Λειτουργικά Συστήματα – Τμήμα Β΄ (Μαξ-Ω)

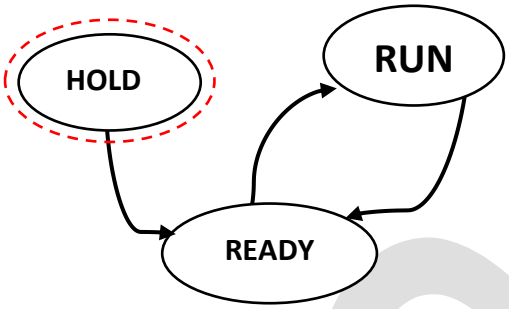
Διάλεξη 3

Job	Χρόνος Άφιξης	Run Time
1	0	50
2	20	20
3	40	60
4	70	30

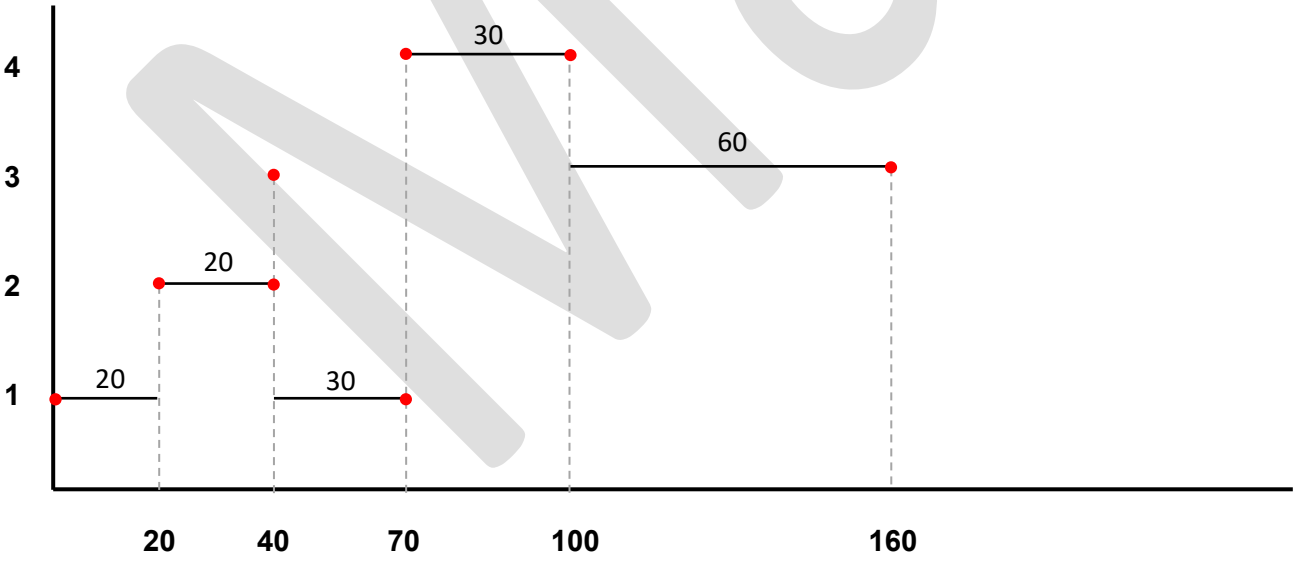
SRTN (Shortest Remaining Time Next)

Μπορεί να χρησιμοποιηθεί με προεκχώρηση (short – term scheduler) αλλά και χωρίς προεκχώρηση (long – term scheduler).

Περίπτωση μακροπρόθεσμου δρομολογητή: από όλες που περιμένουν στο HOLD θα διαλέξει μία διεργασία για να τρέξει, αυτή θα είναι εκείνη με τον μικρότερο χρόνο



Στο μάθημα μελετάμε τη περίπτωση που χρησιμοποιείται με προεκχώρηση από τον βραχυπρόθεσμο δρομολογητή. Αν έρθει μια διεργασία που θέλει λιγότερο χρόνο για να τελειώσει από μια άλλη που ήδη «τρέχει», θα βγει αυτή που «τρέχει» και θα μπει η άλλη



