ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΕΚΠΟΝΗΣΗ ΕΡΓΑΣΙΑΣ

ΤΕΚΜΗΡΙΩΣΗ ΠΡΟΣΟΜΟΙΩΤΗ ΥΠΟΛΟΓΙΣΤΙΚΟΥ ΣΥΣΤΗΜΑΤΟΣ (ΕΚΔΟΣΗ ΣΕ ANSI ΚΑΙ UNIX C)

ΓΕΩΡΓΙΟΣ Β. ΞΥΛΩΜΕΝΟΣ
ΕΠΙΒΛΕΠΩΝ: Ι. Κ. ΚΑΒΟΥΡΑΣ
ΑΘΗΝΑ 1993

1. Εισαγωγή

Το πρόγραμμα προσομοίωσης ενός υπολογιστικού συστήματος, αναπτύχθηκε αρχικά από το Γιάννη Κάβουρα, στα πλαίσια της εκπόνησης της διδακτορικής του διατριβής το 1978, σε PL/1 για σύστημα IBM 370/168. Μία δεύτερη έκδοση αναπτύχθηκε από τον ίδιο, το 1980, σε C για σύστημα DEC VAX 11/780, κάτω από το λειτουργικό σύστημα UNIX Version 7, η οποία όμως δεν μπορεί να εκτελεστεί σε άλλα συστήματα. Ο γράφοντας, ανέλαβε να αναθεωρήσει την έκδοση αυτή σε C, έτσι ώστε να μπορεί να εκτελείται σε οποιοδήποτε σύγχρονο σύστημα UNIX, στα πλαίσια εκπόνησης εργασίας σχετικής με τα λειτουργικά συστήματα, για το τμήμα Εφαρμοσμένης Πληροφορικής του Οικονομικού Πανεπιστημίου Αθηνών, στο ακαδημαϊκό έτος 1992-93. Το βασικό προϊόν της εργασίας αυτής είναι μία έκδοση του προγράμματος σε ANSI C και μία σε UNIX C, με μικρές διαφορές μεταξύ τους. Η μεταφορά του προγράμματος είχε σαν αποτέλεσμα σημαντικές τροποποιήσεις στον κώδικά του με στόχο τη βελτιστοποίηση της απόδοσής του και τη βελτίωση της ποιότητάς του, από πλευράς δομής και ευκολίας κατανόησης. Παράλληλα, το πρόγραμμα εξελληνίστηκε και τεκμηριώθηκε εσωτερικά στα ελληνικά. Το παρόν είναι η εξωτερική τεκμηρίωση του κώδικα, μαζί με ορισμένες παρατηρήσεις πάνω στη δομή και τη λειτουργία του προγράμματος που μπορεί να βοηθήσουν τους χρήστες του στη χρήση και την τροποποίησή του. Στα επόμενα θα δούμε πρώτα τους στόχους και τη δομή του προγράμματος, στη συνέχεια την τεκμηρίωσή του και τέλος ορισμένα σχόλια πάνω στην υλοποίηση αυτή μαζί με ορισμένες προτάσεις για παραπέρα δουλειά, με τη μορφή βελτιώσεων ή επεκτάσεων του προγράμματος. Ελπίζω το σύνολο των προϊόντων κώδικα και τεκμηρίωσης να έχει γίνει πιο κατανοητό από την αρχική υλοποίηση και τεκμηρίωση του προγράμματος, αν και ο όγκος, η αρχική σχεδίαση και η πολυπλοκότητα του προγράμματος οπωσδήποτε βάζουν όρια στο τι είναι εφικτό από την άποψη αυτή. Θέλω να πιστεύω ότι σε συνδυασμό με κάποια άλλα κείμενα που αναφέρονται στο πρόγραμμα, και μαζί με το βιβλίο του αρχικού συγγραφέα του προγράμματος ([KAB94]) πάνω στις αρχές σχεδίασης και την υλοποίηση των λειτουργικών συστημάτων, μπορεί να βοηθήσει τους ενδιαφερόμενους να κατανοήσουν καλύτερα το πώς λειτουργεί ένα σύστημα και πώς επηρεάζεται η απόδοσή του από τις αποφάσεις που υιοθετούνται στο στάδιο της σχεδίασης.

Γιώργος Β. Ξυλωμένος
Ιούλιος 1993

2. Στόχοι και δομή του συστήματος

2.1. Στόχοι

Ο βασικός στόχος του προγράμματος είναι να βοηθήσει τους σχεδιαστές και τους σπουδαστές των λειτουργικών συστημάτων να εκτιμήσουν το αποτέλεσμα της εφαρμογής διαφορετικών πολιτικών ή στρατηγικών σε διάφορα μέρη ενός συστήματος. Για να επιτευχθεί αυτό, προσομοιώνουμε ένα σύστημα γενικού σκοπού με τέσσερις συσκευές (δίσκο, εκτυπωτή, συσκευή ανταλλαγής και αναγνώστη δελτίων) που χειρίζεται εργασίες δεσμίδων και αλληλεπίδρασης, μέσω spooling ή / και τερματικών. Το σύστημα δέχεται μία πληθώρα παραμέτρων οι οποίες μπορούν να αλλάζουν χωρίς να επαναμεταγλωττίζεται ο κώδικας, και δίνει μία ποικιλία αποτελεσμάτων και μετρήσεων ανάλογα με τις απαιτήσεις του χρήστη. Πολλά άλλα σημεία μπορούν να αλλάξουν μετά από επαναμεταγλώττιση (αυτό ισχύει για παραμέτρους που προβλέπεται ότι θα χρειάζεται να αλλάζουν σπάνια). Τέλος, μπορούν να αντικατασταθούν ή να τροποποιηθούν ολόκληρα τμήματα κώδικα για να μεταβληθεί η πολιτική που εφαρμόζει το σύστημα κατά τη διαχείριση των απαιτήσεων των χρηστών του και κατά την κατανομή των πόρων του μεταξύ τους.

Το πρόγραμμα είναι παραμετρικό ως προς το υλικό στο οποίο "εκτελείται", ως προς τις βασικές παραμέτρους του λειτουργικού συστήματος που προσομοιώνει, ως προς τα χαρακτηριστικά των χρηστών του και ως προς τα χαρακτηριστικά της ίδιας της προσομοίωσης. Μεταβάλλοντας κάθε φορά ένα από αυτά τα χαρακτηριστικά (εκτός του τελευταίου), μπορούμε να εκτιμήσουμε τις μεταβολές που αυτό επιφέρει, απομονώνοντάς τις από τις υπόλοιπες. Έτσι, μπορούμε, για παράδειγμα, να εκτιμήσουμε τα αποτελέσματα από την τοποθέτηση ενός ταχύτερου δίσκου, από την υιοθέτηση βραχύτερου μεσοδιαστήματος χρονοπρογραμματισμού ή από την αύξηση της κίνησης στα τερματικά. Μεταβάλλοντας τα στοιχεία της ίδιας της προσομοίωσης μπορούμε να πάρουμε αναλυτικότερα στατιστικά στοιχεία ανά διάφορα χρονικά μεσοδιαστήματα, για διαφορετικούς συνολικούς χρόνους προσομοίωσης (για δείγματα των αποτελεσμάτων που παράγει το πρόγραμμα δείτε [CAV78] και [MOH84]).

Η αξία του προγράμματος έγκειται στη δυνατότητα που δίνει για πειραματισμό με διάφορες πολιτικές διαχείρισης των πόρων του συστήματος, χωρίς να απαιτεί την κατασκευή ενός συστήματος από την αρχή και χωρίς να χρειάζεται να απασχολούμε πραγματικούς χρήστες ή πραγματικό υλικό. Ο φόρτος εργασίας ενός πραγματικού συστήματος και η συμπεριφορά του υλικού του υπολογιστή προσομοιώνονται από το πρόγραμμα, αφήνοντας το

σχεδιαστή να πειραματιστεί εκ του ασφαλούς. Βασική επίτευξη των σχεδιαστών του συστήματος είναι να απομονώνεται η διαχείριση της προσομοίωσης από τον κώδικα πολιτικής των διαχειριστών του συστήματος. Έτσι παρέχεται μία μηχανή προσομοίωσης η οποία υποστηρίζει πολύ γενικές πρωτογενείς κλήσεις και διακοπές, πάνω στην οποία το σύστημα χτίζεται σε ανεξάρτητα τμήματα κώδικα που είναι πολύ παρόμοια με αυτά που υπάρχουν σε ένα πραγματικό λειτουργικό σύστημα.

Η προσομοίωση του συνολικού λειτουργικού περιβάλλοντος με λογικό κόστος (στο σύστημα VAX του Οικονομικού Πανεπιστημίου Αθηνών, πετυχαίνουμε λόγο χρόνου προσομοίωσης προς πραγματικό χρόνο πάνω από 3.5 σε συνθήκες χαμηλής φόρτωσης, ενώ σε μία μηχανή MS-DOS με επεξεργαστή 386 στα 40 MHz πετυχαίνουμε λόγο πάνω από 1.5, με ενσωματωμένο κώδικα για εκσφαλμάτωση και χωρίς κάποιον ειδικά βελτιστοποιημένο μεταγλωττιστή) χωρίς να χρειαζόμαστε χρήστες ή υλικό, σημαίνει ότι είναι εύκολο στους χρήστες να πειραματιστούν αλλάζοντας τις παραμέτρους ή τον κώδικα του συστήματος, έτσι ώστε να εκτιμήσουν αμεσότερα τα πλεονεκτήματα και τα μειονεκτήματα των διάφορων πολιτικών διαχείρισης των πόρων του συστήματος. Μία τέτοια μελέτη είναι πολύ χρήσιμη σε όσους θέλουν να κατανοήσουν την εσωτερική δομή ενός λειτουργικού συστήματος και την επίδραση που έχουν οι διάφοροι αλγόριθμοι καταμερισμού πόρων στην απόδοσή του.

