ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΑΘΗΜΑ : ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΑΝΤΙΚΕΙΜΕΝΟ: 1η Εργαστηριακή Άσκηση 2018 - 2019

των φοιτητών

Παναγιώτη Καββαδία
ΑΜ: 1054350

pkavadia13@gmail.com
up1054350@upnet.gr

Θωμά Χατζόπουλου
ΑΜ: 1054288

chatzothomas@gmail.com
up1054288@upnet.gr

Διδάσκοντες Καθηγητές: **Σπυρίδων Σιούτας**

Χρήστος Μακρής

Αριστείδης Ηλίας

ΠΑΤΡΑ | Νοέμβριος 2018

Περιεχόμενα:

Μέρος 1ο: Πρόγραμμα BASH shell για τη διαχείριση ενός αρχείου χρηστών καταγραφής γεγονότων (log files) για την πλοήγηση σε κοινωνικά δίκτυα 2

Ερώτημα 1: προβολή ΑΜ ή περιεχόμενα αρχείου 2

Ερώτημα 2: προβολή ονόματος, επωνύμου και ημ. γέννησης βάση id 2

Ερώτημα 3: προβολή διακριτών ονομάτων ή επωνύμων με αλφαβητική σειρά 3

Ερώτημα 4: προβολή γραμμών βάση ημερομηνίας . . . . . .

Ερώτημα 5: προβολή μέσων κοινωνικής δικτύωσης (αλφαβητικά) και αριθμός ατόμων που τα χρησιμοποίησαν . . . . . .

Μέρος 2ο

Ερώτημα Α: Αριθμητική έκφραση 5

i) Γράφος προτεραιοτήτων (precedence graph) 5

ii) Κώδικας (co)begin, (co)end για την ζητούμενη έκφραση 6

iii) Κώδικας με σημαφόρους για τον υπολογισμό της έκφρασης 7

Ερώτημα Β: Παράλληλη εκτέλεση δύο διεργασιών (Δ1, Δ2) 8

i) Αποτελεσμάτων παράλληλης εκτέλεσης των διεργασιών Δ1, Δ2 8

ii) Χρήση σημαφόρων για εκτύπωση επιθυμητού αποτελέσματος 12

Ερώτημα Γ: Παράλληλη εκτέλεση διεργασιών και αμοιβαίος αποκλεισμός 13

Μέρος 3ο

Ερώτημα Α: Φοιτητές – Βιβλιοθήκη, σημαφόροι 14

Ερώτημα Β: Βάση Δεδομένων βιβλιοθήκης 14

Μέρος 4ο: Διαγράμματα εκτέλεσης διεργασιών (Gantt), Μέσος Χρόνος Διεκπεραίωσης (ΜΧΔ) και Μέσος Χρόνος Αναμονής (ΜΧΑ) 15

i) FCFS (First Come First Served) 16

ii) SJF (Shortest Job First) 17

iii) SRTF (Shortest Remaining Time First) 18

iv) Priority Scheduling – μη προεκχωρητικός (non-preemptive priority) 19

v) RR (Round Robin) με κβάντο χρόνου 4 χρονικές μονάδες 20

Μέρος 1ο

```
#!/bin/bash
if [ ${#} -eq 0 ]; then
  echo "1054350-1054288"
  exit 1
fi

while [[ $# -gt 0 ]]; do
   key="$1"
   case "$key" in
      # Flag for surnames argument
      --firstnames)
            FIRSTNAMES=1
      ;;
      # Flag for last name argument
      --lastnames)
      LASTNAMES=1
      ;;
      # Pass filepath to FILE variable
      -f)
      shift # past the argument to get the value
      FILE="$1"
      ;;
      # Pass id to ID variable
      -id)
      shift
      ID="$1"
      ;;
            # Pass born since date to BORN_SINCE variable
      --born-since)
      shift
      BORN_SINCE="$1"
      ;;
            # Pass born until date to BORN_UNTIL variable
      --born-until)
      shift
      BORN_UNTIL="$1"
      ;;
            # Flag for social media argument
            --socialmedia)
      SOCIALMEDIA=1
      ;;
            # Pass id of the user,column to be replaced and the value replacing it to
their respectable variables
      --edit)
      shift
      ID="$1"
#           shift # To get second value
            COLUMN="$2"
#           shift # To get third value
            VALUE="$3"
            EDIT=1
      ;;
      *)
   esac
   # After checking all cases shift to get next option
   shift
```

```
done

while read line ; do
	[[ $line = \#* ]] && continue # Ignore lines starting with #
	LINEARRAY+=("$line") # Insert every line into array

done <$FILE
	: '
     For every line processed the elements of each column go to the variable with the appropriate name(ids,lastnames etc)
	Then this variable is put in the appropriate array(IDARRAY,LASTNAMESARRAY etc)
	'
while IFS=$'|' read -r ids lastnames firstnames genders birthdays joindates ips browsersused socialmedias
do
	[[ $ids = \#* ]] && continue
	IDARRAY+=("$ids")
	LASTNAMESARRAY+=("$lastnames")
	FIRSTNAMESARRAY+=("$firstnames")
	GENDERSARRAY+=("$genders")
	BIRTHDAYSARRAY+=("$birthdays")
	JOINDATESARRAY+=("$joindates")
	IPSARRAY+=("$ips")
	BROWSERUSEDARRAY+=("$browserused")
	SOCIALMEDIAARRAY+=("$socialmedias")
done <$FILE

if [[ -z $ID ]] && [[ -z $FIRSTNAMES ]] && [[ -z $LASTNAMES ]] && [[ -z $BORN_SINCE ]] && [[ -z $BORN_UNTIL ]] && [[ -z $SOCIALMEDIA ]] # If no other arguments given then only print every user
then
	for each in "${LINEARRAY[@]}"
	do
	echo "$each"
	done
#for ((index=0;index<${#IDARRAY[@]};++index));do
#	echo -e ${IDARRAY[index]}'\t'${LASTNAMESARRAY[index]}'\t'${FIRSTNAMESARRAY[index]}'\t'${GENDERSARRAY[index]}'\t'${BIRTHDAYSARRAY[index]}'\t'${JOINDATESARRAY[index]}'\t'${IPSARRAY[index]}'\t'${BROWSERUSEDARRAY[index]}'\t'${SOCIALMEDIAARRAY[index]}
#done

fi

if [[ $EDIT ]] && [[ -z $ID ]]
then
	for ((i=0;i< "${#IDARRAY[@]}";++i)); # Iterate through all IDs
	do
		if [[ $ID ==  ${IDARRAY[i]} ]] # If there is a match prints corresponding first name,last name and date of birth
		then
			echo -e ${FIRSTNAMESARRAY[i]}'\t'${LASTNAMESARRAY[i]}'\t'${BIRTHDAYSARRAY[i]}
		fi
	done
fi

if [[ $FIRSTNAMES ]]
then
```

```
	IFS=$'\n' sortedFirstnames=($(sort -u  <<<"${FIRSTNAMESARRAY[*]}")) # Create a new array with sorted firstnames
	for sortedfirstname in "${sortedFirstnames[@]}"
	do
		echo "$sortedfirstname"
	done
fi

if [[ $LASTNAMES ]]
then
	IFS=$'\n' sortedLastnames=($(sort -u  <<<"${LASTNAMESARRAY[*]}")) # Create a new array with sorted lastnames
	for sortedlastname in "${sortedLastnames[@]}"
	do
		echo "$sortedlastname"
	done
fi

if [[ $BORN_SINCE ]] || [[ $BORN_UNTIL ]]
then
	if [[ -z $BORN_UNTIL ]]
	then
		BORN_UNTIL=$(date +%Y-%m-%d) # If no BORN_UNTIL argument given set current day as BORN_UNTIL
	fi
	if [[ -z $BORN_SINCE ]]
	then
		BORN_SINCE="1800-01-01" # If no BORN_SINCE argument given then set a date earlier than every other possible date(no social media users existed before 1800)
	fi
	for ((i=0;i< "${#BIRTHDAYSARRAY[@]}";++i)); # Equality is checked separately because inside [[ ]] operators ">=" and "<=" can't be used
	do
		if [[  "${BIRTHDAYSARRAY[i]}" > $BORN_SINCE ]]  && [[ "${BIRTHDAYSARRAY[i]}" < $BORN_UNTIL ]]  || [[  "${BIRTHDAYSARRAY[i]}" == $BORN_SINCE ]] || [[  "${BIRTHDAYSARRAY[i]}" == $BORN_UNTIL ]]
		then
			echo "${LINEARRAY[i]}"
		fi
	done

fi

if [[ $EDIT ]]
then
idcheck="$(grep -c $ID $FILE)"
if [[ $idcheck -eq 0 ]] || [[ $COLUMN -lt 2 ]] || [[ $COLUMN -gt 8 ]];then  # If given id doesn't exist or wrong column number exit
	echo "Wrong parametres"
	exit 0
else
	#For every line starting with the id replaces value of the desired column with the new value and creates a backup of the original file called (original filename).bak
	sed -i.bak -e  "/^$ID|/s/[^|]*/$VALUE/$COLUMN" $FILE

fi
fi
```