** Με κόκκινη βουλίτσα σημειώνονται ο χρόνος άφιξης και η χρονική στιγμή που τελειώνει η κάθε διεργασία*

$TT_1 = 70 - 0 = 70$
$TT_2 = 40 - 20 = 20$
$TT_3 = 160 - 40 = 120$
$TT_4 = 100 - 70 = 30$

$$ATT = \frac{70+20+120+30}{4} = 60$$

$WTT_1 = 70/50 = 1,4$
$WTT_2 = 20/20 = 1$
$WTT_3 = 120/60 = 2$
$WTT_4 = 30/30 = 1$

$AWTT = \frac{1,4+1+2+1}{4} = 1,35$

$WT_1 = 70 – 50 = 20$
$WT_2 = 20 – 20 = 0$
$WT_3 = 120 – 60 = 60$
$WT_4 = 30 – 30 = 0$

$AWT = \frac{20+0+60+0}{4} = 20$

$WWT_1 = 20/50 = 0,4$
.
.
.
AWWT = 0,35

SRTN μειονεκτήματα:

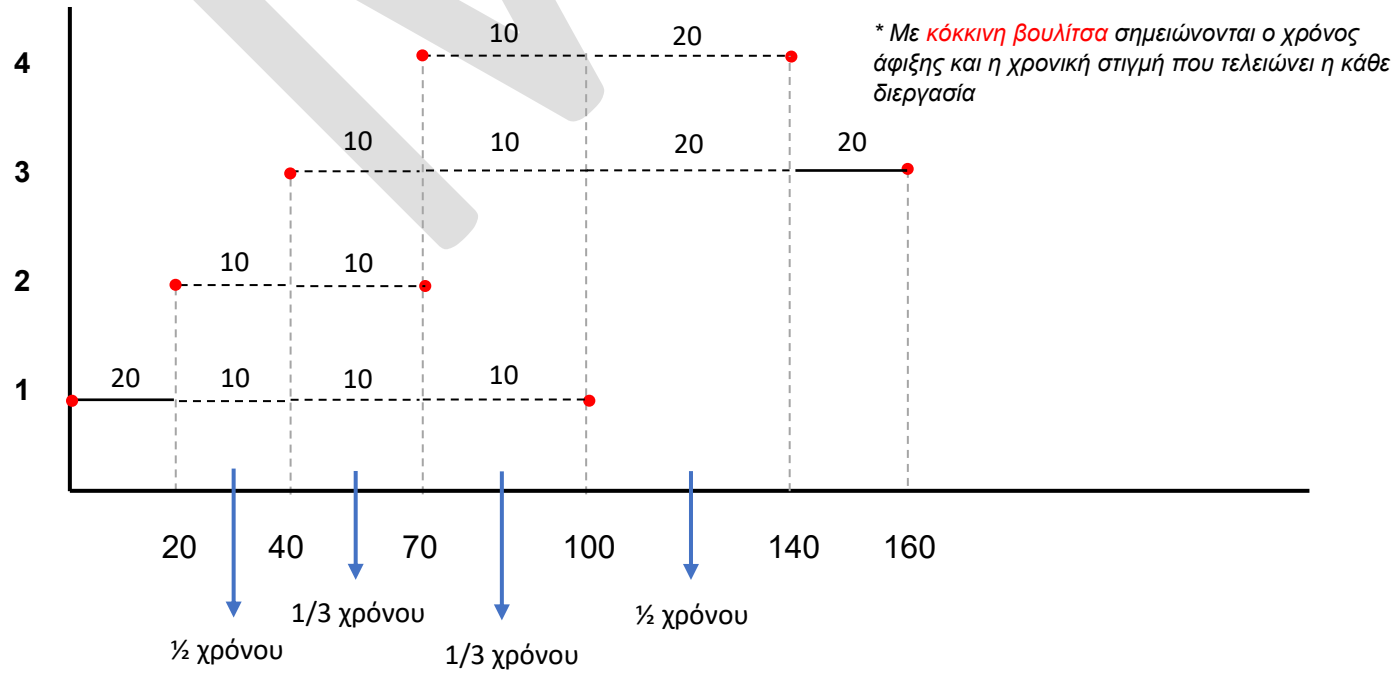
- Όσο έρχονται μικρότερες διεργασίες στο ΛΣ, οι μεγαλύτερες δε θα τρέχουν
- Πρέπει να γνωρίζουμε τα Run Times

Round Robin (εκ περιτροπής)

Ο αλγόριθμος αυτός είναι πάντα με προεκχώρηση.