Παράλληλα με την προσομοίωση του συστήματος για διευκόλυνση στον πειραματισμό, η ελεύθερη διανομή του κώδικα του συστήματος σε πηγαία μορφή, μπορεί να βοηθήσει τους ενδιαφερόμενους να δουν πώς υλοποιούνται τα ανώτερα επίπεδα ενός λειτουργικού συστήματος (διαχειριστές πόρων). Η ευκολία αυτή (που παρεχόταν παλιότερα για το σύστημα UNIX, αν και έχει πια καταργηθεί) μπορεί να βοηθήσει πολύ τους σχεδιαστές σαν μία αφετηρία για τη συγγραφή κώδικα (φυσικά, υπάρχουν και πληρέστερες, αλλά και δυσκολότερες στην κατανόηση, προσεγγίσεις, όπως αυτή που δίνεται στο [TAN87], η οποία καλύπτει όλο το σύστημα, από τις διακοπές μέχρι τις διεργασίες).

Ο τελευταίος στόχος, που επιτυγχάνεται με τη νέα έκδοση αυτή σε C, είναι η ευκολία μεταφοράς από μηχανή σε μηχανή, μέσω της τυποποίησης της γλώσσας που χρησιμοποιείται (πρότυπο ANSI). Με διαφορετικό κόστος, ο προσομοιωτής μπορεί να εκτελεστεί πρακτικά σε οποιαδήποτε σύγχρονη μηχανή που διαθέτει ένα σύγχρονο μεταγλωττιστή της C, με ελάχιστες αλλαγές. Για παράδειγμα, οι μεταβολές από ANSI C σε C για το ULTRIX-32 χρειάστηκαν την πρώτη φορά περίπου 2 ώρες, εξαιρουμένου του χρόνου που χρειάστηκε για να βρεθεί ένα αρκετά λεπτό σφάλμα ασυμβατότητας, αν και τώρα πια υπάρχει μόνο μία έκδοση και για τους δύο τύπους μεταγλωττιστών.

Ταυτόχρονα, τα αποτελέσματα (θα έπρεπε να) είναι πανομοιότυπα από μηχανή σε μηχανή, τουλάχιστον όπου έχουμε παρόμοια μεγέθη λέξεων. Δυστυχώς, τα διαφορετικά μεγέθη λέξεων, οι διαφορετικές υλοποιήσεις των βασικών τύπων της γλώσσας και οι διαφορετικές μέθοδοι που ακολουθούνται για την εκτέλεση των αριθμητικών πράξεων πάνω στους πραγματικούς αριθμούς σε κάθε σύστημα, κάνουν τα αποτελέσματα να αποκλίνουν (αν και όχι σημαντικά, τουλάχιστον για μικρό χρόνο εκτέλεσης).

Μία τελευταία παρατήρηση είναι ότι σε ένα σύστημα όπως αυτό, το οποίο έχει σχεδιαστεί για βοηθάει στη σύγκριση διαφορετικών πολιτικών, θα πρέπει να έχουμε επαναλαμβανόμενη συμπεριφορά σε εκτελέσεις με τα ίδια δεδομένα. Αυτό το πετυχαίνουμε χρησιμοποιώντας σειρές ψευδοτυχαίων αριθμών με μία καθορισμένη διαδικασία η οποία έχει σαν σπόρο παραμέτρους που δίνει ο χρήστης του προγράμματος. Έτσι, παρά το ότι έχουμε εκτιμήσεις για τη συμπεριφορά των διεργασιών των χρηστών (τους οποίους προσομοιώνουμε) τα αποτελέσματα είναι πάντα τα ίδια σε εκτελέσεις με τις ίδιες παραμέτρους και συγκρίσιμα σε εκτελέσεις με τις ίδιες παραμέτρους εκτίμησης (στις οποίες περιλαμβάνονται μεταξύ άλλων και οι σπόροι των ψευδοτυχαίων ακολουθιών).

2.2. Δομή του προγράμματος

2.2.1. Δομή του υπό προσομοίωση συστήματος

Η δομή του προγράμματος επηρεάζεται αποφασιστικά από τη δομή του συστήματος που προσομοιώνουμε. Το σύστημα που προσομοιώνουμε έχει μία σειρά από διαχειριστές πόρων οι οποίοι κατανέμονται σε ιεραρχικά επίπεδα και επικοινωνούν μέσω μηνυμάτων. Το σύστημα, γενικά, παρουσιάζει μία ομοιομορφία στους μηχανισμούς επικοινωνίας και στις μεθόδους υλοποίησης των διαφόρων τμημάτων του. Στη βάση, έχουμε τον πυρήνα, που παρέχει ένα ελάχιστο σύνολο υπηρεσιών, επιτρέποντας στους διαχειριστές να εφαρμόζουν τις πολιτικές τους. Ο πυρήνας, εξυπηρετεί τις διακοπές, ελέγχει την είσοδο και έξοδο και παρέχει το βραχυχρόνιο χρονοπρογραμματισμό (με τη μορφή ενός διανομέα ελέγχου). Αμέσως μετά, έχουμε το διαχειριστή της μνήμης, ο οποίος παρέχει την εικονική μνήμη στις διεργασίες. Στο επόμενο επίπεδο, έχουμε τους διαχειριστές συσκευών, οι οποίοι παρέχουν εικονικές συσκευές στις διεργασίες, και το διαχειριστή της ΚΜΕ, που δίνει τη δυνατότητα δημιουργίας και διαγραφής περιγραφητών διεργασιών και χρονοπρογραμματίζει τις διεργασίες. Ακόμη παραπάνω, έχουμε το σύστημα αρχείων, το οποίο παρέχει εικονικά αρχεία, και το δημιουργό διεργασιών, που δημιουργεί και καταστρέφει τις διεργασίες στο σύνολό τους. Στο επόμενο επίπεδο έχουμε τους spoolers, και τέλος έχουμε το χρονοπρογραμματιστή εργασιών. Πάνω από

αυτούς τους διαχειριστές έχουμε τις διεργασίες των χρηστών (για την περιγραφή του συστήματος που προσομοιώνουμε δείτε [CAV78] και [MOH84]).

Σε γενικές γραμμές, κάθε επίπεδο παρέχει ένα πιο αφηρημένο σύνολο υπηρεσιών στα ανώτερά του, με αποτέλεσμα το σύστημα να τμηματοποιεί το πρόβλημα διαχείρισης και κατανομής των πόρων σε υποπροβλήματα που αντιμετωπίζονται και επιλύονται ξεχωριστά. Η υιοθέτηση της μεταβίβασης μηνυμάτων μεταξύ όλων των διεργασιών του συστήματος παρέχει απλότητα και ομοιομορφία στις επικοινωνίες. Στο επίπεδο των χρηστών, κάθε χρήστης δημιουργεί δένδρα (ομάδες) διεργασιών για να εκτελέσει τις εργασίες του, με τις διεργασίες του να έχουν την ίδια γενική μορφή με τις διεργασίες του συστήματος και να λειτουργούν παρόμοια. Η συγκέντρωση των ελάχιστων δυνατών αρμοδιοτήτων στον πυρήνα κάνει το σύστημα εύκολα τροποποιήσιμο για μεταβολή πολιτικών, αφού ο πυρήνας υλοποιεί μηχανισμούς μόνο (κατά το δυνατόν). Επειδή, τέλος, ορισμένα χαρακτηριστικά κρίθηκε σημαντικό να επιλέγονται από το χρήστη, αλλά η υλοποίησή τους μέσω παραμέτρων κρίθηκε προβληματική, δίνεται η επιλογή κατά τη σύνδεση του προγράμματος ο χρήστης να κατασκευάσει είτε ένα σύστημα σελιδοποίησης είτε ένα σύστημα τεμαχισμού, να χρησιμοποιεί αποκλειστικά καταμεριζόμενες ή καταμεριζόμενες και μη καταμεριζόμενες συσκευές και να μετράει το χρόνο εξυπηρέτησης των πρωτογενών κλήσεων είτε κατ' εκτίμηση είτε με πραγματικά δεδομένα από τη μηχανή εκτέλεσης.

2.2.2. Υλοποίηση του συστήματος

Το βασικό πρόβλημα κατά την προσομοίωση οποιουδήποτε συστήματος, είναι πώς θα μιμηθούμε τα γεγονότα που συμβαίνουν στο σύστημα αυτό. Το πρόβλημα περιπλέκεται από το ότι θα πρέπει ταυτόχρονα να προσομοιώνουμε και το σύστημα και τους χρήστες του. Για το σύστημα, μπορούμε να γράψουμε τον κώδικα των διεργασιών με τρόπο παρόμοιο με τον τρόπο που θα τον γράφαμε για ένα κανονικό λειτουργικό σύστημα. Το πρόβλημα είναι στους χρήστες, οι οποίοι έχουν γενικά απρόβλεπτες απαιτήσεις, εκτελώντας προγράμματα με άγνωστη μορφή και συμπεριφορά. Ταυτόχρονα, δεν έχουμε το υλικό ενός πραγματικού υπολογιστή στη διάθεσή μας να μας προμηθεύει διακοπές, χρονόμετρα κ.λπ. Ας δούμε τα προβλήματα και τις λύσεις τους με τη σειρά. Αναγκαστικά η παρουσίαση είναι σύντομη και δεν έχει σκοπό να εξηγήσει πλήρως ούτε τις γενικές αρχές της προσομοίωσης, ούτε τις ειδικές λύσεις για την περίπτωσή μας, αλλά πιο πολύ αποσκοπεί στο να σκιαγραφήσει τα ειδικά προβλήματα που αντιμετωπίζονται και τις αρχές για τη λύση τους (όλες οι λεπτομέρειες δίνονται στο [CAV78] και μία εισαγωγική παρουσίαση της γενικής ιδέας της μεθόδου προσομοίωσης που χρησιμοποιείται δίνεται στο [CAV81-1]).