Μέρος 2ο

Ερώτημα Α

Μας δίνεται η αριθμητική έκφραση: Y = G + X * (C + D) / ((E - F) * (H + I)) , η οποία υπολογίζεται από τον παρακάτω κώδικα, που εκτελεί ακολουθιακά τις εξής 7 εντολές:

I1: Y11 = C + D;
I2: Y12 = E - F;
I3: Y13 = H + I;
I4: Y21 = X * Y11;
I5: Y22 = Y12 * Y13;
I6: Y31 = Y21 / Y22;
I7: Y = G + Y31;

i) Είναι γνωστό από τα μαθηματικά ότι η προτεραιότητα των πράξεων είναι η εξής: πρώτα εκφράσεις μέσα σε παρενθέσεις, μετά δυνάμεις και ρίζες, έπειτα πολλαπλασιασμοί και διαιρέσεις (με τη σειρά που εμφανίζονται στην έκφραση, αριστερά προς δεξιά) και τέλος προσθέσεις και αφαιρέσεις (με τη σειρά που εμφανίζονται στην έκφραση, αριστερά προς δεξιά).

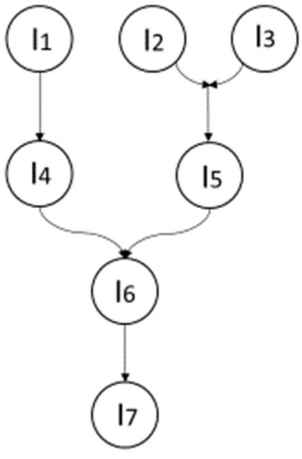
Έτσι, με βάση τις προτεραιότητες των πράξεων τις εντολές (I1; I2; ….; I7;) και τις ενδιάμεσες μεταβλητές Y11, Y12, Y13, Y21, Y22, Y31 για την έκφραση Y προκύπτει:

$$Y = G+X*\frac{(C+D)}{(E-F)*(H+I)} \xrightarrow{I1,I2,I3} G+X*\frac{Y11}{Y12*Y13} \xrightarrow{I4,I5} G+\frac{Y21}{Y22} \xrightarrow{I6} G+Y31 \xrightarrow{I7} Y$$

ή ακόμα καλύτερα

$$Y = G+X*\frac{(C+D)}{(E-F)*(H+I)} \xrightarrow{I1,I2,I3} G+X*\frac{Y11}{Y12*Y13} \xrightarrow{I4,I5,I6} G+Y31 \xrightarrow{I7} Y$$

Έτσι κατανοούμε ότι οι εντολές (I2, I3) μπορούν να εκτελεστούν παράλληλα και μετά να ακολουθήσει η εκτέλεση της εντολής I5, η I4 μπορεί να εκτελεστεί μόλις ολοκληρωθεί η εκτέλεση της εντολής I1, ενώ συνολικά μπορούν να εκτελεστούν παράλληλα οι εντολές [(I1, I4), (I3, I5, I6)]· η I6 μπορεί να εκτελεστεί μετά την ολοκλήρωση των εντολών I4 και I5 και μετά από αυτή να εκτελεστεί και η I7. Με βάση τα παραπάνω προκύπτει ο παρακάτω γράφος προτεραιοτήτων:



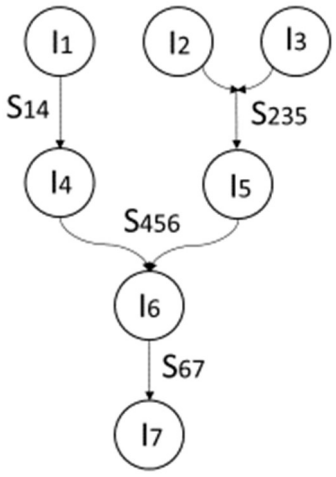
ii) Ο κώδικας που υπολογίζει με τη μέγιστη δυνατή παραλληλία την ανωτέρω έκφραση, χωρίς την χρήση σημαφόρων και εντολών P (ή wait ή down) / V (ή signal ή up) είναι ο ακόλουθος (βλ πλαίσιο *«Παράλληλος κώδικας χωρίς σημαφόρους»*):

```
cobegin
   begin
      I1;
      I4;
   end;
   begin
      cobegin
         I2;
         I3;
      coend;
      I5;
   end;
coend;
   I6;
   I7;
```

ή

```
cobegin
   begin
      Y11 = C + D;
      Y21 = X * Y11;
   end;
   begin
      cobegin
         Y12 = E - F;
         Y13 = H + I;
      coend;
      Y22 = Y12 * Y13;
   end;
coend;
   Y31 = Y21 / Y22;
   Y = G + Y31;
```

iii) Ένας σημαφόρος είναι μια ειδική μεταβλητή που ανήκει και ελέγχεται από το Λειτουργικό Σύστημα και δημιουργεί τα σήματα συγχρονισμού. Δύο ή περισσότερες διεργασίες μπορούν να συγχρονιστούν με τη χρήση απλών σημάτων, οπότε μια διεργασία σταματά σε μια συγκεκριμένη θέση την εκτέλεσή της και ξεκινά μόνο όταν λάβει ένα σήμα αφύπνισης. Σε έναν σημαφόρο s μπορούμε να κάνουμε down(s) (η διεργασία που εκτελεί το down παραλαμβάνει ένα σήμα μέσω του σημαφόρου s), μειώνοντας την τιμή του σημαφόρου κατά 1 και αν γίνει αρνητική, τότε η διεργασία που εκτελεί τη down αναστέλλεται ή up(s) (οπότε η διεργασία που εκτελεί το up στέλνει ένα σήμα μέσω του σημαφόρου s), αυξάνοντας κατά 1 την τιμή του σημαφόρου και αν είναι θετική, τότε μια διεργασία που είχε περάσει σε κατάσταση αναστολής (εξαιτίας της εκτέλεσης μιας εντολής down) αφυπνίζεται.