Καθορίζεται ένα κβάντο χρόνου q. Ένα κβάντο χρόνου είναι μια συγκεκριμένη χρονική περίοδος, παράμετρος του συστήματος, η οποία καθορίζει πόσο θα τρέξει η κάθε διεργασία όταν μπαίνει στην CPU
Τρέχουν λίγο – λίγο όλες οι διεργασίες δίνοντας από ένα κβάντο στην καθεμιά.

Μικρό κβάντο (σε σχέση με τα Run Times)



$TT_1 = 100 – 0 = 100$
$TT_2 = 70 – 20 = 50$
$TT_3 = 160 – 40 = 120$
$TT_4 = 140 – 70 = 70$

$ATT = \frac{100+50+120+70}{4} = 85$

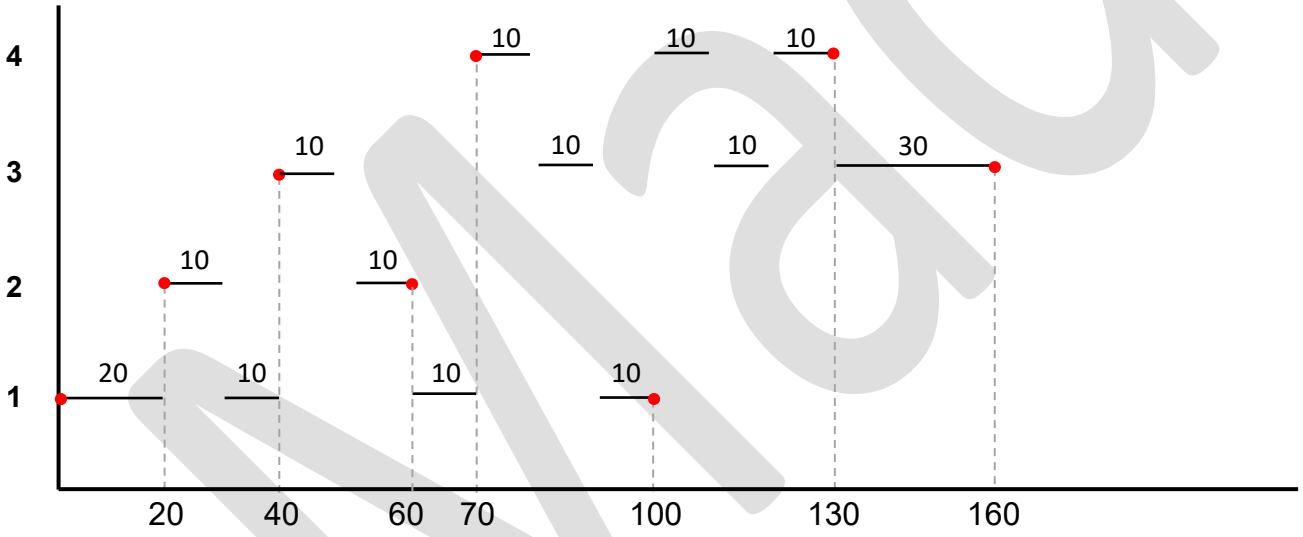
$WTT_1 = 100/50 = 2$
$WTT_2 = 50/20 = 2,5$
$WTT_3 = 120/60 = 2$
$WTT_4 = 70/30 = 2,3$

