2η Εργασία Λειτουργικών Συστημάτων  
Φοιτητές: Γρηγόρης Ηλιάδης, Δημήτρης Κεφαλάς, Φοίβος Πουρναρόπουλος  
ΑΕΜ: 2522, 2533, 2614  
Διδάσκων: Εμμανουήλ Κουτσουμπέλιας  
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Ηλεκτρονικών Υπολογιστών   
Πανεπιστήμιο Θεσσαλίας

Profile 1:

Στο profile 1 δημιουργούμε μόνο non interactive διεργασίες με μη αμελητέο χρόνο εκτέλεσης. Η διεργασία (1) δημιουργείται πρώτη και τρέχει για αρκετό χρονικό διάστημα μόνη της στον επεξεργαστή. Μετά από πολλούς κύκλους μηχανής και ενώ η διεργασία 1 δεν έχει τελειώσει ακόμα δημιουργούμε διαδοχικά έξι διεργασίες ανά ένα κύκλο μηχανής.

Κώδικας: example1.conf

[endTime, rqWaitTime] [endTime, rqWaitTime]

Pi: SPAWNTIME: WAITING\_RQ: EXP\_BURST:

1 100ms [1991ms, 1000ms] [4010ms, 3000ms]

2 999ms [3918ms, 2430ms] [3649ms, 2160ms]

3 1009ms [4001ms, 2480ms] [3629ms, 2130ms]

4 1019ms [4010ms, 2470ms] [3609ms, 2100ms]

5 1029ms [3938ms, 2430ms] [3589ms, 2070ms]

6 1039ms [4008ms, 2460ms] [3569ms, 2040ms]

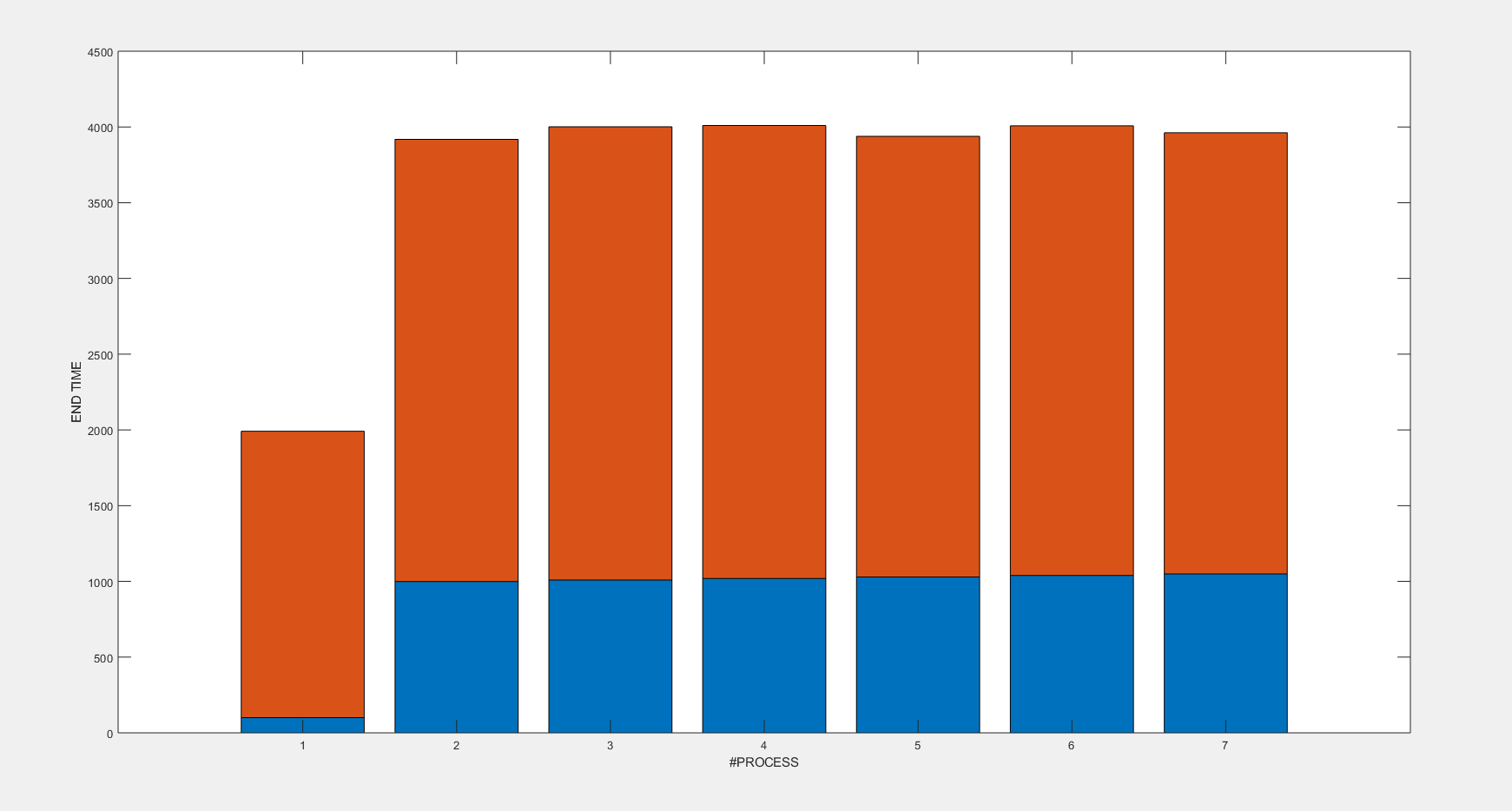
7 1049ms [3962ms, 2430ms] [3999ms, 2450ms]

Στα πρώτα δύο διαγράμματα που ακολουθούν ο κάθετος άξονας συμβολίζει τη χρονική στιγμή τερματισμού κάθε διεργασίας και ο οριζόντιος άξονας αναφέρεται για το κάθε process, πχ η 1η μπάρα συμβολίζει την διεργασία 1 κοκ.

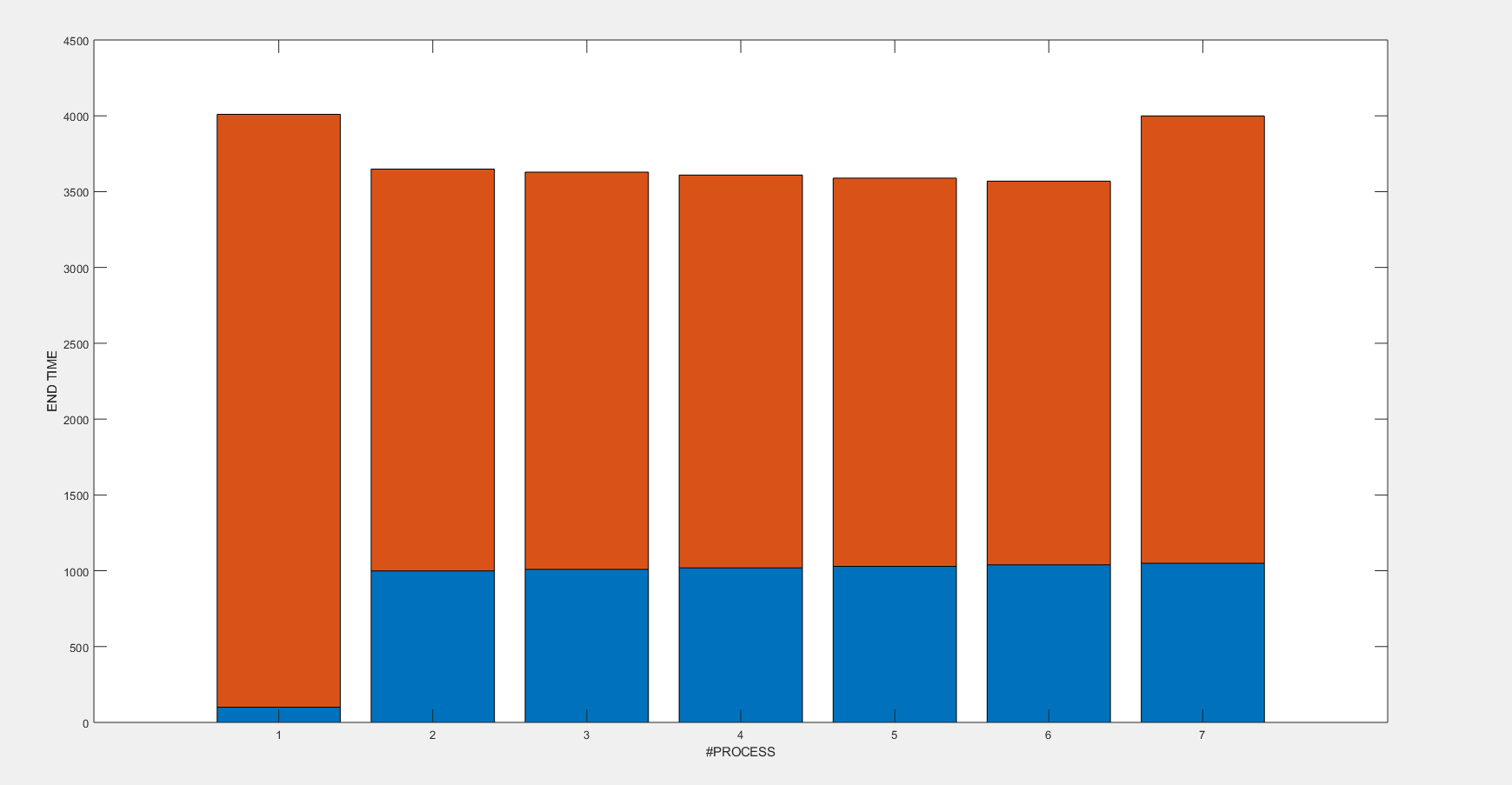
Στα άλλα δύο διαγράμματα που απεικονίζονται, ο κάθετος άξονας συμβολίζει το χρονικό διάστημα αναμονής κάθε διεργασίας στην ουρά δρομολόγησης ενώ ο οριζόντιος αναφέρεται πάλι στο κάθε process.

