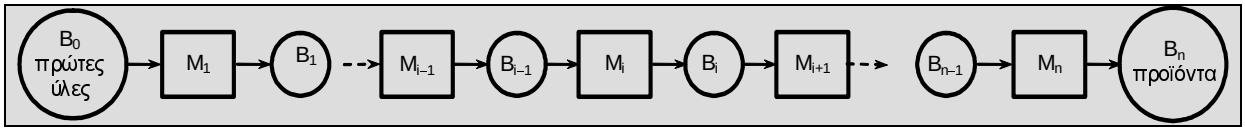
ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ
ΠΑΡΑΓΩΓΗΣ & ΔΙΟΙΚΗΣΗΣ

Διπλωματική Εργασία:

«Ένας Ταχύς Αλγόριθμος Προσομοίωσης Γραμμών Παραγωγής»



Αναλυτής: Χριστοφορούδης Αντώνης

Τριμελής εξεταστική επιτροπή:

Κουϊκόγλου Βασίλης (επιβλέπων)

Νικολός Ιωάννης

Κοσματόπουλος Ηλίας

Χανιά, 2005

ΠΡΟΛΟΓΟΣ

Στην αρχή της εργασίας μου, νιώθω την ανάγκη να ευχαριστήσω ορισμένους ανθρώπους:

Τον πατέρα μου, Δημήτρη, τη μητέρα μου, Πηνελόπη και την αδελφή μου, Κατερίνα. Την οικογένειά μου.

Τον επιβλέποντα Καθηγητή μου Β. Κουικόγλου, γιατί μου πρόσφερε παραπάνω από όσα δίσταζα να ζητήσω.

Τους Καθηγητές Ι. Νικολό και Η. Κοσματόπουλο για τη συμμετοχή τους στην εξεταστική επιτροπή.

Όλους τους φίλους και γνωστούς που έκανα στα Χανιά, για την ηθική και, πολλές φορές, υλική υποστήριξη που συνεχίζουν να μου προσφέρουν.

Εμένα, που το 1999 έβαλα στη σωστή θέση του Μηχανογραφικού μου τη σχολή των ΜΠΔ. Καλή επιλογή.

ΕΙΣΑΓΩΓΗ

Βασικοί ορισμοί και προβλήματα στις γραμμές παραγωγής

Στην αρχή της εργασίας περιγράφονται τα βασικά στοιχεία από τα οποία αποτελούνται οι γραμμές παραγωγής. Έπειτα, διατυπώνονται τα προβλήματα ανάλυσης και σύνθεσης που αφορούν τη βέλτιστη λειτουργία τέτοιων γραμμών. Τέλος, περιγράφεται συνοπτικά η μέθοδος προσομοίωσης που χρησιμοποιείται για την ανάλυση. Για περισσότερες λεπτομέρειες, βλέπε [1].

Οι *σταθμοί παραγωγής* αποτελούνται από τις μηχανές που δέχονται ακατέργαστα κομμάτια ή πρώτες ύλες και παράγουν έτοιμα προϊόντα.

Χώροι αποθήκευσης διαμορφώνονται μεταξύ των σταθμών παραγωγής συμβάλλοντας στο συγχρονισμό των λειτουργιών των σταθμών και στην αύξηση της παραγωγικότητας.

Γραμμή παραγωγής είναι ένα σύνολο από σταθμούς παραγωγής συνδεδεμένους σε σειρά με ενδιάμεσες αποθήκες, στην οποία κάποιο είδος δέχεται διαδοχικές κατεργασίες μέχρι τον τελευταίο σταθμό και κατόπιν φεύγει από τη γραμμή.

Δύο είναι τα προβλήματα που αφορούν τη λειτουργία των γραμμών παραγωγής: η *ανάλυση* και η *σύνθεση*. Η ανάλυση των γραμμών παραγωγής γίνεται με τη βοήθεια μοντέλων προσομοίωσης, τα οποία εξετάζουν τα διάφορα γεγονότα που λαμβάνουν χώρα στο σύστημα. *Ανάλυση* είναι η μέθοδος με την οποία υπολογίζονται οι μέσοι ρυθμοί παραγωγής, οι μέσες στάθμες στις αποθήκες κι άλλες παράμετροι που χαρακτηρίζουν μία γραμμή παραγωγής. *Σύνθεση*, εξάλλου, είναι το σύνολο των αποφάσεων που λαμβάνονται για τη σχεδίαση μιας νέας γραμμής παραγωγής, τη μεταβολή μιας υπάρχουσας και τον προγραμματισμό λειτουργίας της. Η λήψη αποφάσεων για το μέγεθος των αποθηκών, το είδος των μηχανών και τον τρόπο κυκλοφορίας των προϊόντων μέσα στη γραμμή, επηρεάζει τους ρυθμούς παραγωγής, τις μέσες στάθμες αποθηκών και κατά συνέπεια το κόστος λειτουργίας της. Έτσι, τις τελευταίες δεκαετίες υπάρχει συνεχές ερευνητικό ενδιαφέρον σχετικά με την ανάλυση γραμμών παραγωγής.

Η κατάσταση του συστήματος κάποια χρονική στιγμή περιγράφεται συνήθως από τις στάθμες των αποθηκών και τις καταστάσεις των μηχανών (λειτουργικές ή υπό επισκευή).

Για την επίλυση του προβλήματος χρησιμοποιούνται οι αναλυτικές μέθοδοι, καθώς επίσης και τεχνικές προσομοίωσης σε υπολογιστή. Η εργασία επικεντρώνεται στη δεύτερη περίπτωση αντιμετώπισης, στην οποία το μοντέλο μιμείται τη λειτουργία του συστήματος για μία ορισμένη χρονική περίοδο. Ο υπολογιστής παράγει μία ακολουθία «τυχαίων αριθμών», οι οποίοι αντιστοιχούν στις διαδοχικές χρονικές στιγμές που επισκευάζονται οι μηχανές ή πληρούνται ή αδειάζουν οι αποθήκες.

Η γραμμή αποτελείται από μηχανές για τη διαδοχική κατεργασία και μετατροπή ακατέργαστων αρχικά κομματιών σε έτοιμα προϊόντα. Στα ενδιάμεσα στάδια της παραγωγής τα κομμάτια αποθηκεύονται σε χώρους πεπερασμένης χωρητικότητας. Κατά τη διάρκεια της λειτουργίας του συστήματος, οι μηχανές υφίστανται βλάβες κι επισκευάζονται τυχαία, ενώ οι χώροι των αποθηκών γεμίζουν ή αδειάζουν. Για την αναπαράσταση των γεγονότων αυτών χρησιμοποιείται κατάλληλος συμβολισμός. Στη συνέχεια εξετάζεται αναλυτικά η επίδραση των γεγονότων στον τρόπο λειτουργίας των μηχανών κι εξάγονται οι εξισώσεις κατάστασης της γραμμής παραγωγής. Οι εξισώσεις αυτές είναι στοχαστικές εξισώσεις για διακεκριμένα γεγονότα και η υλοποίησή τους γίνεται με την προσομοίωση του συστήματος στον υπολογιστή.

Η έρευνα αυτή έγινε πάνω σε γραμμές παραγωγής, που παράγουν ένα είδος προϊόντος σε τεμάχια. Αναπτύσσονται δύο αλγόριθμοι για την μελέτη τους, ένας λεπτομερής κι ακριβής, βασιζόμενος σε κλασικό μοντέλο προσομοίωσης, κι ένας νέος, που συνδυάζει την ταχύτητα σε υπολογιστικό χρόνο με την ακρίβεια στα αποτελέσματα για σύστημα δύο μηχανών και την πολύ μικρή απόκλιση για συστήματα τριών μηχανών και πάνω.

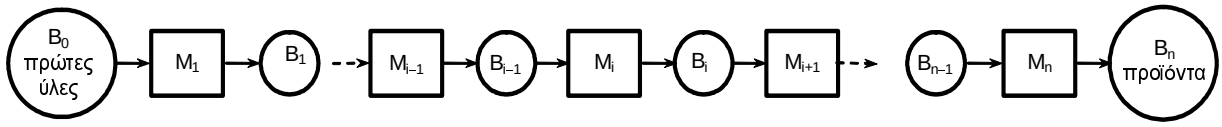
1

ΓΡΑΜΜΕΣ ΠΑΡΑΓΩΓΗΣ

Στο κεφάλαιο αυτό μελετάμε μια απλή γραμμή παραγωγής με n μηχανές και n–1 αποθήκες, την οποία και αναλύουμε παρακάτω με δύο μοντέλα προσομοίωσης. Το πρώτο είναι απλό και λεπτομερές, ενώ το δεύτερο είναι γρήγορο και αποτελεσματικότερο, χάρη στη χρήση πιο πολύπλοκων εξισώσεων για τη μείωση των γεγονότων που πρέπει να παρακολουθούνται.

1.1. ΠΕΡΙΓΡΑΦΗ ΣΥΣΤΗΜΑΤΟΣ

Θεωρούμε μια γραμμή παραγωγής με n μηχανές M_1, M_2, ..., M_n που υπόκεινται σε βλάβες και n – 1 ενδιάμεσες αποθήκες B_1, B_2, …, B_{n-1}. Τα κομμάτια προς επεξεργασία φορτώνονται στην πρώτη μηχανή από μια πηγή άπειρων πρώτων υλών B_0. Όταν ολοκληρωθεί η κατεργασία τους από τη μηχανή προχωρούν και αποθηκεύονται στην αποθήκη που ακολουθεί. Κατόπιν συνεχίζουν προς τη μηχανή M_2, την αποθήκη B_2, ..., τη μηχανή M_n και καταλήγουν σε μια αποθήκη άπειρης χωρητικότητας B_n. Η διαδικασία συνεχίζεται έως ότου το σύστημα ολοκληρώσει έναν προκαθορισμένο όγκο παραγωγής. Θεωρούμε ότι ο χρόνος μεταφοράς ενός κομματιού από μια μηχανή στην επόμενη είναι αμελητέος. Το σύστημα απεικονίζεται στο Σχ. 1.1.



Σχήμα 1.1. Γραμμή παραγωγής με n μηχανές

Τα κομμάτια είναι πανομοιότυπα και ο χρόνος κατεργασίας για κάθε μηχανή είναι ίσος προς μια σταθερή ποσότητα Δ. Η μηχανή M_i τροφοδοτεί την αποθήκη B_i που έχει χωρητικότητα BC_i κομμάτια, που περιμένουν να επεξεργαστούν από τη M_{i+1}. Η μηχανή M_i αποκλείεται όταν είναι έτοιμη να ελευθερώσει ένα κομμάτι και η αποθήκη της είναι γεμάτη. Δυαδικά, η M_{i+1} αποστερείται όταν είναι έτοιμη να δεχθεί ένα κομμάτι και η αποθήκη που την τροφοδοτεί είναι άδεια. Παρόλα αυτά, η πρώτη μηχανή δεν αποστερείται ποτέ ενώ η τελευταία δεν μπλοκάρει ποτέ. Τα φαινόμενα αποκλεισμού και αποστέρησης αναγκάζουν τις μηχανές που επηρεάζονται να προσαρμόσουν το χρόνο κατεργασίας τους, παρατείνοντάς τον ανάλογα με την αιτία. Στην πράξη, εαν μια μηχανή αποστερηθεί, περιμένει μέχρι να υπάρξει διαθέσιμο προς επεξεργασία κομμάτι. Μετά το επεξεργάζεται με ρυθμό Δ και το ελευθερώνει προς την επόμενη αποθήκη, αλλά περιμένει ξανά μέχρι το επόμενο διαθέσιμο κομμάτι και

ούτω καθεξής. Η μακροσκοπική της συμπεριφορά αναπαριστά παραγωγή με βραδύτερο ρυθμό.

Μαζί με τον αποκλεισμό και την αποστέρηση, οι μηχανές μπορεί να αναγκαστούν να πάψουν να λειτουργούν προσωρινά, εξαιτίας διακοπής ρεύματος, μηχανικής βλάβης, αλλαγής εργαλείων, και προληπτικής συντήρησης. Διακοπές ρεύματος συμβαίνουν σε τυχαίες χρονικές στιγμές και είναι γνωστές ως βλάβες που εξαρτώνται από το χρόνο (χρονικής εξάρτησης). Τα άλλα γεγονότα εξαρτώνται από τη λειτουργία ή μη της μηχανής. Οι διακοπές εξαιτίας μηχανικής βλάβης και αλλαγής εργαλείων οφείλονται στη φθορά χρήσης και λαμβάνουν χώρα μετά την παραγωγή ορισμένου αριθμού κομματιών. Η προληπτική συντήρηση συνήθως προβλέπεται μετά από έναν ορισμένο όγκο παραγωγής. Όλα αυτά τα φαινόμενα μπορούν εύκολα να ληφθούν υπόψη κατά τη διάρκεια της προσομοίωσης με την εισαγωγή κατάλληλών γεγονότων και μεταβλητών κατάστασης.

Για να διατηρήσουμε το μοντέλο απλό, εξετάζουμε μόνον βλάβες που εξαρτώνται από τη λειτουργία. Επιπρόσθετα, και χωρίς απώλεια της γενικότητας, υποθέτουμε ότι η πιθανότητα παραγωγής ενός κομματιού σε έναν κύκλο παραγωγής $1 - f_i$ είναι σταθερή για κάθε μηχανή M_i. Η συμπληρωματική αυτής της πιθανότητας είναι η πιθανότητα βλάβης f_i σε έναν κύκλο παραγωγής. Έστω F_i ο αριθμός κομματιών μέχρι τη βλάβη της M_i. Τότε

$$P(F_i = n) = (1 - f_i)^n \; f_i \tag{1.1}$$

που είναι η γεωμετρική κατανομή με παράμετρο f_i, $0 \le f_i < 1$.

Σύμφωνα με τον αντίστροφο μετασχηματισμό της γεωμετρικής κατανομής [2], μια γεννήτρια τυχαίων τιμών για τον αριθμό των κομματιών μέχρι τη βλάβη, είναι η ακόλουθη:

(a) Γέννησε έναν τυχαίο αριθμό $u \in (0, 1)$.

(b) Θέσε

$$\text{αριθμός κομματιών μέχρι τη βλάβη} = \left\lfloor \frac{\ln u}{\ln (1 - f_i)} \right\rfloor \tag{1.2}$$

όπου $\lfloor x \rfloor$ είναι ο μικρότερος ακέραιος, τέτοιος ώστε $\lfloor x \rfloor < x$.

Τέλος, για απλούστευση και μόνον, υποθέτουμε ότι ο χρόνος επισκευής TTR_i (Time-To-Repair) της M_i ακολουθεί επίσης τη γεωμετρική κατανομή, με πιθανότητα επισκευής r_i, οπότε

$$P(\text{χρόνος επισκευής} = t) = (1 - r_i)^t \; r_i$$

Η παράμετρος r_i παριστάνει τον αριθμό επισκεύων που μπορούν να ολοκληρωθούν σε μια χρονική μονάδα. Ξανά, εφαρμόζοντας αντίστροφο μετασχηματισμό, παίρνουμε τη γεννήτρια για τη διάρκεια μίας επισκευής

$$\text{Διάρκεια επισκευής} = \left\lfloor \frac{\ln u}{\ln (1 - r_i)} \right\rfloor \tag{1.3}$$

Το μοντέλο μπορεί να δεχθεί οποιεσδήποτε κατανομές βλαβών κι επισκευών.

2

ΛΕΠΤΟΜΕΡΕΣ ΜΟΝΤΕΛΟ ΠΡΟΣΟΜΟΙΩΣΗΣ

Το πρώτο μοντέλο είναι απλό και λεπτομερές. Χρησιμοποιεί κλασική μεθοδολογία προσομοίωσης, ενώ η περιγραφή του είναι παρόμοια με εκείνη που παρουσιάζεται στο [4].

2.1. ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΚΡΙΤΩΝ ΓΕΓΟΝΟΤΩΝ PP (PIECE-BY-PIECE SIMULATION)

Τώρα θα αναπτύξουμε ένα τυπικό μοντέλο για να προσομοιώσουμε τη λειτουργία του συστήματος παραγωγής σε χρόνο $[0, t_{max}]$. Έστω ότι t παριστάνει τον χρόνο του ρολογιού προσομοίωσης. Ξεκινάμε με τις μεταβλητές που περιγράφουν την κατάσταση του σύστηματος σε οποιοδήποτε χρόνο t κατά τη διάρκεια μιας προσομοίωσης: $BL_i(t)$ είναι η στάθμη της αποθήκης B_i σε χρόνο t, $P_i(t)$ η συνολική παραγωγή της μηχανής M_i σε χρόνο t, και $s_i(t)$ η κατάσταση της M_i όπου

$$s_i(t) = \begin{cases} 0 & \text{εαν η } M_i \text{ είναι αποστερημένη} \\ 1 & \text{εαν η } M_i \text{ δεν είναι ούτε αποστερημένη ούτε αποκλεισμένη} \\ 2 & \text{εαν η } M_i \text{ είναι αποκλεισμένη} \end{cases}$$

Το μοντέλο βασίζεται στην παρατήρηση κάθε κομματιού όταν αναχωρεί από κάποια μηχανή. Αυτό ισοδυναμεί με ένα γεγονός που το ονομάζουμε αναχώρηση. Αναχώρηση συμβαίνει όταν η M_i παράγει ένα κομμάτι, το στέλνει στην αποθήκη B_i, κι αφαιρεί ένα νέο από την προηγούμενη αποθήκη B_{i-1} για να αρχίσει νέο κύκλο παραγωγής.

Θεμελιώδης στην ανάπτυξη του μοντέλου είναι η διαδοχή των στιγμών που ολοκληρώνεται η κατεργασία των κομματιών σε μια μηχανή οπότε και είναι έτοιμα να περάσουν στην επόμενη αποθήκη. Έστω t ο χρόνος στον οποίο η μηχανή M_i αρχίζει να επεξεργάζεται ένα κομμάτι. Το t μπορεί να είναι η στιγμή που η M_i ελευθέρωσε το προηγούμενο κομμάτι στην αποθήκη B_i και δέχεται αυτό το κομμάτι από τη B_{i-1}, ή η στιγμή που το κομμάτι αυτό ελευθερώνεται από τη M_{i-1} και πηγαίνει απ' ευθείας στην M_i, εαν η M_i τυχαίνει να είναι αποστερημένη. Η διάρκεια του κύκλου παραγωγής για το κομμάτι αυτό είναι ίση προς το άθροισμα του καθαρού χρόνου κατεργασίας Δ και του συνολικού χρόνου για επισκευές, εφόσον κατά τη διάρκεια της παραγωγής του κομματιού η M_i υποστεί μία ή περισσότερες βλάβες. Η σχέση (1.3) δίνει τη διάρκεια μίας περιόδου επισκευής. Ο συνολικός χρόνος διακοπής κατά την παραγωγή του κομματιού, είναι

$$\begin{pmatrix} \text{συνολική διακοπή κατά} \\ \text{έναν κύκλο παραγωγής} \end{pmatrix} = \sum_{n=1}^{\substack{\text{αριθμός} \\ \text{βλαβών}}} (\text{διάρκεια επισκευής της n} - \text{οστής βλάβης}) \qquad (2.1)$$

Επομένως, ο χρόνος στον οποίο ολοκληρώνεται η κατεργασία για το υπό εξέταση κομμάτι, οπότε κι αυτό είναι έτοιμο να αναχωρήσει από την M_i, είναι

$$T_{Mi} = t + \begin{pmatrix}\text{συνολική διακοπή κατά}\\ \text{έναν κύκλο παραγωγής}\end{pmatrix} + \Delta \qquad (2.2)$$

Ο χρόνος T_{Mi} είναι ο χρόνος επόμενου γεγονότος της μηχανής M_i. Κάθε χρονική στιγμή, η κάθε μηχανή έχει το δικό της χρόνο επόμενου γεγονότος. Ο προσομοιωτής παρακολουθεί την εξέλιξη του συστήματος προσαυξάνοντας το ρολόι της προσομοίωσης t μέχρι το χρόνο που θα συμβεί το πιο άμεσο γεγονός, δηλαδή

$$t = \min_i T_{Mi} \qquad (2.3)$$

Οι εξισώσεις (2.1)–(2.3) είναι οι εξισώσεις προγραμματισμού επόμενου γεγονότος για το λεπτομερές μοντέλο προσομοίωσης.