Η αντιμετώπιση του προβλήματος της προσομοίωσης γίνεται με τη δημιουργία μίας κεντρικής λίστας γεγονότων που κρατάει όλα τα γεγονότα του συστήματος διατεταγμένα με βάση το χρόνο. Κάθε φορά που εκτελείται το κεντρικό πρόγραμμα, ανάλογα με το επόμενο γεγονός εκτελούμε κάποιες ενέργειες, οι οποίες μπορεί να δρομολογήσουνε άλλα γεγονότα, κ.ο.κ. Να σημειώσουμε εδώ ότι όταν μιλάμε για δρομολόγηση εννοούμε ότι βάζουμε μία καταχώρηση που περιγράφει το γεγονός στην κατάλληλη λίστα γεγονότων στη σωστή θέση (σύμφωνα με το χρόνο του γεγονότος). Το θέμα τώρα είναι ποιες ενέργειες πρέπει να γίνονται για τα γεγονότα. Μία λύση είναι να γράψουμε διαχειριστές οι οποίοι να εκτελούν κώδικα όπως αυτόν που περιέχεται σε ένα πραγματικό σύστημα. Αυτό βοηθάει στον πειραματισμό με διάφορες πολιτικές και είναι η βασική ιδέα πίσω από το πρόγραμμα. Βέβαια, αυτό δεν είναι απόλυτα εφαρμόσιμο, αφού δεν έχουμε στη διάθεσή μας το πραγματικό υλικό και τους πραγματικούς χρήστες. Έτσι, ορισμένα τμήματα του συστήματος έχουν σχέση μόνο με την προσομοίωση, ορισμένα μόνο με το πραγματικό σύστημα, και ορισμένα χρησιμεύουν στη διεπαφή μεταξύ των δύο. Το πρόβλημά μας τώρα είναι πώς να συνταιριάξουμε την προσομοίωση των γεγονότων με το διακοπτόμενο χρονοπρογραμματισμό που απαιτεί ένα λειτουργικό σύστημα (εδώ, το διακοπτόμενος δεν αναφέρεται στον τρόπο με τον οποίο καταχωρείται η ΚΜΕ στις διεργασίες, αλλά στο ότι ένα σύστημα πολλών χρηστών πρέπει πάντα να μπορεί να διακόπτει την εκτέλεση των διεργασιών των χρηστών, είτε λόγω διακοπών είτε λόγω παγίδων).

Ας υποθέσουμε ότι το πρόβλημά μας είναι μόνο η έλλειψη του υλικού. Το πρόγραμμά μας θα πρέπει να διακόπτεται μόνο του όποτε πρέπει, μεταφέροντας τον έλεγχο από σημείο σε σημείο. Για να το πετύχουμε αυτό, τεμαχίζουμε τον κώδικα των διαχειριστών σε αυτόνομα τμήματα, μεταξύ των οποίων έχουμε κλήσεις που φτάνουν στον πυρήνα. Κατά την επανενεργοποίηση των διαχειριστών, θα πρέπει να συνεχίζουμε από το κατάλληλο σημείο. Αυτό μπορεί να υλοποιηθεί με το να δώσουμε ετικέτες στα σημεία αυτά, και κάθε φορά που γίνεται μία κλήση προς τον πυρήνα, να αποθηκεύουμε σε πίνακες του συστήματος το επόμενο σημείο ενεργοποίησης (ετικέτα). Κατά την ενεργοποίηση μίας διεργασίας, αρκεί με μία εντολή *switch* να επιστρέψουμε στο κατάλληλο σημείο. Τα σημεία στα οποία επανέρχεται ο έλεγχος μετά από κάθε νέα κλήση / ενεργοποίηση μίας διεργασίας λέγονται *είσοδοι* και έχουν έναν αριθμό το καθένα, μοναδικό μέσα στη διεργασία. Ουσιαστικά τα σημεία εισόδου είναι ετικέτες *case* (case 0, case 1 κ.ο.κ), τις οποίες έχουμε μετονομάσει με μακροαντικαταστάσεις (εντολή *#define* του προεπεξεργαστή της C) σε ENTRY(0), ENTRY(1) κ.ο.κ. Στα παρακάτω λοιπόν, όταν λέμε "είσοδος" εννοούμε την ετικέτα στην οποία μεταφέρεται ο έλεγχος με ένα switch, όταν ξαναξεκινάει η διεργασία. Έτσι, προσομοιώνουμε το διακοπτόμενο χρονοπρογραμματισμό για τις διεργασίες του συστήματος. Η προσομοίωση τώρα των χαρακτηριστικών του υλικού είναι εύκολη, μέσα από

άμεσους υπολογισμούς και τοποθέτηση των σχετικών γεγονότων στην κατάλληλη θέση στις λίστες της προσομοίωσης. Το βασικό είναι όλες οι κλήσεις που μπορεί να εμποδίσουν μία διεργασία να κατευθύνονται τελικά στον πυρήνα. Μετά από την εξυπηρέτησή τους, μπορούμε να συνεχίζουμε την εκτέλεση της διεργασίας ή να εκλέγουμε μία άλλη μέσω του διανομέα. Αν διακοπεί μία διεργασία του συστήματος, μπορούμε να βάλουμε το επόμενο γεγονός της σε μία λίστα η οποία κρατάει όλα τα γεγονότα των υπό διακοπή εργασιών. Κατά την επανεκκίνηση της διεργασίας, τα γεγονότα θα επαναδρομολογούνται, δηλαδή θα μεταφέρονται από τη λίστα των γεγονότων των διεργασιών που διακόπηκαν στη λίστα των γεγονότων της προσομοίωσης. Έτσι, έχουμε ανάμειξη της μεθόδου των γεγονότων με τη μέθοδο της αλληλεπίδρασης διεργασιών. Παρατηρήστε ότι για να μπορούμε μετά από κάθε τέτοια κλήση να επιστρέφουμε στον πυρήνα, από τον οποίο ξεκίνησε η εκτέλεση της αρχικής διεργασίας, απαιτείται μία εντολή μη τοπικού *goto*, πράγμα που επιβάλλει ειδικές απαιτήσεις στην κωδικοποίηση (η υλοποίηση των διεργασιών περιγράφεται αναλυτικότερα στο [CAV83-1]).

Το τελευταίο μας πρόβλημα είναι η προσομοίωση των χρηστών. Φυσικά δεν μπορούμε να γράψουμε κώδικα για όλες τις διεργασίες των χρηστών, μπορούμε όμως να κάνουμε εκτιμήσεις για την συμπεριφορά τους. Η βασική ιδέα είναι να έχουμε ένα τμήμα κώδικα το οποίο να εκτιμά τις απαιτήσεις που έχουν οι εργασίες και διεργασίες των χρηστών από το σύστημα. Επειδή, βέβαια, ο κάθε χρήστης έχει τις δικές του απαιτήσεις, θα κρατάμε για τον καθένα μία λίστα με τα γεγονότα που θα αφορούν τη δική του διεργασία. Όταν εκτελείται η διεργασία αυτή, τότε τα γεγονότα της θα δρομολογούνται στην κύρια λίστα της προσομοίωσης. Αν η διεργασία σταματήσει πριν την χρονική στιγμή δρομολόγησης των γεγονότων, τα γεγονότα θα μεταφέρονται στην ιδιωτική της λίστα πάλι, αφού οι χρόνοι των γεγονότων ισχύουν μόνο σε σχέση με τον χρόνο που εκτελείται η διεργασία. Η πρόβλεψη της συμπεριφοράς των χρηστών γίνεται κατά την άφιξή τους στο σύστημα (η οποία φυσικά γίνεται μέσω στατιστικών εκτιμήσεων), όπου εκτιμώνται οι απαιτήσεις τους σε διεργασίες, μνήμη, ε/ε, χρόνο εκτέλεσης κ.ο.κ., και τοποθετούνται σε *πίνακες σκιάς*, οι οποίοι χρησιμοποιούνται κατά τη δρομολόγηση των γεγονότων των διεργασιών στις εσωτερικές τους λίστες για να υπολογίζονται τα στατιστικά χαρακτηριστικά της κάθε διεργασίας. Ουσιαστικά, στον κάθε πίνακα σκιάς, έχουμε τα στοιχεία που αφορούν την προσομοίωση της κάθε διεργασίας και δεν βρίσκονται από τους πίνακες του πραγματικού συστήματος. Τελικά, ο κώδικας των διεργασιών των χρηστών θα προβλέπει και θα αντιμετωπίζει γεγονότα εσωτερικά στις διεργασίες των χρηστών σύμφωνα με τις εκτιμώμενες απαιτήσεις τους, συγχρονίζοντάς τα με την πραγματική εκτέλεση των διεργασιών στο σύστημα και τον πραγματικό χρόνο του συστήματος. Προσέξτε ότι σε κάθε μεταφορά γεγονότων από λίστα σε λίστα απαιτείται μετατροπή του χρόνου από χρόνο συστήματος σε χρόνο διεργασίας ή

αντίστροφα (η μέθοδος προσομοίωσης των απαιτήσεων των χρηστών περιγράφεται αναλυτικά και στο [CAV81-2]).

Βλέπουμε λοιπόν ότι έχουμε δύο επίπεδα προσομοίωσης. Το ένα είναι η προσομοίωση των διεργασιών των χρηστών η οποία γίνεται από ένα τμήμα κοινού κώδικα και με τη χρήση πολλών λιστών γεγονότων, μίας για κάθε διεργασία χρήστη. Το δεύτερο είναι η κύρια προσομοίωση του συστήματος η οποία αντλεί γεγονότα από τις διεργασίες του συστήματος και από την εκτελούμενη διεργασία του χρήστη σε κάθε χρονική στιγμή. Τελικά, το σύστημα προσομοιώνει τον πραγματικό φόρτο εργασίας αναμιγνύοντάς τον με τον κώδικα του λειτουργικού συστήματος, ο οποίος αλλού είναι ίδιος με τον πραγματικό και αλλού αποκλίνει με σκοπό να εξυπηρετήσει τις ανάγκες της προσομοίωσης. Το τελικό αποτέλεσμα είναι η ανάμειξη των γεγονότων του συστήματος με τα γεγονότα της τρέχουσας διεργασίας χρήστη, με στόχο να έχουμε τα ίδια γεγονότα με αυτά του πραγματικού υπολογιστικού συστήματος (η μέθοδος αυτή για την συνένωση τμημάτων προσομοίωσης και πραγματικού συστήματος περιγράφεται στο [CAV83-2]).