Με πορτοκαλί χρώμα μαρκάρετε το χρονικό διάστημα από την δημιουργία της κάθε διεργασίας μέχρι τον τερματισμό της.

Scheduling based on Goodness (expected burst and waiting time in rq) :

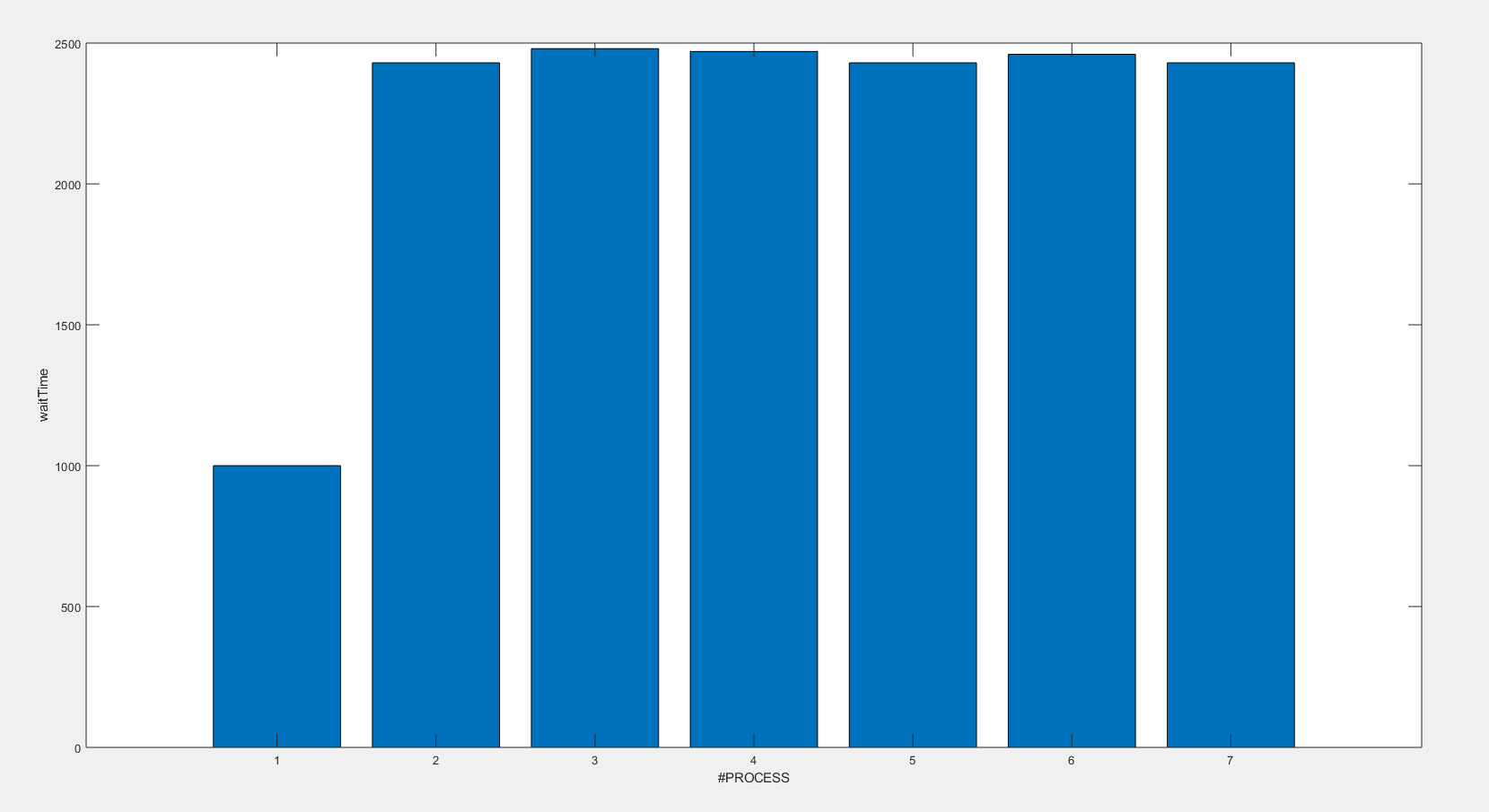


Scheduling based on expected burst:



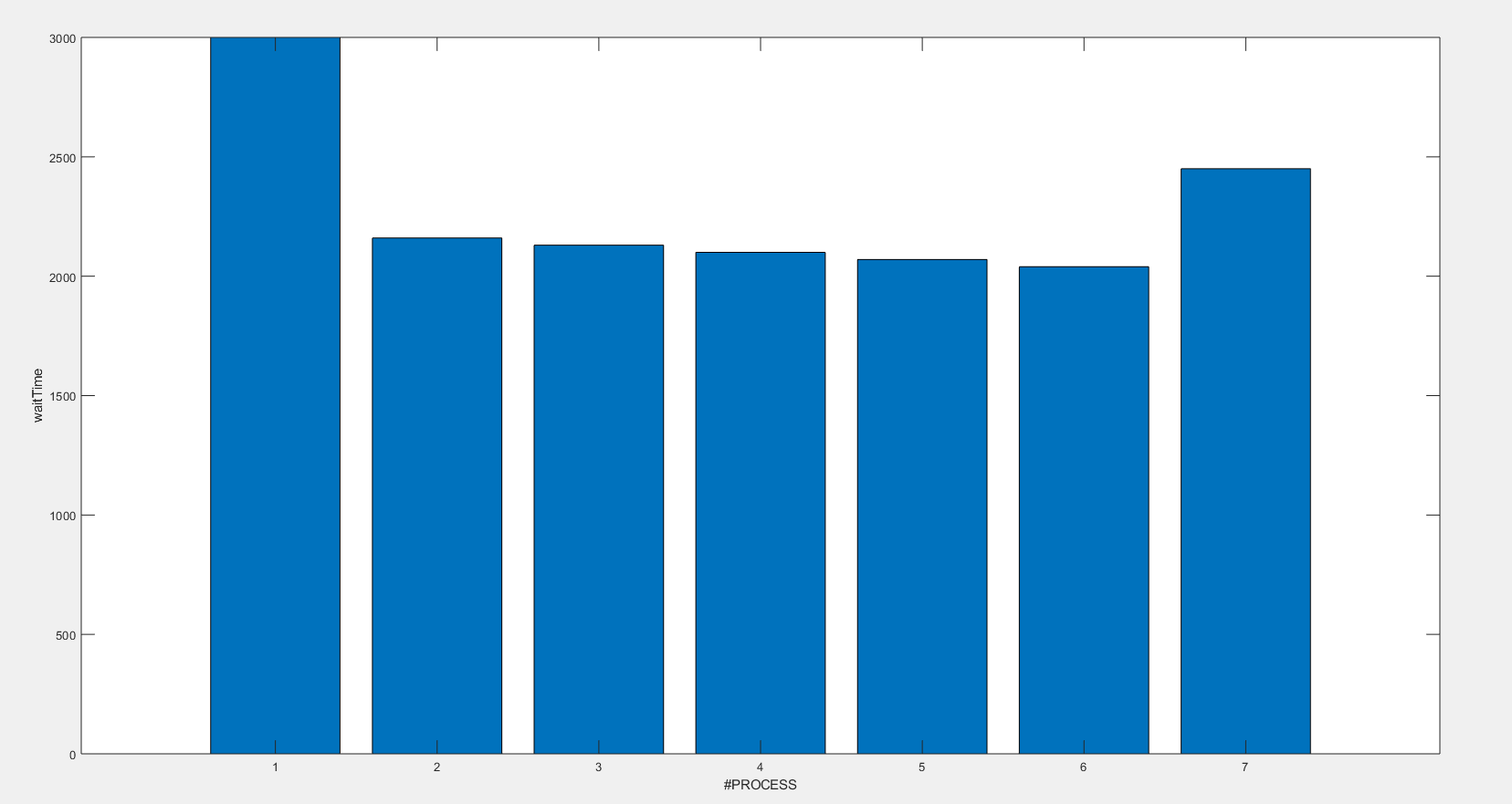
Scheduling based on Goodness (expected burst and waiting time in rq)