Κατόπιν, περιγράφουμε πώς τα γεγονότα επηρεάζουν την κατάσταση του συστήματος. Μελετούμε μία γενική μηχανή M_i του συστήματος με μία προηγούμενη αποθήκη B_{i-1} και μία επόμενη B_i. Έτσι, θα μπορέσουμε να περιγράψουμε εύκολα τη γραμμή παραγωγής που, όπως αναφέραμε, είναι της μορφής $M_1 \to B_1 \to \ldots \to B_{i-1} \to M_i \to B_i \to \ldots$.

Υποθέτουμε ότι τη στιγμή $t = T_{Mi}$ λαμβάνει χώρα το γεγονός i, δηλαδή η μηχανή M_i τελειώνει με την κατεργασία ενός κομματιού και είναι έτοιμη να το στείλει στη επόμενη αποθήκη B_i. Θεωρούμε τις ακόλουθες περιπτώσεις για το γεγονός i:

(A) η επόμενη αποθήκη είναι γεμάτη και η μηχανή αποκλείεται

(B) το επεξεργασμένο κομμάτι ελευθερώνεται στη B_i ή στέλνεται απευθείας στη M_{i+1}, αν αυτή η μηχανή ήταν αποστερημένη στο χρόνο t

(C) η M_i είναι έτοιμη να αρχίσει νέο κύκλο παραγωγής αλλά γίνεται αποστερημένη γιατί δεν υπάρχει διαθέσιμο προς κατεργασία κομμάτι

(D) η M_i δεν είναι αποστερημένη και παίρνει ένα κομμάτι από τη B_{i-1} σε χρόνο t· αν η προηγούμενη μηχανή M_{i-1} τυχαίνει να είναι αποκλεισμένη, τότε αυτόματα παύει να είναι.

Οι περιπτώσεις A-D περιγράφονται αναλυτικά παρακάτω.

(A) Η M_i αποκλείεται: Εαν η επόμενη αποθήκη B_i είναι γεμάτη, τότε η μηχανή αποκλείεται αμέσως. Παρόλα αυτά, πρέπει να διαχωρίσουμε την περίπτωση όπου η χωρητικότητα της B_i είναι μηδέν, η επόμενη μηχανή M_{i+1} είναι αποστερημένη, και το επεξεργασμένο κομμάτι ελευθερώνεται απευθείας προς τη M_{i+1}. Λόγω αυτού, η συνθήκη αποκλεισμού είναι $\{BL_i = BC_i\}$ και $\{s_{i+1} \neq 0\}$. Όταν η συνθήκη ισχύει, η λειτουργία της M_i αναστέλλεται μέχρι το χρόνο T_{Mi+1} οπότε η M_{i+1} ελευθερώνει το κομμάτι που κατεργάζεται και παίρνει ένα νέο από την ενδιάμεση αποθήκη ή από τη M_i απευθείας (εαν $BC_i = 0$). Επειδή ο χρόνος αυτός μπορεί να είναι μην γνωστός εκ των προτέρων (εξαιτίας της πιθανότητας η M_{i+1} να είναι επίσης αποκλεισμένη), μπορούμε με ασφάλεια να θέσουμε $T_{Mi} = \infty$ για να εξαιρέσουμε το χρόνο T_{Mi} από το σύνολο των υποψήφιων χρόνων επόμενου γεγονότος στην Εξ. (2.3). Τότε, η Περίπτωση D, που περιγράφουμε στη συνέχεια, διασφαλίζει ότι ο αλγόριθμος θα εκτελέσει μια αναχώρηση από

την M_i αμέσως μετά την αναχώρηση κομματιού από την M_{i+1}. Συνοψίζοντας, ο αποκλεισμός της M_i εκφράζεται ως ακολούθως:

$$\text{αν } \{BL_i = BC_i\} \text{ και } \{s_{i+1} \neq 0\}$$

$$\text{τότε } \{s_i = 2\} \text{ και } \{T_{Mi} = \infty\}$$

Στον υπολογιστή, το ∞ αναπαρίσταται από το μέγιστο επιτρεπόμενο σταθερό αριθμό, ή από έναν αριθμό μεγαλύτερο από την καθορισμένη διάρκεια της προσομοίωσης t_{max}, για παράδειγμα $t_{max} + 1$.

(B) Ελευθερώνεται ένα κομμάτι: Αν η M_i δεν είναι αποκλεισμένη, το κομμάτι προωθείται στην B_i και η συνολική παραγωγή P_i θα αυξηθεί κατά ένα. Το κατεργασμένο κομμάτι θα μπει στην B_i της οποίας το επίπεδο θα αυξηθεί κατά ένα, εκτός αν η επόμενη μηχανή είναι αποστερημένη, δηλαδή $s_{i+1} = 0$. Στην τελευταία περίπτωση, το τεμάχιο θα σταλεί απ' ευθείας στην M_{i+1}, που θα αρχίσει έναν νέο κύκλο παραγωγής άμεσα. Ο αντίστοιχος χρόνος T_{Mi+1} πραγματοποίησης του επόμενου γεγονότος υπολογίζεται από την Εξ. (2.2).

(C) Η M_i αποστερείται: Αφότου το κατεργασμένο κομμάτι ελευθερωθεί, η M_i είναι έτοιμη να πάρει ένα νέο από τη B_{i-1} και να ξεκινήσει έναν νέο κύκλο παραγωγής. Εαν η B_{i-1} είναι άδεια και η M_{i-1} δεν είναι αποκλεισμένη*, τότε η M_i θα γίνει αποστερημένη. Το φαινόμενο αυτό είναι το δυαδικό του αποκλεισμού και, με την ίδια λογική όπως στην Περίπτωση Α, το εκφράζουμε ως εξής:

$$\text{αν } \{BL_{i-1} = 0\} \text{ και } \{s_{i-1} \neq 2\}$$

$$\text{τότε } \{s_i = 0\} \text{ και } \{T_{Mi} = \infty\}$$

(D) Η M_i ξεκινά νέο κύκλο παραγωγής: Αν η M_i δεν είναι αποστερημένη, τότε η στάθμη της B_{i-1} μειώνεται κατά ένα, και ο χρόνος T_{Mi} υπολογίζεται από την Εξ. (2.2). Αν η M_{i-1} τυχαίνει να ήταν αποκλεισμένη, απελευθερώνεται αμέσως μια και τώρα υπάρχει μια μονάδα χώρου διαθέσιμη στην B_{i-1} για το κομμάτι που την απόκλεισε. Έτσι, η M_{i-1} τίθεται άμεσα σε λειτουργία και ο αλγόριθμος εκτελεί το γεγονός $(i-1)$ στο χρόνο t. Το γεγονός αυτό μπορεί να προκαλέσει μια σειρά από παρόμοια γεγονότα πριν την M_i έτσι ώστε να επανεργοποιήσει την αλυσίδα των μηχανών M_{i-1}, M_{i-2}, …, που είχαν αποκλειστεί από την M_i.

Από τα παραπάνω, είναι προφανές ότι η παραγματοποίηση ενός γεγονότος στη M_i μπορεί να ενεργοποιήσει δευτερεύοντα γεγονότα που επηρεάζουν τις προηγούμενες κι επόμενες αποθήκες. Το ροϊκό διάγραμμα του Σχ. 2.2 απεικονίζει τις παραπάνω περιπτώσεις. Συνοπτικά, το κλασικό μοντέλο προσομοίωσης λειτουργεί ως εξής:

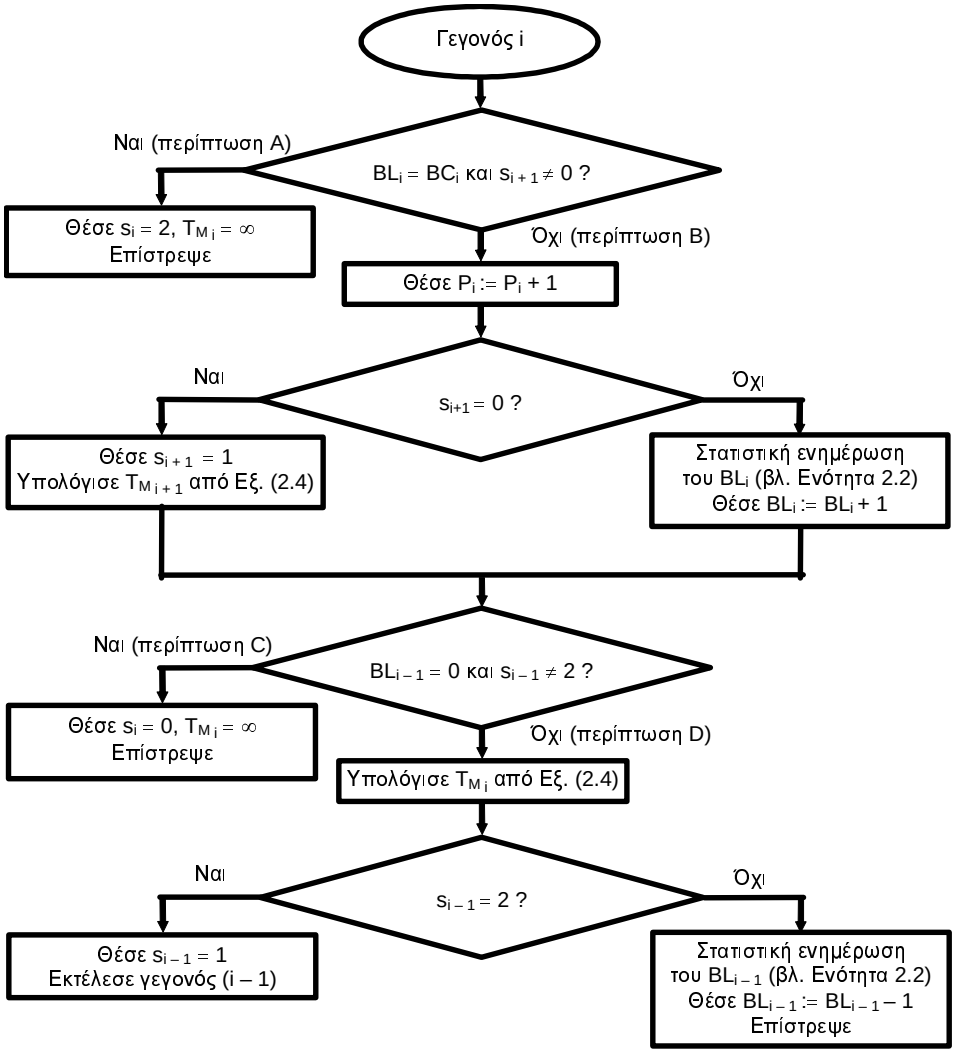
Αλγόριθμος 2.1. *Τυπικό μοντέλο προσομοίωσης*

(a) *Έναρξη*. Είσοδος παραμέτρων των μηχανών, χωρητικότητας των αποθηκών, και συνολικού όγκου παραγωγής TITEMS. Θέσε $t = 0$ και υπολόγισε χρόνους επόμενων γεγονότων για κάθε μηχανή από την Εξ. (2.2).

---

* Αυτές οι συνθήκες δεν είναι αμοιβαία αποκλειόμενες: αν δεν υπάρχει ενδιάμεση αποθήκη μεταξύ των M_{i-1} και M_i, τότε $BL_i = 0$ πάντα και η M_{i-1} είναι δυνατόν να αποκλείεται συχνά από την M_i.

(b) *Ενημέρωση ρολογιού*. Κατάγραψε το χρόνο εμφάνισης του πιο πρόσφατου γεγονότος, $\tau = t$. Ο χρόνος αυτός απαιτείται για τη στατιστική ενημέρωση του συστήματος (βλέπε Ενότητα 2.2). Βρες τη μηχανή, έστω M_i, στην οποία θα συμβεί το πιο άμεσο γεγονός και προσαύξησε την τιμή του ρολογιού t όπως στην Εξ. (2.3). Αν ο όγκος παραγωγής της τελευταίας μηχανής υπερβεί τον προκαθορισμένο ή $t > t_{max}$, τότε τερμάτισε την προσομοίωση.
(c) *Εκτέλεσε το γεγονός i* (βλέπε Σχ. 2.2).
(d) *Επίστρεψε στο Βήμα* (b).



Σχήμα 2.2. Ρουτίνα γεγονότος-i για το τυπικό μοντέλο διακριτών κομματιών.

2.2. ΥΠΟΛΟΓΙΣΜΟΣ ΤΩΝ ΚΡΙΤΗΡΙΩΝ ΑΠΟΔΟΣΗΣ

Ο στόχος της προσομοίωσης είναι να βοηθήσει τη διοίκηση στο να αποφασίσει πώς να σχεδιάσει, να επεκτείνει, και να διαχειριστεί τα συστήματα παραγωγής. Η διοίκηση προσδιορίζει τις επιπτώσεις τέτοιων αποφάσεων συγκρίνοντας το καθαρό κέρδος, την απόδοση της επένδυσης, και την ταμειακή ροή για έναν προκαθορισμένο μεγάλο όγκο

παραγωγής ή για μια συγκεκριμένη περίοδο, ας πούμε, t_{max} χρονικών μονάδων. Αυτοί οι οικονομικοί δείκτες σχετίζονται στενά με έναν αριθμό λειτουργικών δεικτών, που λέγονται *κριτήρια απόδοσης*, κι αφορούν ένα σύστημα παραγωγής. Συνηθισμένα κριτήρια απόδοσης είναι: ο μέσος ρυθμός παραγωγής, το μέσο απόθεμα, το ποσοστό χρόνου που οι μηχανές λειτουργούν, και ο μέσος χρόνος μετατροπής μιας πρώτης ύλης σε έτοιμο προϊόν. Στην προσομοίωση, αυτές οι ποσότητες υπολογίζονται ως μέσες τιμές των συναρτήσεων της κατάστασης που βρίσκεται το σύστημα, μέσα σε μια συγκεκριμένη χρονική περίοδο.

Έστω f [x(t)] η συνάρτηση της κατάστασης x(t) της οποίας την αναμενόμενη τιμή θέλουμε να υπολογίσουμε. Στο μοντέλο διακριτών γεγονότων που παρουσιάσαμε πριν, η κατάσταση x(t) του συστήματος αλλάζει μόνο τις στιγμές $t_0 = 0$, t_1, t_2, …, t_K που συμβαίνουν γεγονότα, όπου t_K είναι είτε ο χρόνος που η συνολική παραγωγή P_n της τελευταίας μηχανής φθάσει κάποιο όριο, είτε $t_K = t_{max}$, ανάλογα με το κριτήριο τερματισμού που έχουμε ορίσει. Ο χρονικός μέσος της f [x(t)] στην περίοδο [0, t_K] υπολογίζεται από τη σχέση

$$\bar{f} = \frac{1}{t_K} \int_0^{t_K} f[x(t)]dt$$

Η f [x(t)] είναι σταθερή κατά τμήματα, επειδή η κατάσταση x(t) του συστήματος είναι σταθερή σε κάθε διάστημα [t_{k-1}, t_k) μεταξύ διαδοχικών γεγονότων, κι εμφανίζει βηματικές μεταβολές τις στιγμές t_k. Άρα

$$\bar{f} = \frac{1}{t_K} \sum_{k=1}^{K} \left\{ \int_{t_{k-1}}^{t_k} f[x(t)]dt \right\}$$

$$= \frac{1}{t_K} \sum_{k=1}^{K} (t_k - t_{k-1}) f[x(t_{k-1})] \qquad (2.4)$$

Κατόπιν, δίνουμε εκτιμήσεις των δύο πιο συνηθισμένων κριτηρίων απόδοσης.

Μέσος ρυθμός παραγωγής: Η παραγωγικότητα TH (throughput) μιας γραμμής παραγωγής προσεγγίζεται με το μέσο ρυθμό παραγωγής της τελευταίας μηχανής. Επομένως

$$TH = \frac{P_n \text{ στο } [0, t_K]}{t_K}$$

Μέση στάθμη της αποθήκης B_i: Το μέσο απόθεμα, που είναι εκτίμηση της μέσης στάθμης της αποθήκης, ορίζεται από τη σχέση

$$\bar{B}_i = \frac{1}{t_K} \int_0^{t_K} BL_i(t)dt$$

Σε ένα σύστημα διακριτών κομματιών, η στάθμη $BL_i(t)$ της B_i είναι σταθερή στο χρονικό διάστημα [t_{k-1}, t_K), μεταξύ διαδοχικών γεγονότων. Εφαρμόζοντας την Εξ. (2.4) παίρνουμε

$$\overline{B}_i = \frac{1}{t_K} \sum_{k=1}^{K} (t_k - t_{k-1}) BL_i(t_{k-1})$$

3

ΤΑΧΥ ΜΟΝΤΕΛΟ ΠΡΟΣΟΜΟΙΩΣΗΣ

3.1. ΑΝΑΠΤΥΞΗ ΤΟΥ ΜΟΝΤΕΛΟΥ

Τώρα θα αναπτύξουμε ένα ταχύ μοντέλο προσομοίωσης της διακριτής κυκλοφορίας κομματιών μέσα στο σύστημα. Από το προηγούμενο κεφάλαιο γίνεται σαφές ότι τα γεγονότα γεμάτων και άδειων αποθηκών λειτουργούν σαν «συνδετικά» μηνύματα, που επιβάλλουν τις γύρω μηχανές να συμβαδίσουν με τις γειτονικές τους. Τα κομμάτια που κατεργάζονται παριστάνουν τους διακομιστές αυτών των μηνυμάτων. Λόγω διακριτότητας των κομματιών, τα γεγονότα αυτά πραγματοποιούνται μετά από μια μεταβατική περίοδο που αντιστοιχεί στο μεσοδιάστημα του χρόνου που μια αποθήκη πάυει να είναι διαθέσιμη και του χρόνου που μια μηχανή ζητά μια μονάδα χώρου ή ένα κομμάτι προς επεξεργασία. Για την ανάλυση τέτοιων φαινομένων ορίζουμε τις παρακάτω μικροσκοπικές μεταβλητές

a χρόνος μέχρι την επόμενη άφιξη στην αποθήκη B_i (χρόνος μέχρι να παραγάγει η M_i)
d χρόνος μέχρι την επόμενη αναχώρηση από την B_i (χρόνος μέχρι να πάρει νέο κομμάτι η M_{i+1})

Αυτές οι ποσότητες θα αναφέρονται ως *μεταβατικοί χρόνοι* των αποθηκών και μηχανών. Το μοντέλο που θα αναπτύξουμε αμέσως μετά είναι θεωρητικά ακριβές, μιας και περικλείει όλα τα μεταβατικά φαινόμενα. Επίσης, είναι γρηγορότερο από την κλασική προσομοίωση γιατί παρακολουθεί μικρό αριθμό γεγονότων.