Για να μπορέσουμε να προσομοιώσουμε το σύστημα χρειάζεται να έχουμε δεδομένα για το πραγματικό σύστημα (κατάσταση διεργασιών και μνήμης), αλλά και για την προσομοίωση (ποιες είναι οι απαιτήσεις που θα έχουν οι διεργασίες). Πολλά δεδομένα της προσομοίωσης βέβαια μπαίνουν σε δομές του συστήματος (για παράδειγμα, τα σημεία επανεκκίνησης). Επίσης χρειάζεται να έχουμε κώδικα που να υλοποιεί πολιτικές διαχείρισης πόρων (μνήμη, ΚΜΕ) και κώδικα που να υλοποιεί μηχανισμούς προσομοίωσης (εκτίμηση απαιτήσεων, δρομολόγηση γεγονότων). Μεταξύ των δύο, υπάρχουν διεργασίες που εκτελούν εργασίες που βρίσκονται και σε ένα πραγματικό σύστημα, αλλά επηρεάζονται από την προσομοίωση. Έτσι, η παρουσίαση του προγράμματος γίνεται σε τρία επίπεδα. Στο επίπεδο προσομοίωσης, έχουμε τις λειτουργίες που ασχολούνται καθαρά με την προσομοίωση, όπως ο κώδικας για τις διεργασίες των χρηστών. Στο επίπεδο διεπαφής έχουμε το χρονοπρογραμματιστή εργασιών ο οποίος εκτιμάει τις αρχικές απαιτήσεις των διεργασιών, αλλά και τις δημιουργεί (στην πραγματικότητα δημιουργεί τον οργανωτή της εργασίας που δημιουργεί με τη σειρά του τις άλλες διεργασίες) και τον πυρήνα, ο οποίος εξυπηρετεί διακοπές δρομολογώντας ταυτόχρονα και άλλα γεγονότα. Τέλος, στο υψηλότερο επίπεδο έχουμε κώδικα πολιτικής, για τη διαχείριση της ΚΜΕ, της μνήμης, των συσκευών ε/ε, των αρχείων και των τερματικών. Σε όλα τα επίπεδα υπάρχουν βέβαια στοιχεία από την προσομοίωση, απλά όσο ανεβαίνουμε γίνονται λιγότερα και απομονώνονται ευκολότερα. Παρατηρήστε ότι μεγαλύτερο ενδιαφέρον, από πλευράς πειραματισμού, έχουν τα τμήματα που υλοποιούν πολιτικές και είναι πιο μακριά από την προσομοίωση.

Το μόνο πράγμα που περιορίζει τον πειραματισμό με ένα τέτοιο εργαλείο είναι το ποσοστό μηχανισμών που έχει ενσωματωμένους. Μία ποικιλία μηχανισμών μπορεί να δίνει ευελιξία στα πραγματικά συστήματα, είναι όμως περιοριστικός παράγοντας σε ένα σύστημα που προσομοιώνει άλλα συστήματα. Έτσι, έχουμε δώσει τις ελάχιστες δυνατές κλήσεις, διακοπές και παγίδες στο σύστημα, με στόχο να μεταφέρεται μεγάλο μέρος της υλοποίησης στους διαχειριστές. Βέβαια, ορισμένες στρατηγικές αποφάσεις έχουν ληφθεί, όπως η μορφή του μηχανισμού επικοινωνίας, η μορφή των πινάκων του συστήματος, η μορφή των ομάδων των διεργασιών, και άλλα πολλά που είναι απαραίτητα για να μπορεί να λειτουργεί το σύστημα σε λογικά χρονικά πλαίσια. Σε γενικές γραμμές πάντως, το πρόγραμμα είναι αρκετά ευέλικτο και δεν επιβάλλει ιδιαίτερους περιορισμούς στο χρήστη του.

3. Τεκμηρίωση του κώδικα

Ο κώδικας κατανέμεται σε μία σειρά από αρχεία, τα οποία περιέχουν τα ακόλουθα:

defs.h	ορισμοί καθολικών δομών δεδομένων
vardefs.η	ορισμοί καθολικών μεταβλητών (δηλώσεις)

Τα δύο παραπάνω αρχεία διαβάζονται (με #include) από όλα τα αρχεία του συστήματος, εκτός από τα vars.c και utils.c που κάνουν #include μόνο το defs.h και times.h αντίστοιχα.

vars.c	οι πραγματικές μεταβλητές που δηλώνονται στο vardefs.h
utils.c	βοηθητικές συναρτήσεις για μεταφορά του προγράμματος
kernel.c	πυρήνας, διεργασίες χρηστών και αρχική φόρτωση
simr.c	βοηθητικές συναρτήσεις του προσομοιωτή
pir.c	πρωτογενείς κλήσεις (έκδοση με εκτίμηση χρόνων)
opir.c	πρωτογενείς κλήσεις (έκδοση με πραγματικό χρόνο)
psvc.c	χρονοπρογραμματιστής των εργασιών (έκδοση με σελιδοποίηση)
ssvc.c	χρονοπρογραμματιστής των εργασιών (έκδοση με τεμαχισμό)
procr.c	δημιουργός διεργασιών
fsys.c	σύστημα αρχείων
cpuman.c	διαχειριστής ΚΜΕ
cmman.c	διαχειριστής μνήμης
outsp.c	spooler εξόδου
insp.c	spooler εισόδου
terman.c	διαχειριστής τερματικών

devman.c	διαχειριστής συσκευών (έκδοση με κανονικές συσκευές)
sdevman.c	διαχειριστής συσκευών (έκδοση με καταμεριζόμενες συσκευές)
polfn.c	συναρτήσεις πολιτικής

Τα αρχεία περνάνε χωριστά από το μεταγλωττιστή και χρειάζονται όλα κατά τη σύνδεση, με επιλογή μεταξύ ενός από κάθε ένα από τα ζεύγη psvc / ssvc, pir / opir και devman / sdevman (δείτε τις κατάλληλες ενότητες για τα κριτήρια επιλογής). Σημειώστε ότι υπάρχει μία μόνο έκδοση των αρχείων, ανεξαρτήτως του μεταγλωττιστή που πρόκειται να χρησιμοποιηθεί.

Στις ακόλουθες ενότητες θα δούμε χωριστά τα τρία επίπεδα του προγράμματος, δηλαδή το επίπεδο προσομοίωσης, το επίπεδο διεπαφής συστήματος-προσομοίωσης και το επίπεδο συστήματος. Η τεκμηρίωση δεν είναι η καλύτερη δυνατή, αλλά δυστυχώς γράφτηκε μετά από τον κώδικα και σίγουρα δεν θα μπορούσε να είναι καλύτερη από αυτόν. Στόχος της είναι να υποβοηθήσει το έργο της εσωτερικής τεκμηρίωσης, δίνοντας μία πιο μακροσκοπική εικόνα του συνόλου. Ταυτόχρονα, εδώ δίνονται σαφέστερα οι λόγοι για τους οποίους γίνονται ορισμένα πράγματα, αλλά και παρέχεται μία πιο ολοκληρωμένη εικόνα των αλγορίθμων που χρησιμοποιούνται. Σε καμία περίπτωση πάντως η τεκμηρίωση αυτή δεν μπορεί να χρησιμοποιηθεί χωρίς τον κώδικα, αφού περισσότερο συμπληρώνει την εσωτερική τεκμηρίωση (άλλωστε γράφτηκε μεταγενέστερα) παρά την αντικαθιστά. Επίσης, για κατανόηση της συμπεριφοράς του συστήματος και του τρόπου με τον οποίο γίνεται η προσομοίωση (θέμα που είναι ιδιαίτερα δύσκολο, όπως φαίνεται και από την προηγούμενη ενότητα), χρειάζεται αναδρομή στα κείμενα που δώσαμε στην αρχή.

Για τα περιεχόμενα, έχει ακολουθηθεί η αρχική κατανομή σε ενότητες από το συγγραφέα του προγράμματος. Σε κάθε ενότητα, γίνεται μία εισαγωγική αναφορά, ακολουθεί μία περιγραφή των καθολικών και τοπικών δομών και μεταβλητών που χρησιμοποιούνται και τέλος δίνεται η περιγραφή του κώδικα, αν χρειάζεται, σε ενότητες. Για συμφωνία με τον κώδικα, ακολουθείται το όνομα είσοδος i για την ετικέτα ENTRY(i) του κώδικα, η οποία τελικά είναι απλά η ετικέτα *case i* μίας εντολής *switch*.

3.1. Τμήμα προσομοίωσης

Ο κώδικας που ανήκει στο επίπεδο αυτό του προγράμματος, περιλαμβάνει τον κώδικα αρχικοποίησης του συστήματος (παράμετροι λειτουργίας προσομοίωσης και συστήματος, αλλά και αρχικοποίηση ορισμένων πινάκων του πραγματικού συστήματος και της εξωτερικής και εσωτερικής προσομοίωσης), τον κώδικα επίκλησης διεργασιών, τον κώδικα

που προσομοιώνει τις διεργασίες των χρηστών (δηλαδή προβλέπει, δρομολογεί και αντιμετωπίζει τα γεγονότα που προσομοιώνουν τη συμπεριφορά των χρηστών) και τις βοηθητικές ρουτίνες για την προσομοίωση. Όλο αυτό το τμήμα είναι ο πυρήνας της προσομοίωσης και έχει στο μεγαλύτερο βαθμό του μόνο εννοιολογικές συγγένειες με τα τμήματα ενός πραγματικού συστήματος.

3.1.1. Εξωτερική προσομοίωση

3.1.1.1. Εισαγωγή

Η εξωτερική προσομοίωση ασχολείται με την αρχικοποίηση των παραμέτρων της και των πινάκων του συστήματος, με την παρακολούθηση και εκτύπωση των στατιστικών στοιχείων του συστήματος καθώς και με την υλοποίηση και παρακολούθηση των δομών που σχετίζονται με την προσομοίωση σε επίπεδο λειτουργικού συστήματος. Το μεγαλύτερο μέρος του κώδικα βρίσκεται στο αρχείο *kernel.c*, σαν το πρώτο μέρος της συνάρτησης *main* (αρχική φόρτωση, είσοδος σε διεργασίες και διεργασίες χρηστών). Οι βοηθητικές συναρτήσεις παρακολούθησης και παρουσίασης της κατάστασης του συστήματος βρίσκονται στο αρχείο *simr.c*.

3.1.1.2. Δομές Δεδομένων

Τα γεγονότα της εξωτερικής προσομοίωσης παριστάνονται με στοιχεία τύπου *event_list* (defs.h) με τα ακόλουθα πεδία:

int event: τύπος γεγονότος.
int eprocess: θέση της σχετικής διεργασίας στον pdt (βλέπε παρακάτω).
double event_time: πραγματικός χρόνος προσομοίωσης στον οποίο θα συμβεί το γεγονός.
msg emessage: το μήνυμα που σχετίζεται με το γεγονός.
event_list *elink: δείκτης στο επόμενο γεγονός.