Running queue waiting time:



Scheduling based on expected burst:

Running queue waiting time:

­­

TURN\_AROUND\_TIME

Scheduling based on Goodness (expected burst and waiting time in rq) : turn around time = 2797ms   
Scheduling based on expected burst: turn around time = 2830ms

Σύμφωνα με τις μετρήσεις που πήραμε παρατηρούμε πολύ μεγάλη διαφορά στο waiting time της διεργασίας 1. Η διεργασία 1 μονοπωλεί τον επεξεργαστή έως ότου δημιουργηθούν οι επόμενες διεργασίες. Στην περίπτωση όπου η δρομολόγηση γίνεται με βάση το exp\_burst παρατηρούμε ότι η πρώτη διεργασία αργεί να πάρει ξανά τον επεξεργαστή σε σχέση με την δρομολόγηση που γίνεται με βάση το goodness. Αυτό συμβαίνει διότι στη δεύτερη περίπτωση δρομολόγησης λαμβάνουμε υπόψιν μας και το waiting time της κάθε διεργασίας, που η σχέση με το goodness κάθε διεργασίας είναι αντιστρόφως ανάλογη. Για το λόγο αυτό ο επεξεργαστής μοιράζεται πιο δίκαια στις διεργασίες στην περίπτωση του goodness και αποφεύγεται η ακραία περίπτωση λιμοκτονίας της πρώτης διεργασίας όταν έρχονται συνεχώς νέες μικρότερες διεργασίες. Σε αντίθεση, η δρομολόγηση με βάση το exp\_burst επιφέρει λιμοκτονία στο ίδιο σενάριο εκτέλεσης. Η πιο δίκαιη κατανομή του επεξεργαστή επιφέρει μείωση και στο turn\_around\_time. Σε περίπτωση όπου θα είχαμε περισσότερες διεργασίες όπου θα δημιουργούνταν ταυτόχρονα με την διεργασία 1, θα παρατηρούσαμε ακόμα μεγαλύτερη μείωση του turn\_around\_time.

Παρατηρούμε ότι όταν η δρομολόγηση γίνεται με βάση το exp\_burst, μια διεργασία όταν δημιουργείται παίρνει πάντα τον επεξεργαστή στον επόμενο κύκλο. Στην δρομολόγηση με βάση το goodness όμως μια νέα διεργασία μπορεί να μην πάρει αμέσως τον επεξεργαστή αλλά να περιμένει κάποιους κύκλους γιατί μπορεί να υπάρχει μια διεργασία που περιμένει για μεγάλο χρονικό διάστημα στην ουρά δρομολόγησης.

Και στις δύο περιπτώσεις παρατηρούμε ότι και οι δύο αλγόριθμοι δρομολόγησης εκφυλίζονται σε Round Robin, στη δρομολόγηση με βάση το goodness εκφυλίζεται από τους πρώτους κιόλας κύκλους, ενώ με βάση το exp burst μετά από κάποιο χρονικό διάστημα (όταν όλες οι διεργασίες έχουν πάρει περίπου ίσο χρόνο τον επεξεργαστή).

Profile 2:

Στο profile 2 δημιουργούμε μόνο interactive διεργασίες με μη αμελητέο χρόνο εκτέλεσης. Οι διεργασίες δημιουργούνται με μικρή χρονική καθυστέρηση περίπου 2 κύκλους μηχανής μεταξύ των πρώτων τριών διεργασιών και με 4 κύκλους μηχανής μεταξύ της τρίτης και της τελευταίας διεργασίας.

Κώδικας: example2a.conf

SEED: 500

[endTime, rqWaitTime] [endTime, rqWaitTime]

Pi: SPAWNTIME: WAITING\_RQ: EXP\_BURST:

1 59ms [1270ms, 650ms] [1700ms, 1100ms]

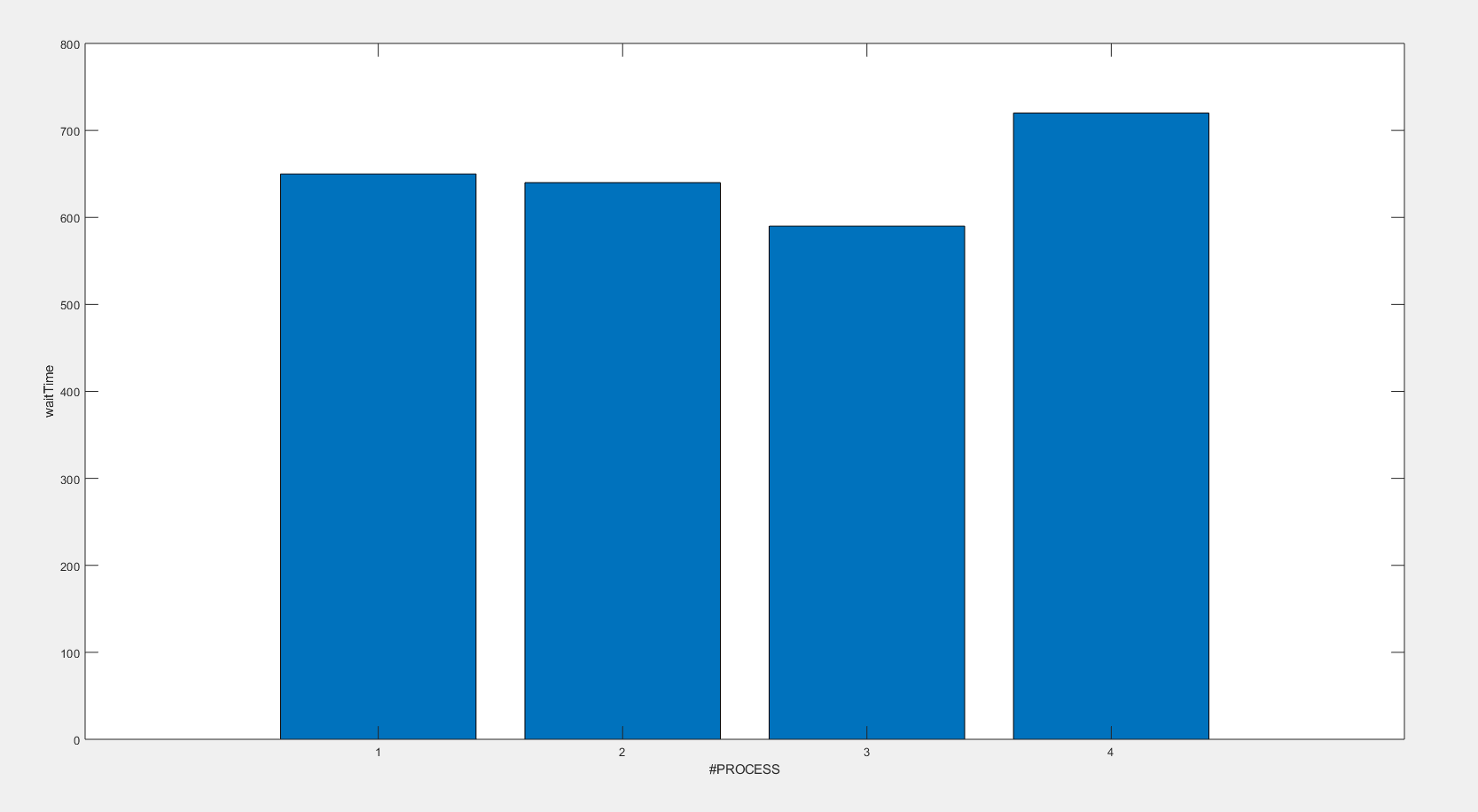
2 129ms [1582ms, 640ms] [1568ms, 510ms]