Το μοντέλο παρατηρεί μόνο τρεις τύπους γεγονότων, συν ένα ειδικό γεγονός

(a) μια μηχανή υφίσταται βλάβη
(b) μια αποθήκη γεμίζει (προηγούμενη μηχανή αποκλείεται)
(c) μια αποθήκη αδειάζει (επόμενη μηχανή αποστερείται)
(d) μια αποθήκη γίνεται μη-άδεια

Στη συνέχεια, θα χρησιμοποιούμε τους όρους *μηχανή αποκλείεται* ή *αποθήκη γεμίζει* ισοδύναμα, για να ορίσουμε την έναρξη μιας σειράς περιόδων παραγωγής όπου μια μηχανή παράγει, «μπλοκάρεται», κατόπιν ξεμπλοκάρεται και διώχνει ένα κομμάτι, μετά παράγει άλλο κομμάτι, μπλοκάρεται, ξεμπλοκάρεται κ.ο.κ..

Επίσης, θα κάνουμε χρήση των όρων *μηχανή αποστερείται* ή *αποθήκη αδειάζει* για να περιγράψουμε την έναρξη μιας σειράς λειτουργικών περιόδων μίας μηχανής ακολουθούμενων από αδρανή διαστήματα, λόγω αποστέρησης.

Πρέπει να αναφερθεί ότι το μοντέλο δεν χρησιμοποιεί γεγονότα επισκευών. Η απαλοιφή τους επιτυγχάνεται ως εξής: όταν η μηχανή M_i χαλάει, επεκτείνουμε το μεταβατικό χρόνο a για να συμπεριληφθεί σε αυτόν ο χρόνος επισκευής της. Ο χρόνος αυτός δίνεται από την Εξ. (2.1):

$$\begin{pmatrix}\text{συνολική διακοπή κατά}\\ \text{έναν κύκλο παραγωγής}\end{pmatrix} = \sum_{n=1}^{\text{αριθμός βλαβών}} (\text{διάρκεια επισκευής της i – οστής μηχανής})$$

Συνεπώς όταν η M_i υφίσταται βλάβη, ο μεταβατικός χρόνος a που θα μεσολαβήσει μέχρι να ολοκληρώσει την παραγωγή του κομματιού της είναι

$$a = \Delta + \begin{pmatrix}\text{συνολική διακοπή κατά}\\ \text{έναν κύκλο παραγωγής}\end{pmatrix}$$

Όμοια, όταν η M_{i+1} χαλάει, ο μεταβατικός χρόνος d προσαρμόζεται κατάλληλα. Έτσι, οι περίοδοι επισκευών συμπεριλαμβάνονται στους χρόνους a και d. Αποτέλεσμα αυτού είναι οι μηχανές να θεωρούνται ότι λειτουργούν «συνεχώς».

Η διακριτή κίνηση μπορεί να προσομοιωθεί με χρήση του Αλγορίθμου 2.1 της προηγούμενης ενότητας αλλά υλοποιώντας διαφορετικές εξισώσεις για την ενημέρωση και προσαρμογή των καταστάσεων και προγραμματισμού των επόμενων γεγονότων. Θα ξεκινήσουμε με τις εξισώσεις για τον καθορισμό των επόμενων γεγονότων. Κατόπιν, θα εξάγουμε τις εκφράσεις για την ενημέρωση των μικροσκοπικών μεταβλητών πριν την εμφάνιση ενός γεγονότος και την προσαρμογή της κατάστασης αμέσως μετά.

3.1.1. Προγραμματισμός γεγονότων για τις μηχανές

Για κάθε μηχανή πρέπει να γνωρίζουμε τον χρόνο της επόμενης βλάβης της, Θεωρούμε τη μηχανή M_i σε έναν (αυθαίρετο) χρόνο t. Από την παραδοχή ότι οι βλάβες συμβαίνουν αν η μηχανή λειτουργεί, προκύπτει ότι ο χρόνος μεταξύ βλαβών σε μια μηχανή εξαρτάται από τον αριθμό των κομματιών που επεξεργάζεται. Όταν μια μηχανή M_i υφίσταται βλάβη, το μοντέλο παράγει έναν τυχαίο αριθμό για τα κομμάτια μέχρι την επόμενη βλάβη, εφαρμόζοντας την Εξ. (1.2). Έστω ότι στο χρόνο t ο αριθμός των κομματιών μέχρι την επόμενη βλάβη είναι F_i. Τότε μια εκτίμηση του χρόνου μέχρι τη βλάβη, είναι

$$T_{M_i} = t + a + \Delta\,(F_i - 1) \qquad (3.1)$$

όπου a είναι ο χρόνος μέχρι να ελευθερωθεί το επόμενο κομμάτι, F_i ο αριθμός των κομματιών μέχρι τη βλάβη και Δ ο χρόνος κατεργασίας της M_i. Το "– 1" στην εξίσωση μπαίνει επειδή τη στιγμή t + a *θα παραχθεί το πρώτο* από τα F_i κομμάτια. Θα αναφερθούμε αναλυτικότερα σε αυτό παρακάτω. Ο χρόνος μέχρι τη βλάβη επανεκτιμάται κάθε φορά που ο αποκλεισμός και η αποστέρηση επηρεάζουν τη M_i.

Στη συνέχεια μελετάμε τα γεγονότα γεμάτων και άδειων αποθηκών.

3.1.2. Προγραμματισμός γεγονότος αποκλεισμού

Αρχικά, εξετάζουμε το μηχανισμό αποκλεισμού των μηχανών. Η μηχανή M_i θα αποκλειστεί εαν επιχειρήσει να ελευθερώσει ένα κομμάτι προς μια γεμάτη αποθήκη. Στο Σχήμα 3.1 φαίνεται μια πιθανή περίπτωση που οδηγεί σε αποκλεισμό της M_i. Έστω ότι στο χρόνο t, είναι γνωστοί οι μεταβατικοί χρόνοι a και d και η στάθμη BL_i. Ορίζουμε τις παρακάτω ποσότητες:

T χρόνος στον οποίο η M_i επιχειρεί να ελευθερώσει ένα κομμάτι και αποκλείεται

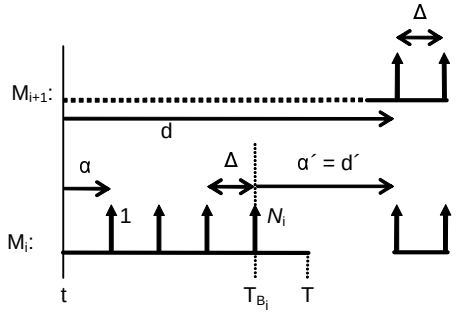
T_{Bi} χρόνος στον οποίο το *προηγούμενο* κομμάτι είχε ελευθερωθεί από την M_i, $T_{Bi} < T$

N_i αριθμός κομματιών που θα κατεργαστεί η M_i, i = 1, 2, ..., στο διάστημα $(t, T_{Bi}]$.

Αναζητούμε μια έκφραση του χρόνου T_{Bi} που θα βασίζεται στην κατάσταση του συστήματος στο χρόνο t. Ο χρόνος αυτός θα είναι για τον αλγόριθμο ο χρόνος εκτέλεσης επόμενου γεγονότος της αποθήκης B_i. Στο σύστημά μας οι μηχανές παράγουν με τον ίδιο ρυθμό Δ, οπότε δεν υφίσταται αποκλεισμός λόγω διαφοράς στην ταχύτητα παραγωγής των μηχανών. Εξετάζουμε τις ακόλουθες περιπτώσεις.

Περίπτωση Α: $N_{i+1} = 0$. Στον αλγόριθμο, το γεγονός της πλήρωσης της αποθήκης συμβαίνει *πριν* η M_{i+1} παραγάγει ένα κομμάτι, δηλαδή, $T_{Bi} < t + d$. Το γεγονός αυτό λαμβάνει χώρα όταν ο χρόνος επισκευής d της M_{i+1} είναι πολύ μεγάλος ή αν η M_{i+1} έχει αποκλεισθεί κι έχει αποκτήσει μία καθυστέρηση $d > \Delta$. Μια τυπική απεικόνιση της περίπτωσης φαίνεται στο Σχ. 3.1, όπου η M_i παράγει το πρώτο της κομμάτι στο χρόνο t + a, το δεύτερο στο χρόνο t + a + Δ, κτλ. Το N_i-οστό κομμάτι ολοκληρώνεται στο χρόνο T_{B1} και γεμίζει τη χωρητικότητα της B_i. Το επόμενο είναι έτοιμο πριν το χρόνο t + d, ενώ η M_{i+1} εξακολουθεί να κατεργάζεται ένα κομμάτι. Για το λόγο αυτό, μια αναγκαία και ικανή συνθήκη για τον αποκλεισμό μέσα στη μεταβατική περίοδο d είναι

$$(d - a) / \Delta > BC_i - BL_i$$



Σχήμα 3.1. Αποκλεισμός μέσα στο μεταβατικό χρόνο d ($N_{i+1} = 0$).

Στο μοντέλο ο χρόνος T_{Bi} θεωρείται ως *χρόνος γεγονότος*. Αργότερα θα δούμε πως εκτός του ότι στο χρόνο αυτό η αποθήκη είναι γεμάτη, ο T_{Bi} ικανοποιεί μια ισχυρότερη συνθήκη που δικαιολογεί την επιλογή του για χρόνο γεγονότος. Από το Σχ. 3.1 προκύπτει ότι ο χρόνος T_{Bi} είναι ίσος προς

$$T_{Bi} = t + a + \Delta (N_i - 1) \tag{3.2}$$

Επιπρόσθετα, εφόσον δεν χάνονται κομμάτια, ο ισολογισμός κομματιών στην αποθήκη B_i είναι

$$BL_i + N_i - N_{i+1} = BC_i \tag{3.3}$$

Εφόσον $N_{i+1} = 0$, η Εξ. (3.3) συνεπάγεται ότι $N_i = BC_i - BL_i$, που, με αντικατάσταση στην Εξ. (3.2), δίνει

$$T_{Bi} = t + a + \Delta\,(BC_i - BL_i - 1)$$

Περίπτωση Β: $N_{i+1} = 0$ και $BC_i = BL_i$. Αυτή η συνθήκη αντιστοιχεί στην περίπτωση που η αποθήκη B_i είναι γεμάτη τη στιγμή t και η M_{i+1} έχει μεγάλη καθυστέρηση d. Για $BC_i = BL_i$ η παραπάνω εξίσωση προγραμματισμού θα έδινε

$$T_{Bi} = t + a - \Delta < t$$

που σημαίνει ότι στο χρόνο t, το επόμενο γεγονός πρέπει να εκτελεστεί πριν το t! Για να αποφευχθεί αυτή η ασυνέπεια, το μοντέλο προγραμματίζει να συμβεί το γεγονός αποκλεισμού άμεσα, δηλαδή $T_{Bi} = t$.

Η επόμενη πρόταση συνοψίζει τις Περιπτώσεις Α και Β.

Πρόταση 3.1. Ο χρόνος ενός γεγονότος αποκλεισμού προγραμματίζεται ως εξής:

$$T_{Bi} = \begin{cases} t & \text{αν} \quad BL_i = BC_i \ \text{ και } \ d - a > 0 \\ t + a + \Delta(BC_i - BL_i) - 1 & \text{αν} \ \ (d - a)/\Delta > BC_i - BL_i > 0 \end{cases} \tag{3.4}$$

Το μοντέλο αποφεύγει τους λεπτομερείς υπολογισμούς των χρόνων παραγωγής των κομματιών N_i, μια και, γενικά, οι χρόνοι t και T_{Bi} μπορεί να απέχουν αρκετούς κύκλους παραγωγής.

3.1.3. Προγραμματισμός γεγονότος αποστέρησης

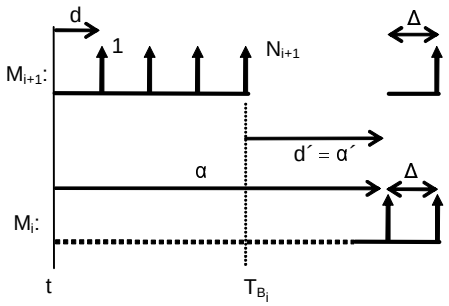
Με ανάλογο τρόπο θα βρούμε εκφράσεις για το χρόνο που η M_{i+1} αποστερείται (βλέπε Σχ. 3.2). Υποθέτουμε ότι στο χρόνο t, οι μεταβατικοί χρόνοι a και d είναι γνωστοί, η αποθήκη περιέχει BL_i κομμάτια κι ένα ακόμη κομμάτι βρίσκεται στην M_{i+1}. Έστω T_{Bi} ο χρόνος που η M_{i+1} γίνεται αποστερημένη και N_i ο αριθμός των κομματιών που παράγει η M_i, i = 1, 2, ..., στο διάστημα $(t, TB_i]$. Τότε

$$N_{i+1} = BL_i + N_i + 1 \tag{3.5}$$

Το "+ 1" προκύπτει από την υπόθεση ότι στην αρχή υπάρχει 1 κομμάτι στην M_{i+1}, εκτός από τα BL_i στην αποθήκη B_i. Ο χρόνος T_{Bi} υπολογίζεται από τη σχέση

$$T_{Bi} = t + d + \Delta\,(N_{i+1} - 1) \tag{3.6}$$

Η Εξ. (3.6) είναι η ανάλογη της Εξ. (3.2) για το γεγονός άδειας αποθήκης.



Σχήμα 3.2. Αποστέρηση μέσα στο μεταβατικό χρόνο a ($N_i = 0$).

Έστω a' και d' οι μεταβατικοι χρόνοι αμέσως μετά το T_{Bi}. Στο χρόνο T_{Bi} + a', ένα κομμάτι θα ελευθερωθεί από τη M_i προς την αποθήκη και στη συνέχεια προς την M_{i+1} χωρίς καθυστέρηση. Εδώ έχουμε την περίπτωση στην οποία σημειώνονται ταυτόχρονα μια άφιξη και μια αναχώρηση από την B_i. Έτσι, στο γρήγορο μοντέλο το d' τίθεται ίσο προς το a'.

Στη συνέχεια, εξετάζουμε την γενική περίπτωση που οδηγεί σε γεγονός αποστέρησης.

Περίπτωση C: $N_i = 0$. Η αποστέρηση σημειώνεται *πριν* η M_i παραγάγει ένα προϊόν, δηλαδή, $T_{Bi} < t + a$. Αυτό συμβαίνει όταν η M_i τύχει να έχει πολύ μακρύ χρόνο επισκευής ή να είναι η ίδια αποστερημένη. Από το Σχ. 3.2, βλέπουμε ότι η M_{i+1} παράγει το πρώτο της κομμάτι στο χρόνο t + d, το δεύτερο στο χρόνο t + d + Δ, κτλ. Στο χρόνο T_{Bi}, η M_{i+1} έχει πάρει όλα τα κομμάτια που υπήρχαν στην B_i κι αποστερείται αμέσως. Έτσι, η συνθήκη για το γεγονός είναι

$$(a - d) > \Delta\, BL_i$$

Από την Εξ. (3.5) κι επειδή $N_i = 0$,

$$N_{i+1} - 1 = BL_i$$

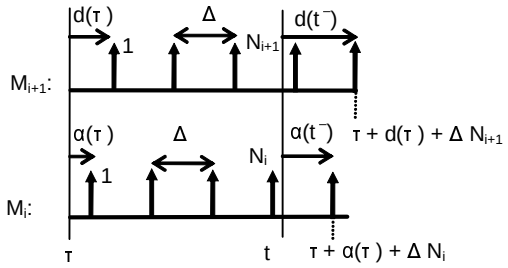
Εισάγοντας τα παραπάνω στην Εξ. (3.6) παίρνουμε

Πρόταση 3.2. Ο χρόνος ενός γεγονότος αποστέρησης προγραμματίζεται ως εξής:

$$T_{Bi} = t + d + \Delta\, BLi\,,\ \text{αν}\ (a - d) > \Delta\, BL_i \qquad (3.7)$$

3.1.4. Εξισώσεις ενημέρωσης

Σε αυτή την ενότητα θα περιγράψουμε την εξέλιξη των μικροσκοπικών καταστάσεων στο σύστημα, ειδικότερα, τους μεταβατικούς χρόνους, τη στάθμη των αποθηκών, τη συνολική παραγωγή των μηχανών, τα εναπομείναντα κομμάτια μέχρι τη βλάβη και σχετικά μεγέθη απόδοσης, στο διάστημα μεταξύ δύο διαδοχικών γεγονότων.



Σχήμα 3.3. Εξέλιξη συστήματος μεταξύ δύο διαδοχικών γεγονότων.

Θεωρούμε 2 γεγονότα που επηρεάζουν την αποθήκη B_i τις στιγμές τ και t. Έστω N_i ο αριθμός των αφίξεων στην B_i στο διάστημα [τ, t) και N_{i+1} ο αριθμός των αναχωρήσεων από την B_i την ίδια χρονική περίοδο. Εξ ορισμού, N_i είναι οι αφίξεις μέχρι το χρόνο t. N_{i+1} είναι οι αναχωρήσεις από την αποθήκη στο διάστημα [τ, t). Μετρώντας τα βέλη του Σχ. 3.3, παίρνουμε

$$\begin{aligned} N_i &= \begin{cases} 1+\lfloor [t-\tau-a(\tau)]/\Delta \rfloor & \text{αν } t \geq \tau + a(\tau) \\ 0 & \text{αλλιώς} \end{cases} \\ N_{i+1} &= \begin{cases} 1+\lfloor [t-\tau-d(\tau)]/\Delta \rfloor & \text{αν } t \geq \tau + d(\tau) \\ 0 & \text{αλλιώς} \end{cases} \end{aligned} \tag{3.8}$$

όπου $\lfloor x \rfloor$ ο μεγαλύτερος ακέραιος μικρότερος του x. Στο χρόνο t^-, το ισοζύγιο υλικού στην B_i είναι

$$BL_i(t^-) = BL_i(\tau) + N_i - N_{i+1}$$

Ο εκθέτης «μείον» στο χρόνο t^- χρησιμοποιείται λόγω του ότι το μοντέλο ενημερώνει τις μεταβλητές κατάστασης ακριβώς *πριν* την εκτέλεση των γεγονότων.

Εφόσον η M_i θα παραγάγει N_i κομμάτια, η συνολική παραγωγή και το πλήθος των κομματιών μέχρι τη βλάβη ενημερώνονται από τις εξισώσεις

$$\begin{aligned} P_i(t^-) &= P_i(\tau) + N_i \\ F_i(t^-) &= F_i(\tau) - N_i \end{aligned} \tag{3.9}$$

Τώρα υπολογίζουμε το νέο μεταβατικό χρόνο $a(t^-)$ της M_i. Από το Σχ. 3.5 έχουμε

$$a(t^-) = \tau + a(\tau) + \Delta N_i - t \tag{3.10}$$

Αν το διάστημα [τ, t) τύχει να είναι μικρότερο από τον αρχικό μεταβατικό χρόνο a(τ) (διαφορετική περίπτωση από αυτή στο Σχ. 3.3), τότε $N_i = 0$ κι, επομένως, η Εξ. (3.10) εξακολουθεί να ισχύει.

Αντικαθιστώντας τον a(τ) με τον d(τ) και το δείκτη i με i + 1 στις Εξ. (3.9) και (3.10), παίρνουμε τις μικροσκοπικές μεταβλητές της M_{i+1} στο χρόνο t^-.

