοφόρων θα προκαλέσει τον κώδικα να τρέξει σχετικά ταυτόχρονα με όλες του τις διεργασίες.

cobegin

shared var K = L = 1;

Process_1	Process_2	Process_3	Process_4
while (TRUE) {	while (TRUE) {	while (TRUE) {	while (TRUE) {
L:=K;	L:=K;	L:=K;	L:=K;
K:=K+11;	K:=K+11;	K:=K+11;	K:=K+11;
print_num(L,L+10);	print_num(L,L+10);	print_num(L,L+10);	print_num(L,L+10);
}	}	}	}

coend

Όταν θα ξεκινήσει να γίνεται η πρώτη διεργασία υπάρχει μεγάλη πιθανότητα να αρχίσει η δεύτερη διεργασία πριν ολοκληρώσει η πρώτη, κ.ο.κ.

```
Please give a number of how many times shall we ran the Process: 4

1 2 4 5 12 13 14 15 16 17 18 23 24 34 35 36 37 38 39 40 41 42 43 44
```

Εδώ βλέπουμε τα αποτελέσματα του κώδικα. Μπορούμε να προσέξουμε η πρώτη διεργασία σταματάει στον αριθμό 5 και εκείνη την στιγμή μπαίνει η δεύτερη διεργασία. Το καταλαβαίνουμε από την διαφορά επειδή μετά το 5 πηγαίνει κατευθείαν στο 12.

Αυτό θα συμβεί επειδή στον κώδικα κάθε διεργασίας έχουμε τις μεταβλητές K, L οι οποίες δέχονται τιμές σύμφωνα με την πορεία του κώδικα. Η μεταβλητή L θα λάβει την τιμή της K όταν θα τρέξει η διεργασία και θα τυπώσει τα ανάλογα, ενώ η K σε κάθε διεργασία αυξάνεται κατά 11. Οπότε στο παράδειγμα αυτό όταν η print_num της πρώτης διεργασίας έφτασε στο 5 τότε η δεύτερη διεργασία έτρεξε, έχοντας την τιμή του K = 12, όπου το θέσαμε στο L και έπειτα διατυπώθηκε όπως και φαίνεται.

Παρομοίως και για τις υπόλοιπες τιμές στις οποίες φαίνονταν απότομες αλλαγές.

Στην περίπτωση λοιπόν της απουσίας σημαφόρων, είναι μερικώς αδύνατο να καταφέρουμε να εμφανίσουμε όλες τις επιθυμητές τιμές.

Σημείωση: Αξίζει να ειπωθεί το γεγονός ότι υπάρχει μία μικρή πιθανότητα να εκτελεστούν όλες οι διεργασίες κανονικά, η οποία πιθανότητα είναι αντιστρόφος ανάλογη του αριθμού των διεργασίων που θα εκτελεστούν.

(Β) Στην περίπτωση της χρήσης των σημαφόρων τα αποτελέσματα θα είναι τα επιθυμητά. Αυτό θα συμβεί επειδή με την χρήση τους, θα τρέξουν τις διεργασίες η μία μετά την άλλην ώστε να βγει το αποτέλεσμα που θέλουμε.

```
var s1, s2, s3,  : semaphores;
   s1=s2=s3=0;
   cobegin
```

```
    begin P1; up(s1);end;
    begin down(s1); P2; up(s2); end;
    begin down(s2); P3; up(s3); end;
    begin down(s3); P4; end;
  coend;
```

Στον κώδικα αυτόν έχουμε πάρει το N να ισούται με 4, δηλαδή θα τρέξουνε 4 παράλληλες διεργασίες με το ίδιο περιεχόμενο.

Ανάλογα εάν θα είχαμε περισσότερες διεργασίες (παράδειγμα N=5 ή N= 7) θα είχαμε περισσότερους σημαφόρους ώστε να έχουμε και το ανάλογο αποτέλεσμα.

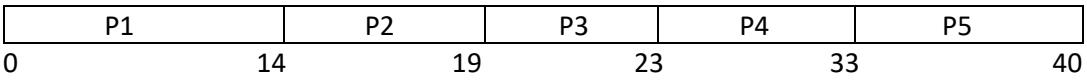
Τα αποτελέσματα θα είναι αυτήν την φορά αυτά που θέλουμε

```
1 2 3 4 5 6 7 8 9 10 11
12 13 14 15 16 17 18 19 20 21 22
23 24 25 26 27 28 29 30 31 32 33
34 35 36 37 38 39 40 41 42 43 44
```

Σημείωση: Στο παράδειγμα έτρεξαν 4 διεργασίες.