Στην περίπτωσή μας, θέλουμε να εκτελέσουμε παράλληλα κάποιες εντολές, οι οποίες υπολογίζουν μέρη μια αριθμητικής παράστασης και επόμενο είναι να μοιράζονται κάποιες μεταβλητές ή να προηγείται μια εντολή από κάποια άλλη. Αρκεί να χρησιμοποιήσουμε 4 σημαφόρους, S_{14}, S_{235}, S_{456}, S_{67} όπως φαίνονται και στο γράφο προτεραιότητας. Ο «παράλληλος» κώδικας, ο οποίος υπολογίζει την ανωτέρω έκφραση, χρησιμοποιώντας την εντολή cobegin, coend, καθώς και εντολές P (wait ή down) και V (signal ή up) σε σημαφόρους είναι ο ακόλουθος:

```
var S14, S235, S456, S67: semaphores;
S14 = S67 = 0;
S235 = S456 = -1;
Y, X, C, D, E, F, G, H, I, Y11, Y12, Y13, Y21, Y22: shared integer;
Y31 shared float;
cobegin
    I1: begin
            Y11 = C + D;
            up(S14);
        end;
    I2: begin
            Y12 = E - F;
            up(S235);
        end;
    I3: begin
            Y13 = H + I;
            up(S235);
        end;
    I4: begin
            down(S14);
            Y21 = X * Y11;
            Up(S456);
        End;
    I5: begin
            Down(S235);
            Y22 = Y12 * Y13;
            Up(S456);
        end;
    I6: begin
            down(S456);
            Y31 = Y21 / Y22;
            up(S67);
        end;
    I7: begin
            down(S67);
            Y = G + Y31;
        end;
coend;
```

Ερώτημα Β

i) Για τις διεργασίες Δ1 και Δ2, που εκτελούνται «παράλληλα» (cobegin Δ1; Δ2; coend), θεωρούμε ότι κάθε εντολή εκχώρησης τιμής εκτελείται αδιαίρετα και ότι κάθε εντολή if εκτελείται σε 2 βήματα (1ο: έλεγχος συνθήκη if, 2ο: με ικανοποίηση της συνθήκη της if εκτέλεσης της εντολής print. Θεωρούμε ότι η εντολή if μπορεί να διασπαστεί στα δύο βήματα εκτέλεσής της· πιο αναλυτικά, γίνεται έλεγχος της συνθήκης και μπορεί να διακοπεί η εκτέλεσή της, ενώ όταν επανέλθει η ροή σε αυτή την διεργασία, αν η συνθήκη ήταν αληθής θα εκτελεστεί η εντολή print, ενώ σε διαφορετική περίπτωση, που η συνθήκη ήταν ψευδής, δεν θα εκτελεστεί. Οι διεργασίες χρησιμοποιούν δύο κοινά διαμοιραζόμενες ακέραιες μεταβλητές (τις x και y).

Αρίθμηση εντολών

```
shared int x, y;
cobegin
   Δ1: begin
1     x = 1;
2     y = 2;
3     if (x == y)
4        print "X";
   end;
   Δ2: begin
5     x = 2;
6     y = 2;
7     if (x == y)
8        print "Y";
   end;
coend;
```

Ακολουθούν όλα τα τελικά αποτελέσματα που είναι δυνατό να εμφανιστούν μετά το πέρας της παράλληλης εκτέλεσης των διεργασιών Δ1 και Δ2. Αρχικά παραθέτουμε κάθε πιθανή σειρά με την οποία μπορούν να εκτελεστούν οι εντολές των διεργασιών, λαμβάνοντας υπ' όψιν τους περιορισμούς που υπάρχουν, δηλαδή ότι κάθε διεργασία εκτελείται σειριακά («από πάνω προς τα κάτω»), ενώ οι δύο διεργασίες εκτελούνται παράλληλα.

Εκκίνηση από την πρώτη διεργασία (Δ1)

1 εντολή από Δ1 πρώτα				2 εντολές από Δ1 πρώτα				3 εντολές από Δ1 πρώτα				
1	1	1	1	1	1	1	1	1	1	1	1	1
5	5	5	5	2	2	2	2	2	2	2	2	2
2	6	6	6	5	5	5	5	3	3	3	3	3
3	2	7	7	3	6	6	6	5	5	5	5	4
4	3	2	8	4	3	7	7	4	6	6	6	5
6	4	3	2	6	4	3	8	6	4	7	7	6
7	7	4	3	7	7	4	3	7	7	4	8	7
8	8	8	4	8	8	8	4	8	8	8	4	8

Εκκίνηση από την δεύτερη διεργασία (Δ2)

1 εντολή από Δ2 πρώτα				2 εντολές από Δ2 πρώτα				3 εντολές από Δ1 πρώτα				
5	5	5	5	5	5	5	5	5	5	5	5	5
1	1	1	1	6	6	6	6	6	6	6	6	6
6	2	2	2	1	1	1	1	7	7	7	7	7
7	6	3	3	7	2	2	2	1	1	1	1	8
8	7	6	4	8	7	3	3	8	2	2	2	1
2	8	7	6	2	8	7	4	2	8	3	3	2
3	3	8	7	3	3	8	7	3	3	8	4	3
4	4	4	8	4	4	4	8	4	4	4	8	4

Κατά την εκτέλεση των διεργασιών αναμένουμε με την εκτέλεση της Δ1 να μην εκτυπώνεται κάτι, καθώς x=1 ≠ y=2, ενώ με την εκτέλεση της Δ2 αναμένουμε να εκτυπώνεται η τιμή 2 καθώς x=y=2. Όμως λόγω της παράλληλης εκτέλεσης των διεργασιών χωρίς την χρήση σημαφόρων υπάρχουν διαφοροποιήσεις, λόγω των διαμοιραζομένων μεταβλητών.
Παρακάτω παραθέτουμε αναλυτικά όλα τα παραπάνω σενάρια εκτέλεσης των εντολών ομαδοποιημένα.

Εκκίνηση από την πρώτη διεργασία (Δ1)

Εκτελείται η πρώτη εντολή της Δ1:

command	var x	var y	if	print
1 x=1	1	-		
5 x=2	2	-		
2 y=2	2	2		
3 if(x)	2	2	true	
4 printx	2	2		2
6 y=2	2	2		
7 if(y)	2	2	true	
8 printy	2	2		2

command	var x	var y	if	print
1 x=1	1	-		
5 x=2	2	-		
6 y=2	2	2		
2 y=2	2	2		
3 if(x)	2	2	true	
4 printx	2	2		2
7 if(y)	2	2	true	
8 printy	2	2		2

command	var x	var y	if	print
1 x=1	1	-		
5 x=2	2	-		
6 y=2	2	2		
7 if(y)	2	2	true	
2 y=2	2	2		
3 if(x)	2	2	true	
4 printx	2	2		2
8 printy	2	2		2

command	var x	var y	if	print
1 x=1	1	-		
5 x=2	2	-		
6 y=2	2	2		
7 if(y)	2	2	true	
8 printy	2	2		2
2 y=2	2	2		
3 if(x)	2	2	true	
4 printx	2	2		2

Παρατηρούμε ότι δεν πραγματοποιείται ορθή εκτέλεση των διεργασιών, καθώς η συνθήκη της εντολής 3, λανθασμένα, αληθεύει, εκτυπώνοντας την τιμή του x.