Τώρα θα εξάγουμε την εξίσωση ενημέρωσης για το μέσο απόθεμα στην B_i. Έστω $n_i(s)$ και $n_{i+1}(s)$ οι αριθμοί των αφίξεων και αναχωρήσεων αντίστοιχα, στο διάστημα $[\tau, s)$, $s \le t$. Τότε, αν τ και t είναι οι χρόνοι των γεγονότων,

$$\overline{B}_i = \frac{\int_0^{t_{max}} BL_i(s)\,ds}{t_{max}} = \frac{\sum \int_\tau^{t^-} BL_i(s)\,ds}{t_{max}}$$

$$= \frac{\sum \int_\tau^{t^-} [BL_i(\tau) + n_i(s) - n_{i+1}(s)]\,ds}{t_{max}} \qquad (3.11)$$

Οι συναρτήσεις $n_i(s)$ και $n_{i+1}(s)$ έχουν μορφή σκαλοπατιών και δίνονται από την Εξ. (3.8) αντικαθιστώντας το s με το t.

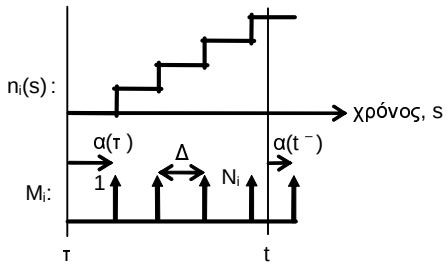
Θεωρούμε τη συνάρτηση $n_i(s)$ που απεικονίζεται στο Σχ. 3.4, και το εμβαδόν της περιοχής που περικλείεται από την $n_i(s)$ ανάμεσα στα $[\tau, t)$,

$$\int_\tau^{t^-} n_1(s)\,ds$$

Η περιοχή αποτελείται από N_i ορθογώνια με ύψη 1, 2, …, N_i – θυμηθείτε ότι N_i είναι η συνολική παραγωγή στο διάστημα $[\tau, t)$, συνεπώς $N_i = n_i(t^-)$. Το μήκος της βάσης των πρώτων $N_i - 1$ ορθογωνίων είναι Δ και του τελευταίου είναι $\Delta - a(t^-)$. Ειδικότερα,

$$\int_\tau^{t^-} n_i(s)\,ds = \Delta[1 + 2 + \ldots + (N_i - 1)] + [\Delta - a(t^-)]N_i$$

$$= \Delta \frac{N_i(N_i + 1)}{2} - a(t^-)\,N_i$$



Σχήμα 3.4. Εξέλιξη της $n_i(s)$.

Με όμοιο τρόπο, το εμβαδόν της $n_{i+1}(s)$ δίνεται από τη σχέση

$$\int_{\tau}^{t^-} n_{i+1}(s)\,ds = \Delta\,\frac{N_{i+1}(N_{i+1}+1)}{2} - d(t^-)\,N_{i+1}$$

Εισάγοντας τα παραπάνω στην Εξ. (3.11), παίρνουμε την εκτίμηση για το μέσο απόθεμα στην αποθήκη.

Οι μικροσκοπικές καταστάσεις και το μέσο απόθεμα κάθε αποθήκης, ενημερώνονται ακριβώς πριν την εκτέλεση των γεγονότων. Στη συνέχεια, θα δημιουργήσουμε τις εξισώσεις προσαρμογής κατάστασης, που καλούνται με την εμφάνιση των γεγονότων.

3.1.5. Μεταβολές κατάστασης λόγω γεγονότων

Όταν συμβαίνει ένα γεγονός αποκλεισμού, οι μεταβατικοί χρόνοι επόμενης άφιξης κι αναχώρησης είναι ίσοι. Έτσι, θέτουμε

$$a = d \tag{3.12}$$

Η λειτουργία της M_{i+1} δεν επηρεάζεται από το γεγονός αυτό. Το επόμενο γεγονός που θα συμβεί στην M_i σχεδιάζεται σύμφωνα με την Ενότητα 3.1.1. Τέλος, όλες οι *μελλοντικές* αλλαγές στην B_i, π.χ. λογω εμφάνισης βλάβης πριν το τέλος της προσομοίωσης, δεν χρειάζεται να ληφθούν υπόψη τη στιγμή αυτή. Επομένως, εφόσον οι μεταβατικοί χρόνοι και οι ρυθμοί παραγωγής είναι ίσοι, τα ρολόγια που συνδέονται με τα γεγονότα που συμβαίνουν στις αποθήκες παγώνουν και θέτουμε $T_{Bi} = \infty$.

Με αντίστοιχο τρόπο, επεξεργαζόμαστε το γεγονός αποστέρησης, θέτοντας

$$d = a \tag{3.13}$$

Μετά, θέτουμε $T_{Bi} = \infty$ και προγραμματίζουμε το επόμενο γεγονός για την M_{i+1}.

Όπως θα αναλυθεί παρακάτω στην ενότητα, το μοντέλο θεωρεί ότι μια μηχανή μπορεί να χαλάσει μόνο στην αρχή ενός κύκλου κατεργασίας. Αυτή η παραδοχή γίνεται για ευκολία και, όπως θα δούμε, επηρεάζει τους χρόνους που φεύγουν τα κομμάτια από τη μηχανή. Όταν μια μηχανή χαλάει, ο μεταβατικός χρόνος της αποθήκης της (a ή d) παρατείνεται όσο διαρκεί η επισκευή, και υπολογίζεται από την Εξ. (2.1),

$$\begin{pmatrix}\text{συνολική διακοπή κατά}\\ \text{έναν κύκλο παραγωγής}\end{pmatrix} = \sum_{n=1}^{\substack{\text{αριθμός}\\ \text{βλαβών}}} (\text{διάρκεια επισκευής της n} - \text{οστής βλάβης})$$

Ας υποθέσουμε, για παράδειγμα, ότι η M_i λειτουργεί στο διάστημα [τ, t) και ότι παθαίνει βλάβη στο χρόνο t, οπότε και ξεκινά ένα νέο κύκλο παραγωγής.

Στο χρόνο t^-, φορτώνεται ένα κομμάτι στη M_i κι ο μεταβατικός χρόνος a είναι ίσος προς τον κύκλο παραγωγής, δηλαδή

$$a(t^-) = \Delta$$

Στο χρόνο t, το μοντέλο παρατηρεί μια βλάβη στην M_i κι επαναπροσδιορίζει το μεταβατικό χρόνο σαν τον υπολειπόμενο χρόνο για να ολοκληρώσει ένα κομμάτι

$$a(t) = \begin{pmatrix} \text{συνολική διακοπή κατά} \\ \text{έναν κύκλο παραγωγής} \end{pmatrix} + \Delta \qquad (3.14)$$

Τότε το μοντέλο υπολογίζει μια νέα τιμή για τον αριθμό των κομματιών μέχρι την επόμενη βλάβη. Η ποσότητα αυτή είναι ένας *θετικός* ακέραιος, αφού όλες οι βλάβες που θα συμβούν κατά τη διάρκεια της κατεργασίας του τρέχοντος κομματιού, συμπεριλαμβάνονται στο μεταβατικό χρόνο a. Τέλος, το μοντέλο προγραμματίζει τα επόμενα γεγονότα στις M_i, B_i και M_{i+1}.

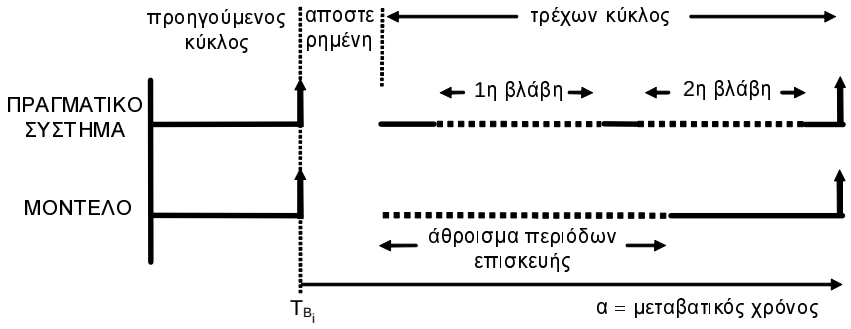
3.2. ΚΑΝΟΝΕΣ ΠΡΟΤΕΡΑΙΟΤΗΤΑΣ ΓΕΓΟΝΟΤΩΝ

Επειδή ο αλγόριθμος προγραμματίζει βλάβες μηχανών και όλα τα γεγονότα που σχετίζονται με αποθήκες με βάση τις αφίξεις ή αναχωρήσεις σε αυτές, υπάρχει η πιθανότητα δύο διαφορετικά γεγονότα να συμβούν ταυτόχρονα. Εξετάζουμε τα παρακάτω φαινόμενα.

3.2.1. Κανόνες κυριαρχίας

Το πρώτο φαινόμενο συμβαίνει όταν ο χρονος στον οποίο το μοντέλο θα παρατηρήσει μια βλάβη στη μηχανή M_{i+1} τυχαίνει να είναι ο πραγματικός χρόνος T_{Bi} στον οποίο η M_{i+1} θα γίνει αποστερημένη. Όμως, αφού οι βλάβες σημειώνονται μόνον όταν λειτουργούν οι μηχανές, δεν μπορούν να συμβούν σε νεκρές (αποστερημένες) περιόδους. Αυτό που πραγματικά συμβαίνει είναι ότι η βλάβη σημειώνεται *αφού* η μηχανή φορτώσει ένα κομμάτι. Αυτό ορίζει μια κυριαρχική σχέση μεταξύ των φαινομένων αποστέρησης και βλάβης, όπου αν μια μηχανή γίνει αποστερημένη και ταυτόχρονα πάθει βλάβη, τότε το το πρώτο γεγονός θα εκτελεστεί πριν το δεύτερο. Δηλαδή,

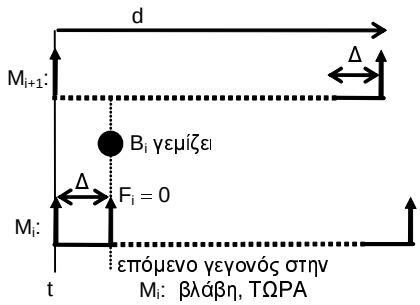
1ος κανόνας προτεραιότητας γεγονότων*:* Η αποστέρηση κυριαρχεί της βλάβης.



Σχήμα 3.5. Η αποστέρηση επικρατεί της βλάβης. Η συγκέντρωση των χρόνων επισκευής στα αριστερά δεν επηρεάζει τους χρόνους αναχώρησης.

Αυτή η κατάσταση φαίνεται στο Σχ. 3.5. Το σχήμα δείχνει επίσης πώς το μοντέλο χειρίζεται την εμφάνιση συνεχόμενων βλαβών. Σε τέτοιες περιπτώσεις, είναι υπολογιστικά πιο αποτελεσματικό να συμπεριλάβουμε την περίοδο αποστέρησης, όλες τις περιόδους επισκευής και το χρόνο κατεργασίας για το τρέχον κομμάτι, στο μεταβατικό χρόνο, παρά να εκτελέσουμε κάθε μία βλάβη χωριστά. Είναι προφανές από το Σχ. 3.5 ότι η προσέγγιση αυτή

δεν επηρεάζει τους χρόνους ελευθέρωσης των κομματιών κι έτσι τα δείγματα χρόνων του μοντέλου και του συστήματος συμπίπτουν.



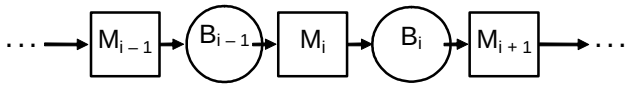
Σχήμα 3.6. Η βλάβη επικρατεί του αποκλεισμού

Το δεύτερο φαινόμενο αφορά την περίπτωση όπου, στο χρόνο T_{Bi}, η μηχανή M_i θα αποκλειστεί και θα χαλάσει ταυτόχρονα. Παρατηρήστε ότι η ποσότητα T_{Bi}, που ορίστηκε στην Ενότητα 3.1.2, είναι στην πραγματικότητα ο χρόνος που η μηχανή *ξεκινά* να κατεργάζεται το κομμάτι που πρόκειται να αποκλειστεί. Στην περίπτωση αυτή, το γεγονός της βλάβης πρέπει να εκτελεστεί πρώτο. Αυτό συμβαίνει γιατί ο χρόνος επισκευής μπορεί να είναι αρκετά μακρύς ώστε, τελικά, ο αποκλεισμός να ακυρωθεί. Το φαινόμενο περιγράφεται στο Σχ. 3.6. Έτσι,

2^{ος} κανόνας προτεραιότητας γεγονότων*:* Η βλάβη κυριαρχεί του αποκλεισμού.

3.2.2. Μηχανές που είναι συγχρόνως αποστερημένες και αποκλεισμένες

Θεωρούμε ένα τμήμα της γραμμής παραγωγής που αποτελείται από τρεις μηχανές και δύο ενδιάμεσες αποθήκες (βλέπε Σχ. 3.7). Έστω ότι η B_{i-1} είναι άδεια και η B_i γεμάτη. Αυτή η κατάσταση παρατηρείται όταν η M_i είναι, λόγω βλαβών, ταχύτερη από τις εκατέρωθέν της. Τότε, η M_i αναγκάζεται να περιμένει μέχρι να της έρθει ένα κομμάτι από την προηγούμενη αποθήκη και, μόλις τελειώνει την κατεργασία, το κομμάτι αποκλείεται μέχρι να αδειάσει μια μονάδα χώρου στον ακόλουθη αποθήκη. Η μηχανή εναλλάσσεται μεταξύ αποστερημένων και αποκλεισμένων καταστάσεων περιοδικά, μέχρι να ακυρωθεί μια από τις δύο.



Σχήμα 3.7. Τμήμα που περιλαμβάνει μια αποστερημένη και αποκλεισμένη μηχανή M_i.

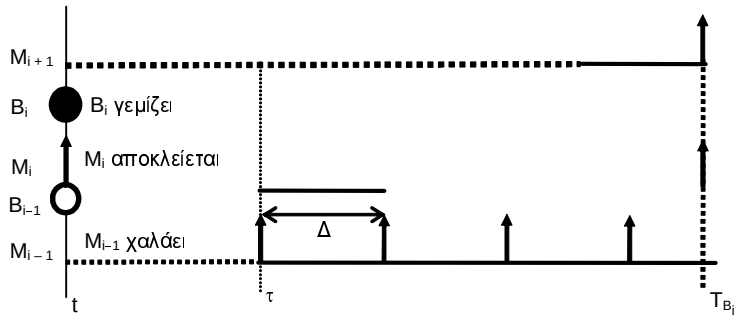
Κατά την περίοδο αποστέρησης και αποκλεισμού, η παραγωγή της M_i καθορίζεται από τις γύρω μηχανές, ενώ οι στάθμες των εκατέρωθεν αποθηκών παίρνουν ακραίες τιμές, $BL_{i-1} = 0$ και $BL_i = BC_i$, αντίστοιχα. Επομένως, αν μπορέσουμε να υπολογίσουμε το μήκος της περιόδου αυτής, θα έχουμε ολοκληρωμένη γνώση πάνω στις δυναμικές του τμήματος της αλυσίδας, άρα θα αποφυγουμε την κλασική λεπτομερή προσομοίωση.

Έστω t ο χρόνος στον οποίο η μηχανή μπαίνει σε κατάσταση αποστέρησης και αποκλεισμού. Θέλουμε να προβλέψουμε το χρόνο επόμενου γεγονότος στο τμήμα. Ξεχωρίζουμε της ακόλουθες περιπτώσεις:

1. Μια κατάσταση αποστέρησης και αποκλεισμού *ακυρώνεται αμέσως μετά την εμφάνισή της*, είτε γιατί η B_i παύει να είναι γεμάτη ή γιατί η B_{i-1} παύει να είναι άδεια.
2. Μια κατάσταση αποστέρησης και αποκλεισμού *ακυρώνεται μετά από την παραγωγή μερικών κομματιών*, πάλι επειδή η B_i παύει να είναι γεμάτη ή η B_{i-1} άδεια.

Έχουμε αποδείξει ότι όταν οι κύκλοι παραγωγής των μηχανών είναι ίσοι με Δ, τότε η περίπτωση 2 δεν μπορεί να συμβεί. Η απόδειξη παραλείπεται για συντομία αλλά είναι παρόμοια με την εξέταση των Περιπτώσεων 1_A κι 1_B που περιγράφονται στη συνέχεια. Τώρα εξετάζουμε την άμεση ακύρωση των καταστάσεων αποστέρησης ή αποκλεισμού.

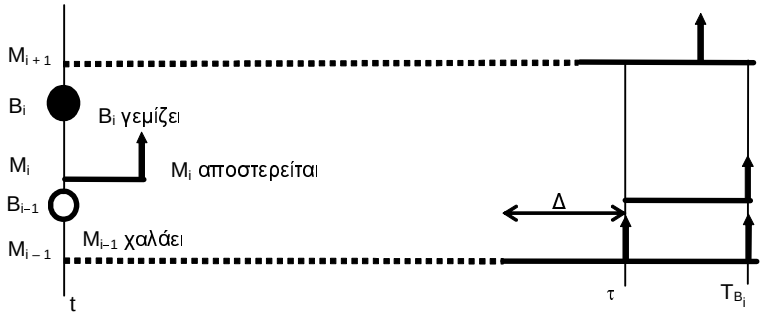
Περίπτωση 1_A: Έστω ότι τη χρονική στιγμή t, η αποθήκη B_{i-1} αδειάζει, η μηχανή M_i παράγει ένα κομμάτι που γεμίζει την αποθήκη της, και η M_{i-1} χαλάει, οπότε παύει να τροφοδοτεί την αποθήκη B_{i-1}. Παρατηρώντας το Σχ. 3.8, βλέπουμε πως η επόμενη άφιξη στην B_i θα επιτραπεί μόνο τη στιγμή T_{Bi} κι όχι νωρίτερα (λόγω βλάβης ή αποκλεισμού της M_{i+1}). Παρόλα αυτά, η μηχανή M_{i-1} εν τω μεταξύ αρχίζει να τροφοδοτεί πάλι την B_{i-1} από τη στιγμή τ. Ενώ λοιπόν το πρώτο νέο κομμάτι θα έπρεπε να σταλεί άμεσα στην M_i που αποστερείται, λόγω αποκλεισμού της από τη γεμάτη B_i κάτι τέτοιο δεν είναι εφικτό, οπότε η M_i αναγκάζεται να περιμένει μέχρι τη στιγμή T_{Bi} για να μπορέσει να ελευθερώσει το κομμάτι της. Για τη σωστή ενημέρωση των στατιστικών μεταβλητών όμως, η αναχώρηση του κομματιού από την B_{i-1}, πρέπει να σημειωθεί στο χρόνο τ. Έτσι, τελικά, συμπεραίνουμε πως σε μια τέτοια κατάσταση, ο αποκλεισμός υπερισχύει της αποστέρησης. Για την ειδική αυτή περίπτωση είναι απαραίτητη η χρήση του ειδικού γεγονότος «αποθήκη μη-άδεια», που αναφέρθηκε αρχικά. Το γεγονός αυτό στον αλγόριθμο λαμβάνει χώρα τη στιγμή τ.



Σχήμα 3.8. Υπερίσχυση του αποκλεισμού έναντι της αποστέρησης.

Περίπτωση 1_B: Μια δεύτερη περίπτωση έχουμε όταν ο χρόνος παραγωγής κομματιού από την M_{i-1} είναι αρκετά μεγάλος λόγω βλάβης ή αποστέρησης, ώστε η μηχανή M_i να πάψει να είναι αποκλεισμένη, όπως απεικονίζεται στο Σχ. 3.9. Τώρα, η αποστέρηση επικρατεί του αποκλεισμού, μιας και για να αποκλειστεί η μηχανή M_i χρειάζεται νέο κομμάτι προς επεξεργασία. Εδώ, τη στιγμή t, η M_i κληρονομεί μια μεγάλη καθυστέρηση από την M_{i-1}, μέχρις ότου τροφοδοτηθεί. Η μηχανή παύει να είναι αποκλεισμένη ακαριαία, οπότε η κατάσταση s_i της αποθήκης B_i γίνεται ενδιάμεση (1) από γεμάτη (2) που ήταν μέχρι τώρα.