3 149ms [1700ms, 590ms] [1478ms, 750ms]

4 189ms [1469ms, 720ms] [1685ms, 630ms

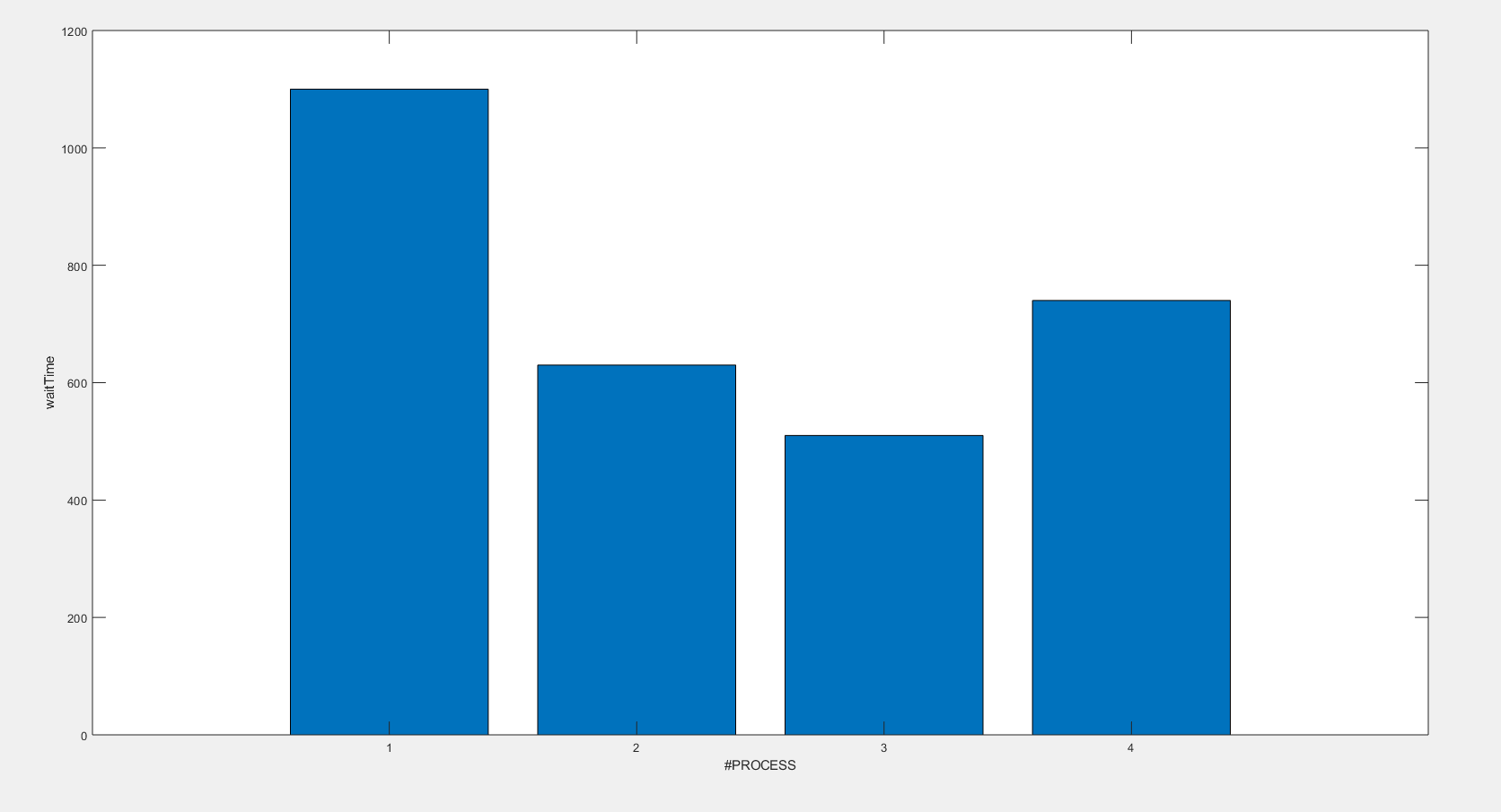
Scheduling based on Goodness (expected burst and waiting time in rq)

Running queue waiting time:



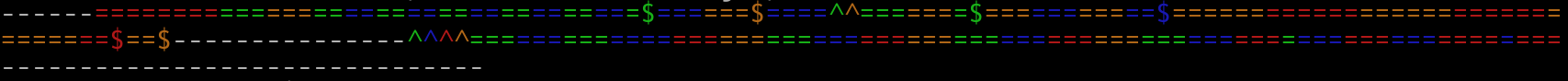
Scheduling based on expected burst:

Running queue waiting time:

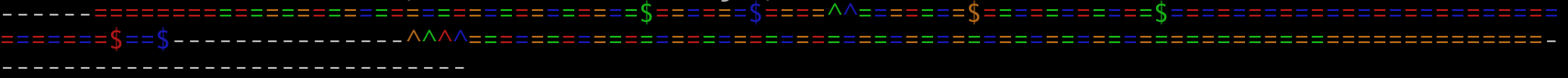


PROCESSES\_TIME\_LINE

Scheduling based on expected burst:



Scheduling based on Goodness (expected burst and waiting time in rq) :