Εκτελούνται οι δύο πρώτες εντολές της Δ1:

command	var x	var y	if	print
1 x=1	1	-		
2 y=2	1	2		
5 x=2	2	2		
3 if(x)	2	2	true	
4 printx	2	2		2
6 y=2	2	2		
7 if(y)	2	2	true	
8 printy	2	2		2

command	var x	var y	if	print
1 x=1	1	-		
2 y=2	1	2		
5 x=2	2	2		
6 y=2	2	2		
3 if(x)	2	2	true	
4 printx	2	2		2
7 if(y)	2	2	true	
8 printy	2	2		2

command	var x	var y	if	print
1 x=1	1	-		
2 y=2	1	2		
5 x=2	2	2		
6 y=2	2	2		
7 if(y)	2	2	true	
3 if(x)	2	2	true	
4 printx	2	2		2
8 printy	2	2		2

command	var x	var y	if	print
1 x=1	1	-		
2 y=2	1	2		
5 x=2	2	2		
6 y=2	2	2		
7 if(y)	2	2	true	
8 printy	2	2		2
3 if(x)	2	2	true	
4 printx	2	2		2

Παρατηρούμε ότι δεν πραγματοποιείται ορθή εκτέλεση των διεργασιών, καθώς η συνθήκη της εντολής 3, λανθασμένα, αληθεύει, εκτυπώνοντας την τιμή του x.

Εκτελούνται οι τρεις πρώτες εντολές της Δ1 ή ολόκληρη η Δ1:

command	var x	var y	if	print
1 x=1	1	-		
2 y=2	1	2		
3 if(x)	1	2	false	
5 x=2	2	2		
4 printx	2	2		-
6 y=2	2	2		
7 if(y)	2	2	true	
8 printy	2	2		2

command	var x	var y	If	print
1 x=1	1	-		
2 y=2	1	2		
3 if(x)	1	2	false	
5 x=2	2	2		
6 y=2	2	2		
4 printx	2	2		-
7 if(y)	2	2	true	
8 printy	2	2		2

command	var x	var y	if	print
1 x=1	1	-		
2 y=2	1	2		
3 if(x)	1	2	false	
5 x=2	2	2		
6 y=2	2	2		
7 if(y)	2	2	true	
4 printx	2	2		-
8 printy	2	2		2

command	var x	var y	if	print
1 x=1	1	-		
2 y=2	1	2		
3 if(x)	1	2	false	
5 x=2	2	2		
6 y=2	2	2		
7 if(y)	2	2	true	
8 printy	2	2		2
4 printx	2	2		-

command	var x	var y	if	print
1 x=1	1	-		
2 y=2	1	2		
3 if(x)	1	2	false	
4 printx	1	2		-
5 x=2	2	2		
6 y=2	2	2		
7 if(y)	2	2	true	
8 printy	2	2		2

Παρατηρούμε ότι, παρά την παράλληλη εκτέλεση των διεργασιών, πραγματοποιείται ορθή εκτέλεση, εμφανίζοντας τα επιθυμητά αποτελέσματα. Προφανώς, επιθυμητό αποτέλεσμα εμφανίζεται και όταν εκτελείται πρώτα ολόκληρη η Δ1.

Εκκίνηση από την δεύτερη διεργασία (Δ2)

Εκτελείται η πρώτη εντολή της Δ1:

command	var x	var y	if	print
5 x=2	2	-		
1 x=1	1	-		
6 y=2	1	2		
7 if(y)	1	2	false	
8 printy	1	2		-
2 y=2	1	2		
3 if(x)	1	2	false	
4 printx	1	2		-

command	var x	var y	if	print
5 x=2	2	-		
1 x=1	1	-		
2 y=2	1	2		
6 y=2	1	2		
7 if(y)	1	2	false	
8 printy	1	2		-
3 if(x)	1	2	false	
4 printx	1	2		-

command	var x	var y	if	print
5 x=2	2	-		
1 x=1	1	-		
2 y=2	1	2		
3 if(x)	1	2	false	
6 y=2	1	2		
7 if(y)	1	2	false	
8 printy	1	2		-
4 printx	1	2		-

command	var x	var y	if	print
5 x=2	2	-		
1 x=1	1	-		
2 y=2	1	2		
3 if(x)	1	2	false	
4 printx	1	2		-
6 y=2	1	2		
7 if(y)	1	2	false	
8 printy	1	2		-

Παρατηρούμε ότι δεν πραγματοποιείται ορθή εκτέλεση των διεργασιών, καθώς η συνθήκη της εντολής 7, λανθασμένα, δεν αληθεύει, μη εκτυπώνοντας την τιμή του y.

Εκτελούνται οι δύο πρώτες εντολές της Δ2:

command	var x	var y	if	print
5 x=2	2	-		
6 y=2	2	2		
1 x=1	1	2		
7 if(y)	1	2	false	
8 printy	1	2		-
2 y=2	1	2		
3 if(x)	1	2	false	
4 printx	1	2		-

command	var x	var y	if	print
5 x=2	2	-		
6 y=2	2	2		
1 x=1	1	2		
2 y=2	1	2		
7 if(y)	1	2	false	
8 printy	1	2		-
3 if(x)	1	2	false	
4 printx	1	2		-

command	var x	var y	if	print
5 x=2	2	-		
6 y=2	2	2		
1 x=1	1	2		
2 y=2	1	2		
3 if(x)	1	2	false	
7 if(y)	1	2	false	
8 printy	1	2		-
4 printx	1	2		-

command	var x	var y	if	print
5 x=2	2	-		
6 y=2	2	2		
1 x=1	1	2		
2 y=2	1	2		
3 if(x)	1	2	false	
4 printx	1	2		-
7 if(y)	1	2	false	
8 printy	1	2		-

Παρατηρούμε ότι δεν πραγματοποιείται ορθή εκτέλεση των διεργασιών, καθώς η συνθήκη της εντολής 7, λανθασμένα, δεν αληθεύει, μη εκτυπώνοντας την τιμή του y.

Εκτελούνται οι τρεις πρώτες εντολές της Δ1 ή ολόκληρη η Δ1:

command	var x	var y	if	print
5 x=2	2	-		
6 y=2	2	2		
7 if(y)	2	2	true	
1 x=1	1	2		
8 printy	1	2		2
2 y=2	1	2		
3 if(x)	1	2	false	
4 printx	1	2		-

command	var x	var y	if	print
5 x=2	2	-		
6 y=2	2	2		
7 if(y)	2	2	true	
1 x=1	1	2		
2 y=2	1	2		
8 printy	1	2		2
3 if(x)	1	2	false	
4 printx	1	2		-

command	var x	var y	if	print
5 x=2	2	-		
6 y=2	2	2		
7 if(y)	2	2	true	
1 x=1	1	2		
2 y=2	1	2		
3 if(x)	1	2	false	
8 printy	1	2		2
4 printx	1	2		-

command	var x	var y	if	print
5 x=2	2	-		
6 y=2	2	2		
7 if(y)	2	2	true	
1 x=1	1	2		
2 y=2	1	2		
3 if(x)	1	2	false	
4 printx	1	2		-
8 printy	1	2		2

command	var x	var y	if	print
5 $_{x=2}$	2	-		
6 $_{y=2}$	2	2		
7 $_{if(y)}$	2	2	true	
8 $_{printy}$	2	2		2
1 $_{x=1}$	1	2		
2 $_{y=2}$	1	2		
3 $_{if(x)}$	1	2	false	
4 $_{printx}$	1	2		-

Παρατηρούμε ότι, παρά την παράλληλη εκτέλεση των διεργασιών, πραγματοποιείται ορθή εκτέλεση, εμφανίζοντας τα επιθυμητά αποτελέσματα. Προφανώς, επιθυμητό αποτέλεσμα εμφανίζεται και όταν εκτελείται πρώτα ολόκληρη η Δ2.