Συνεπώς, αυτή η μεταβολή κατάστασης γίνεται στιγμιαία και δεν χρειάζεται ο προγραμματισμός νέου γεγονότος «Αποθήκη μη–γεμάτη», όπως στην 1_A. Στο χρόνο T_{Bi}, ελευθερώνει κανονικά το κομμάτι προς την B_i, μια κι έχει δημιουργηθεί μια μονάδα χώρου από την M_{i+1} που έχει αρχίσει πάλι να λειτουργεί.



Σχήμα 3.9. Υπερίσχυση της αποστέρησης έναντι του αποκλεισμού.

Τέτοιες ειδικές περιπτώσεις έχουν ληφθεί υπόψη στον αλγόριθμο, ωστόσο το μοντέλο παρουσίασε μικρά σφάλματα για γραμμές με τρεις ή περισσότερες μηχανές. Τα σφάλματα αυτά δεν οφείλονται σε λογικές αδυναμίες του μοντέλου αλλά στη διαδοχή με την οποία εκτελεί ταυτόχρονα γεγονότα. Οι αποκλίσεις των αποτελεσμάτων του αλγορίθμου από εκείνα του κλασικού και ακριβούς μοντέλου προσομοίωσης, βρέθηκαν να είναι μικρότερες του 2%. Παρά τα μικρά σφάλματα όμως, είναι απαραίτητη η περαιτέρω διερεύνηση των Περιπτώσεων 1_A και 1_B για την τελειοποίηση της μεθόδου. Αυτό ακριβώς το ζήτημα αποτελεί αντικείμενο μελλοντικής έρευνας.

3.3. ΛΟΓΙΚΗ ΤΟΥ ΜΟΝΤΕΛΟΥ ΠΡΟΣΟΜΟΙΩΣΗΣ

Τώρα θα παρουσιάσουμε αναλυτικά τη δομή του μοντέλο προσομοίωσης. Λόγω της ύπαρξης πολλών μεταβλητών που περιγράφουν την κατάσταση του συστήματος, το μοντέλο δεν θα ήταν υπολογιστικά αποτελεσματικό εαν, για κάθε γεγονός, έπρεπε να ενημερώσουμε και να προσαρμόσουμε όλο το διάνυσμα κατάστασης. Στην περίπτωση του γρήγορου μοντέλου κάτι τέτοιο δεν είναι ανάγκη να συμβεί, εφόσον το σύστημα είναι αποδομήσιμο. Πράγματι, όπως αναλύθηκε στις προηγούμενες ενότητες, τα γεγονότα προκαλούν τοπικές διαταραχές στο σύστημα, αλλάζοντας τους μεταβατικούς χρόνους των παρακείμενων μηχανών κι αποθηκών. Με τον τρόπο αυτό, όταν σημειώνεται ένα γεγονός, πρέπει να να ενημερωθούν και να ρυθμιστούν οι καταστάσεις μόνο των γειτονικών συνιστωσών. Οι διαταραχές αυτές μεταφέρονται εμπρός και πίσω στη γραμμή παραγωγής, μέσω μιας σειράς δευτερεύοντων γεγονότων που παρακολουθούνται κι εκτελούνται σειριακά. Τα βήματα του αλγορίθμου διακριτών γεγονότων είναι τα ακόλουθα:

Αλγόριθμος 3.1. *Ταχύς αλγόριθμος προσομοίωσης*

(a) *Απόδοση αρχικών τιμών στη γραμμή*. Θέσε: συνολικό όγκο παραγωγής TITEMS, χωρητικότητες αποθηκών κι αρχικές στάθμες, μεταβατικούς χρόνους, πιθανότητες βλάβης κι επισκευής για τις αποθήκες, μήκος n της γραμμής παραγωγής. Σάρωσε τη γραμμή από την αρχή προς το τέλος και προγραμμάτισε τα επόμενα γεγονότα.

(b) *Προσδιόρισε το επόμενο γεγονός*. Κατάγραψε το χρόνο του πιο πρόσφατου γεγονότος $\tau = t$. Βρες τα γεγονότα με τον μικρότερο χρόνο εμφάνισης κι επίλεξε ένα που συμβαδίζει με τους κανόνες προτεραιότητας της Ενότητας 3.2. Εαν η παραγωγή της τελευταίας μηχανής φθάσει ή ξεπεράσει τη συνολική προκαθορισμένη, πήγαινε στο βήμα (d), αλλιώς πήγαινε στο (c).

(c) *Εκτέλεσε την κατάλληλη ρουτίνα γεγονότος* (βλέπε παρακάτω) και πήγαινε στο βήμα (b):

- Ακριβώς πριν την εκτέλεση του γεγονότος αυτού, το μοντέλο *ενημερώνει* τις μεταβλητές κατάστασης των συνιστωσών (μηχανών και αποθηκών) που θα επηρεαστούν από το γεγονός. Οι εξισώσεις ενημέρωσης δίνουν τις νέες τιμές για τη συνολική παραγωγή των μηχανών, τους μεταβατικούς χρόνους, τις στάθμες των αποθηκών, και τα στατιστικά τους μεγέθη, χρησιμοποιώντας τις τιμές των μεταβλητών κατάστασης στο χρόνο του πιο πρόσφατου (προηγούμενου) γεγονότος. Ο χρόνος αυτός είναι τ_{Mi} για τη μηχανή M_i και τ_{Bi} για την αποθήκη B_i.
- Με την εμφάνιση ενός γεγονότος στο χρόνο, έστω, t, ρυθμίζονται *κατάλληλα* οι μεταβατικοί χρόνοι των επηρεασμένων στοιχείων. Ο χρόνος t είναι τώρα ο νέος χρόνος του πιο πρόσφατου γεγονότος για τα στοιχεία αυτά, δηλαδή, $\tau_{Mi} = t$ και $\tau_{Bi} = t$.
- Τα επόμενα γεγονότα για τα επηρεασμένα στοιχεία επαναπρογραμματίζονται υπολογίζοντας νέους χρόνους επόμενων γεγονότων. Επίστρεψε στο βήμα (b).

(d) *Τερμάτισε την προσομοίωση*. Σάρωσε τη γραμμή προς το τέλος κι ενημέρωσε όλες τις μεταβλητές του συστήματος. Σταμάτα.

Οι σχετικές ρουτίνες γεγονότων είναι οι παρακάτω:

1. Η M_i υφίσταται βλάβη

- Ενημέρωσε τα M_i, B_i και το τμήμα B_{i-1}, M_{i-1}, …, B_{i-k}, όπου B_{i-1}, B_{i-2}, ... μια αλυσίδα τυχόν γεμάτων αποθηκών που προηγούνται της M_i και B_{i-k} είναι η πρώτη μη γεμάτη αποθήκη της αλυσίδας, με τη βοήθεια των εξισώσεων ενημέρωσης της Ενότητας 3.1.4.
- Παράτεινε το μεταβατικό χρόνο παραγωγής επόμενου κομματιού από την M_i, προσθέτοντας στο Δ την καθυστέρηση λόγω επισκευής. Αυτή η παράταση θα επηρεάσει τον μεταβατικό χρόνο άφιξης στην B_i καθώς και τον μεταβατικό χρόνο αναχώρησης επόμενου κομματιού από την προηγούμενη αποθήκη B_{i-1}. Αν οι αποθήκες B_{i-1}, B_{i-2}, ... είναι γεμάτες τότε όλες οι ενδιάμεσες μηχανές θα καθυστερήσουν την παραγωγή τους λόγω αποκλεισμού και θα ελευθερώσουν το επόμενο κομμάτι τους τη στιγμή που η M_i θα παραγάγει το κομμάτι πάνω στο οποίο υπέστη βλάβη.
- Προγραμμάτισε επόμενα γεγονότα σε B_i, M_i, B_{i-1}, M_{i-1}, …, B_{i-k}, όπως στην Ενότητα 3.1.

2. Η B_i γεμίζει

- Ενημέρωσε τα M_i, B_i και το τμήμα B_{i-1}, M_{i-1}, …, B_{i-k}, όπου B_{i-1}, B_{i-2}, ... μια αλυσίδα τυχόν γεμάτων αποθηκών που προηγούνται της M_i και B_{i-k} είναι η πρώ-

τη μη γεμάτη αποθήκη της αλυσίδας, με τη βοήθεια των εξισώσεων ενημέρωσης της Ενότητας 3.1.4.

- Παράτεινε το μεταβατικό χρόνο παραγωγής επόμενου κομματιού από την M_i, προσθέτοντας στο Δ την καθυστέρηση λόγω αποκλεισμού. Αυτή η παράταση θα επηρεάσει τον μεταβατικό χρόνο αναχώρησης επόμενου κομματιού από την προηγούμενη αποθήκη B_{i-1}. Αν οι αποθήκες B_{i-1}, B_{i-2}, ... είναι γεμάτες, τότε όλες οι ενδιάμεσες μηχανές θα καθυστερήσουν την παραγωγή τους λόγω αποκ-λεισμού και θα ελευθερώσουν το επόμενο κομμάτι τους τη στιγμή που η M_i θα παραγάγει το κομμάτι πάνω στο οποίο αποκλείστηκε.
- Προγραμμάτισε επόμενα γεγονότα σε B_i, M_i, B_{i-1}, M_{i-1}, …, B_{i-k}, όπως στην Ενότητα 3.1.

3. Η B_i αδειάζει

- Ενημέρωσε τα B_i, M_{i+1}, B_{i+1}.
- Αφού η B_i αδειάζει, παρατείνεται ο μεταβατικός χρόνος διέλευσης κομματιού μέσα από αυτήν. Συνεπώς, παρατείνεται κι ο μεταβατικός χρόνος παραγωγής επόμενου κομματιού από την M_{i+1}, λόγω αναμονής πρώτης ύλης από την προηγούμενη μηχανή M_i. Αυτή η παράταση θα επηρεάσει τον μεταβατικό χρόνο άφιξης στην B_{i+1}.
- Προγραμμάτισε επόμενα γεγονότα σε B_i και M_{i+1}, όπως στην Ενότητα 3.1.

4. Η i B_i γίνεται μη-άδεια (ειδικό γεγονός)

- Ενημέρωσε τα B_i, M_{i+1}, B_{i+1}.
- Αποδέσμευσε την M_{i+1} από την M_i θέτοντας το μεταβατικό χρόνο παραγωγής επόμενου κομματιού ίσο προς το μέγιστο χρόνο ανάμεσα σε αυτόν και στον μεταβατικό χρόνο άφιξης στην B_{i+1}. Η παράταση αυτή αποδίδεται και στον μεταβατικό χρόνο αναχώρησης από την B_i.
- Προγραμμάτισε επόμενα γεγονότα σε B_i και M_{i+1}, όπως στην Ενότητα 3.1.

4

ΑΡΙΘΜΗΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

Στην ενότητα αυτή εξετάζουμε τα θέματα ακρίβειας και υπολογιστικής αποδοτικότητας για τα μοντέλα διακριτών γεγονότων που αναλύσαμε. Οι παρακάτω προσομοιωτές αναπτύχθηκαν σε κώδικα C++: ένας λεπτομερής (PP) προσομοιωτής, βασισμένος στον Αλγόριθμο 2.1 και στο διάγραμμα ροής του Σχ. 2.2, κι ένας ακόμη, ταχύς (FAST), που αντιστοιχεί στο μοντέλο διακριτών κομματιών. Ο δεύτερος επίσης βασίστηκε στον Αλγόριθμο 3.1 αλλά, όπως αναλύσαμε στην Ενότητα 3.1, χρησιμοποιεί διαφορετικές εξισώσεις για ενημέρωση και ρύθμιση των καταστάσεων και για τον προγραμματισμό των επόμενων γεγονότων.

Για να συγκρίνουμε τα μοντέλα κάτω από τις ίδιες πειραματικές συνθήκες, κάνουμε χρήση της τεχνικής *κοινών τυχαίων αριθμών* [3]. Αυτό σημαίνει πως, για συγκεκριμένο τύπο γεγονότος και μια δοθείσα μηχανή, οι προσομοιωτές χρησιμοποιούν μια κοινή ακολουθία τυχαίων αριθμών. Η χρήση των κοινών τυχαίων αριθμών επιτρέπει τις δίκαιες συγκρίσεις διαφορετικών μοντέλων που θα γίνουν πάνω σε σύντομες προσομοιώσεις.

Ο υπολογιστικός χρόνος (CPU time) που απαιτείται για την εκτέλεση μιας προσομοίωσης εξαρτάται από τον αριθμό των γεγονότων που σημειώνονται κατά τη διάρκειά της. Για το κλασικό (PP) μοντέλο, ο αριθμός των γεγονότων είναι οι φορές που οι μηχανές M_i, i = 1, 2, …, παράγουν κομμάτια. Εάν η χωρητικότητα των ενδιάμεσων αποθηκών είναι πεπερασμένη, τότε η παραγωγές των μηχανών είναι κατά προσέγγιση ίσες. Έτσι, ο υπολογιστικός χρόνος είναι ανάλογος της συνολικής παραγωγής του συστήματος, επαυξημένος κατά το χρόνο που απαιτεί η εκτέλεση των φαινομένων αποκλεισμού κι αποστέρησης, που χρειάζονται κάποιους παραπάνω υπολογισμούς. Αντίθετα, ο αριθμός των γεγονότων του ταχέως μοντέλου δεν εξαρτάται άμεσα από τη συνολική παραγωγή, αλλά από τον αριθμό των βλαβών, και τον αριθμό των γεγονότων αποκλεισμού κι αποστέρησης.

Διεξάγαμε πολλά πειράματα για την εξέταση των διάφορων παραγόντων που επηρεάζουν τις υπολογιστικές απαιτήσεις του μοντέλου. Σε κάθε πείραμα, αλλάζουμε μία παράμετρο του συστήματος και βλέπουμε την επίδραση στον υπολογιστικό χρόνο που απαιτείται για την παραγωγή 10,000,000 κομματιών (υπολογιστικός χρόνος ανά δέκα εκατομμύρια κομμάτια). Οι δεδομένες τιμές των παραμέτρων είναι οι παρακάτω

ρυθμός παραγωγής $\Delta_i = 10$ μονάδες χρόνου/κομμάτι, i = 1, 2, ...

πιθανότητα βλάβης $f_i = 0.01$

πιθανότητα επισκευής $r_i = 0.01$

χωρητικότητες αποθηκών $BC_i = 10$

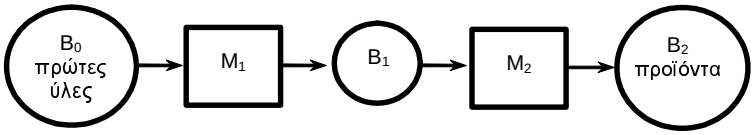
συνολική παραγωγή = 10 000 000 κομμάτια

Τα πειράματα έγιναν αρχικά για σύστημα δύο μηχανών, όπου ο ταχύς αλγόριθμος είναι ακριβής σε σχέση με τον κλασικό. Λόγω των ειδικών περιπτώσεων που προαναφέραμε όμως, οι οποίες και υφίστανται για πάνω από τρεις μηχανές, ο ταχύς αλγόριθμος παρουσιάζει

απόκλιση της τάξεως του 1–1.5% από τα ακριβή αποτελέσματα του λεπτομερούς. Παρακάτω παρουσιάζονται τα πειράματα για σύστημα δύο μηχανών, αλλά και για άνω των τριών, και παραθέτονται οι όποιες αποκλίσεις μεταξύ τους.

4.1. ΣΥΣΤΗΜΑ ΔΥΟ ΜΗΧΑΝΩΝ

Το σύστημα με δύο μηχανές είναι της μορφής του Σχ. 4.1



Σχήμα 4.1. Γραμμή παραγωγής με δύο μηχανές.

Το Σχ. 4.2 δείχνει τις εκτιμήσεις για το μέσο ρυθμό παραγωγής (TH) του λεπτομερούς (PP) μοντέλου, και τους υπολογιστικούς χρόνους (CPU times) ανά δέκα εκατομμύρια κομμάτια, ως συναρτήσεις των πιθανοτήτων επισκευής r_1, r_2. Παρατηρούμε πως με αύξηση της πιθανότητας επισκευής, η παραγωγικότητα του συστήματος αυξάνει, κάτι λογικό εφόσον μειώνεται ο χρόνος επισκευής των μηχανών για κάθε φορά που υποστούν βλάβη.

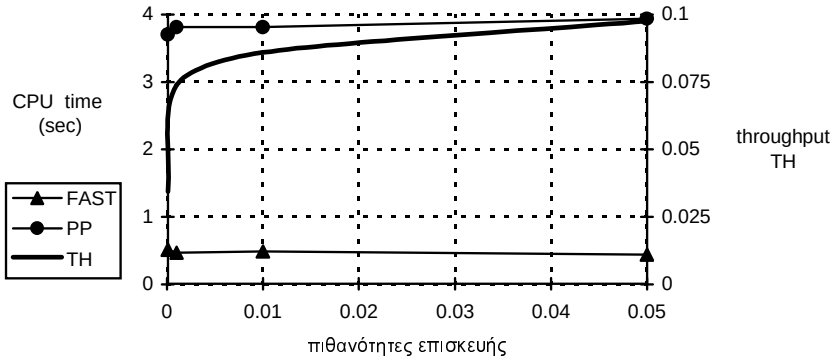
Η παραγωγικότητα του συστήματος καθορίζεται από τη λειτουργικότητα των μηχανών, κι επομένως από τις πιθανότητες επισκευής και βλάβης. Η λειτουργικότητα των μηχανών (η) υπολογίζεται ως εξής

$$\eta = \frac{1}{\text{μέσος χρόνος παραγωγής ενός κομματιού}}$$

$$= \frac{1}{\Delta + \left(\begin{array}{c}\text{μέσος αριθμός βλαβών κατά}\\ \text{την παραγωγή ενός κομματιού}\end{array}\right) \times \left(\begin{array}{c}\text{μέσος}\\ \text{χρόνος επισκευής}\end{array}\right)}$$

$$= \frac{1}{\Delta + \frac{f_i}{1-f_i} \times \frac{r_i}{1-r_i}} = 0.099989 \approx 0.1 \text{ κομμάτια/ μονάδα χρόνου}$$

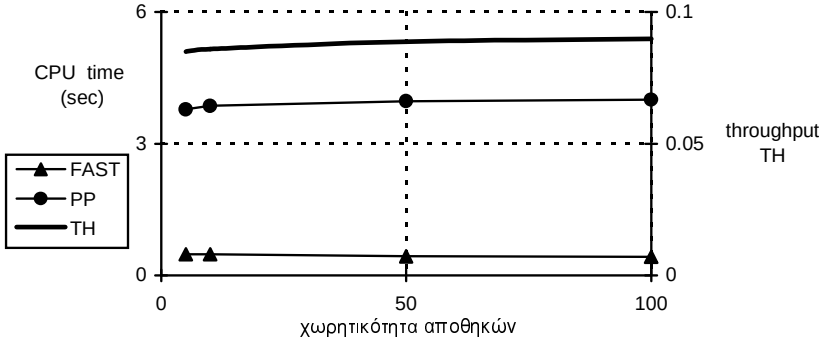
Για πολύ μεγάλους ρυθμούς παραγωγής Δ, ο αριθμός των γεγονότων που παρατηρεί το ταχύ μοντέλο είναι ανάλογος του αριθμού των βλαβών, που, σύμφωνα με την υπόθεση ότι οι βλάβες εξαρτώνται από τη λειτουργία των μηχανών, εξαρτάται από την παραγωγικότητα TH και είναι ανεξάρτητη του Δ.