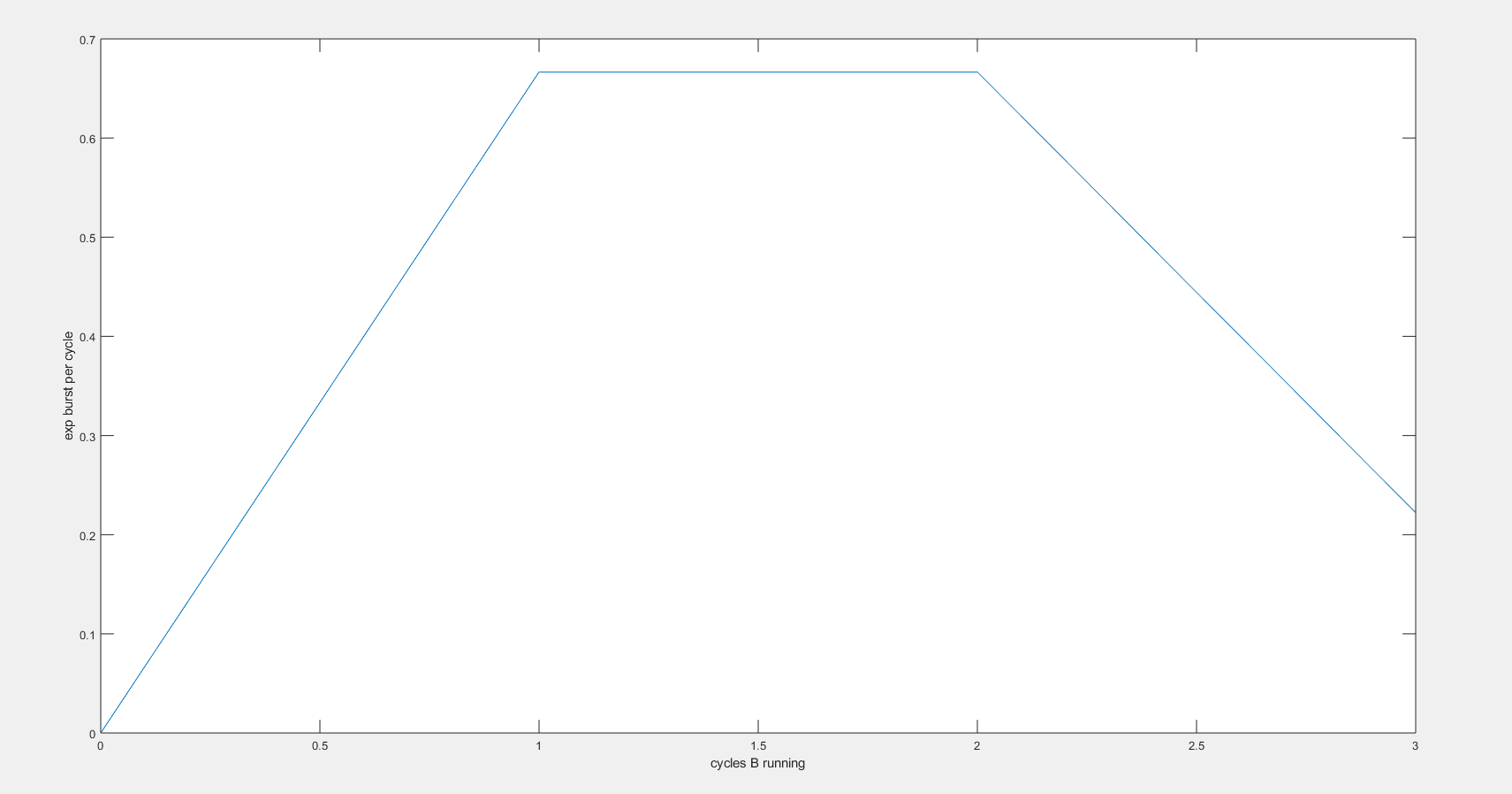


Υπόμνημα:  
- / = : INIT / Current Process Running P1 P2 P3 P4  
$ : Interrupt ( process going to sleep )  
^ : Interrupt ( process wake up )

Στην περίπτωση που η δρομολόγηση γίνεται μόνο με βάση το expected burst όλες οι διεργασίες μόλις δημιουργηθούν έχουν την ελάχιστη τιμή του expected burst, με αποτέλεσμα να παίρνουν κατευθείαν τον επεξεργαστή στον επόμενο κύκλο. Αν οι προϋπάρχουσες διεργασίες έχουν τρέξει για μεγάλο χρονικό διάστημα, οι καινούργιες διεργασίες μονοπωλούν τον επεξεργαστή μέχρι να εξισορροπηθεί το expected burst με αυτό των υπολοίπων. Στην περίπτωση που η δρομολόγηση γίνεται με βάση και το waiting time in run queue, όταν δημιουργείται μία διεργασία   δεν είναι απαραίτητο να πάρει τον επεξεργαστή στον επόμενο κύκλο παρ' όλου που έχει μηδενικό expected burst, διότι μπορεί να υπάρχουν διεργασίες που περιμένουν στην ουρά δρομολόγησης για μεγάλο χρονικό διάστημα. Επίσης οι καινούργιες διεργασίες δεν μονοπωλούν τον επεξεργαστή κατά τους πρώτους κύκλους της εκτέλεσης τους. Αυτό συμβαίνει διότι, η κάθε διεργασία έχει αρχικά μηδενικό expected burst αλλά και μηδενικό waiting time in run queue, συνεπώς μικρό goodness, ενώ μετά από την πρώτη φορά που θα πάρει τον επεξεργαστή αυξάνεται σημαντικά το expected burst ενώ διατηρεί ακόμη μηδενικό wait\_time\_in\_rq (πολύ μικρότερο σε σχέση με τις υπόλοιπες διεργασίες) **(βλ. Σχήμα 1, Σχήμα 2)**. Συνεπώς το goodness αυξάνεται και μειώνεται η προτεραιότητα της διεργασίας με αποτέλεσμα να χάνει τον επεξεργαστή **(βλ. Σχήμα 2)**. Αυτό προκαλεί μία εξισορρόπηση μεταξύ των goodness των διεργασιών με αποτέλεσμα στους επόμενος κύκλους να εκφυλίζεται η δρομολόγηση σε Round Robin. Την ίδια συμπεριφορά παρατηρούμε και όταν μια διεργασία ξυπνάει μετά από κάποιο interrupt και εισέρχεται εκ νέου στο run\_queue, δηλαδή η δρομολόγηση εκφυλίζεται σε Round Robin, αφού η νεοεισαχθείσα διεργασία έχει μηδενικό waiting time in run queue και expected burst εξισορροπημένο σε σχέση με τις υπόλοιπες διεργασίες, εφόσον έχει ήδη τρέξει πιο πριν για κάποιους κύκλους. ***Το γεγονός ότι ο SJF εκφυλίζεται σε Round Robin στην περίπτωση που ο υπολογισμός του goodness γίνεται με βάση και το*** waiting time in run queue ***, βελτιώνει δραματικά την αποκρισιμότητα του συστήματος.***  Οι interactive διεργασίες χρησιμοποιούν για μικρό χρονικό διάστημα τον επεξεργαστή σε σχέση με τις non-interactive διεργασίες για αυτό είναι ιδιαίτερα σημαντικό η κατανομή του επεξεργαστή να γίνεται “δίκαια”, πράγμα που επιτυγχάνεται με τον συνυπολογισμό του waiting time in run queue. Από τα διαγράμματα, παρατηρούμε μικρή αυξομείωση των χρονικών διαστημάτων που περιμένουν οι διεργασίες στο running queue στην περίπτωση της δρομολόγησης με βάση το expected burst και waiting time in rq (min waiting time = 590ms, max waiting time = 720 ms). Όλες οι διεργασίες έχουν χρόνο αναμονής κοντά στο μέσο χρόνο αναμονής. Αντιθέτως στη δρομολόγησης μόνο με βάση το expected burst παρατηρούμε μεγαλύτερη αυξομείωση (min waiting time = 510ms, max waiting time = 1100ms), γεγονός που δείχνει την δικαιότερη διαμοίραση του επεξεργαστή στις διεργασίες στη πρώτη περίπτωση.

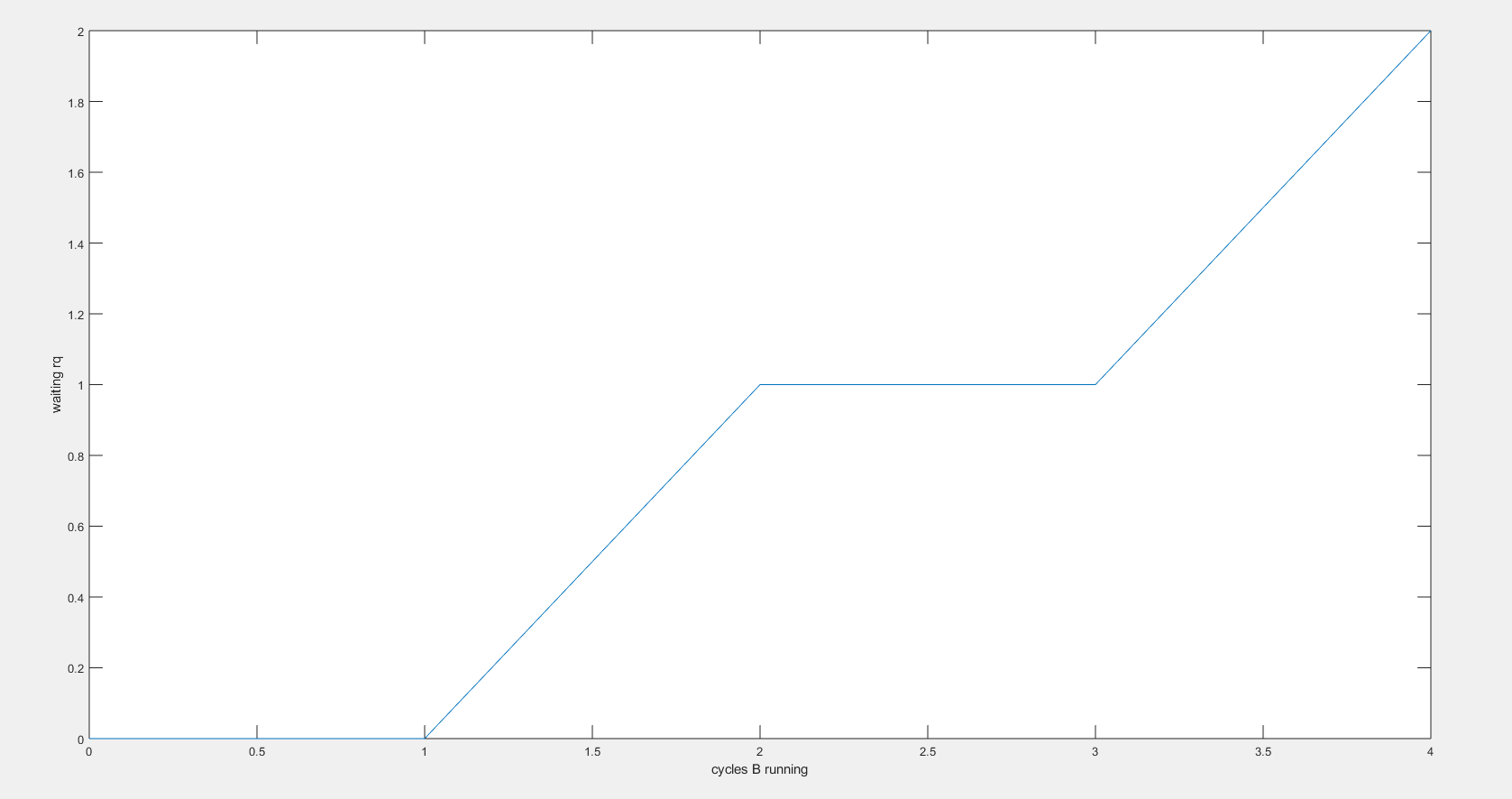
Για την διεργασία 2 κατασκευάσαμε τα εξής διαγράμματα για τους πρώτους κύκλους μετά την δημιουργία της:

Στο παρακάτω διάγραμμα ο κάθετος άξονας συμβολίζει το expected burst per cycle, ενώ ο οριζόντιος τους κύκλους που έχει τρέξει η διεργασία 2.

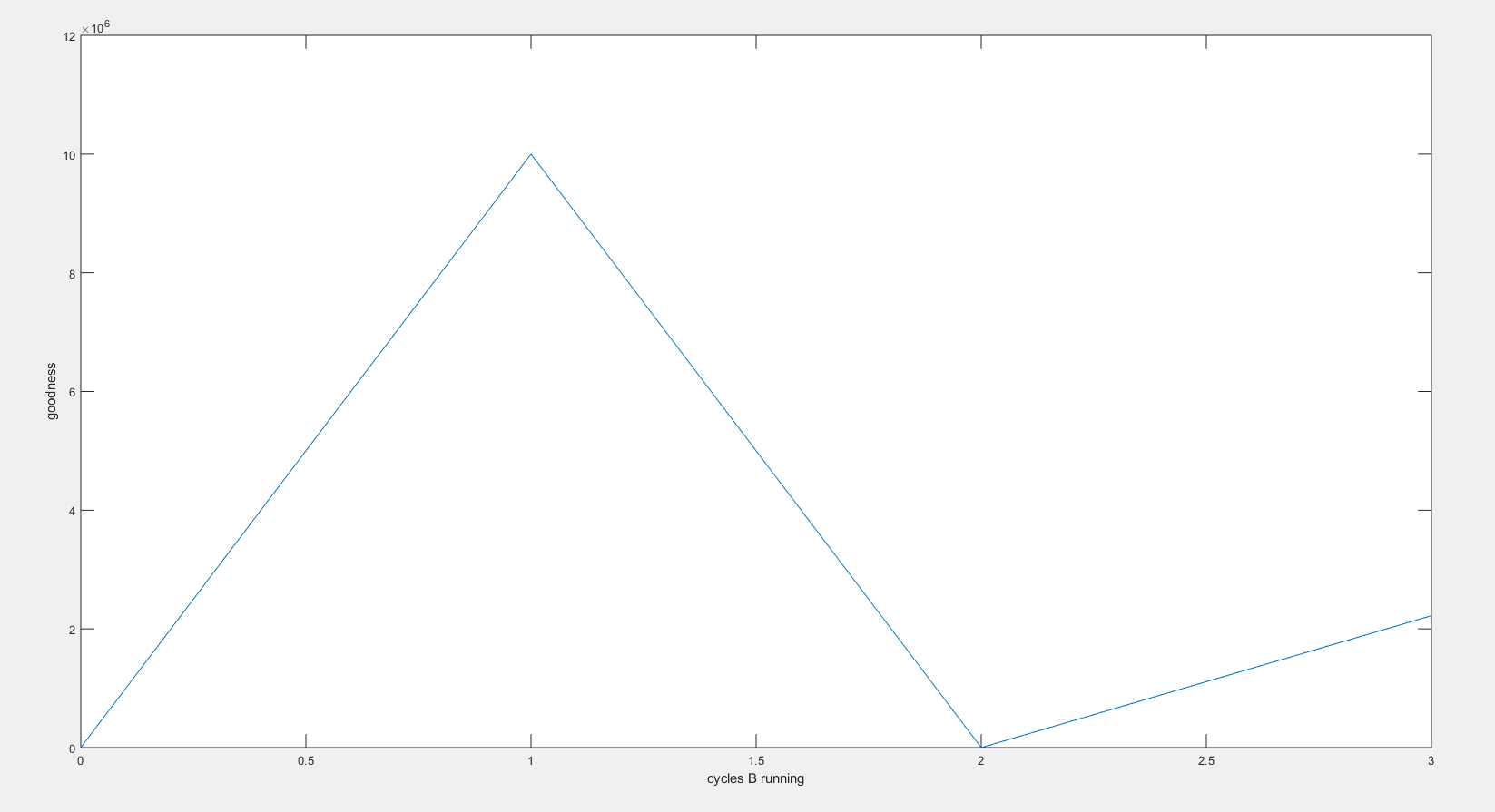


Σχήμα1

Στο παρακάτω διάγραμμα ο κάθετος άξονας συμβολίζει το waiting time in run queue, ενώ ο οριζόντιος τους κύκλους που έχει τρέξει η διεργασία 2.



Σχήμα 2

Στο παρακάτω διάγραμμα ο κάθετος άξονας συμβολίζει το goodness, ενώ ο οριζόντιος τους κύκλους που έχει τρέξει η διεργασία 2. 

Σχήμα 3

Στις χρονικές στιγμές t=0 και t = 2, οι τεταγμένες είναι ίσες με 1, δηλαδή είναι τα σημεία (0,1) και (2,1). Το οποίο λόγω κλίμακας δεν είναι διακριτό στο σχήμα.

Από τις μετρήσεις που πήραμε συμπεραίνουμε ότι στην περίπτωση που συνυπολογίζουμε το waiting time in run queue βελτιώνεται το μέσο Turn Around Time. Συγκεκριμένα:

Process(i)TurnAroundTime = WorkDuration(i) + RQ\_wait\_time(i).

AvgTurnAroundTime = Σ(Proces(i)TurnAroundTime) / #Processes  ( i = 0, 1, …, N ).

EXP\_BURST: AvgTurnAroundTime = 1.120 ms.

RQ\_WAIT\_TIME: AvgTurnAroundTime = 1.025 ms.

Άρα παρατηρούμε μείωση 8,4 %.

Πραγματοποιήσαμε το ίδιο πείραμα με διαφορετικό SEED στο profile.

SEED: 234

Κώδικας: example2b.conf

[endTime, rqWaitTime] [endTime, rqWaitTime]

Pi: SPAWNTIME: WAITING\_RQ: EXP\_BURST:

1/400 59ms [2406ms, 460ms] [2316ms, 350ms]

2/300 129ms [1407ms, 440ms] [1400ms, 280ms]

3/400 149ms [3655ms, 420ms] [4087ms, 480ms]

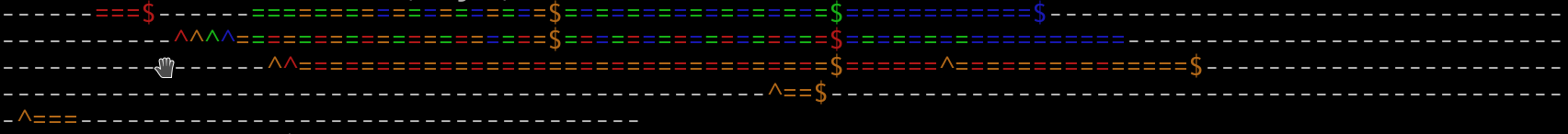
4/400 189ms [1499ms, 460ms] [1560ms, 530ms]

PROCESSES\_TIME\_LINE

Scheduling based on expected burst:



Scheduling based on Goodness (expected burst and waiting time in rq) :



Υπόμνημα:  
- / = : INIT / Current Process Running P1 P2 P3 P4  
$ : Interrupt ( process going to sleep )  
^ : Interrupt ( process wake up )

Scheduling based on Goodness (expected burst and waiting time in rq) : turn around time = 817,5ms   
Scheduling based on expected burst: turn around time = 777,5ms.