Οι τύποι των γεγονότων σχετίζονται με τα σημεία εισόδου στον πυρήνα. Τα σημεία εισόδου έχουν κωδικοποιηθεί με ονόματα που αντικαθιστούν σταθερές και case μαζί, έτσι η εκχώρηση στο πεδίο event γίνεται (αναγκαστικά) αριθμητικά. Οι αντιστοιχίες είναι:

TIMER_INTERRUPT	0
ARRIVAL	1
TRANSFER_COMPLETION_INTERRUPT	2
FAULT	3
ABORT1	4
HALT	5

SEND	6
RECE	7
RECM	8
DESP	9
ELSZ	10
CREP	11
START	12
STOP	13
INITIO	14

Οι καταχωρήσεις στη λίστα αυτή ξεκινούν από την κεφαλή της λίστας, τη μεταβλητή ehead (vardefs.h).

Όταν διακόπτεται μία διεργασία του συστήματος, δημιουργείται μία καταχώρηση στη λίστα διακοπών. Τα στοιχεία αυτής της λίστας είναι τύπου *interrupt_list* (defs.h), με πεδία:

int next_event: το γεγονός που αναβλήθηκε.
int iprocess: η διεργασία που διακόπηκε.
double rem_time: ο χρόνος μέχρι το γεγονός.
msg imessage: το σχετικό μήνυμα.
interrupt_list *ilink: το επόμενο γεγονός.

Και εδώ η αρχή της λίστας εντοπίζεται από μία μεταβλητή, την κεφαλή, με το όνομα ihead (vardefs.h). Οι τύποι των γεγονότων ορίζονται όπως στα παραπάνω.

Τα στατιστικά στοιχεία για τις ουρές μηνυμάτων των διεργασιών του συστήματος περιέχονται σε έναν πίνακα, με ένα στοιχείο ανά διεργασία. Κάθε τέτοιο στοιχείο είναι μία δομή τύπου *statistics* (defs.h), με πεδία:

char process_name[35]: όνομα διεργασίας.
double max_wait: μέγιστος χρόνος αναμονής ενός μηνύματος.
double w_acc: συσσωρευμένος χρόνος αναμονής στην ουρά.
int max_length: μέγιστο μήκος ουράς.
int length: τρέχον μήκος ουράς.
double tl_acc: συσσωρευμένο γινόμενο χρόνου επί μήκος.
double tlast: τελευταία μεταβολή στην ουρά.
double length_distribution[21]: συχνότητα χρόνου παραμονής σε κάθε μήκος.
long int entry_count: σύνολο καταχωρήσεων στην ουρά.

Όλα τα στοιχεία βρίσκονται στον πίνακα queue_statistics (vardefs.h), ο οποίος έχει μήκος ίσο με no_of_system_processes+1 (defs.h).

Τέλος, υπ' ευθύνη του πυρήνα έχουμε καθολικές μεταβλητές για τη διεργασία που εκτελείται (int j), αυτή που διανεμήθηκε τελευταία (int k) και το τρέχον γεγονός (int e), το χρόνο μέχρι το επόμενο γεγονός (double t). Όλα αυτά βρίσκονται στο αρχείο επικεφαλίδων vardefs.h. Για την παρακολούθηση της προσομοίωσης, έχουμε το χρόνο (double) current_time και το χρόνο της επόμενης εκτύπωσης στατιστικών (double) plot_time. Επίσης, έχουμε τα μέγιστα για τις καταχωρήσεις σε όλους τους πίνακες, λίστες και ουρές του συστήματος, τους συνολικούς μέσους χρόνους σε διάφορες καταστάσεις για τις διεργασίες του συστήματος και μερικά τρέχοντα πλήθη καταχωρήσεων σε δομές του συστήματος. Τις παραμέτρους του συστήματος που ορίζονται από το χρήστη κατά την αρχική φόρτωση, θα τις δούμε στον κώδικα που σχετίζεται με την αρχικοποίηση.

3.1.1.3. Αλγόριθμοι

3.1.1.3.1. Δρομολόγηση εξωτερικών γεγονότων

Τα γεγονότα εισάγονται στην εξωτερική λίστα από τη συνάρτηση *insert_in_event_list* (simr.c), η οποία δημιουργεί ένα νέο στοιχείο για το γεγονός και το γεμίζει με τις τρέχουσες τιμές των σχετικών καθολικών μεταβλητών, εκτός από το πεδίο χρόνου, το οποίο παίρνει την τιμή τρέχοντα χρόνου συν το διάστημα μέχρι το γεγονός (αυτό δίνεται από την καθολική μεταβλητή t, από το vardefs.h). Η εισαγωγή γίνεται σύμφωνα με το χρόνο που θα συμβεί το γεγονός, με τα γεγονότα που θα συμβούν στον ίδιο χρόνο να εισάγονται με τη σειρά άφιξης (εισαγωγής) τους στη λίστα (FIFO).

3.1.1.3.2. Παρακολούθηση ουρών του συστήματος

Κατά την είσοδο μηνυμάτων στις ουρές των διεργασιών του συστήματος, καλείται η συνάρτηση *in_statistics*, με παράμετρο την ουρά που μας ενδιαφέρει (int que). Η συνάρτηση αυτή ενημερώνει τις κατανομές των μηκών της ουράς, τα πεδία μήκους και συσσωρευμένου γινομένου χρόνου και μήκους, το μετρητή συνόλου καταχωρήσεων και το χρόνο τελευταίας αλλαγής κατάστασης. Κατά την έξοδο στοιχείων από την ουρά, καλείται η συνάρτηση *out_statistics*, η οποία είναι όμοια με την προηγούμενη συνάρτηση, με τη διαφορά ότι έχει σαν παράμετρο και το χρόνο εισαγωγής του μηνύματος στην ουρά (double tin). Και οι δύο συναρτήσεις βρίσκονται στο αρχείο simr.c.

3.1.1.3.3. Παραγωγή ψευδοτυχαίων αριθμών

Η παραγωγή ψευδοτυχαίων αριθμών γίνεται από τη συνάρτηση *myrandom*, με παράμετρο τον τρέχοντα αριθμό της σειράς (int *x). Η σειρά έχει πολλαπλασιαστικό όρο το 16453, αθροιστικό όρο το 6925 και μέτρο το

32768. Οι αριθμοί που επιστρέφονται δεν είναι οι αριθμοί της σειράς, αλλά αριθμοί στο διάστημα (0,1), μετά από διαίρεση. Επειδή η σειρά δίνει κλειστό διάστημα, αποκόπτουμε τα άκρα 0 και 1. Φυσικά ενημερώνεται και ο τρέχων αριθμός της σειράς x. Βρίσκεται στο αρχείο simr.c.

3.1.1.3.4. Ιστογράμματα

Η συνάρτηση *histogram* (simr.c), σχεδιάζει ένα ιστόγραμμα στην έξοδο το οποίο παριστάνει την κατανομή συχνοτήτων για ένα στοιχείο (μία διεργασία) του πίνακα queue_statistics. Παίρνει σαν παραμέτρους τον πίνακα της κατανομής (double v[]), τη μέγιστη τιμή μήκους (int xmax), δηλαδή το πλήθος τιμών που θα ληφθούν υπόψη από τον πίνακα v, και τα ονόματα των αξόνων x και y (char *x_axis και char *y_axis). Η συνάρτηση βρίσκει το μέγιστο στοιχείο του διανύσματος μηκών και επιχειρεί να κάνει κανονικοποίηση των μηκών, έτσι ώστε να γίνεται η κατάλληλη συγκριτική παρουσίαση. Αν το πετύχει, τυπώνει το διάνυσμα με τους χαρακτήρες '*'. Αν δεν μπορεί να γίνει κανονικοποίηση, τυπώνεται ένα μήνυμα που ενημερώνει για το γεγονός. Στην έκδοση αυτή μπορούμε να δούμε τα ιστογράμματα σε τερματικό πλάτους 80 στηλών.

3.1.1.3.5. Ρουτίνες πρωτογενών κλήσεων

Οι ρουτίνες αυτές καλούνται όταν θέλουμε να κάνουμε μία πρωτογενή κλήση και είναι μία σειρά συναρτήσεων με τα ονόματα *send_message*, *receive_event*, *receive_message*, *create_path*, *change_eligible_set_number*, *destroy_path*, *start_process*, *stop_process* και *initiate_io*. Κάθε συνάρτηση παίρνει μία παράμετρο (int event2) που είναι το σημείο εισόδου από το οποίο θα συνεχίσει την εκτέλεσή της η διεργασία στην επόμενη ενεργοποίησή της. Οι κλήσεις receive_event και receive_message παίρνουν και μία δεύτερη παράμετρο (int event1) που δίνει το σημείο ενεργοποίησης της διεργασίας αν δεν υπάρχει κατάλληλη ενεργοποίηση ή μήνυμα μετά την κλήση. Το σημείο αυτό είναι το σημείο από όπου θα επαναληφθεί η κλήση αυτή. Όλες οι συναρτήσεις, σημειώνουν το σημείο ενεργοποίησης στον pdt, ενημερώνουν την καθολική μεταβλητή e για το αντίστοιχο γεγονός (χρησιμοποιώντας τις ίδιες τιμές με τις εισόδους του πυρήνα), βρίσκουν το χρόνο μέχρι το γεγονός, εισάγουν το γεγονός στη λίστα γεγονότων της εξωτερικής προσομοίωσης και τελικά μεταφέρουν με μη τοπικό *goto* (συνάρτηση longjmp στη C) τον έλεγχο στον πυρήνα (είσοδος kernel). Οι συναρτήσεις με τις δύο παραμέτρους αποθηκεύουν το εναλλακτικό σημείο εισόδου στο message.mport. Παρατηρήστε ότι για να δουλέψει το μη τοπικό goto θα πρέπει σε κάθε στιγμή στη μεταβλητή kernel_entry να έχουμε αποθηκευμένες τις κατάλληλες πληροφορίες για το σημείο προορισμού (ετικέτα kernel) μέσω της κλήσης *setjmp*. Έτσι, με την *longjmp* έχουμε επιστροφή στο σημείο όπου κλήθηκε η

setjmp, όπου μπορούμε να την ξανακαλέσουμε για αν ενημερώσουμε την κατάσταση του συστήματος και μετά να προχωρήσουμε την εκτέλεση. Το μόνο σημείο που θέλει προσοχή είναι ότι πρέπει να είμαστε βέβαιοι ότι οι πληροφορίες είναι έγκυρες και κατά την πρώτη της κλήση (αυτό αντιμετωπίζεται με τη μεταβλητή flag στη συνάρτηση main).