Σχήμα 4.2. Υπολογιστικός χρόνος και μέσος ρυθμός παραγωγής ως προς τις πιθανότητες επισκευής.

Επίσης, από το ίδιο σχήμα, παρατηρούμε ότι, όσο αυξάνει η πιθανότητα επισκευής, ο υπολογιστικός χρόνος (CPU time) αυξάνει για το λεπτομερές μοντέλο αλλά μειώνεται για το ταχύ. Αυτό οφείλεται στο ότι μειώνονται οι ενημερώσεις που πρέπει το ταχύ μοντέλο να κάνει για τους μεταβατικούς χρόνους, μιας και οι χρόνοι επισκευής γίνονται μικρότεροι.

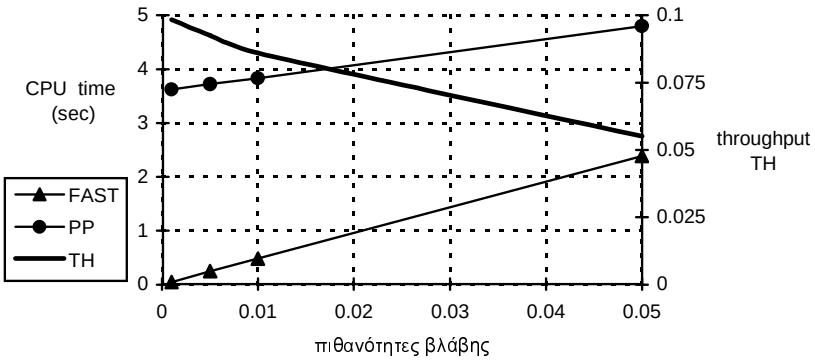
Εφόσον τα πειράματα προσομοίωσης χρησιμοποιούν κοινούς τυχαίους αριθμούς, τα δύο μοντέλα αποδίδουν τις ίδιες ακριβώς εκτιμήσεις για την παραγωγικότητα TH και τα μέσα αποθέματα B_{mi} των μηχανών. Το Σχ. 4.3 δείχνει την εξάρτηση της παραγωγικότητας και του υπολογιστικού χρόνου από την χωρητικότητα της ενδιάμεσης αποθήκης BC_1. Όπως είναι αναμενόμενο, η μείωση της χωρητικότητας αυξάνει τη συχνότητα φαινομένων αποκλεισμού κι, επομένως, αυξάνονται οι υπολογιστικές απαιτήσεις. Επίσης, όσο η χωρητικότητα τείνει στο άπειρο, η παραγωγικότητα τείνει στο $0.099989 \approx 0.1$, που είναι η παραγωγικότητα της M_1 ή της M_2 (οι βασικές παράμετροι των δύο μηχανών είναι ίσες).



Σχήμα 4.3. Υπολογιστικός χρόνος και μέσος ρυθμός παραγωγής ως προς την χωρητικότητα BC_1.

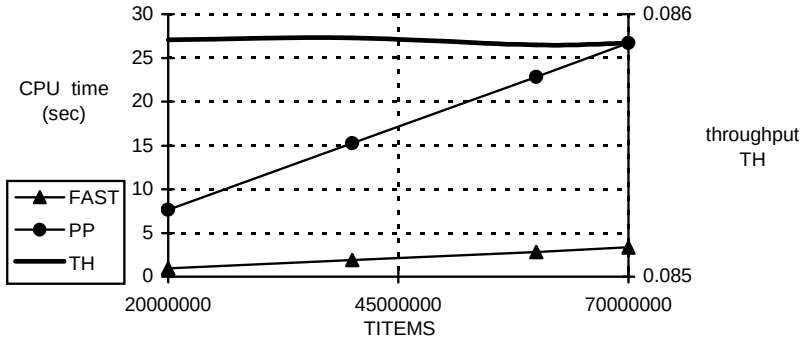
Το Σχ. 4.4 περιέχει τα αποτελέσματα προσομοίωσης για διάφορες τιμές των πιθανοτήτων βλάβης f_1 κι f_2, όπου $f_1 = f_2$. Με αύξηση των πιθανοτήτων βλάβης, σημειώνεται όπως είναι αναμενόμενο και αύξηση στη συχνότητα των βλαβών κι, επομένως, αύξηση στα φαινόμενα αποκλεισμού κι αποστέρησης. Έτσι, οι υπολογιστικοί χρόνοι των μοντέλων είναι αύξουσες συναρτήσεις των πιθανοτήτων βλάβης. Παρόλα αυτά, το λεπτομερές μοντέλο δεν εμφανίζει μείωση στην απόδοσή του. Από το σχήμα παρατηρούμε ότι για όλες τις τιμές πιθανότητας βλάβης, το ταχύ μοντέλο είναι αποδοτικότερο του απλού, όσο όμως αυξάνεται η πιθανότητα αυτή, η ταχύτητα των δύο μοντέλων τείνει να διασταυρωθεί. Μεγαλύτερες πιθανότητες βλάβης σπάνια υφίστανται σε πραγματικές γραμμές παραγωγής. Εαν η γραμμή δεν υφίσταται καθόλου βλάβες, από μια μεταβατική περίοδο και μετά, δεν θα σημειώνονται καθόλου

γεγονότα στο ταχύ μοντέλο μέχρι το τέλος της προσομοίωσης. Στην περίπτωση αυτή, το νέο μοντέλο είναι απείρως ταχύτερο του λεπτομερούς.



Σχήμα 4.4. Υπολογιστικός χρόνος και μέσος ρυθμός παραγωγής ως προς τις πιθανοτήτες βλάβης.

Τέλος, ένα ακόμη πείραμα αφορά στο συνολικό όγκο παραγωγής, όπως φαίνεται στο Σχ. 4.5, κι έγινε κυρίως για την έμφαση στη διαφορά υπολογιστικού χρόνου ανάμεσα στα δύο μοντέλα, όταν προσομοιώνουν με τις ίδιες τιμές για τις βασικές παραμέτρους τους. Με αύξηση των συνολικών κομματιών προς παραγωγή, παρατηρούμε πως η παραγωγικότητα παραμένει πρακτικά σταθερή, κάτι λογικό εφόσον και τα δύο μοντέλα είναι ανεξάρτητα της συνολικής τους παραγωγής. Εξάλλου, και στα δύο ο υπολογιστικός χρόνος αυξάνει όσο περισσότερα κομμάτια καλούνται να παράγουν, με τη διαφορά στην ταχύτητα όμως να παραμένει σημαντική.



Σχήμα 4.5. Υπολογιστικός χρόνος και μέσος ρυθμός παραγωγής ως προς τη συνολική παραγωγή.

4.2. ΣΥΣΤΗΜΑ ΠΟΛΛΩΝ ΜΗΧΑΝΩΝ

Τώρα θα παρουσιάσουμε αποτελέσματα ενδεικτικών πειραμάτων που έγιναν για σύστημα με έξι (6) και περισσότερες μηχανές. Οι αποκλίσεις του δεύτερου μοντέλου έχουν ήδη αναφερθεί κι αναλυθεί σε προηγούμενη ενότητα, καθώς και οι αιτίες εμφάνισής τους.

Τα νέα δεδομένα λοιπόν, είναι

αριθμός μηχανών n = 6 (60)

ρυθμός παραγωγής $\Delta_i = 10$ μονάδες χρόνου/κομμάτι, i = 1, 2, ..., 6 (60)

πιθανότητα βλάβης $f_i = 0.01$

πιθανότητα επισκευής $r_i = 0.01$

χωρητικότητες αποθηκών $BC_i = 10$

συνολική παραγωγή = 10^7 ($7*10^7$) κομμάτια

Πίνακας 4.1: Σύγκριση αποτελεσμάτων FAST και PP για γραμμές παραγωγής με 6 μηχανές (10^7 κομμάτια)

	TH	**B_{m1}**	**B_{m2}**	**B_{m3}**	**B_{m4}**	**B_{m5}**	CPU (sec)
PP	0.076655	7.217	5.959	5.008	4.057	2.788	13.875
FAST	0.076674	7.219	5.970	5.014	4.053	2.781	2.187
σφάλμα (%)	**0.025**	**0.028**	**0.184**	**0.120**	**0.098**	**0.251**	

Παρατηρούμε πως για γραμμή παραγωγής με έξι μηχανές, το ταχύ μοντέλο είναι πάνω από έξι φορές πιο γρήγορο από το λεπτομερές. Οι αποκλίσεις κυμαίνονται σε πολύ χαμηλά επίπεδα, με τη μεγαλύτερη να είναι περίπου 0.250 %. Μάλιστα, όσο προχωρούμε προς το τέλος της αλυσίδας, βλέπουμε τις αποκλίσεις από τις ακριβείς τιμές να αυξάνονται. Αυτό οφείλεται στη μη σωστή διαδοχή εκτέλεσης ορισμένων γεγονότων που λαμβάνουν χώρα την ίδια στιγμή. Τέτοια γεγονότα συμβαίνουν συχνότερα προς τις τελευταίες μηχανές, όπου η πολυπλοκότητα του συστήματος είναι αυξημένη. Όσο λοιπόν προχωρά το ρολόι της προσομοίωσης, τα σφάλματα συσσωρεύονται και προκαλούν αύξηση στην απόκλιση.

Πίνακας 4.2: Σύγκριση αποτελεσμάτων FAST και PP για γραμμές παραγωγής με 6 μηχανές ($7*10^7$ κομμάτια)

	TH	**B_{m1}**	**B_{m2}**	**B_{m3}**	**B_{m4}**	**B_{m5}**	CPU (sec)
PP	0.076564	7.229	5.972	5.003	4.025	2.768	95.890
FAST	0.076578	7.230	5.975	5.007	4.032	2.770	15.265
σφάλμα (%)	**0.018**	**0.014**	**0.050**	**0.080**	**0.174**	**0.072**	

Για σύστημα έξι μηχανών και πολύ μεγάλο όγκο συνολικής παραγωγής ($7*10^7$ κομμάτια), τα σφάλματα παραμένουν σε χαμηλά επίπεδα (μέγιστη απόκλιση ≈ 0.175 %). Η διαφορά στην ταχύτητα μεταξύ των δύο μοντέλων είναι επίσης προφανής, με το FAST μοντέλο να είναι περίπου έξι φορές ταχύτερο του PP.

Πίνακας 4.3: Σύγκριση αποτελεσμάτων FAST και PP για γραμμές παραγωγής με 60 μηχανές (10^7 κομμάτια)

	TH	**B_{m1}**	**B_{m2}**	**B_{m3}**	**B_{m29}**	**B_{m30}**	**B_{m31}**	**B_{m58}**	**B_{m59}**	**CPU (sec)**
PP	0.066171	8.291	7.644	7.280	5.081	5.031	4.985	2.382	1.712	273.422
FAST	0.066913	8.321	7.696	7.349	5.304	5.266	5.229	2.476	1.781	36.576
σφάλμα (%)	**0.074**	**0.362**	**0.680**	**0.948**	**4.389**	**4.671**	**4.895**	**3.946**	**4.030**	

Μία δοκιμή έγινε και για πολύ μεγάλη γραμμή παραγωγής (60 μηχανές, συνολική παραγωγή 10^7 κομμάτια), όπου φαίνεται ότι οι αποκλίσεις ξεπερνούν τα αποδεκτά όρια, κάτι που καθιστά το FAST μοντέλο μη εφαρμόσιμο σε τέτοιες, ακραίες, περιπτώσεις. Σε συνήθεις γραμμές παραγωγής όμως, αποδεικνύεται ότι το νέο μοντέλο μπορεί να λειτουργήσει με ικανοποιητικά αποτελέσματα και με πολύ μεγάλη βελτίωση στο κριτήριο CPU time.

5

ΣΥΜΠΕΡΑΣΜΑΤΑ

Στην εργασία αυτή αναπτύξαμε ένα μοντέλο προσομοίωσης γραμμών παραγωγής με μηχανές που έχουν ίσους ρυθμούς παραγωγής, αλλά υφίστανται βλάβες και επισκευάζονται σε τυχαίους χρόνους. Μεταξύ των μηχανών υπάρχουν αποθήκες πεπερασμένης χωρητικότητας.

Οι συμβατικοί αλγόριθμοι προσομοίωσης που χρησιμοποιούνται για την εκτίμηση της απόδοσης συστημάτων παραγωγής σήμερα είναι αργοί γιατί παρακολουθούν την κίνηση κάθε κομματιού καθώς αυτό εισέρχεται κι εξέρχεται από κάθε μηχανή.

Το μοντέλο που αναπτύχθηκε αποφεύγει την παρατήρηση τέτοιων γεγονότων. Αντί γι' αυτά παρατηρεί το σύστημα μόνον όταν οι μηχανές υφίστανται βλάβες, αποκλείονται ή αποστερούνται και στα ενδιάμεσα χρονικά διαστήματα υπολογίζει τα αποθέματα των αποθηκών και την παραγωγή κάθε μηχανής με αναλυτικό τρόπο.

Η φιλοσοφία του αλγορίθμου βασίζεται σε προηγούμενο αλγόριθμο που έχει αναπτυχθεί στο Εργαστήριο CAM του Τμήματος ΜΠΔ του Πολυτεχνείου Κρήτης. Ο νέος αλγόριθμος παρατηρεί λιγότερα γεγονότα από τον προηγούμενο και είναι περίπου δύο φορές ταχύτερος.

Από πειραματικά αποτελέσματα προκύπτει ότι ο προτεινόμενος αλγόριθμος είναι πάνω από 6 φορές ταχύτερος από έναν συμβατικό αλγόριθμο προσομοίωσης, ενώ τα σφάλματα εκτίμησης της παραγωγικότητας ή των μέσων αποθεμάτων είναι μικρά (<2% στις περισσότερες περιπτώσεις).

Η εργασία αυτή αποτελεί αφετηρία για την ανάπτυξη ενός βελτιωμένου αλγορίθμου προσομοίωσης ο οποίος θα είναι ακριβής και θα μπορεί να περιγράψει μηχανές με διαφορετικούς ρυθμούς παραγωγής.