Άρα παρατηρούμε αύξηση 4,89%.

SEED: 1002

Κώδικας: example2b.conf

[endTime, rqWaitTime] [endTime, rqWaitTime]

Pi: SPAWNTIME: WAITING\_RQ: EXP\_BURST:

1 59ms [1723ms, 360ms] [1792ms, 380ms]

2 129ms [ 966ms, 320ms] [1755ms, 300ms]

3 149ms [1585ms, 470ms] [3052ms, 840ms]

4 189ms [2087ms, 140ms] [1412ms, 280ms]

PROCESSES\_TIME\_LINE

Scheduling based on expected burst:



Scheduling based on Goodness (expected burst and waiting time in rq) :



Scheduling based on Goodness (expected burst and waiting time in rq) : turn around time = 697.5ms.    
Scheduling based on expected burst: turn around time = 761.5ms.  .

Άρα παρατηρούμε μείωση 8,4 %.

Μέσα από αυτά τα πειράματα που ακολούθησαν, παρατηρούμε ότι δεν μπορούμε να βγάλουμε ασφαλή συμπεράσματα σχετικά με τη μεταβολή του turn around time συγκριτικά με τους διαφορετικούς αλγόριθμους δρομολόγησης του SJF. Και στα τρία σενάρια, στην περίπτωση της δρομολόγησης με βάση το expected burst και waiting time in rq παρατηρήσαμε ότι ο χρόνος αναμονής στο running queue για κάθε διεργασία είναι πιο κοντά στο μέσο χρόνο αναμονής, σε σχέση με τους χρόνους αναμονής στην περίπτωση δρομολόγησης μόνο με βάση το expected burst. Το γεγονός αυτό, δείχνει τη δικαιότερη διαμοίραση του επεξεργαστή στις διεργασίες στην πρώτη περίπτωση, κάτι που είναι αναγκαίο για τις interactive διεργασίες.

Profile 3:

Στο profile 3 δημιουργούμε δύο interactive και δύο non interactive διεργασίες. Η πρώτη interactive διεργασία δημιουργείται ταυτόχρονα με την πρώτη non interactive, ενώ η δεύτερη interactive μετά από χρονικό διάστημα 150ms, ενώ η δεύτερη non interactive, θυγατρική διεργασία της πρώτης non interactive περίπου στα 1000ms.

Πραγματοποιήθηκαν τρία πειράματα του ίδιου profile, αλλάζοντας κάθε φορά το SEED, προκειμένου να δούμε τις διαφορές μεταξύ των εκτελέσεων.

SEED:234 Κώδικας: example3a.conf

[End\_Time, rqWaitTime] [EndTime, rqWaitTime]

Pi: SPAWNTIME: TYPE: WAITING\_RQ: EXP\_BURST:

1 50ms I [5913ms, 1650ms] [6581ms, 1760ms]

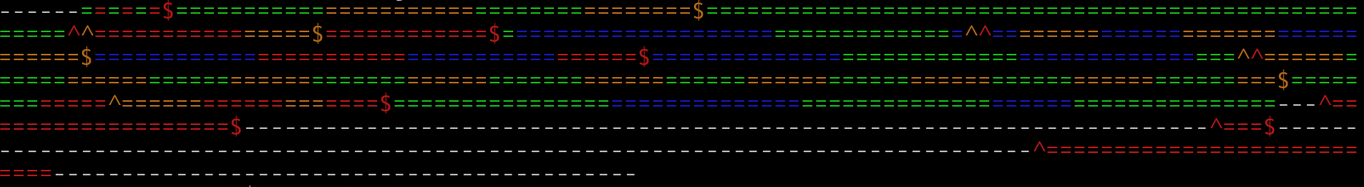
2 50ms N [4590ms, 2520ms] [4596ms, 2530ms]

3 200ms I [4995ms, 2000ms] [3937ms, 870ms]

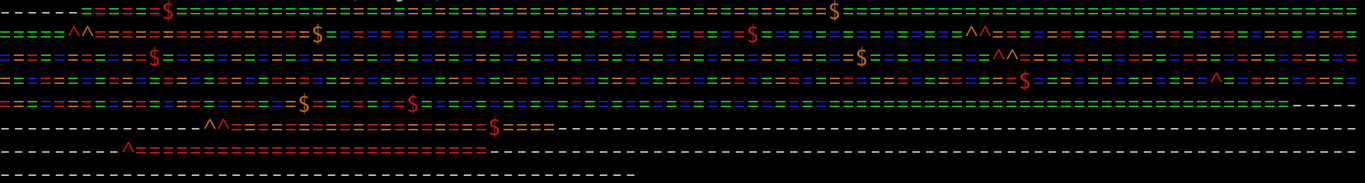
4 1138ms /1245ms N [4263ms, 2120ms] [4452ms, 2190ms]

PROCESSES\_TIME\_LINE

Scheduling based on expected burst:



Scheduling based on Goodness (expected burst and waiting time in rq) :



SEED:1 Κώδικας: example3b.conf

[End\_Time, rqWaitTime] [EndTime, rqWaitTime]

Pi: SPAWNTIME: TYPE: WAITING\_RQ: EXP\_BURST:

1/1000 50ms I [3984ms, 2280ms] [4677ms, 3550ms]

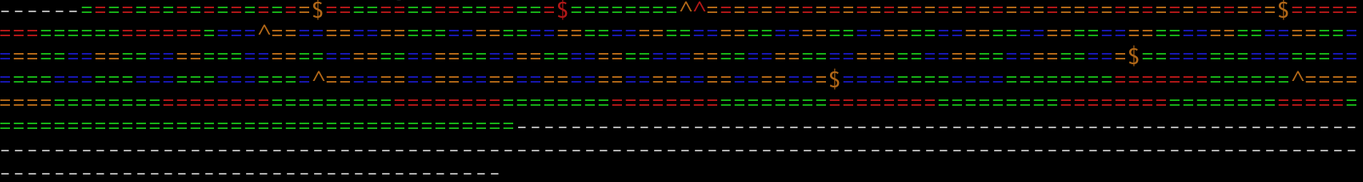
2/2000 50ms N [5050ms, 2970ms] [5050ms, 2980ms]

3/1000 200ms I [4381ms, 2390ms] [3754ms, 1420ms]

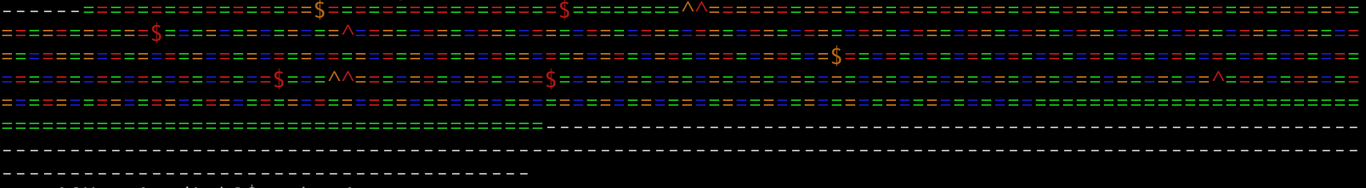
4/1000 1008ms /1030ms N [4441ms, 2420ms] [3483ms, 1460ms]

PROCESSES\_TIME\_LINE

Scheduling based on expected burst:



Scheduling based on Goodness (expected burst and waiting time in rq) :



SEED: 5656 Κώδικας: example3c.conf

[End\_Time, rqWaitTime] [EndTime, rqWaitTime]

Pi: SPAWNTIME: TYPE: WAITING\_RQ: EXP\_BURST:

1/1000 50ms I [4327ms, 2490ms] [4146ms, 2240ms]

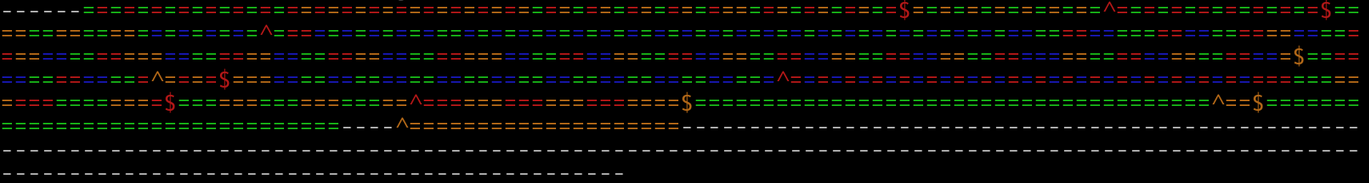
2/2000 50ms N [4954ms, 2870ms] [4865ms, 2790ms]