Υπάρχουν δύο εκδόσεις των ρουτινών αυτών. Η μία έκδοση χρησιμοποιεί σταθερό χρόνο για κάθε πρωτογενή κλήση (primitive_time). Η δεύτερη έκδοση χρησιμοποιεί αυτό το χρόνο για τις διεργασίες χρήστη, αλλά για τις διεργασίες του συστήματος χρησιμοποιεί το χρόνο που πέρασε από την τελευταία είσοδο σε διεργασία του συστήματος, ουσιαστικά δηλαδή χρησιμοποιεί τον πραγματικό χρόνο που δαπάνησε αυτή η διεργασία για την εκτέλεσή της σαν χρόνο μέχρι το γεγονός. Ο χρόνος υπολογίζεται με χρήση της συνάρτησης cputim. Οι απλές εκδόσεις των ρουτινών αυτών βρίσκονται στο αρχείο pir.c, ενώ οι ρουτίνες με τον πραγματικό χρόνο στο opir.c.

3.1.1.3.6. Διαχείριση λαθών

Τα λάθη που εντοπίζονται με σήματα (signals) στη γλώσσα C, παγιδεύονται αυτόματα από το σύστημα. Χρησιμοποιήσαμε μόνο τα σήματα που υποστηρίζονται από την ANSI C και υπάρχουν και στο UNIX. Οι ρουτίνες παγίδευσης αυτών των σφαλμάτων βρίσκονται στο αρχείο simr.c και απλά τυπώνουν ένα μήνυμα λάθους πριν καλέσουν τη συνάρτηση error. Οι συναρτήσεις είναι οι *sigint*, *sigill*, *sigfpe*, *sigsegv*, *sigterm*, *sigabrt* και παγιδεύουν τα σήματα με το ίδιο όνομα (με κεφαλαία όμως).

Σε περίπτωση λάθους, ανεξάρτητα από το αν έχει βρεθεί αυτόματα, μέσα από τα σήματα του λογισμικού, ή αν έχει βρεθεί από κάποιον έλεγχο στο σύστημα, καλείται η συνάρτηση *error* (simr.c). Η συνάρτηση αυτή τυπώνει την ενεργή διεργασία, τις θύρες της, τα περιεχόμενα του μηνύματος που είναι τρέχον, τη λίστα γεγονότων της διεργασίας (για διεργασίες χρήστη), τη λίστα γεγονότων της εξωτερικής προσομοίωσης, τη λίστα διακοπών (διεργασίες συστήματος), τις θύρες των διεργασιών του συστήματος, τους πίνακες διεργασιών και τεμαχίων, τις λίστες οπών και κατειλημμένων και τέλος τα στατιστικά στοιχεία του συστήματος (καλώντας την plot_statistics για τον τρέχοντα χρόνο), εγκαταλείποντας στη συνέχεια την εκτέλεση με επιστρεφόμενο κωδικό το 1 (σφάλμα εκτέλεσης), χρησιμοποιώντας τη συνάρτηση exit (σε κανονικό τερματισμό επιστρέφεται 0, κατά τη συνήθη σύμβαση του UNIX).

3.1.1.3.7. Εκτύπωση στατιστικών

Αυτή η εργασία γίνεται από τη συνάρτηση *plot_statistics* (simr.c) η οποία παίρνει σαν παράμετρο το χρόνο για την εκτύπωση των στατιστικών (double time). Η συνάρτηση βρίσκει τον πραγματικό χρόνο εκτέλεσης του προγράμματος και τυπώνει γενικά στοιχεία της προσομοίωσης (επιδόσεις προσομοίωσης, απόδοση συστήματος), το βαθμό χρήσης των συσκευών και τερματικών, στοιχεία για τις διεργασίες που απέτυχαν να ολοκληρωθούν, το πλήθος χρηστών που απορρίφθηκαν, στοιχεία για τις δομές του συστήματος, την απόδοση για τα διάφορα είδη διεργασιών και αναλυτικά στοιχεία εκτέλεσης και ουρών για όλες τις διεργασίες του συστήματος, πιθανά και με ιστογράμματα.

3.1.1.3.8. Αρχικοποίηση συστήματος

Οι παράμετροι που ζητούνται για την προσομοίωση είναι οι ακόλουθες (όλες στο vardefs.h):

int option: αρνητικό για ιστογράμματα, 0 για τίποτα, 1 για στοιχεία δημιουργίας και διαγραφής εργασιών, 2 για στοιχεία δημιουργίας και διαγραφής εργασιών και διεργασιών, 3 για πλήρη παρακολούθηση. Σε όλες τις επιλογές τυπώνονται στατιστικά και λάθη.
long int no_of_jobs: πλήθος εργασιών (όριο). Μικρό πλήθος μπορεί να δείξει αν το σύστημα πέφτει σε αδιέξοδο.
double simulation_period: χρόνος προσομοίωσης (όριο).
double statistics_period: μεσοδιάστημα εκτύπωσης στατιστικών.
float mean_interarrival_time: μεσοδιάστημα άφιξης χρηστών.
int arnd: αρχικός τυχαίος αριθμός.

Ο κώδικας ξεκινάει από την αρχή της *main* (kernel.c). Αρχικά ρυθμίζουμε την παγίδευση των διακοπών σφαλμάτων (signals). Διαβάζουμε τις παραμέτρους και αρχικοποιούμε το πλήθος των εργασιών που απομένουν (int job_count, στο vardefs.h) και το χρόνο της επόμενης εκτύπωσης στατιστικών. Οι λίστες γεγονότων και διακοπών είναι κενές. Καλούμε τη cputim για να αρχίσουμε τη μέτρηση του πραγματικού χρόνου εκτέλεσης.

Αμέσως μετά διαβάζουμε τις ακόλουθες παραμέτρους για το υλικό (όλες στο vardefs.h):

long int store_size: μέγεθος μνήμης διαθέσιμης για διεργασίες των χρηστών.
float page_size: μέγεθος σελίδας (σελιδοποίηση) ή μέσο μέγεθος τμημάτων (τεμαχισμός).

float time_to_copy_byte: χρόνος αντιγραφής ενός byte μεταξύ θέσεων της κύριας μνήμης.
float context_switching_time: χρόνος της ΚΜΕ για τη μεταγωγή των περιεχομένων των διεργασιών.
float enter_time: χρόνος εισαγωγής σε διεργασία (επανενεργοποίηση διεργασίας).
float primitive_time: χρόνος εξυπηρέτησης πρωτογενούς κλήσης.

Ορίζουμε τις υπόλοιπες παραμέτρους του πίνακα περιγραφητών των συσκευών με εντολές του προγράμματος. Για το υλικό έχουμε επίσης μόνιμα ρυθμισμένες τις σταθερές no_of_terminals (defs.h) για το πλήθος τερματικών και no_of_devices (defs.h) για το πλήθος των συσκευών.

Στη συνέχεια, διαβάζουμε τις χαρακτηριστικές παραμέτρους του συστήματος, δηλαδή (όλες στο vardefs.h):

float timer_interval: το χρονικό μεσοδιάστημα χρονοπρογραμματισμού (χρόνος μεταξύ διακοπών του χρονομέτρου).
float maxresp: μέγιστη επιτρεπτή απόκριση (πάνω από αυτό το όριο, απορρίπτουμε νέους χρήστες στα τερματικά).
int physical_blocksize: φυσικό μέγεθος μεταφορών από / προς το δίσκο.
float time_to_service_routine[15]: χρόνος εξυπηρέτησης για όλα τα σημεία εισόδου του πυρήνα.

Ο πίνακας περιγραφητών τεμαχίων δεν χρειάζεται αρχικοποίηση (μόνο οι δείκτες στα ελεύθερα στοιχεία του) αφού οι διεργασίες του συστήματος δεν περιέχονται σε αυτόν. Για τον πίνακα περιγραφητών των διεργασιών, αρχικοποιούμε τις διεργασίες του συστήματος με αναγνωριστικό τον αριθμό θέσης τους. Οι διεργασίες είναι στη μνήμη και έτοιμες για εκτέλεση. Αρχικά δεν έχουν παιδιά και πατέρας τους είναι ο χρονοπρογραμματιστής των εργασιών. Τα στοιχεία μετρήσεων μηδενίζονται και οι χρόνοι μεταβολών τίθενται στον τρέχοντα χρόνο. Οι διεργασίες συνδέονται με τη σειρά σε μία λίστα δύο κατευθύνσεων. Στη συνέχεια ορίζονται κατάλληλα μερικές θύρες (οι μόνιμες) των διεργασιών του συστήματος και οι υπόλοιπες θύρες γίνονται αόριστες. Ετοιμάζονται οι πίνακες ports και ports0to. Η λίστα του διανομέα αρχίζει από τη διεργασία μηδέν του συστήματος, και το σύνολο καταλλήλων (int elsetno, στο vardefs.h) γίνεται ίσο με το πλήθος των διεργασιών του συστήματος. Ο πίνακας εργασιών είναι κενός. Αρχικοποιούνται οι μετρήσεις στον πίνακα queue_statistics και ο χρόνος αδράνειας (double cpuit), τελευταίας χρήσης (double cpulbt) και τελευταίας μέτρησης στατιστικών για την ΚΜΕ (float time_last, όλα στο vardefs.h). Η ΚΜΕ δεσμεύεται (int cpu_busy, στο vardefs.h) και μηδενίζονται τα στατιστικά στοιχεία για τις συσκευές. Τα τερματικά γίνονται διαθέσιμα, η επιβάρυνση του πυρήνα (float

kernel_overhead, στο vardefs.h) μηδενίζεται και προχωράμε να προγραμματίσουμε το πρώτο γεγονός στο σύστημα.