ΠΑΡΑΡΤΗΜΑ

ΚΩΔΙΚΕΣ ΠΡΟΣΟΜΟΙΩΣΗΣ

Α. ΛΕΠΤΟΜΕΡΕΣ ΜΟΝΤΕΛΟ

```
//                    PROGRAM "RPL"
// FAST PIECE-BY-PIECE SIMULATOR
// ONLY DEPARTURE EVENTS ARE EXAMINED

// piece-driven simulator for lines
// processing times equal to ten (10) for each machine
// parts-to-failure = geometric r.v's

//                    OCTOBER 2005

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <time.h>
#include <sys/timeb.h>

struct domi1 {
     int *MSTATE;
     int *NRNG;
     long *ntf;
     long **icrn;
     long *NPROD;
     float *RPROB;
     float *FPROB;
     } mac;
struct domi2 {
     long *TBO;
     long *MEVTIME;
     long *ncap;
     long *leve;
     double *MBLEV;
```

```
     } buf;
struct domi3 {
     long timnow;
     long oldtim;
     long SIMTIM;
     long stoptim;
     } tim;

int nm;
int iendsim;
long titems;
long XREP=0;

void nextevt(int *n);
void departure(int n);
void process(int n);
long zinext(long zi);

void main (void)
{
   FILE *input_fp;
   FILE *output_fp;
   FILE *test;
   int i, j, k, N;
// char choice;  //epilogi gia dimiourgia i mi anal. arxeiou prosomoiwsis
   int ie;
   long zi,zinew;
   float u;
   long double zx,z1;
   struct _timeb tstruct;
   char tmbuf_start[128], tmbuf_end[128];
   time_t ltime_start, ltime_end;
   int msec1,msec2;

   if((input_fp=fopen("input.txt","r"))==NULL)
   {
     printf("Problem in opening file input.txt.\n\n");
     exit(2);
   }
   else
     printf("File input.txt was opened and read successfully.\n\n");
   fscanf(input_fp,"%d %d %ld\n\n", &ie, &nm, &titems);
   printf("\nInitial Generator Seed:%d  N. Of Machines=%d \
     Total Items=%ld\n\n",ie,nm,titems);

// printf("Type 'y' or 'n' to create or not a detailed simulation process file\n");
```

```
// choice = getchar();

   printf("Processing times are equal to ten for each machine: R(i)=10, i=1,..,nm\n\n\n");

   mac.MSTATE=(int *)malloc((nm+2)*sizeof(int));
   mac.ntf=(long *)malloc((nm+2)*sizeof(long));
   mac.NRNG=(int *)malloc((nm+2)*sizeof(int));
   mac.NPROD=(long *)malloc((nm+2)*sizeof(long));
   mac.icrn=(long **)malloc((nm+2)*sizeof(long));
     for(i=0;i<=(nm+1);i++)
        mac.icrn[i]=(long *)malloc(4*sizeof(long));
   mac.RPROB=(float *)malloc((nm+2)*sizeof(float));
   mac.FPROB=(float *)malloc((nm+2)*sizeof(float));
   buf.MBLEV=(double *)malloc((nm+2)*sizeof(double));
   buf.TBO=(long *)malloc((nm+2)*sizeof(long));
   buf.MEVTIME=(long *)malloc((nm+2)*sizeof(long));
   buf.ncap=(long *)malloc((nm+2)*sizeof(long));
   buf.leve=(long *)malloc((nm+2)*sizeof(long));

   if( !mac.MSTATE || !mac.NPROD || !mac.NRNG || !mac.RPROB || !mac.FPROB  || !mac.icrn ||
     !buf.MBLEV || !buf.TBO || !buf.MEVTIME || !buf.ncap || !buf.leve)
   {
     printf("Memory allocation error...\n\n");
     exit(1);
   }
   tim.SIMTIM = 2147483646;
   for(i=1;i<=nm;i++)
   {
     fscanf(input_fp,"%ld",&buf.ncap[i]);
     fscanf(input_fp,"%ld",&buf.leve[i]);
     fscanf(input_fp,"%f",&mac.FPROB[i]);
     fscanf(input_fp,"%f\n",&mac.RPROB[i]);
     mac.NPROD[i] = 0;
     buf.TBO[i] = 0;
     buf.MBLEV[i] = 0;
     mac.NRNG[i] = 0;
     mac.ntf[i] = 0;
   }
   fclose(input_fp);
   if((output_fp=fopen("output RPL.txt","w"))==NULL)
   {
     printf("Problem in opening file output.txt.\n\n");
     exit(2);
   }
   else
     printf("File output.txt was opened successfully.\n\n");
```

```
   fprintf(output_fp,"                    ******************  EXACT  PIECE-DRIVEN  ALGORITHM
******************\n\n");
   fprintf(output_fp,"          P A R A M E T E R S  O F  M A C H I N E S  A N D  B U F F E R
S :\n\n");
   fprintf(output_fp,"                -----I--------M A C H I N E--------I-----B U F F E R----
\n");
   fprintf(output_fp,"                      I    FAILURE    |    REPAIR   I          | INITIAL
\n");
   fprintf(output_fp,"                   N. I  PROBABILITY  | PROBABILITY I CAPACITY |  LEVEL
\n");
   fprintf(output_fp,"                -----I---------------|-------------I----------|---------
\n");


   for(N=1;N<=nm;N++)
     fprintf(output_fp,"%19d  I%10.3f     |%9.3f    I%6ld    |%4ld\n",
     N,mac.FPROB[N],mac.RPROB[N],buf.ncap[N],buf.leve[N]);
   fprintf(output_fp,"\n\n\n");
   fclose(output_fp);
   tim.stoptim = 2147483647;
   iendsim=0;
   if(ie>21474)
     ie=1;
   zx=1973272912;
   z1=2147483647;
   for(i=1;i<=ie;i++)
   {
     zx = (long) ((715*zx/z1 - (long) (715*zx/z1))*z1);
     zx = (long) ((1058*zx/z1 - (long) (1058*zx/z1))*z1);
     zx = (long) ((1385*zx/z1 - (long) (1385*zx/z1))*z1);
   }
   for(i=1;i<=nm;i++)
   {
     for(j=1;j<=2;j++)
     {
        zx = (long) ((715*zx/z1 - (long) (715*zx/z1))*z1);
        zx = (long) ((1058*zx/z1 - (long) (1058*zx/z1))*z1);
        zx = (long) ((1385*zx/z1 - (long) (1385*zx/z1))*z1);
        mac.icrn[i][j] = (long) zx;
     }
   }
   _tzset();
   _strtime(tmbuf_start);
   time(&ltime_start);
   _ftime(&tstruct);
   msec1 = tstruct.millitm;


   for(N=1;N<=nm;N++)
```

```
   {
     zi = (long) mac.icrn[N][1];
     if(mac.FPROB[N]>0.000001)
     {
        zinew = zinext(zi);
        u = (float) (2*(zinew/256)+1)/16777216;
        mac.ntf[N] = (int) (log(u)/log(1-mac.FPROB[N])) + 1;
        mac.icrn[N][1] = zinew;
     }
     else
        mac.ntf[N]=2147483646;

     if((N==1) || (buf.leve[N-1]>0))
     {
        process(N);
        mac.MSTATE[N] = 1;
     }
     else
     {
        buf.MEVTIME[N]=tim.stoptim;
        mac.MSTATE[N] = 0;
     }
   }
   buf.ncap[nm]=titems;
   buf.leve[nm]=0;
   tim.timnow = 0;
   tim.oldtim = 0;
   mac.MSTATE[0] = 1;
   mac.MSTATE[nm+1] = 1;
   buf.leve[0] = 5*titems;
   if((test=fopen("testRPL.txt","w"))==NULL)
   {
     printf("Problem in opening file testRPL.txt.\n\n");
     exit(3);
   }
   else
     printf("File testRPL.txt was opened successfully.\n\n");

LABEL_CONTINUE:
/* switch(choice)     // commented part
   {
     case 'y':
     {
        fprintf(test,"%ld %2d  ",tim.timnow,N);
        if(XREP>0)
        {
```

```
          fprintf(test,"XREP:%ld",XREP);
          XREP=0;
        }
        fprintf(test,"\n");
        fprintf(test,"    N:");
        for(k=1;k<=nm;k++)
          fprintf(test,"%8d ",k);
        fprintf(test,"\n");
        fprintf(test," PROD:");
        for(k=1;k<=nm;k++)
          fprintf(test,"%8ld ",mac.NPROD[k]);
        fprintf(test,"\n");
        fprintf(test,"MTIME:");
        for(k=1;k<=nm;k++)
          fprintf(test,"%8ld ",__min(buf.MEVTIME[k],9999));
        fprintf(test,"\n");
        fprintf(test,"    B:");
        for(k=1;k<=nm;k++)
          fprintf(test,"%8ld ",__max(0,buf.leve[k]-1));
        fprintf(test,"\n");
        fprintf(test,"STATE:");
        for(k=1;k<=nm;k++)
          fprintf(test,"%8ld ",mac.MSTATE[k]);
        fprintf(test,"\n\n");
      }
      case 'n':
        break;
      default:
        break;
    }*/

    nextevt(&N);
    if(iendsim==1)
      goto LABEL_STOP;
    departure(N);
    goto LABEL_CONTINUE;

LABEL_STOP:
    fclose(test);
    _strtime(tmbuf_end);
    time(&ltime_end);
    _ftime(&tstruct);
    msec2 = tstruct.millitm;
    for(N=1;N<=nm-1;N++)
    {
      buf.leve[N] = __max((buf.leve[N]-1), 0);
```

```
     buf.MBLEV[N] = (buf.MBLEV[N] +
        (buf.leve[N]*(tim.oldtim-buf.TBO[N])))/tim.oldtim;
   }
   if((output_fp=fopen("output RPL.txt","a"))==NULL)
   {
     printf("Problem in opening file output.txt.\n\n");
     exit(4);
   }
   else
     printf("File output.txt was opened successfully.\n\n");

   for(i=0;i<85;i++)
     fprintf(output_fp,"%c",'-');
   fprintf(output_fp,"\n%23cS I M U L A T I O N   R E S U L T S\n",' ');
     for(i=0;i<85;i++)
     fprintf(output_fp,"%c",'-');
   fprintf(output_fp,"\n\n");
   fprintf(output_fp,"%13c-----I------ M A C H I N E S ------I---- B U F F E R S ----\n",' ');
   fprintf(output_fp,"%18cI                              I- - - -L E V E L- - - -\n",' ');
   fprintf(output_fp,"%15cN. I  TOT. PROD. I  N. OF FAILS  I  FINAL  |   AVERAGE\n",' ');
   fprintf(output_fp,"%13c-----I-------------I---------------I---------|-------------\n",' ');
   for(N=1;N<=nm-1;N++)
     fprintf(output_fp,"%14c%2d  I %10ld  I    %5d      I  %3ld    | %8.3lf\n",
                ' ',N,mac.NPROD[N],mac.NRNG[N],buf.leve[N],buf.MBLEV[N]);
   fprintf(output_fp,"%14c%2d      I   %10ld      I            %5d                    I\n\n",'
',nm,mac.NPROD[nm],mac.NRNG[nm]);
   for(i=0;i<43;i++)
     fprintf(output_fp,"%c%c",'-',' ');
   fprintf(output_fp,"\n   S I M U L A T I O N   P E R I O D: %ld\n",tim.oldtim);
   fprintf(output_fp,"MEAN   THROUGHPUT   RATE   OF   LAST   MACHINE:   %.6lf\n\n",((double)
mac.NPROD[nm])/tim.oldtim);
   fprintf(output_fp," STARTING TIME:%s\n",tmbuf_start);
   fprintf(output_fp,"    FINAL TIME:%s\n",tmbuf_end);
   if(ltime_end>ltime_start)
   {
     if(msec2<msec1)
     {
       if(ltime_end-ltime_start==1)
          fprintf(output_fp,"      CPU TIME:%u msec\n\n",
          msec2 + 1000 - msec1);
       else
          fprintf(output_fp,"      CPU TIME:%ld sec plus %u msec\n\n",
          (ltime_end - ltime_start-1),msec2 + 1000 - msec1);
     }
     else if(msec2>=msec1)
       fprintf(output_fp,"      CPU TIME:%ld sec plus %u msec\n\n",
```

```
        ltime_end - ltime_start,msec2 - msec1);
   }
   else
     fprintf(output_fp,"      CPU TIME:%u msec\n\n",msec2 - msec1);
   fclose(output_fp);
   free(mac.MSTATE);
   free(mac.ntf);
   free(mac.icrn);
   free(mac.NRNG);
   free(mac.RPROB);
   free(mac.FPROB);
   free(mac.NPROD);
   free(buf.MBLEV);
   free(buf.TBO);
   free(buf.MEVTIME);
   free(buf.ncap);
   free(buf.leve);
}

void nextevt(int *n)
{
   int i;

   tim.oldtim = tim.timnow;
   tim.timnow = tim.stoptim;
   for(i=1;i<=nm;i++)
   {
     if(buf.MEVTIME[i] < tim.timnow)
     {
        tim.timnow = buf.MEVTIME[i];
        *n = i;
     }
   }
   if((tim.timnow > tim.SIMTIM) ||
     ((buf.leve[nm]) >= (buf.ncap[nm])))
   {
     iendsim = 1;
   }
}

void departure(int n)
{
   int k, j;
   long DURATION;

   if((buf.leve[n]) >= (buf.ncap[n])+1)
```

```
   {
     mac.MSTATE[n] = 2;
     buf.MEVTIME[n] = tim.stoptim;
   }
   else
   {
     if((mac.MSTATE[n+1]) == 0)
     {
       mac.MSTATE[n+1] = 1;
       process(n+1);
     }
     else if(n<nm)
     {
       DURATION = buf.MEVTIME[n] - buf.TBO[n];
       buf.MBLEV[n] += DURATION*((buf.leve[n])-1);
       buf.TBO[n] = buf.MEVTIME[n];
     }
     k=n;
     j=n-1;

LABEL_R:
     buf.leve[k]++;
     mac.NPROD[k]++;
     mac.ntf[k]--;
     if(j>0)
       buf.leve[j]--;
     if((j==0) || ((buf.leve[j])>0) ||
       ((mac.MSTATE[j])==2))
     {
       process(k);
       if((j>0) && ((mac.MSTATE[j])==2))
       {
         mac.MSTATE[j] = 1;
         k--;
         j--;
         goto LABEL_R;
       }
       else if(j>0)
       {
         DURATION = tim.timnow - buf.TBO[j];
         buf.MBLEV[j] += DURATION*(buf.leve[j]);
         buf.TBO[j] = tim.timnow;
       }
     }
     else
     {
```

```
        mac.MSTATE[k] = 0;
        buf.MEVTIME[k] = tim.stoptim;
     }
   }
}

void process(int n)
{
   int i,nfail;
   long zp,zinew;
   float u;

   buf.MEVTIME[n] = tim.timnow + 10;
   if((mac.ntf[n]) <= 0)
   {
     nfail=0;
     do{
        zp = mac.icrn[n][1];
        zinew = zinext(zp);
        u = (float) (2*(zinew/256)+1)/16777216;
        mac.ntf[n] = (int) (log(u)/log(1-(mac.FPROB[n])));
        mac.icrn[n][1] = zinew;
        nfail++;
     }while((mac.ntf[n]) < 1);
     XREP = 0;
     for(i=1;i<=nfail;i++)
     {
        zp = mac.icrn[n][2];
        zinew = zinext(zp);
        u = (float) (2*(zinew/256)+1)/16777216;
        XREP += (long) (log(u)/log(1-(mac.RPROB[n])));
        mac.icrn[n][2] = zinew;
     }
     buf.MEVTIME[n] += XREP;
     mac.NRNG[n] += nfail;
   }
}

long zinext(long zi)
{
   long XHI, XALO, LEFTLO, FHI, K;
   XHI=zi/65536;
   XALO=(zi-XHI*65536)*24112;
   LEFTLO=XALO/65536;
   FHI=XHI*24112+LEFTLO;
   K=FHI/32768;
```

```
   zi = (((XALO-LEFTLO*65536)-2147483647)+
      (FHI-K*32768)*65536)+K;
   if(zi<0)
     zi += 2147483647;

   XHI=zi/65536;
   XALO=(zi-XHI*65536)*26143;
   LEFTLO=XALO/65536;
   FHI=XHI*26143+LEFTLO;
   K=FHI/32768;
   zi = (((XALO-LEFTLO*65536)-2147483647)+
      (FHI-K*32768)*65536)+K;
   if(zi<0)
     zi += 2147483647;
   return zi;
}
```