ii) Για να εξασφαλίσουμε ότι μετά το πέρας της εκτέλεσης των διεργασιών Δ1 και Δ2 θα εμφανίζεται πάντοτε το αποτέλεσμα «XY» είναι απαραίτητο να χρησιμοποιήσουμε σημαφόρους και εντολές signal (up ή V) και wait (down ή P). Συνεπώς θα πρέπει να εξασφαλίσουμε ότι θα εκτελείται πάντοτε πρώτα η Δ1 και μετά η Δ2. Έτσι προκύπτει:

```
var s1, s2: semaphores;

begin
   s1=1;
   s2=0;
   shared int x, y;
   x:=0;
   y:=0;
   cobegin

      Δ1: begin
          down(s1);
          x = 1;
          y = 2;
          if (x == y)
             print "X";
          up(s2);
       end;

      Δ2: begin
          down(s2);
          x = 2;
          y = 2;
          if (x == y)
              print "Y";
      end;

   coend;
end
```

Ερώτημα Γ

Οι διεργασίες Δ1 και Δ2 εκτελούνται παράλληλα (cobegin Δ0; Δ1; coend). Παρακάτω εξετάζουμε αν ο κώδικας των διεργασιών, όπως δίνεται, εξασφαλίζει (ή όχι) αμοιβαίο αποκλεισμό για τις διεργασίες Δ0 και Δ1, δηλαδή εάν υπάρχει ή δεν υπάρχει περίπτωση οι δύο διεργασίες να εκτελούν ταυτόχρονα το τμήμα εντολών <ΚΡΙΣΙΜΟ ΤΜΗΜΑ> του κώδικά τους.

```
shared boolean flag[2];  /* αρχικά flag[0]=flag[1]=FALSE*/
shared turn[2];          /* αρχικά turn[0]=0 και turn[1]=0*/
Διεργασία Δ0
    flag[0]=TRUE;
    turn[0]=(turn[1]+0)mod2;
    repeat noop
    until (flag[1]==FALSE)OR(turn[0]<>(turn[1]+0)mod2);
    <ΚΡΙΣΙΜΟ ΤΜΗΜΑ>
    flag[0]= FALSE;
Διεργασία Δ1
    flag[1]=TRUE;
    turn[1]=(turn[0]+1)mod2;
    repeat noop
    until (flag[0]==FALSE)OR(turn[1]<>(turn[0]+1)mod2);
    <ΚΡΙΣΙΜΟ ΤΜΗΜΑ>
    flag[1]= FALSE;
```

Φαίνεται πως οι διεργασίες δεν μπαίνουν ταυτόχρονα στην κρίσιμη περιοχή, δηλαδή επιτυγχάνεται ο αμοιβαίος αποκλεισμός. Ο τρόπος που επιτυγχάνεται ο αμοιβαίος αποκλεισμός φαίνεται να είναι όμοιος, αλλά όχι ίδιος με την λύση που δίνει ο Peterson. Πιο συγκεκριμένα η λύση του Peterson για τον αμοιβαίο αποκλεισμό είναι:

```
shared boolean flag[2];     /* αρχικά FALSE */
shared in turn;             /* αρχικά 0 ή 1 */

Διεργασία Δ0
   flag[0] = TRUE;
   turn = 0;
   while (turn == 0 AND flag[1] == TRUE) do noop;
   <κρίσιμο τμήμα>;
   flag[0] = FALSE;
Διεργασία Δ1
   flag[1] = TRUE;
   turn = 1;
   while (turn == 1 AND flag[0] == TRUE) do noop;
   <κρίσιμο τμήμα>;
   flag[1] = FALSE;
```

Παρατηρούμε ότι ισχύει ο Κανόνας De Morgan με την συμπληρωματικότητα, καθώς υπάρχουν οι παρακάτω αντιστοιχίσεις στην εντολή για να μπει η διεργασία στην κρίσιμη περιοχή:

until	while
AND	OR
==	<>
repeat	do
FALSE	TRUE

Στον κώδικα που μας δόθηκε για εξέταση ισχύει (turn[1]+0)mod2 = (turn[1]+0)mod2 = 0, από βασικές πράξεις άλγεβρας (1mod2=0) και λόγω εκφώνησης (turn[0]=turn[1]=0).

Συνεπώς καταλήγουμε στο συμπέρασμα ότι ο δοθέν κώδικας με την λύση του Peterson είναι ισοδύναμοι, άρα ο δοθέν κώδικας εξασφαλίζει αμοιβαίο αποκλεισμό για τις διεργασίες Δ0 και Δ1, δηλαδή δεν υπάρχει περίπτωση οι δύο διεργασίες να εκτελούν ταυτόχρονα το τμήμα εντολών <ΚΡΙΣΙΜΟ ΤΜΗΜΑ> του κώδικά τους.

Μέρος 3ο

Ερώτημα Α

```
/**

Αν πολλα processes εκτελεσουν wait(condition) τοτε η signal(condition) ξυπναει μονο 1 εξ'αυτών.Επομένως επειδή για κάθε condition υπάρχουν 2 wait και θέλω να ξυπνάνε όλα κάθε signal εντολή υπάρχει 2 φορές

/

monitor study
condition student1_ready,student2_ready,student3_ready;
Student_1{
        Search_Book();
        signal(student1_ready);
        signal(student1_ready);
        wait(student2_ready);
        wait(student3_ready);
        Study_Project();
}
Student_2{
        Search_Book();
        signal(student2_ready);
        signal(student2_ready);
        wait(student1_ready);
        wait(student3_ready);
        Study_Project();
}
Student_3{
        Search_Book();
        signal(student3_ready);
        signal(student3_ready);
        wait(student1_ready);
        wait(student2_ready);
        Study_Project();
}
```

Ερώτημα Β

```
binary semaphore insertion=1;
Insertion
while(TRUE){
        down(insertion);
        if(count==0) down(library);
        Insert_Book();
        up(insertion);
        up(library);
}
```

Μέρος 4ο

Στο υπολογιστικό σύστημα που μελετούμε καταφθάνουν πέντε διεργασίες για τις οποίες γνωρίζουμε τα ακόλουθα δεδομένα:

Όνομα Διεργασίας	**Χρονική Στιγμή Άφιξης**	**Απαιτήσεις Χρόνου Εκτέλεσης**	**Προτεραιότητα**
Α	0	7	2
Β	3	11	3
Γ	9	5	1
Δ	12	14	5
Ε	14	4	4

Ως Χρόνος Διεκπεραίωσης μιας διεργασίας ορίζεται το χρονικό διάστημα που καταναλώνει από τη χρονική στιγμή εισόδου της στο σύστημα μέχρι την ολοκλήρωση της εκτέλεσής της από την CPU.

Επομένως εάν είναι t_1 η χρονική στιγμή εισόδου και t_2 η χρονική στιγμή εξόδου και t_{CPU} είναι ο χρόνος που χρειάζεται η διεργασία για εξυπηρέτηση από τη CPU τότε:

$$\text{Χρόνος Διεκπεραίωσης ΧΔ} = t_2 - t_1$$

και Μέσος Χρόνος Διεκπεραίωσης (**ΜΧΔ**) θα είναι:

$$\text{ΜΧΔ} = (\text{ΧΔ}_1 + \text{ΧΔ}_2 + \text{ΧΔ}_3 + \text{ΧΔ}_4 + \text{ΧΔ}_5) / 5$$

Ως Χρόνος Αναμονής μιας διεργασίας ορίζεται το χρονικό διάστημα που καταναλώνει αναμένοντας στο σύστημα, χωρίς να πραγματοποιείται εξυπηρέτησή της από την CPU.