Το πρώτο γεγονός είναι η πρώτη διακοπή χρονομέτρου που θα συμβεί μετά από το μεσοδιάστημα που ορίστηκε. Το γεγονός χρονοπρογραμματίζεται, η διεργασία 0 (διαχειριστής ΚΜΕ) ορίζεται ως εκτελούμενη και υπό διανομή και προχωράμε στην είσοδο kernel για να αποθηκεύσουμε το περιβάλλον. Επειδή η flag είναι 1 (αρχική τιμή), θα επιστρέψουμε αμέσως στην επόμενη εντολή, που είναι η είσοδος enter σε μία διεργασία (έτσι λύνεται το πρόβλημα της αρχικής αποθήκευσης του περιβάλλοντος της ετικέτας kernel). Εκεί, ενημερώνουμε το χρόνο του συστήματος (προχωράει όσο χρόνο χρειάζεται για εισαγωγή σε μία διεργασία). Ακολουθεί η ετικέτα enter1, στην οποία ερχόμαστε όταν δεν θέλουμε να μπούμε κανονικά στη διεργασία (δηλαδή, να μην περάσει ο χρόνος εισαγωγής, πράγμα χρήσιμο σε περίπτωση που δεν έπρεπε να έχουμε φύγει καθόλου από τη διεργασία που εκτελούσαμε). Αν η διεργασία j είναι μία διεργασία του συστήματος, βρίσκουμε το χρόνο από την αρχή της κλήσης και καλούμε την κατάλληλη συνάρτηση υλοποίησης της διεργασίας. Αλλιώς, προχωράμε άμεσα στον κώδικα των διεργασιών χρήστη, αφού εντοπίσουμε τον πίνακα σκιάς της διεργασίας.

3.1.2. Εσωτερική προσομοίωση

3.1.2.1. Εισαγωγή

Όλες οι διεργασίες των χρηστών προσομοιώνονται από ένα κοινό κομμάτι κώδικα, το οποίο υλοποιεί την εσωτερική προσομοίωση. Η εσωτερική προσομοίωση έχει τις δικές της δομές για την παρακολούθηση των διεργασιών του χρήστη. Κύριο μέλημα της εσωτερικής προσομοίωσης είναι η δρομολόγηση γεγονότων των διεργασιών των χρηστών, μέσω των οποίων προσομοιώνουμε το φόρτο εργασίας ενός πραγματικού συστήματος. Τα βασικά γεγονότα για τις διεργασίες των χρηστών είναι σφάλματα σελίδων, απαιτήσεις ε/ε και δημιουργία και διαγραφή παιδιών. Ο κώδικας βρίσκεται στο *kernel.c*, αμέσως μετά από τον κώδικα της αρχικής φόρτωσης στη συνάρτηση *main*, με ορισμένες βοηθητικές συναρτήσεις να βρίσκονται στο *simr.c*.

3.1.2.2. Δομές Δεδομένων

Για να μπορούμε να βρίσκουμε τα στοιχεία που υπολογίζει το πρόγραμμα της προσομοίωσης για τα χαρακτηριστικά των διεργασιών των χρηστών, με κάθε διεργασία σχετίζεται μία καταχώρηση στους πίνακες σκιάς του συστήματος. Αυτοί οι πίνακες περιέχουν τις εκτιμήσεις που σχετίζονται με τη διεργασία και χρησιμοποιούνται για τη δρομολόγηση των γεγονότων της διεργασίας. Οι καταχωρήσεις αυτές συνδέονται μεταξύ τους όπως και οι

περιγραφητές των διεργασιών και είναι προσπελάσιμες από τον pdt. Για κάθε διεργασία έχουμε ένα στοιχείο τύπου *user_process* (defs.h), με τα ακόλουθα πεδία:

double run_time: ζητούμενος χρόνος εκτέλεσης της διεργασίας.
double load_time: χρόνος δημιουργίας της διεργασίας.
double remaining_time: χρόνος που απομένει μέχρι την ολοκλήρωση της διεργασίας (αν εκτελεστεί μόνη της στο σύστημα).
int requests[8]: πλήθος απαιτήσεων ανά αρχείο (μέχρι 8 αρχεία).
int remaining_requests[8]: απαιτήσεις που απομένουν.
float mean[10]: μέσοι των κατανομών διαστημάτων μεταξύ απαιτήσεων, θέση 0 για σφάλματα τεμαχίων, θέση 1 για δημιουργία παιδιών, θέσεις 2 μέχρι 9 για τα αρχεία.
int files: πλήθος αρχείων της διεργασίας.
int no_of_sons: πλήθος παιδιών.
int remaining_sons: παιδιά που απομένουν.
int working_set_size: μέγεθος (τρέχοντος) συνόλου εργασίας.
jdt *job: περιγραφητής αντίστοιχης εργασίας.
int prnd: τυχαίος αριθμός της διεργασίας. user_process *brother_pointer, *son_pointer: δείκτης προς καταχώρηση σκιάς για αδελφό και γιο.
evlist *procptr: λίστα γεγονότων της διεργασίας.
int no_of_sections: πλήθος τεμαχίων.
int no_of_sections_not_in_core: τεμάχια εκτός μνήμης.
int out_of_ws_sections: τεμάχια εκτός μνήμης αλλά εντός συνόλου εργασίας.
int sections[no_of_pages+1]: μεγέθη των τεμαχίων της διεργασίας. Η σταθερά no_of_pages ορίζεται στο defs.h και είναι το άνω όριο τεμαχίων για κάθε διεργασία χρήστη.

Τα στοιχεία της εσωτερικής λίστας γεγονότων της διεργασίας περιέχουν χρόνους που μετρούνται από την αρχή της εκτέλεσης της διεργασίας (είναι σχετικοί χρόνοι). Αυτοί οι χρόνοι δεν είναι ίδιοι με το χρόνο που έχουμε στα στοιχεία της λίστας γεγονότων της εξωτερικής προσομοίωσης, γιατί δεν μπορούν να οριστούν κατά την αρχική δρομολόγηση σαν απόλυτες χρονικές στιγμές, αφού δεν ξέρουμε πότε θα εκτελεστεί η διεργασία και πόσες φορές και για πόσο θα διακοπεί. Οι καταχωρήσεις στις εσωτερικές λίστες είναι στοιχεία τύπου *evlist* (defs.h) με πεδία:

int evtype: τύπος εσωτερικού γεγονότος.
double evtime: χρόνος για το γεγονός (χρόνος διεργασίας).
int suspension_marker: ψηφίο που δείχνει αν το γεγονός αυτό προκάλεσε την αναστολή της διεργασίας.
msg evmessage: σχετικό μήνυμα.
evlist *evptr: δείκτης στο επόμενο γεγονός.

Οι δείκτες στην αρχή των λιστών βρίσκονται στο σχετικό πεδίο του pdt, στον κάθε περιγραφητή χωριστά.

Τέλος έχουμε τη μεταβλητή int sswitch, που δείχνει το τρέχον γεγονός χρήστη (χρησιμοποιείται και για την επιλογή του κατάλληλου κώδικα προς εκτέλεση), την user_process *p3 που δείχνει την τρέχουσα καταχώρηση στον πίνακα σκιάς για την διεργασία που μας ενδιαφέρει και την double t1 για το χρόνο της διεργασίας κατά τον οποίο θα συμβεί το γεγονός (όλες στο vardefs.h). Αυτές οι μεταβλητές χρησιμοποιούνται και για έμμεσο πέρασμα παραμέτρων στη συνάρτηση που δρομολογεί τα εσωτερικά γεγονότα των διεργασιών των χρηστών.

3.1.2.3. Αλγόριθμοι

3.1.2.3.1. Εκτίμηση τυχαίων αριθμών

Έχουμε τη συνάρτηση *dist* η οποία επιστρέφει έναν double με τιμή από μία αρνητική εκθετική κατανομή, παίρνοντας σαν παραμέτρους τα όρια (float a το κάτω και double b το άνω), τον μέσο της κατανομής (float m) και τον τυχαίο αριθμό (int *r). Για διακριτές τιμές, έχουμε την *ddist* που επιστρέφει int και έχει όρια τους ακεραίους (int) a και b, μέσο το float m και τυχαίο αριθμό τον int *r (και οι δύο στο simr.c).

3.1.2.3.2. Δρομολόγηση εσωτερικών γεγονότων

Η συνάρτηση *insert_in_list* (simr.c) δημιουργεί μία νέα καταχώρηση εσωτερικού γεγονότος και την ενημερώνει με τις τρέχουσες τιμές των t1, sswitch και message. Το γεγονός κατά την εισαγωγή του με αυτό τον τρόπο δεν προκαλεί ποτέ αναστολή. Η εισαγωγή του γεγονότος στη λίστα της διεργασίας (ο σχετικός πίνακας σκιάς βρίσκεται από το p3) γίνεται με LIFO εισαγωγή για τα στοιχεία που θα συμβούν στην ίδια χρονική στιγμή, δηλαδή το πιο πρόσφατα εισηγμένο στοιχείο μπαίνει πρώτο.

3.1.2.3.3. Κώδικας εσωτερικής προσομοίωσης

Ο κώδικας των διεργασιών του χρήστη ακολουθεί άμεσα τις εντολές εισαγωγής σε διεργασία στη συνάρτηση *main* (kernel.c). Η πρώτη είσοδος στον κώδικα αυτό αρχικοποιεί τις μεταβλητές που είναι κοινές για όλες τις διεργασίες χρήστη (ο έλεγχος για την πρώτη είσοδο γίνεται μέσω της ακέραιας μεταβλητής first_time). Στο τμήμα αυτό διαβάζουμε τις ακόλουθες παραμέτρους (ορίζονται στο vardefs.h):

float reference_string: μέσο μήκος ακολουθίας αναφοράς του προγράμματος μίας διεργασίας.
double ws_quantum: μεσοδιάστημα αλλαγής συνόλου εργασίας.
float reduction_of_mean: μείωση των μέσων τιμών.