B. ΤΑΧΥ ΜΟΝΤΕΛΟ

```
//                    ROGRAM "FRL"
// WAITING TIMES ARE TRANSIENT AND ARE EXAMINED WITH THE PROCESSING
// RATES. THUS, REPAIR EVENTS DO NOT HAVE TO BE TAKEN INTO ACCOUNT
// AND "FRL" IS MORE EFFICIENT THAN "RPL"

// EVENT-DRIVEN SIMULATOR OF A DISCRETE EVENT
// PRODUCTION LINE WITH DETERMINISTIC PROCESSING TIMES
// PROCESSING TIMES ARE EQUAL TO 10 (TEN) FOR ALL MACHINES

//                    OCTOBER 2005

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include <time.h>
#include <sys/timeb.h>

struct times {
     long *TE;
     long *TEM;
     long *TEB;
     long *TMO;
     long *TBO;
     long *TM;
     long **TB;
     } tim;
struct machines {
     long *PNE;
     long *PROD;
     long *NRNG;
     long **ICRN;
     float *RP;
     float *FP;
     } mac;
struct buffers {
     int *LSTATE;
     long *B;
     long *BC;
     double *BMLEV;
     } buf;
struct event {
     int *MOB;
```

```
     int *NEB;
     int MAORBUF;
     int NEL;
     } sel;

int nm;
int iendsim;
long titems;
long XREP=0;
long SIMTIM;
long STOPTIM;

void initial(long *tep);
void nextevt(long *tep);
void fails(long tep,int n);
void fills(long tep,int n);
void empties(long tep,int n);
void nempty(long tep,int n);
void felap(long tep,int n);
void feven(long tep,int n);
void macelap(long tep,int n);
void maceven(long tep,int n);
void bufelap(long tep,int n);
void bufeven(long tep,int n);
long emvadon(int n,int i,long dur,double *area);
long  zinext(long zi);

void main (void)
{
   FILE *input_fp;
   FILE *output_fp;
   FILE *test;
   int i, j, k, n;
   struct _timeb tstruct;
   char tmbuf_start[128], tmbuf_end[128];
   time_t ltime_start, ltime_end;
   int msec1,msec2;
// char choice;     //epilogi gia dimiourgia i mi anal. arxeiou prosomoiwsis
   int ie;
   long double zx,z1;
   long TEP;

   if((input_fp=fopen("input.txt","r"))==NULL)
   {
     printf("Problem in opening file input.txt.\n\n");
     exit(1);
```

```
   }
   else
     printf("File input.txt was opened and read successfully.\n\n");
   fscanf(input_fp,"%d %d %ld\n", &ie, &nm, &titems);
   printf("\nInitial Generator Seed:%d  N. Of Machines=%d \
     Total Items=%ld\n\n",ie,nm,titems);

// printf("Type 'y' or 'n' to create or not a detailed simulation process file\n");
// choice = getchar();

   printf("Processing times are equal to ten for each machine: R(i)=10, i=1,..,nm\n\n\n");
   tim.TE=(long *)malloc((nm+2)*sizeof(long));
   tim.TEM=(long *)malloc((nm+2)*sizeof(long));
   tim.TEB=(long *)malloc((nm+2)*sizeof(long));
   tim.TMO=(long *)malloc((nm+2)*sizeof(long));
   tim.TBO=(long *)malloc((nm+2)*sizeof(long));
   tim.TM=(long *)malloc((nm+2)*sizeof(long));
   tim.TB=(long **)malloc(3*sizeof(long));
   for(i=0;i<=2;i++)
     tim.TB[i]=(long *)malloc((nm+2)*sizeof(long));
   mac.PNE=(long *)malloc((nm+2)*sizeof(long));
   mac.PROD=(long *)malloc((nm+2)*sizeof(long));
   mac.NRNG=(long *)malloc((nm+2)*sizeof(long));
   mac.ICRN=(long **)malloc((nm+2)*sizeof(long));
   for(i=0;i<=(nm+1);i++)
     mac.ICRN[i]=(long *)malloc(2*sizeof(long));
   mac.RP=(float *)malloc((nm+2)*sizeof(float));
   mac.FP=(float *)malloc((nm+2)*sizeof(float));
   buf.LSTATE=(int *)malloc((nm+2)*sizeof(int));
   buf.B=(long *)malloc((nm+2)*sizeof(long));
   buf.BC=(long *)malloc((nm+2)*sizeof(long));
   buf.BMLEV=(double *)malloc((nm+2)*sizeof(double));
   sel.MOB=(int *)malloc((nm+2)*sizeof(int));
   sel.NEB=(int *)malloc((nm+2)*sizeof(int));

   if( !tim.TE || !tim.TEM || !tim.TEB || !tim.TMO || !tim.TBO   || !tim.TM || !tim.TB ||
!mac.PNE ||
     !mac.PROD || !mac.NRNG || !mac.ICRN || !mac.RP || !mac.FP || !buf.LSTATE || !buf.B ||
!buf.BC ||
     !buf.BMLEV || !sel.MOB || !sel.NEB)
   {
     printf("Memory allocation error...\n\n");
     exit(2);
   }
   for(i=1;i<=nm;i++)
   {
```

```
     fscanf(input_fp,"%ld",&buf.BC[i]);
     fscanf(input_fp,"%ld",&buf.B[i]);
     fscanf(input_fp,"%f",&mac.FP[i]);
     fscanf(input_fp,"%f\n",&mac.RP[i]);
   }
   fclose(input_fp);
   if((output_fp=fopen("output FRL nempty.txt","w"))==NULL)
   {
     printf("Problem in opening file output.txt.\n\n");
     exit(2);
   }
   else
     printf("File output.txt was opened successfully.\n\n");
   fprintf(output_fp,"           FAST DISCRETE PART MODEL OF PRODUCTION LINES WITHOUT WAITING
TIMES\n\n");
   fprintf(output_fp,"          P A R A M E T E R S  O F  M A C H I N E S  A N D  B U F F E R
S :\n\n");
   fprintf(output_fp,"               -----I--------M A C H I N E--------I-B U F F E R-\n");
   fprintf(output_fp,"                    I    FAILURE    |    REPAIR   I            \n");
   fprintf(output_fp,"                  N. I  PROBABILITY  | PROBABILITY I   CAPACITY  \n");
   fprintf(output_fp,"               -----I---------------|-------------I-------------\n");
   for(n=1;n<=nm;n++)
     fprintf(output_fp,"%19d  I%10.3f     |%9.3f    I%7ld\n",
     n,mac.FP[n],mac.RP[n],buf.BC[n]);
   fprintf(output_fp,"\n\n\n");
   fclose(output_fp);
   SIMTIM = 2147483646;
   STOPTIM = 2147483647;
   iendsim=0;
   if(ie>21474)
     ie=1;
   zx=1973272912;
   z1=2147483647;
   for(i=1;i<=ie;i++)
   {
     zx = (long) ((715*zx/z1 - (long) (715*zx/z1))*z1);
     zx = (long) ((1058*zx/z1 - (long) (1058*zx/z1))*z1);
     zx = (long) ((1385*zx/z1 - (long) (1385*zx/z1))*z1);
   }
   for(i=1;i<=nm;i++)
   {
     for(j=1;j<=2;j++)
     {
        zx = (long) ((715*zx/z1 - (long) (715*zx/z1))*z1);
        zx = (long) ((1058*zx/z1 - (long) (1058*zx/z1))*z1);
        zx = (long) ((1385*zx/z1 - (long) (1385*zx/z1))*z1);
```

```
        mac.ICRN[i][j] = (long) zx;
      }
    }
    for(n=0;n<=nm+1;n++)
    {
      mac.NRNG[n]=0;
      mac.PNE[n]=0;
      mac.PROD[n]=0;
      buf.BMLEV[n]=0;
      tim.TMO[n]=0;
      tim.TBO[n]=0;
    }
    _tzset();
    _strtime(tmbuf_start);
    time(&ltime_start);
    _ftime(&tstruct);
    msec1 = tstruct.millitm;

    initial(&TEP);
    if((test=fopen("testFRL.txt","w"))==NULL)
    {
      printf("Problem in opening file testFRL.txt.\n\n");
      exit(3);
    }
    else
      printf("File testFRL.txt was opened successfully.\n\n");

LABEL_CONTINUE:

/* switch(choice)   // commented part
    {
      case 'y':
      {
        fprintf(test,"\n%ld %d ",TEP,sel.NEL);

        if(sel.MAORBUF==1)
          fprintf(test,"fails XREP:%ld\n",XREP);
        else if(sel.NEB[sel.NEL]==0)
          fprintf(test,"empties\n");
        else if(sel.NEB[sel.NEL]==2)
          fprintf(test,"fills\n");
        else
          fprintf(test,"nempty\n");

        fprintf(test,"    N:");
        for(k=1;k<=nm;k++)
```

```
        fprintf(test,"%8d ",k);
      fprintf(test,"\n");

      fprintf(test," PROD:");
      for(k=1;k<=nm;k++)
        fprintf(test,"%8ld ",mac.PROD[k]);
      fprintf(test,"\n");

      fprintf(test,"   TM:");
      for(k=1;k<=nm;k++)
        fprintf(test,"%8ld ",tim.TM[k]);
      fprintf(test,"\n");

      fprintf(test,"  TB1:");
      for(k=1;k<=nm;k++)
      fprintf(test,"%8ld ",tim.TB[1][k]);
      fprintf(test,"\n");

      fprintf(test,"  TB2:");
      for(k=1;k<=nm;k++)
        fprintf(test,"%8ld ",tim.TB[2][k]);
      fprintf(test,"\n");

      fprintf(test,"    B:");
      for(k=1;k<=nm;k++)
        fprintf(test,"%8ld ",buf.B[k]);
      fprintf(test,"\n");

      fprintf(test,"STATE:");
      for(k=1;k<=nm;k++)
        fprintf(test,"%8ld ",buf.LSTATE[k]);
      fprintf(test,"\n");

      fprintf(test,"  TBO:");
      for(k=1;k<=nm;k++)
        fprintf(test,"%8ld ",tim.TBO[k]);
      fprintf(test,"\n");

      fprintf(test,"  TMO:");
      for(k=1;k<=nm;k++)
        fprintf(test,"%8ld ",tim.TMO[k]);
      fprintf(test,"\n");
      break;
    }
    case 'n':
      break;
```

```
     default:
        break;
   }*/

   nextevt(&TEP);
   if(TEP>=SIMTIM || iendsim==1)
     goto LABEL_STOP;

   if(sel.MAORBUF==1)
     fails(TEP,sel.NEL);

   else if(sel.NEB[sel.NEL]==0)
     empties(TEP,sel.NEL);

   else if(sel.NEB[sel.NEL]==2)
     fills(TEP,sel.NEL);
   else
     nempty(TEP,sel.NEL);
   goto LABEL_CONTINUE;

LABEL_STOP:
   _strtime(tmbuf_end);
   time(&ltime_end);
   _ftime(&tstruct);
   msec2 = tstruct.millitm;
   fclose(test);
   if((output_fp=fopen("output FRL nempty.txt","a"))==NULL)
   {
     printf("Problem in opening file output.txt.\n\n");
     exit(4);
   }
   else
     printf("File output.txt was opened successfully.\n\n");
   for(i=0;i<85;i++)
     fprintf(output_fp,"%c",'-');
   fprintf(output_fp,"\n%23cS I M U L A T I O N   R E S U L T S\n",' ');
     for(i=0;i<85;i++)
     fprintf(output_fp,"%c",'-');
   fprintf(output_fp,"\n\n");
   fprintf(output_fp,"%12c-----I------ M A C H I N E S ------I---- B U F F E R S ----\n",' ');
   fprintf(output_fp,"%17cI                              I- - - -L E V E L- - - -\n",' ');
   fprintf(output_fp,"%14cN. I TOT. PROD.  |  N. OF FAILS  I  FINAL  |  AVERAGE\n",' ');
   fprintf(output_fp,"%12c-----I-------------|---------------I---------|-------------\n",' ');
   for(n=1;n<=nm-1;n++)
   {
     bufelap(TEP,n);
```

```
  macelap(TEP,n);
  buf.BMLEV[n] = buf.BMLEV[n]/TEP;
  fprintf(output_fp,"%13c%2d  I%10ld   |     %5ld     I  %3ld    | %8.3lf\n",
             ' ',n,mac.PROD[n],mac.NRNG[n],buf.B[n],buf.BMLEV[n]);
}
macelap(TEP,nm);
fprintf(output_fp,"%13c%2d  I%10ld   |     %5ld     I\n\n",
  ' ',nm,mac.PROD[nm],mac.NRNG[nm]);
for(i=0;i<43;i++)
  fprintf(output_fp,"%c%c",'-',' ');
fprintf(output_fp,"\n   S I M U L A T I O N   P E R I O D: %ld\n",TEP);
fprintf(output_fp,"MEAN THROUGHPUT RATE OF LAST MACHINE: %.6lf\n\n",
  ((double)mac.PROD[nm])/TEP);
fprintf(output_fp," STARTING TIME:%s\n",tmbuf_start);
fprintf(output_fp,"    FINAL TIME:%s\n",tmbuf_end);
if(ltime_end>ltime_start)
{
  if(msec2<msec1)
  {
    if(ltime_end-ltime_start==1)
      fprintf(output_fp,"      CPU TIME:%u msec\n\n",
      msec2 + 1000 - msec1);
    else
      fprintf(output_fp,"      CPU TIME:%ld sec plus %u msec\n\n",
      (ltime_end - ltime_start-1),msec2 + 1000 - msec1);
  }
  else if(msec2>=msec1)
    fprintf(output_fp,"      CPU TIME:%ld sec plus %u msec\n\n",
    ltime_end - ltime_start,msec2 - msec1);
}
else
  fprintf(output_fp,"      CPU TIME:%u msec\n\n",msec2 - msec1);

fclose(output_fp);
free(tim.TE);
free(tim.TEM);
free(tim.TEB);
free(tim.TMO);
free(tim.TBO);
free(tim.TM);
free(tim.TB);
free(buf.LSTATE);
free(buf.B);
free(buf.BC);
free(buf.BMLEV);
free(sel.MOB);
```

```
   free(sel.NEB);
   free(mac.PNE);
   free(mac.PROD);
   free(mac.NRNG);
   free(mac.ICRN);
   free(mac.RP);
   free(mac.FP);
}

void initial(long *tep)
{
   int i;
   long zs,zinew;
   float u;
   *tep=0;

   buf.LSTATE[0] = 1;
   tim.TEM[0] = STOPTIM;
   sel.NEB[0] = 0;
   buf.BC[0] = 5*titems+1;
   buf.B[0] = 5*titems;
   tim.TEB[0] = STOPTIM;
   tim.TM[0] = STOPTIM;
   tim.TB[1][0] = STOPTIM;
   tim.TB[2][0] = 0;
   buf.BC[nm] = titems;
   buf.B[nm] = 0;
   tim.TB[2][nm] = STOPTIM;
   tim.TM[nm+1] = STOPTIM;
   for(i=0;i<=nm-1;i++)
   {
     if(buf.B[i]>0)
     {
        buf.LSTATE[i] = 1;
        buf.B[i]--;
     }
     else
        buf.LSTATE[i] = 0;
   }
   buf.LSTATE[nm] = 1;
   for(i=0;i<=nm;i++)
   {
     if(buf.LSTATE[i]==0)
     {
        tim.TB[1][i] = tim.TM[i];
        tim.TB[2][i] = tim.TM[i];
```

```
        tim.TEB[i] = STOPTIM;
        tim.TM[i+1] = tim.TB[2][i] + 10;
      }
      else
      {
        tim.TB[1][i] = tim.TM[i];
        tim.TM[i+1] = 10;
        tim.TB[2][i+1] = tim.TM[i+1];
        if(i==0)
          tim.TEB[i] = STOPTIM;
        else
          bufeven(*tep,i);
      }
    }
    for(i=1;i<=nm;i++)
    {
      if(mac.FP[i]>0)
      {
        zs = (long) mac.ICRN[i][1];
        zinew = zinext(zs);
        u = (float) (2*(zinew/256)+1)/16777216;
        mac.PNE[i] = (long) (log(u)/log(1-mac.FP[i])) + 1;
        mac.ICRN[i][1] = zinew;
      }
      else
        mac.PNE[i] = 214748364;

      tim.TEM[i] = (*tep) + (tim.TM[i]) + 10*(mac.PNE[i] - 1);
    }
    for(i=0;i<=nm;i++)
    {
      if(tim.TEM[i] <= tim.TEB[i])
      {
        tim.TE[i]=tim.TEM[i];
        sel.MOB[i]=1;
      }
      else
      {
        tim.TE[i]=tim.TEB[i];
        sel.MOB[i]=2;
      }
    }
}

void nextevt(long *tep)
{
```

```
   int j;

   *tep = SIMTIM;
   for(j=1;j<=nm;j++)
   {
     if(tim.TE[j] > *tep)
        continue;
     else if(tim.TE[j] < *tep)
        sel.MAORBUF = sel.MOB[j];
     else if(sel.MOB[j]==2 && sel.NEB[j]==0)
        sel.MAORBUF = 2;
     else
        continue;
     sel.NEL = j;
     *tep = tim.TE[j];
   }
   if(sel.NEL==nm && sel.MAORBUF==2 && sel.NEB[sel.NEL]==2)
     iendsim=1;
}

void fails(long tep,int n)
{
   int NFAIL=0;
   int IPTF;
   float u;
   long zf,zinew;

   macelap(tep,n);
   felap(tep,n-1);
   bufelap(tep,n);
   XREP=0;

LABEL_GENERATOR:
   zf = (long) mac.ICRN[n][1];
   zinew = zinext(zf);
   u =  (float) (2*(zinew/256)+1)/16777216;
   IPTF = (int) (log(u)/log(1-(mac.FP[n])));
   mac.ICRN[n][1] = zinew;
   zf = (long) mac.ICRN[n][2];
   zinew = zinext(zf);
   u = (float) (2*(zinew/256)+1)/16777216;
   XREP += (long) (log(u)/log(1-(mac.RP[n])));
   mac.ICRN[n][2] = zinew;
   NFAIL++;
   if(IPTF==0)
     goto LABEL_GENERATOR;
```

```
   mac.NRNG[n] += NFAIL;
   mac.PNE[n]=IPTF;
   if(buf.LSTATE[n]==2)
   {
     if(XREP+10 > tim.TB[2][n])
     {
        tim.TM[n] = XREP + 10;
        buf.LSTATE[n] = 1;
     }
     else
        tim.TM[n] = tim.TB[2][n];
   }
   else
     tim.TM[n] = XREP + 10;
   tim.TEM[n] = tep + tim.TM[n] + 10*(mac.PNE[n]-1);
   tim.TB[2][n-1] = tim.TM[n];
   tim.TB[1][n] = tim.TM[n];
   feven(tep,n-1);
   bufeven(tep,n);
     if(tim.TEM[n] <= tim.TEB[n])
     {
        tim.TE[n] = tim.TEM[n];
        sel.MOB[n] = 1;
     }
     else
     {
        tim.TE[n] = tim.TEB[n];
        sel.MOB[n] = 2;
     }
}

void fills(long tep,int n)
{
   bufelap(tep,n);
   macelap(tep,n);
   felap(tep,n-1);
   buf.LSTATE[n]=2;
   buf.B[n] = buf.BC[n];
   feven(tep,n);
}

void empties(long tep,int n)
{
   int i;
```

```
   bufelap(tep,n);
   macelap(tep,n+1);
   bufelap(tep,n+1);
   buf.LSTATE[n]=0;
   buf.B[n]=0;
   tim.TEB[n] = STOPTIM;
   tim.TB[2][n] = tim.TB[1][n];
   tim.TM[n+1] = tim.TB[2][n] + 10;
   tim.TB[1][n+1] = tim.TM[n+1];
   for(i=n;i<=n+1;i++)
   {
     bufeven(tep,i);
     if(i==n+1)
        maceven(tep,n+1);

     if(tim.TEM[i] <= tim.TEB[i])
     {
        tim.TE[i] = tim.TEM[i];
        sel.MOB[i] = 1;
     }
     else
     {
        tim.TE[i] = tim.TEB[i];
        sel.MOB[i] = 2;
     }
   }
}

void nempty(long tep,int n)
{
   int i;

   bufelap(tep,n);
   macelap(tep,n+1);
   bufelap(tep,n+1);
   buf.LSTATE[n]=1;
   tim.TM[n+1] = __max(tim.TM[n+1],tim.TB[1][n+1]);
   tim.TB[2][n] = tim.TM[n+1];
   bufeven(tep,n);
   maceven(tep,n+1);
   for(i=n;i<=n+1;i++)
   {
     if(tim.TEM[i] <= tim.TEB[i])
     {
        tim.TE[i] = tim.TEM[i];
        sel.MOB[i] = 1;
```

```
     }
     else
     {
        tim.TE[i] = tim.TEB[i];
        sel.MOB[i] = 2;
     }
   }
}

void felap(long tep,int n)
{
   int j=n;

   while(buf.LSTATE[j]==2){
     macelap(tep,j);
     j--;
     bufelap(tep,j);
   }
   bufelap(tep,n);
}

void feven(long tep,int n)
{
   int j=n;

   while(buf.LSTATE[j]==2){
     if(j==0)
        return;
     tim.TB[1][j] = __max(tim.TB[1][j],tim.TB[2][j]);

     if(buf.LSTATE[j-1]!=0 || tim.TB[2][j-1]==10)
     {
        tim.TM[j] = tim.TB[1][j];
        tim.TB[2][j-1] = tim.TB[1][j];
     }
     bufeven(tep,j);
     maceven(tep,j);
     if(tim.TEM[j] <= tim.TEB[j])
     {
        tim.TE[j] = tim.TEM[j];
        sel.MOB[j] = 1;
     }
     else
     {
        tim.TE[j] = tim.TEB[j];
        sel.MOB[j] = 2;
```

```
    }
    j--;
  }
  if(buf.LSTATE[j]!=0 || buf.LSTATE[j+1]!=2)
    bufeven(tep,j);
  else
  {
    tim.TEB[j] = tep + tim.TB[1][j];
    sel.NEB[j] = 1;
  }

  if(tim.TEM[j] <= tim.TEB[j])
  {
    tim.TE[j] = tim.TEM[j];
    sel.MOB[j] = 1;
  }
  else
  {
    tim.TE[j] = tim.TEB[j];
    sel.MOB[j] = 2;
  }
}

void macelap(long tep,int n)
{
  long DUR,DUR1;
  long parts;

  DUR = tep - (tim.TMO[n]);
  if(DUR <= 0)
    goto LABEL_BREAK;

  DUR1 = DUR - tim.TM[n];
  if(DUR1 >= 0)
  {
    parts = DUR1/10 + 1;
    mac.PNE[n] -= parts;
    mac.PROD[n] += parts;
    tim.TM[n] = 10*parts - DUR1;
  }
  else
    tim.TM[n] = -DUR1;

LABEL_BREAK:
  tim.TMO[n] = tep;
}
```

```
void maceven(long tep,int n)
{
   if(buf.LSTATE[n]==2 && mac.PNE[n]==0 && buf.LSTATE[n-1]!=0)
     tim.TEM[n] = tep;
   else
     tim.TEM[n] = tep + tim.TM[n] + 10*(mac.PNE[n]-1);
}

void bufelap(long tep,int n)
{
   if(n==0)
     return;
   int count;
   long B1=0;
   long TMIN;
   long DUR;
   double AREA=0;

   DUR = tep - (tim.TBO[n]);
   if(DUR<=0)
     goto LABEL_BREAK;

   B1 = buf.B[n];
   TMIN = __min(tim.TB[1][n],tim.TB[2][n]);
   if(((tim.TB[1][n]) == (tim.TB[2][n])) || DUR < TMIN)
   {
     buf.BMLEV[n] += B1*DUR;

     if(DUR-tim.TB[1][n] >= 0)
        tim.TB[1][n] =
        ((DUR - tim.TB[1][n])/10 + 1)*10 -
        (DUR - tim.TB[1][n]);
     else
        tim.TB[1][n] = -(DUR - tim.TB[1][n]);

     if(DUR-tim.TB[2][n] >= 0)
        tim.TB[2][n] =
        ((DUR - tim.TB[2][n])/10 + 1)*10 -
        (DUR - tim.TB[2][n]);
     else
        tim.TB[2][n] = -(DUR - tim.TB[2][n]);
   }
   else
   {
     count=1;
```

```
     buf.B[n] += emvadon(n,count,DUR,&AREA);
     buf.BMLEV[n] += (B1*DUR) + AREA;
     count=2;
     buf.B[n] -= emvadon(n,count,DUR,&AREA);
     buf.BMLEV[n] -= AREA;
   }

LABEL_BREAK:
   tim.TBO[n] = tep;
   if((buf.LSTATE[n] == 0) && ((buf.B[n]>0) ||
     (tim.TB[1][n]<tim.TB[2][n])))
     buf.LSTATE[n] = 1;
   else if((buf.LSTATE[n] == 2) &&
     (buf.B[n]<buf.BC[n] || tim.TB[1][n]>tim.TB[2][n]))
     buf.LSTATE[n] = 1;
}



void bufeven(long tep,int n)
{
   long DT;

   if(tim.TB[1][n] == tim.TB[2][n])
     tim.TEB[n] = STOPTIM;
   else
   {
     DT = tim.TB[1][n] - tim.TB[2][n];

     if(DT > buf.B[n]*10)
     {
       tim.TEB[n] = tep + tim.TB[2][n] + buf.B[n]*10;
       sel.NEB[n]=0;
     }
     else if(-DT > (buf.BC[n] - buf.B[n])*10)
     {
       if(buf.BC[n] == buf.B[n])
       {
         tim.TEB[n] = tep;
         sel.NEB[n]=2;
       }
       else
       {
         tim.TEB[n] = tep + tim.TB[1][n] +
           (buf.BC[n] - buf.B[n] - 1)*10;
         sel.NEB[n]=2;
       }
```

```
     }
     else
        tim.TEB[n] = STOPTIM;
   }
}

long emvadon(int n,int i,long dur,double *area)
{
   long ab1;
   long dur1,part;
   double area1;

   dur1 = dur - tim.TB[i][n];
   if(dur1 >= 0)
   {
     ab1 = dur1/10;
     area1 = 10 * ab1 * (ab1+1) / 2;
     part = (dur1 - 10*ab1) * (ab1+1);
     *area = area1+part;
     tim.TB[i][n] = (ab1 + 1)*10 - dur1;
     return (ab1+1);
   }
   else
   {
     *area = 0;
     tim.TB[i][n] = -dur1;
     return(0);
   }

}

long zinext(long zi)
{
   long XHI, XALO, LEFTLO, FHI, K;

   XHI=zi/65536;
   XALO=(zi-XHI*65536)*24112;
   LEFTLO=XALO/65536;
   FHI=XHI*24112+LEFTLO;
   K=FHI/32768;
   zi = (((XALO-LEFTLO*65536)-2147483647)+
      (FHI-K*32768)*65536)+K;
   if(zi<0)
     zi += 2147483647;

   XHI=zi/65536;
```

```
   XALO=(zi-XHI*65536)*26143;
   LEFTLO=XALO/65536;
   FHI=XHI*26143+LEFTLO;
   K=FHI/32768;
   zi = (((XALO-LEFTLO*65536)-2147483647)+
      (FHI-K*32768)*65536)+K;
   if(zi<0)
     zi += 2147483647;
   return zi;
}
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Β. Κουϊκόγλου, *Αποτελεσματικά Μοντέλα Ανάλυσης Δίκτυων Παραγωγής*, Διδακτορική Διατριβή, Πολυτεχνείο Κρήτης, 1989.

[2] Β. Κουϊκόγλου, *Προσομοίωση*, Σημειώσεις μαθήματος, Πολυτεχνείο Κρήτης, 2002.

[3] V. S. Kouikoglou and Y. A. Phillis, *Hybrid Simulation Models of Production Networks*, New York: Kluwer, 2001.

[4] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*, 2nd ed., New York: McGraw-Hill, 1991.