Επομένως εάν είναι t_1 η χρονική στιγμή εισόδου και t_2 η χρονική στιγμή εξόδου και t_{CPU} είναι ο χρόνος που χρειάζεται η διεργασία για εξυπηρέτηση από τη CPU τότε:

$$\text{Χρόνος Αναμονής ΧΑ} = t_2 - t_1 - t_{CPU}$$

και ο Μέσος Χρόνος Αναμονής (**ΜΧΑ**) θα είναι:

$$\text{ΜΧΑ} = (\text{ΧΑ}_1 + \text{ΧΑ}_2 + \text{ΧΑ}_3 + \text{ΧΑ}_4 + \text{ΧΑ}_5) / 5$$

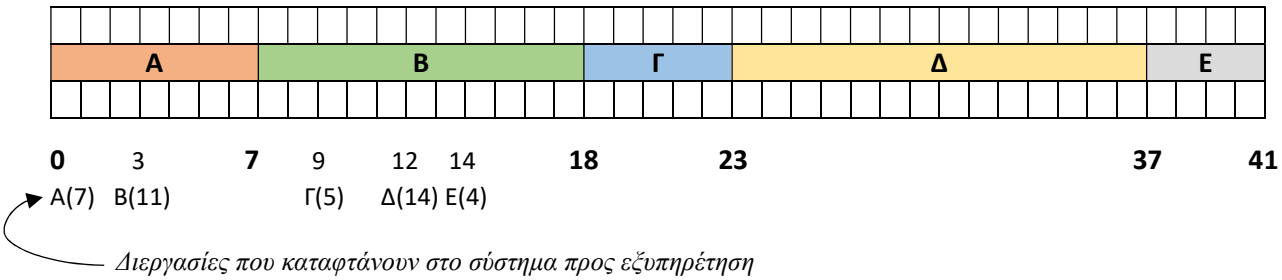
Θεωρούμε ότι ο χρόνος εναλλαγής θέματος (context switch) είναι αμελητέος.

Παρακάτω ακολουθούν διαγράμματα εκτέλεσης των διεργασιών (Gantt) και οι αντίστοιχοι μέσοι χρόνοι διεκπεραίωσης (ΜΧΔ) και μέσοι χρόνοι αναμονής (ΜΧΑ), για καθέναν από τους εξής αλγόριθμους χρονοδρομολόγησης:

- FCFS (First Come First Served)
- SJF (Shortest Job First)
- SRTF (Shortest Remaining Time First)
- Priority Scheduling – μη προεκχωρητικός (non-preemptive priority)
- (Round Robin) με κβάντο χρόνου 4 χρονικές μονάδες

i) FCFS (First Come First Served)

Ο μη προεκτοπιστικός αλγόριθμος εξυπηρέτησης με βάση τη σειρά άφιξης καθορίζει ότι στις διεργασίες ανατίθεται η CPU με τη σειρά που της ζήτησαν. Υπάρχει μία μοναδική ουρά FIFO με τις έτοιμες διεργασίες. Όταν η πρώτη διεργασία εισέρχεται στο σύστημα ξεκινάει αμέσως και εκτελείται για όσο διάστημα θέλει, μέχρι να ανασταλεί από μόνη της. Καθώς καταφθάνουν και άλλες διεργασίες, τοποθετούνται αυτόματα στο τέλος της ουράς. Όταν η εκτελούμενη διεργασία ολοκληρωθεί εκτελείται η διεργασία που είναι η επόμενη στην ουρά.
Στις παρενθέσεις αναφέρονται οι χρονικές μονάδες που απαιτεί ή απομένουν για να εκτελεστεί η αντίστοιχη διεργασία.



Η εκτέλεση των διεργασιών θα ολοκληρωθεί την χρονική στιγμή 41.
Για κάθε διεργασία ο Χρόνος Διεκπεραίωσής της θα είναι:

$ΧΔ_Α = 7 – 0 = 7$
$ΧΔ_Β = 18 – 3 = 15$
$ΧΔ_Γ = 23 – 9 = 14$
$ΧΔ_Δ = 37 – 12 = 25$
$ΧΔ_Ε = 41 – 14 = 27$

Και ο Μέσος Χρόνος Διεκπεραίωσης: ΜΧΔ = (7 + 15 + 14 + 25 + 27) / 5 = 17,6 χρονικές μονάδες

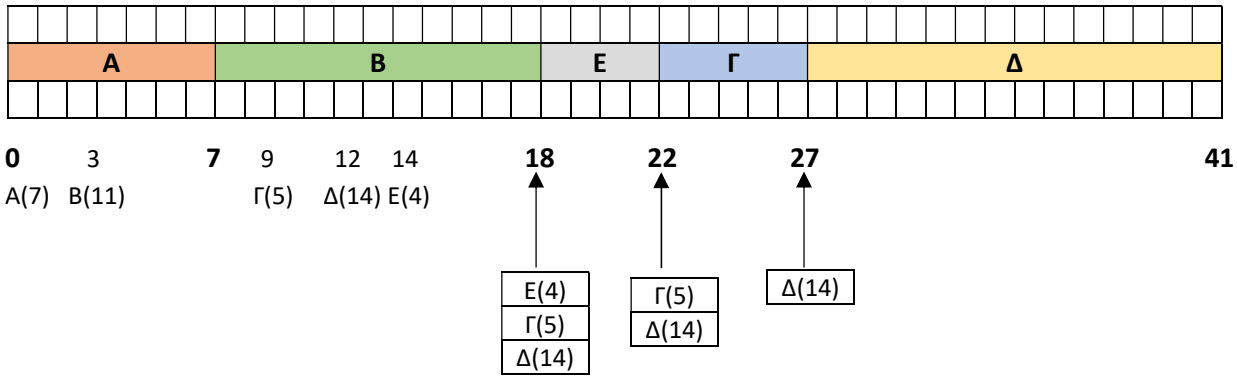
Για κάθε διεργασία ο Χρόνος Αναμονής της στο σύστημα θα είναι:

$ΧΑ_Α = ΧΑ_Α – t_{CPU} = 7 – 7 = 0$
$ΧΑ_Β = ΧΑ_Β – t_{CPU} = 15 – 11 = 4$
$ΧΑ_Γ = ΧΑ_Γ – t_{CPU} = 14 – 5 = 9$
$ΧΑ_Δ = ΧΑ_Δ – t_{CPU} = 25 – 14 = 11$
$ΧΑ_Ε = ΧΑ_Ε – t_{CPU} = 27 – 4 = 23$

Και ο Μέσος Χρόνος Αναμονής: ΜΧΑ = (4 + 9 + 11 + 23) / 5 = 9,4 χρονικές μονάδες

ii) SJF (Shortest Job First)

Αυτός ο αλγόριθμος χρονοπρογραμματισμού υποθέτει ότι οι χρόνοι που χρειάζονται οι διεργασίες για να ολοκληρωθούν είναι γνωστοί προκαταβολικά. Οι διάφορες διεργασίες περιμένουν στην ουρά εισόδου για να ξεκινήσουν και ο χρονοπρογραμματιστής επιλέγει πρώτη την διεργασία με τη μικρότερη διάρκεια. Στην περίπτωση αυτή, όταν το σύστημα αρχίζει να εξυπηρετεί τις διεργασίες, η μοναδική διαθέσιμη διεργασία προς εξυπηρέτηση είναι η Α, παρ' όλο που δεν είναι η μικρότερη χρονικά, ενώ όταν ολοκληρωθεί η εκτέλεσή της, η μοναδική διαθέσιμη διεργασία προς εξυπηρέτηση είναι η Β, η οποία, επίσης, δεν είναι η μικρότερη χρονικά (Ε(4) < Γ(5) < Α(7) < Β(11) < Δ(14)). Κατά την εκτέλεση της διεργασίας Β καταφτάνουν και οι υπόλοιπες διεργασίες και εκτελούνται με βάση την μικρότερη διάρκεια εκτέλεσης, όπως φαίνεται παρακάτω. Στις παρενθέσεις αναφέρονται οι χρονικές μονάδες που απαιτεί ή απομένουν για να εκτελεστεί η αντίστοιχη διεργασία.