3/1000 200ms I [5255ms, 2140ms] [5080ms, 3020ms]

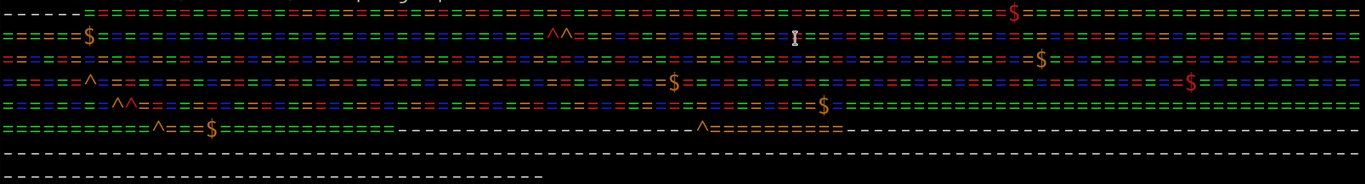
4/1000 1007ms /1010ms N [4338ms, 2310ms] [3668ms, 1660ms]

PROCESSES\_TIME\_LINE

Scheduling based on expected burst:



Scheduling based on Goodness (expected burst and waiting time in rq) :



Ανάλογα με τον τύπο που επιλέγουμε για τον υπολογισμό του goodness κάθε διεργασίας, υπάρχουν διαφορές ακόμα και στο spawn time της διεργασίας 4 που δημιουργείται από την 1, κάτι που προκαλεί διαφορές στη ροή εκτέλεσης. Τα interrupts δημιουργούνται σε διαφορετικά χρονικά διαστήματα ανάλογα με το seed, με αποτέλεσμα σε κάθε πείραμα να υπάρχει διαφορετικό σενάριο δρομολόγησης. Και στα τρία σενάρια, στην περίπτωση της δρομολόγησης με βάση το expected burst και waiting time in rq παρατηρήσαμε ότι ο χρόνος αναμονής στο running queue για κάθε διεργασία είναι πιο κοντά στο μέσο χρόνο αναμονής, σε σχέση με τους χρόνους αναμονής στην περίπτωση δρομολόγησης μόνο με βάση το expected burst. Έτσι ο επεξεργαστής μοιράζεται πιο δίκαια στην πρώτη περίπτωση κάτι που βελτιώνει την αποκρισιμότητα των interactive διεργασιών. Επιπλέον, παρατηρούμε ότι η συμπεριφορά των αλγορίθμων δρομολόγησης εξαρτάται από το workload και σε διαφορετικά σενάρια εκτέλεσης κάποιος αλγόριθμος συμπεριφέρεται καλύτερα από τον άλλον.

Profile 4:

Στο profile4 δημιουργούμε την χρονική στιγμή 59ms μια interactive διεργασία και μετά τη χρονική στιγμή 700ms δημιουργούμε ταυτόχρονα δύο non interactive διεργασίες. Τέλος την χρονική στιγμή 2400ms δημιουργούμε μια ακόμα non interactive διεργασία. Οι διεργασίες δεν έχουν work duration και τερματίζουν με την λήξη του virtual machine.

Kώδικας: example4.conf

[endTime, rqWaitTime] [endTime, rqWaitTime]

Pi: SPAWNTIME: TYPE: WAITING\_RQ: EXP\_BURST:

1 59ms I [3000ms, 1190ms] [3000ms, 2280ms]

2 700ms N [3000ms, 1420ms] [3000ms, 1350ms]

3 700ms N [3000ms, 1420ms] [3000ms, 1180ms]

4 2400ms I [3000ms, 120ms] [3000ms, 220ms]

PROCESSES\_TIME\_LINE

EXP\_BURST:



RQ\_WAIT\_TIME:



Το συγκεκριμένο παράδειγμα δείχνει ένα από τα μεγαλύτερα μειονεκτήματα του αλγορίθμου δρομολόγησης με βάση το exp\_burst. Η διεργασία 1 που αρχικά μονοπωλεί τον επεξεργαστή, χάνει τον επεξεργαστή με το που εμφανιστούν οι δύο νέες διεργασίες που έχουν πολύ μικρότερο exp\_burst. Αυτό μάλιστα μπορεί να οδηγήσει ακόμα και σε λιμοκτονία, στην περίπτωση όπου δημιουργούνται συνεχώς καινούργιες διεργασίες. Το γεγονός ότι η διεργασία 1 είναι interactive, μειώνει δραματικά την αποκρισιμότητα του συστήματος. Στην περίπτωση δρομολόγησης με βάση το waiting\_rq, επειδή η ,παρατηρούμε μια δίκαια διαμοίραση του επεξεργαστή στις διεργασίες, κάτι που αποκλείει την περίπτωση λιμοκτονίας.

ΓΕΝΙΚΕΣ ΠΑΡΑΤΗΡΗΣΕΙΣ

Βασικότερη αδυναμία του SJF είναι ότι χρειάζεται γνώση του χρονικού διαστήματος που απομένει για την κάθε διεργασία. Εφόσον αυτό είναι δύσκολο να γίνει σε συστήματα γενικού σκοπού κάνουμε μια πρόβλεψη για τον χρόνο που θα τρέξει στο μέλλον με βάση το χρόνο που έτρεξε στο παρελθόν. Στην συγκεκριμένη υλοποίηση του VM χρησιμοποιούμε δύο προσεγγιστικούς τύπους για τον υπολογισμό της προτεραιότητας. Στον πρώτο λαμβάνουμε υπ’ όψιν μόνο το προβλεπόμενο μελλοντικό χρόνο ξεσπάσματος, ενώ στον δεύτερο συνυπολογίζουμε και το χρονικό διάστημα που περιμένει μία διεργασία στην ουρά δρομολόγησης. Σύμφωνα με τα πειράματα που πραγματοποιήσαμε παρατηρούμε ότι στην δεύτερη περίπτωση η διαμοίραση του επεξεργαστή στις διεργασίες γίνεται πιο “δίκαια’’, με αποτέλεσμα να βελτιώνεται σημαντικά η αποκρισιμότητα των interactive διεργασιών. Επίσης παρατηρούμε μείωση του μέσου turn around time για profiles non-interactive διεργασιών. Για profiles interactive ή συνδυασμό interactive και non-interactive διεργασιών λόγω των τυχαίων I/O-interrupts δεν μπορούμε να βγάλουμε συμπεράσματα για το μέσο turn around time. Ένα βασικό μειονέκτημα της προσέγγισης με το expected burst είναι και η πιθανότητα της λιμοκτονίας, στην περίπτωση που δημιουργούνται συνεχώς νέες διεργασίες.

Στην προσέγγιση του SJF με βάση τον υπολογισμό του expected burst, δεν γίνεται τόσο “δίκαιη” διαμοίραση του επεξεργαστή με αποτέλεσμα οι διεργασίες να κρατάν τον επεξεργαστή για μεγαλύτερα χρονικά διαστήματα, δηλαδή λιγότερες εναλλαγές διεργασιών στον επεξεργαστή (λιγότερα context switches). Όμως, λόγω του μηδενικού κόστους των context switch στο VM αυτή η διαφορά δεν είναι ορατή, με αποτέλεσμα ο αλγόριθμος δρομολόγησης με βάση το expected burst και waiting time in rq , να φαίνεται πιο αποδοτικός. Επιπλέον, παρατηρούμε ότι στην περίπτωση που το workload περιλαμβάνει interactive διεργασίες και ο χρόνος ενός κύκλου μηχανής είναι πολύ μεγαλύτερος από την διάρκεια του I/O αυτών, δηλαδή το I/O αρχίζει και τελειώνει στον ίδιο κύκλο μηχανής για την διεργασία που έχει τον επεξεργαστή εκείνη τη χρονική στιγμή, ο SJF εκφυλίζεται σε RR. Επομένως η συμπεριφορά είναι παρόμοια με αυτή που βλέπουμε και σε workloads που αποτελούνται μόνο από non-interactive διεργασίες.

Γενικά δεν υπάρχει κάποιος αλγόριθμος δρομολόγησης που να υπερτερεί σε σχέση με τον άλλον για οποιοδήποτε workload.