$AWTT = \frac{2+2,5+2+2,3}{4} = 2,2$

AWWT = 1,2

$WT_1 = 100 – 50 = 50$
$WT_2 = 50 – 20 = 30$
$WT_3 = 60$
$WT_4 = 40$

<u>ΜΕΓΑΛΟ ΚΒΑΝΤΟ</u>
q=10



** Με κόκκινη βουλίτσα σημειώνονται ο χρόνος άφιξης και η χρονική στιγμή που τελειώνει η κάθε διεργασία*

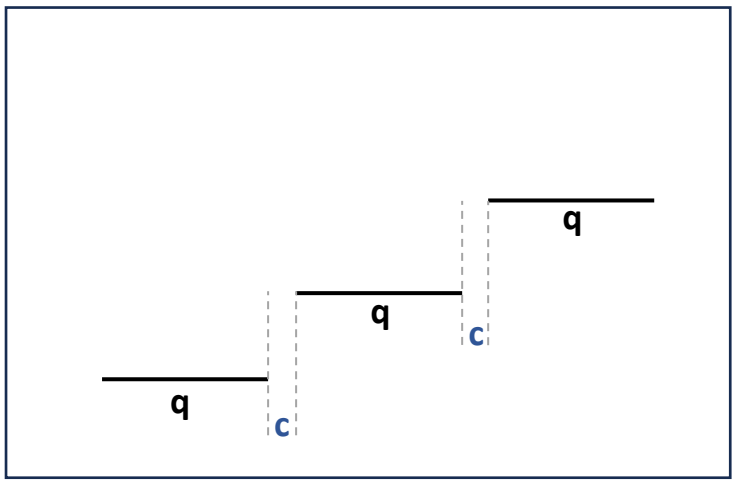
$TT_1 = 0$
$TT_2 = 60 - 20 = 40$
$TT_3 = 160 - 40 = 120$
$TT_4 = 130 - 70 = 60$

Utilization (Βαθμός χρήσης)

Έστω c = 0,001 (όπου c είναι το task switching time ή χρόνος εναλλαγής, δηλαδή ο χρόνος που χρειάζεται για να αλλάξουμε από την διεργασία στην άλλη)
Το c εξαρτάται από το πόσο καλά γραμμένο είναι ένα ΛΣ ή πόσο καλή είναι μια CPU (όσο καλύτερη τόσο και μικρότερο)
Έστω 4 εναλλαγές = 4 * 0,001 = 0,004

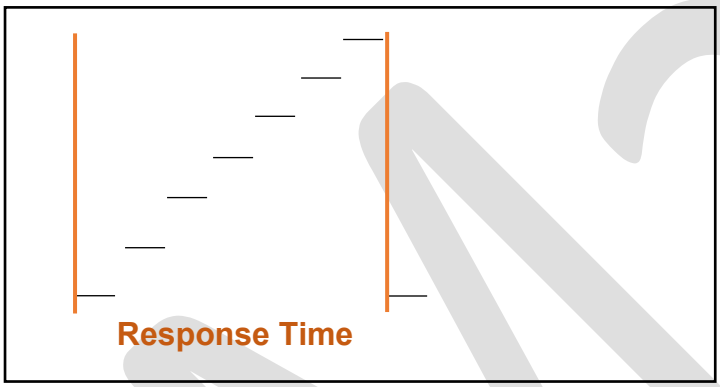
Βαθμός Χρήσης = $\frac{\text{Χρήσιμος}}{\text{Συνολικός}}$ $\quad U = \frac{q}{q+c} = \frac{160}{160,004}$

Υπολογισμός Κβάντου



Ο τύπος $U = \frac{q}{q+c}$ θα με βοηθήσει να υπολογίζω το κβάντο. Το Utilization θέλω να είναι όσο πιο μεγάλο γίνεται. Εφόσον το c είναι σταθερό και δε μπορώ να το πειράξω. Το q ναι μεν πόσο είναι ακριβώς δεν μπορούμε να το γνωρίζουμε, αλλά θέλουμε να είναι όσο πιο μεγάλο γίνεται.

Response Time (Χρόνος ανταπόκρισης)



Ο χρόνος στον οποίο έχει οπωσδήποτε πάρει ένα κβάντο η κάθε διεργασία.