Η εκτέλεση των διεργασιών θα ολοκληρωθεί την χρονική στιγμή 41.
Για κάθε διεργασία ο Χρόνος Διεκπεραίωσής της θα είναι:

$ΧΔ_Α = 7 – 0 = 7$
$ΧΔ_Β = 18 – 3 = 15$
$ΧΔ_Γ = 27 – 9 = 18$
$ΧΔ_Δ = 41 – 12 = 29$
$ΧΔ_Ε = 22 – 14 = 8$

Και ο Μέσος Χρόνος Διεκπεραίωσης: ΜΧΔ = (7 + 15 + 18 + 29 + 8) / 5 = 15,4 χρονικές μονάδες

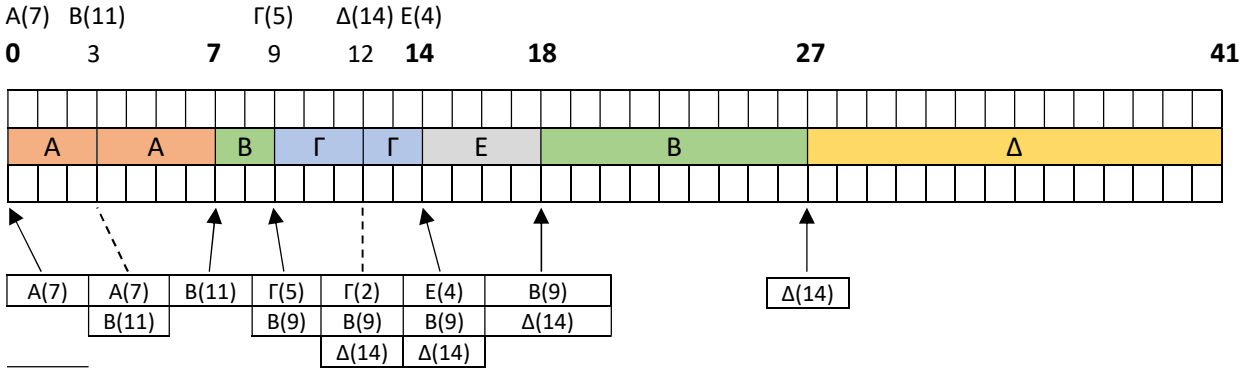
Για κάθε διεργασία ο Χρόνος Αναμονής της στο σύστημα θα είναι:

$ΧΑ_Α = ΧΑ_Α – t_{CPU} = 7 – 7 = 0$
$ΧΑ_Β = ΧΑ_Β – t_{CPU} = 15 – 11 = 4$
$ΧΑ_Γ = ΧΑ_Γ – t_{CPU} = 18 – 5 = 13$
$ΧΑ_Δ = ΧΑ_Δ – t_{CPU} = 29 – 14 = 15$
$ΧΑ_Ε = ΧΑ_Ε – t_{CPU} = 8 – 4 = 4$

Και ο Μέσος Χρόνος Αναμονής: ΜΧΑ = (4 + 13 + 15 + 4) / 5 = 7,2 χρονικές μονάδες

iii) SRTF (Shortest Remaining Time First)

Αυτός ο αλγόριθμος χρονοπρογραμματισμού υποθέτει ότι οι χρόνοι που χρειάζονται οι διεργασίες για να ολοκληρωθούν είναι γνωστοί προκαταβολικά και επιλέγει να εξυπηρετήσει κάθε φορά την διεργασία με το μικρότερο εναπομείναντα χρόνο, δηλαδή τον συνολικό χρόνο καταιγισμού μείον το χρόνο που η διεργασία παρέμεινε προς εκτέλεση στη CPU. Όταν στο σύστημα φθάσει μια διεργασία με μικρότερο χρόνο καταιγισμού από τον υπολειπόμενο χρόνο καταιγισμού της τρέχουσας διεργασίας, τότε η τρέχουσα διεργασία διακόπτεται. Εξετάζουμε ποια διεργασία θα εξυπηρετηθεί κάθε φορά που μια διεργασία έχει ολοκληρώσει το χρόνο καταιγισμού της στη CPU ή μια νέα διεργασία φθάνει στην ουρά των έτοιμων διεργασιών. Στις παρενθέσεις αναφέρονται οι χρονικές μονάδες που απαιτεί ή απομένουν για να εκτελεστεί η αντίστοιχη διεργασία.



------: κατά την άφιξη διεργασίας

⟶ : κατά την ολοκλήρωση διεργασίας

Η εκτέλεση των διεργασιών θα ολοκληρωθεί την χρονική στιγμή 41.
Για κάθε διεργασία ο Χρόνος Διεκπεραίωσής της θα είναι:

$ΧΔ_Α = 7 – 0 = 7$
$ΧΔ_Β = 27 – 3 = 24$
$ΧΔ_Γ = 14 – 9 = 5$
$ΧΔ_Δ = 41 – 12 = 29$
$ΧΔ_Ε = 18 – 14 = 4$

Και ο Μέσος Χρόνος Διεκπεραίωσης: ΜΧΔ = (7 + 24 + 5 + 29 + 4) / 5 = 13,8 χρονικές μονάδες

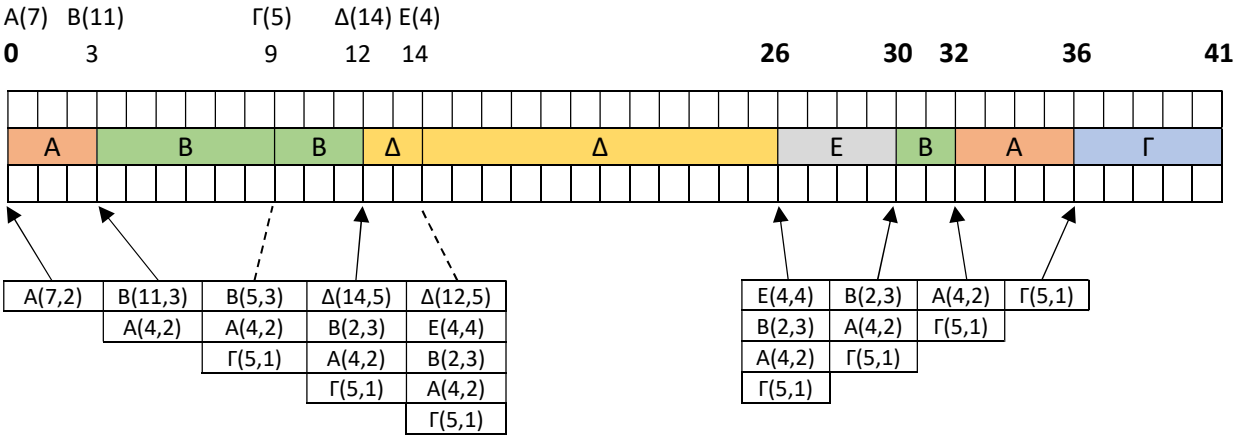
Για κάθε διεργασία ο Χρόνος Αναμονής της στο σύστημα θα είναι:

$ΧΑ_Α = ΧΑ_Α – t_{CPU} = 7 – 7 = 0$
$ΧΑ_Β = ΧΑ_Β – t_{CPU} = 24 – 11 = 13$
$ΧΑ_Γ = ΧΑ_Γ – t_{CPU} = 5 – 5 = 0$
$ΧΑ_Δ = ΧΑ_Δ – t_{CPU} = 29 – 14 = 15$
$ΧΑ_Ε = ΧΑ_Ε – t_{CPU} = 4 – 4 = 0$

Και ο Μέσος Χρόνος Αναμονής: ΜΧΑ = (13 + 15) / 5 = 5,6 χρονικές μονάδες

iv) Priority Scheduling – μη προεκχωρητικός (non-preemptive priority)

Αυτός ο αλγόριθμος υποθέτει ότι κάθε διεργασία προς εκτέλεση διαθέτει κάποια προτεραιότητα και επιλέγει να εξυπηρετήσει κάθε φορά την διεργασία με την υψηλότερη προτεραιότητα. Όταν στο σύστημα φθάσει μια διεργασία με υψηλότερη προτεραιότητα από την προτεραιότητα της τρέχουσας διεργασίας, τότε η τρέχουσα διεργασία διακόπτεται. Εξετάζουμε ποια διεργασία θα εξυπηρετηθεί κάθε φορά που μια διεργασία έχει ολοκληρώσει το χρόνο καταιγισμού της στη CPU ή μια νέα διεργασία φθάνει στην ουρά των έτοιμων διεργασιών. Ο συμβολισμός Χ(x, y) σημαίνει ότι η διεργασία Χ απαιτεί ή της απομένουν για να εκτελεστεί x χρονικές μονάδες, ενώ έχει προτεραιότητα y.



------: κατά την άφιξη διεργασίας
⟶ : κατά την ολοκλήρωση διεργασίας

Η εκτέλεση των διεργασιών θα ολοκληρωθεί την χρονική στιγμή 41.
Για κάθε διεργασία ο Χρόνος Διεκπεραίωσής της θα είναι:

$ΧΔ_A = 36 – 0 = 36$
$ΧΔ_B = 32 – 3 = 29$
$ΧΔ_Γ = 41 – 9 = 32$
$ΧΔ_Δ = 26 – 12 = 14$
$ΧΔ_E = 30 – 14 = 16$

Και ο Μέσος Χρόνος Διεκπεραίωσης: ΜΧΔ = (36 + 29 + 32 + 14 + 16) / 5 = 25,4 χρονικές μονάδες

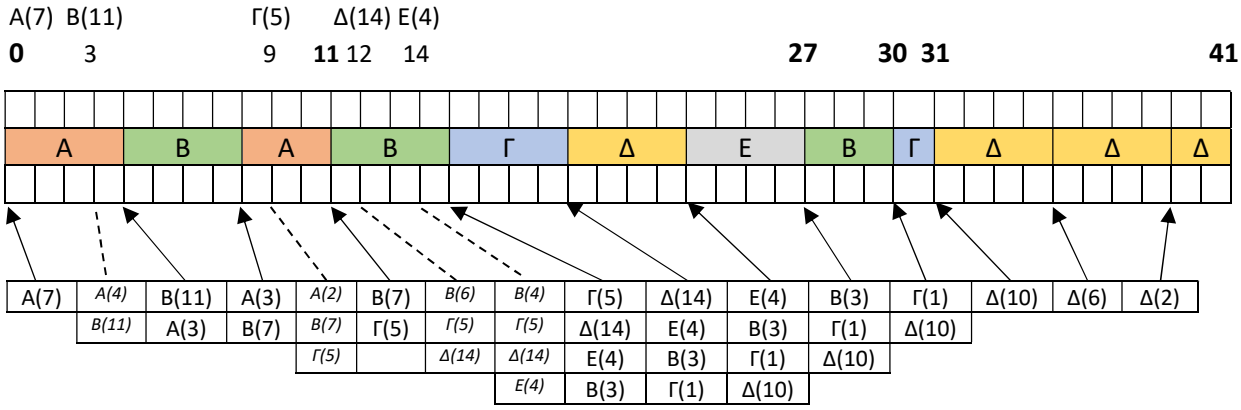
Για κάθε διεργασία ο Χρόνος Αναμονής της στο σύστημα θα είναι:

$ΧΑ_A = ΧΑ_A – t_{CPU} = 36 – 7 = 29$
$ΧΑ_B = ΧΑ_B – t_{CPU} = 29 – 11 = 18$
$ΧΑ_Γ = ΧΑ_Γ – t_{CPU} = 32 – 5 = 27$
$ΧΑ_Δ = ΧΑ_Δ – t_{CPU} = 14 – 14 = 0$
$ΧΑ_E = ΧΑ_E – t_{CPU} = 16 – 4 = 12$

Και ο Μέσος Χρόνος Αναμονής: ΜΧΑ = (29 + 18 + 27 + 12) / 5 = 17,2 χρονικές μονάδες

v) RR (Round Robin) με κβάντο χρόνου 4 χρονικές μονάδες

Σε κάθε εργασία εκχωρείται κάποιο χρονικό διάστημα, το οποίο ονομάζεται κβάντο χρόνου, μέσα στο οποίο επιτρέπεται η εκτέλεση της. Αν η διεργασία εξακολουθεί να εκτελείται όταν τελειώσει το κβάντο χρόνου της, η CPU παραχωρείται υποχρεωτικά σε κάποια άλλη διεργασία. Αν η διεργασία μπλοκαρίστηκε ή ολοκληρώθηκε πριν από το πέρας του κβάντου, είναι εναλλαγή της CPU σε άλλη διεργασία γίνεται τη χρονική στιγμή που η διεργασία σταμάτησε. Προκειμένου αυτό να επιτευχθεί διατηρούμε μία ουρά λίστα με τις διεργασίες προς εξυπηρέτηση. Στις παρενθέσεις αναφέρονται οι χρονικές μονάδες που απαιτεί ή απομένουν για να εκτελεστεί η αντίστοιχη διεργασία.



------: κατά την άφιξη διεργασίας
⟶ : κατά την ολοκλήρωση του κβάντου ή της διεργασίας

Η εκτέλεση των διεργασιών θα ολοκληρωθεί την χρονική στιγμή 41.
Για κάθε διεργασία ο Χρόνος Διεκπεραίωσής της θα είναι:

$ΧΔ_Α = 11 – 0 = 11$
$ΧΔ_Β = 30 – 3 = 27$
$ΧΔ_Γ = 31 – 9 = 23$
$ΧΔ_Δ = 41 – 12 = 29$
$ΧΔ_Ε = 27 – 14 = 13$

Και ο Μέσος Χρόνος Διεκπεραίωσης: ΜΧΔ = (11 + 27 + 23 + 29 + 13) / 5 = 20,6 χρονικές μονάδες

Για κάθε διεργασία ο Χρόνος Αναμονής της στο σύστημα θα είναι:

$ΧΑ_Α = ΧΑ_Α – t_{CPU} = 11 – 7 = 4$
$ΧΑ_Β = ΧΑ_Β – t_{CPU} = 27 – 11 = 16$
$ΧΑ_Γ = ΧΑ_Γ – t_{CPU} = 23 – 5 = 18$
$ΧΑ_Δ = ΧΑ_Δ – t_{CPU} = 29 – 14 = 15$
$ΧΑ_Ε = ΧΑ_Ε – t_{CPU} = 13 – 4 = 9$

Και ο Μέσος Χρόνος Αναμονής: ΜΧΑ = (4 + 16 + 18 + 15 + 9) / 5 = 12,4 χρονικές μονάδες