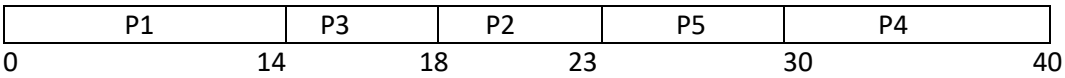
Ερώτημα 2.Ε

1) FCFS



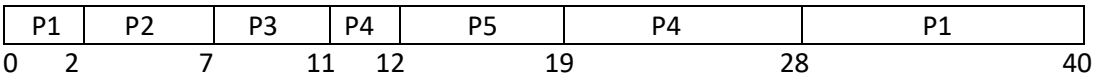
ΔΙΕΡΓΑΣΙΑ	ΧΡΟΝΟΣ ΔΙΕΚΠΕΡΑΙΩΣΗΣ	ΧΡΟΝΟΣ ΑΝΑΜΟΝΗΣ
P1	14-0=14	14-14=0
P2	19-2=17	17-5=12
P3	23-4=19	19-4=15
P4	33-7=26	26-10=16
P5	40-12=28	28-7=21
ΜΕΣΟΣ ΟΡΟΣ	20,8	12,8

2) SJF



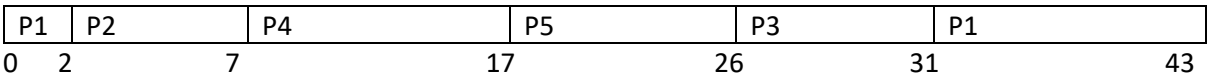
ΔΙΕΡΓΑΣΙΑ	ΧΡΟΝΟΣ ΔΙΕΚΠΕΡΑΙΩΣΗΣ	ΧΡΟΝΟΣ ΑΝΑΜΟΝΗΣ
P1	14-0=14	14-14=0
P2	23-4=21	21-5=16
P3	18-4=14	14-4=10
P4	40-7=33	33-10=23
P5	30-12=18	18-7=11
ΜΕΣΟΣ ΟΡΟΣ	20	12

3) SRTF



ΔΙΕΡΓΑΣΙΑ	ΧΡΟΝΟΣ ΔΙΕΚΠΕΡΑΙΩΣΗΣ	ΧΡΟΝΟΣ ΑΝΑΜΟΝΗΣ
P1	40-0=40	40-14=26
P2	7-2=5	5-5=0
P3	11-7=4	4-4=0
P4	28-11=17	17-10=7
P5	19-12=7	7-7=0
ΜΕΣΟΣ ΟΡΟΣ	14,6	6,6

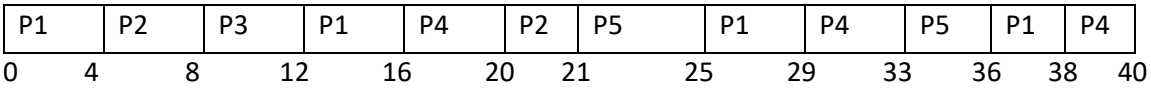
4) PS



ΔΙΕΡΓΑΣΙΑ	ΧΡΟΝΟΣ ΔΙΕΚΠΕΡΑΙΩΣΗΣ	ΧΡΟΝΟΣ ΑΝΑΜΟΝΗΣ
P1	43-0=43	43-14=29
P2	7-2=5	5-5=0
P3	31-4=27	27-4=23
P4	17-7=10	10-10=0

P5	26-12=14	14-7=7
ΜΕΣΟΣ ΟΡΟΣ	19,8	11,8

5) RR



ΔΙΕΡΓΑΣΙΑ	ΧΡΟΝΟΣ ΔΙΕΚΠΕΡΑΙΩΣΗΣ	ΧΡΟΝΟΣ ΑΝΑΜΟΝΗΣ
P1	38-0=38	38-14=24
P2	21-2=19	19-5=14
P3	12-4=8	8-4=4
P4	40-7=33	33-10=23
P5	36-12=24	24-7=17
ΜΕΣΟΣ ΟΡΟΣ	24,4	16,4