Response Time = N*(q+c) = N*q + N*c

- Response Time: το θέλω όσο πιο μικρό γίνεται
- N: Αριθμός διεργασιών

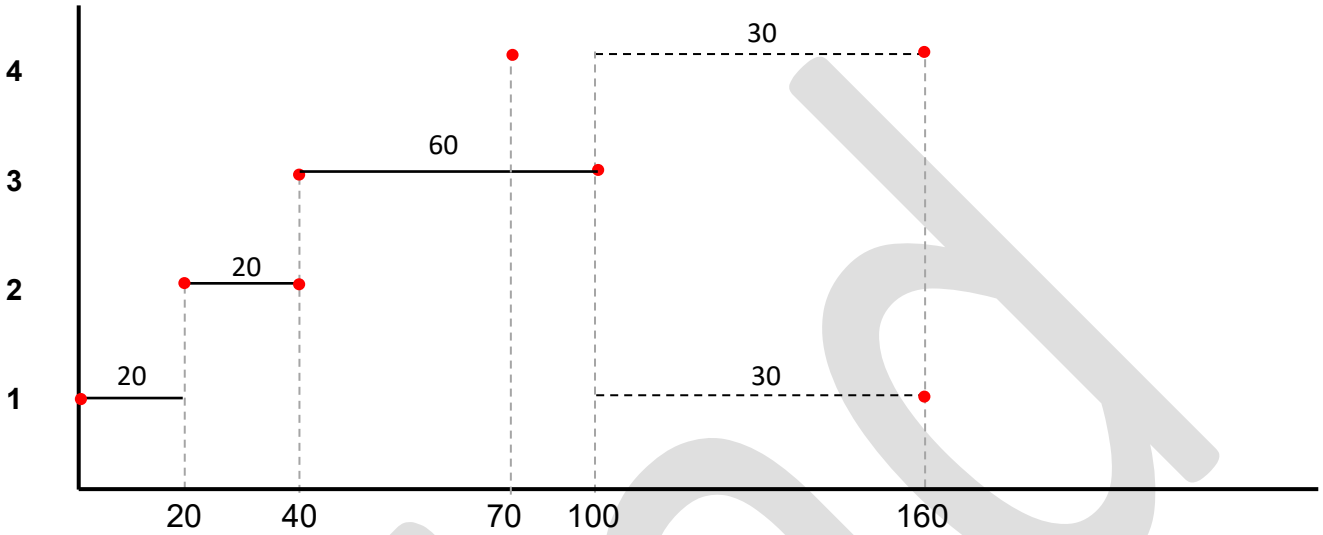
Για να έχω όσο πιο μικρό χρόνο ανταπόκρισης γίνεται θέλω το κβάντο να είναι όσο πιο μικρό γίνεται.

Αυτό, όμως, έρχεται σε αντίθεση με τον βαθμό χρήσης, όπου θέλαμε το κβάντο να είσαι όσο το δυνατόν μεγαλύτερο. Δεν μπορούμε να πετύχουμε όλα τα κριτήρια στα οποία αναφερθήκαμε για να κρίνουμε αν ένας αλγόριθμος χρονοπρογραμματισμού είναι καλός. Επομένως, αυτό που θέλω είναι μια τιμή του κβάντου έτσι ώστε και ο βαθμός χρήσης και ο χρόνος ανταπόκρισης να είναι καλός.

Προτεραιότητες (με Round Robin)

Job	Χρόνος Άφιξης	Run Time	Priority
1	0	50	1
2	20	20	2
3	40	60	2
4	70	30	1

** Θεωρούμε μεγαλύτερο νούμερο = μεγαλύτερη προτεραιότητα*



** Με κόκκινη βουλίτσα σημειώνονται ο χρόνος άφιξης και η χρονική στιγμή που τελειώνει η κάθε διεργασία*

Θα ξεκινήσει η 1η διεργασία παίρνοντας όλα τα κβάντα μέχρι να έρθει η 2η. Η 2η θα πάρει όλα τα κβάντα (αφού έχει μεγαλύτερη προτεραιότητα από την 1η) και θα «τρέξει» μέχρι να ολοκληρωθεί. Όταν ολοκληρωθεί η 2η, έρχεται η 3η και «τρέχει» (αφού επίσης έχει μεγαλύτερη προτεραιότητα από την 1η). Τη χρονική στιγμή 70 έρχεται η 4η, η οποία δε «τρέχει», καθώς εκείνη την ώρα «τρέχει» η 3η που έχει μεγαλύτερη προτεραιότητα. Αφού ολοκληρωθεί η 3η, έχουμε την 1η και την 4η με ίδιες προτεραιότητες. Χρησιμοποιώντας τον αλγόριθμο Round Robin, «τρέχουν» και οι 2 μαζί μέχρι να ολοκληρωθούν.

Μειονέκτημα: όσο έρχονται διεργασίες με χαμηλή προτεραιότητα δε θα τρέχουν αφού θα τρέχουν πρώτα αυτές με την μεγαλύτερη.

Λύση: χρησιμοποιούμε **Γήρανση** για να το αντιμετωπίσουμε. Το ΛΣ ανεβάζει από μόνο του προτεραιότητα όσο περνάει ο χρόνος (μπορεί να την ανεβάσει και πάνω από 1 φορά), μπορεί ακόμα να γίνει και το ανάποδο και να κατεβάζει την προτεραιότητα.

Χρήσιμα Links

- https://www.geeksforgeeks.org/shortest-remaining-time-first-preemptive-sjf-scheduling-algorithm/
- https://www.geeksforgeeks.org/round-robin-scheduling-with-different-arrival-times/