TurnAroundTime (χρόνος παραμονής στο σύστημα):
TT = χρόνος που ολοκληρώνεται - χρόνος άφιξης

Waiting Time (Χρόνος αναμονής):
WT = TT - χρόνος εκτέλεσης
(ο χρόνος εκτέλεσης λέγεται αλλιώς και RT, run time)

Average TurnArround Time (μέσος χρόνος παραμονής στο σύστημα):
ATT = TT1+TT2+... /πλήθος διεργασιών

Average Waiting Time (μέσος χρόνος αναμονής):
AWT = WT1+WT2+... / πλήθος διεργασιών

Weighted TurnArround Time(σταθμισμένος χρόνος παραμονής):
WTT = TT / RT

Weighted Waiting Time(σταθμισμένος χρόνος αναμονής):
WWT = WT / RT

Average Weighted TurnArround Time(μέσος σταθμισμένος χρόνος παραμονής):
AWTT = WTT1+WTT2+... / πλήθος διεργασιών

Average Weighted Waiting Time(μέσος σταθμισμένος χρόνος αναμονής):
AWTT = WWT1+WWT2+... / πλήθος διεργασιών

Βαθμός χρήσης-Utilization
U = q / (q +c)
(υπάρχει άσκηση στις λυμένες του Ρουμελιώτη για τον βαθμό χρήσης)

overhead-χρόνος που χρησιμοποιείται για context switch:
U = 1 - overhead

SLEEP TIME → BONUS:
0-99 ms → 0
100-199 → 1
200-299 → 2
300-399 → 3
400-499 → 4
500-599 → 5
600-699 → 6
700-799 → 7
800-899 → 8
900-999 → 9
1 sec → 10

DYNAMIC PRIORITY
DP = MAX[100,min(SP-bonus+5,139)]

Υπολογισμός Κβάντων:
Αν PR<120 → (140 - Pr) * 20 ms
Αν PR>120 → (140 - Pr) * 5 ms

vruntime = vruntime + (t *weight)

weight = $1{,}25^{nice}$
άρα όσο nice μειώνεται τόσο priority αυξάνεται

Υπερφόρτωση συστήματος:
N * MG > TL

Νέα κβάντα:
K = 1024 / $1{,}25^{nice}$

ΚΑΝΟΝΕΣ RED BLACK TREES

- Η ρίζα είναι πάντα black
- νέα εισαγωγή πάντα red
- Κάθε μονοπάτι από τη ρίζα προς κάθε φύλλο έχει το ίδιο πλήθος μαύρων κόμβων
- κανένα μονοπάτι δεν μπορεί να έχει διαδοχικά red
- φύλλα με τιμή NULL είναι black

ΠΑΡΑΒΙΑΣΗ ΚΑΝΟΝΩΝ
ελέχγουμε τον θείο τον κόμβο που προκάλεσε το πρόβλημα. Αν είναι μαύρος κάνουμε περιστροφή αν είναι κόκκινος αλλάζουμε χρώμα σε μπαμπά και θείο.

ΑΥΣΤΗΡΗ ΕΝΑΛΛΑΓΗ

(α)

```
1   while (TRUE) {
2       while (turn != 0) /* βρόχος */;
3       critical_region();
4       turn = 1;
5       noncritical_region();
    }
```

(β)

```
1    while (TRUE) {
2        while (turn != 1) /* βρόχος */;
3        critical_region();
4        turn = 0;
5        noncritical_region();
     }
```

TSL

```
1     enter_region:
2         TSL REGISTER,LOCK
3         CMP REGISTER,#0
4         JNE enter_region
5         RET
6     leave_region:
7         MOVE LOCK,#0
8         RET
```

PETERSON

```
1    #define FALSE  0
2    #define TRUE  1
3    #define N  2
4    int turn;
5    int interested[N];
6     void enter_region(int process)
7    {
8        int other;
9        other = 1 - process;
10       interested[process] = TRUE;
11       turn = process;
12       while (turn == process && interested[other] == TRUE)
13   }
1    void leave_region(int process)
2    {
3        interested[process] = FALSE;
4    }
```

Producer/Consumer

```
        #define N 100
        int count = 0;
        void producer(void)
        {
1           int item;
2           while (TRUE) {
3               item = produce_item();
4               if (count == N) sleep();
5               insert_item(item);
6               count = count + 1;
7               if (count == 1) wakeup(consumer);
            }}
```

```
     void consumer(void)
     {
          int item;
1         while (TRUE) {
2              if (count == 0) sleep();
3              item = remove_item();
4              count = count - 1;
5              if (count == N - 1) wakeup(producer);
6              consume_item(item);
7         }
     }
```

Λύση με semaphores για το πρόβλημα Producer/Consumer

```
     #define N 100
     typedef int semaphore;
     semaphore mutex = 1;
     semaphore empty = N;
     semaphore full = 0;

     void producer(void)
     {
1        int item;
2        while (TRUE) {
3            item = produce_item();
4            down(&empty);
5            down(&mutex);
6            insert_item(item);
7            up(&mutex);
8        up(&full);
         }
     }

     void consumer(void)
     {
1        int item;
2        while (TRUE) {
3            down(&full);
4            down(&mutex);
5            item = remove_item();
6            up(&mutex);
7            up(&empty);
8            consume_item(item);
         }
     }
```

Αναγνώστες – συγγραφείς

```
semaphore mutex=1, db=1
int rc=0
reader()
    { while (TRUE) {
        down(mutex)
        rc=rc+1
        if (rc=1) down(db)
        up(mutex)
        read_database()
        down(mutex)
        rc=rc-1
        if (rc=0) up(db)
        up(mutex)
        use_data()
    }
}
```

```
writer()
{
1   while (TRUE)
2   {
3   think_up_data()
4   down(db)
5   write_database()
6   up(db)
    }
}
```