Η παράμετρος μείωσης χρειάζεται επειδή κατά την εκτίμηση των διάφορων χρόνων που σχετίζονται με τη δρομολόγηση γεγονότων, τίθεται ένα τεχνητό κάτω όριο ίσο με το χρόνο πρωτογενούς κλήσης, το οποίο κάνει τις διεργασίες να ξεπερνούν το χρόνο που το σύστημα είναι διατεθειμένο να τους παραχωρήσει (έχουμε έναν συστηματικό παράγοντα που μετακινεί τον πραγματικό μέσο προς τα επάνω). Ρυθμίζοντας αυτή την παράμετρο, μπορούμε να ελέγξουμε το πλήθος των διεργασιών που θα διακόπτονται από το σύστημα λόγω χρονικών υπερβάσεων.

Σημειώστε αρχικά ότι τα γεγονότα για το σύστημα της εσωτερικής προσομοίωσης κωδικοποιούνται αριθμητικά στη μεταβλητή sswitch πριν από τη δρομολόγησή τους, ως εξής:

working_set_change	-1 (είναι ετικέτα για άμεση κλήση και αντιμετωπίζεται διαφορετικά από τα άλλα γεγονότα)
SECTION_FAULT	0
IOREQUEST	1
WAIT_FOR_IO	2
CREATE_SON	3
WAIT_FOR_CREATION	4
DELETE_SON	5
WAIT_FOR_DELETION	6

Παρατηρήστε ότι τα γεγονότα 1 έως 6 δίνονται σαν ζεύγη (γεγονός αίτησης, γεγονός αναμονής), πράγμα που είναι πολύ χρήσιμο στην αυτόματη δρομολόγηση ζευγών τέτοιων γεγονότων, όπως θα δούμε παρακάτω. Οι διεργασίες που δημιουργούνται, έχουν αρχικά ορισμένες τις θύρες 0 (γονέας), 1 (δημιουργός διεργασιών), 2 (σύστημα τερματικών) και 3 (σύστημα αρχείων).

Επειδή ο κώδικας είναι καταμεριζόμενος από όλες τις διεργασίες χρηστών, μετά από την αρχικοποίηση των παραμέτρων του συστήματος γίνεται η αρχικοποίηση της τρέχουσας διεργασίας χρήστη. Η διεργασία που δημιουργείται έχει κενή λίστα γεγονότων, είναι εκτός μνήμης και δεν έχει εξυπηρετηθεί ακόμη καθόλου. Αρχικά, δρομολογούμε μία απαίτηση για κάθε αρχείο για το οποίο υπάρχουν απαιτήσεις που περιμένουν να εξυπηρετηθούν (περιλαμβάνονται τα τερματικά και οι spoolers). Οι απαιτήσεις δίνονται θέτοντας mcommand=0. Η θύρα του μηνύματος που σχετίζεται με τα γεγονότα ε/ε είναι είτε η θύρα του συστήματος αρχείων είτε η θύρα του συστήματος

τερματικών. Παρατηρήστε ότι διαβάζουμε πάντα μία φυσική εγγραφή (word7) και ότι οι μέσοι για τις κατανομές των απαιτήσεων βρίσκονται δύο θέσεις παραπάνω από τις καταχωρήσεις για το πλήθος των απαιτήσεων (πίνακες requests και mean), επειδή στον πίνακα μέσων έχουμε μέσους και για απαιτήσεις εκτός από τα αρχεία. Οι μέσοι για την εκτίμηση των μεσοδιαστημάτων μεταξύ των απαιτήσεων βρίσκονται διαιρώντας το συνολικό χρόνο εκτέλεσης με το πλήθος απαιτήσεων από κάθε πηγή (διαιρούμε με το δύο γιατί τα γεγονότα γίνονται στο μέσο των διαστημάτων αυτών). Ο παράγοντας μείωσης μέσου εφαρμόζεται για προσαρμογή προς τα κάτω των μέσων. Τελικά τα πρώτα γεγονότα εκτιμώνται να συμβούν ανάμεσα στη αρχή εκτέλεσης της διεργασίας και το χρόνο που της απομένει ακόμη (με τους μέσους που βρήκαμε) και τα γεγονότα μπαίνουν στη εσωτερική λίστα της διεργασίας. Παρατηρήστε το κάτω όριο χρόνου για τα διαστήματα. Στη συνέχεια δρομολογούμε τη δημιουργία ενός παιδιού (αν χρειάζεται) περνώντας το δείκτη του πρώτου γιου στο word8 και σαν εντολή τη δημιουργία (mcommand=0) προς τη θύρα του δημιουργού διεργασιών. Ο μέσος για τα διαστήματα εκτιμάται όπως και για τα γεγονότα ε/ε και το πρώτο γεγονός δρομολογείται με τον ίδιο ακριβώς τρόπο όπως παραπάνω. Τέλος, προγραμματίζουμε να γίνει αμέσως η πρώτη αλλαγή συνόλου εργασίας, την οποία και κάνουμε προχωρώντας στην είσοδο working_set_change.

Η είσοδος working_set_change καλείται όταν απαιτείται αλλαγή συνόλου εργασίας. Αυτό που κάνουμε είναι να δρομολογήσουμε την επόμενη αλλαγή συνόλου εργασίας (μετά από χρόνο ws_quantum) αν προλαβαίνει βέβαια να συμβεί όσο θα εκτελείται η διεργασία. Σε κάθε περίπτωση, στο χρόνο μέχρι το τέλος της εκτέλεσης ή στο χρόνο μέχρι την επόμενη αλλαγή συνόλου εργασίας (ότι είναι μικρότερο), πρέπει να δρομολογήσουμε μία σειρά από σφάλματα τεμαχίων. Αρχικά τα τεμάχια του τρέχοντος συνόλου εργασίας σημειώνονται ως εκτός συνόλου εργασίας και υπολογίζουμε (εκτίμηση) το νέο μέγεθος (εδώ χρησιμοποιείται το reference_string). Ο μέσος της κατανομής μεταξύ σφαλμάτων εκτιμάται για το επόμενο μεσοδιάστημα (και όχι μόνο κατά τη δημιουργία της διεργασίας, αφού το μέγεθος του συνόλου εργασίας αλλάζει σε κάθε μεσοδιάστημα) και δρομολογούμε ένα σφάλμα για κάθε τεμάχιο του συνόλου εργασίας, για κάποιο τυχαίο τεμάχιο της διεργασίας (στο word3 το τεμάχιο που μας ενδιαφέρει). Τα σφάλματα δρομολογούνται σε τυχαίες χρονικές στιγμές (χωρίς κάποια σειρά), μέσα στα όρια του χρόνου του μεσοδιαστήματος βέβαια. Μετά, προχωράμε άμεσα στην κύρια είσοδο των διεργασιών χρήστη.

Η κύρια είσοδος εξετάζει αρχικά αν υπάρχουν γεγονότα στη λίστα της διεργασίας. Αν όχι, εκτιμούμε το χρόνο ολοκλήρωσης της διεργασίας και δρομολογούμε ένα γεγονός τερματισμού στην εξωτερική λίστα της προσομοίωσης, βγαίνοντας αμέσως προς τον πυρήνα (kernel). Αλλιώς,

εξετάζουμε αν η διεργασία έχει υπερβεί το χρονικό της όριο, οπότε θέτουμε το word6=1 (υπέρβαση χρονικού ορίου) και καλούμε την είσοδο του πυρήνα *abort_now* για να ανασταλεί αμέσως η διεργασία. Σε κάθε άλλη περίπτωση, βρίσκουμε το επόμενο γεγονός της διεργασίας και το αφαιρούμε από τη λίστα της. Αν το γεγονός είναι αλλαγή συνόλου εργασίας, διακλαδωνόμαστε στην είσοδο *working_set_change*. Αλλιώς, υπολογίζουμε το χρόνο μετά τον οποίο θα πρέπει να συμβεί το γεγονός (εξωτερικός χρόνος) και μειώνουμε το χρόνο που μένει στη διεργασία για εκτέλεση κατά την ποσότητα αυτή. Στη συνέχεια, διακλαδωνόμαστε σε μία από τις (υπο)εισόδους της διεργασίας χρήστη, ανάλογα με το γεγονός.

Η υποείσοδος SECTION_FAULT καλείται όταν έχουμε σφάλμα τεμαχίου. Το νέο σημείο εισόδου της διεργασίας χρήστη γίνεται η κύρια είσοδος και αν το σφάλμα πρέπει να συμβεί αμέσως, πάμε κατ' ευθείαν στην είσοδο *fault_now* (στον πυρήνα). Αλλιώς, το γεγονός σφάλματος δρομολογείται στη λίστα της προσομοίωσης και βγαίνουμε προς τον πυρήνα (kernel). Η υποείσοδος IO_REQUEST καλείται για τη δρομολόγηση μίας μεταφοράς ε/ε και εκτιμάει το χρόνο μέχρι την κλήση ε/ε, πριν να μεταφέρει τον έλεγχο στην είσοδο *schedule_both*. Οι είσοδοι CREATE_SON και DELETE_SON προετοιμάζουν μία αποστολή μηνύματος, για δημιουργία και διαγραφή παιδιού αντίστοιχα, σε χρόνο μετά από το χρόνο πρωτογενούς κλήσης. Τελικά, και αυτές μεταφέρουν τον έλεγχο στην είσοδο schedule_both, από όπου δρομολογείται μία αποστολή μηνύματος, με σημείο επαναφοράς την κύρια είσοδο, στην εξωτερική λίστα της προσομοίωσης. Τα γεγονότα αυτά αντιστοιχίζονται σε ένα γεγονός αναμονής, το οποίο δρομολογείται για το χρόνο κατά τον οποίο θα συμβεί η αποστολή του μηνύματος, αλλά στη λίστα της εσωτερικής προσομοίωσης, με τελική έξοδο στον πυρήνα (kernel). Παρατηρήστε ότι η αρίθμηση των γεγονότων αποστολής μηνύματος και αναμονής για εξυπηρέτηση βοηθάει στην εύρεση του αριθμού του γεγονότος αναμονής (απλά προσθέτουμε 1). Έτσι, όλα αυτά τα γεγονότα γίνονται σε ζεύγη για την εσωτερική προσομοίωση και όταν το ένα δρομολογείται εξωτερικά το δεύτερο δρομολογείται εσωτερικά.

Η υποείσοδος WAIT_FOR_IO καλείται όταν αναμένουμε απάντηση σε αίτηση για ε/ε και ορίζει αναμονή από το σύστημα αρχείων ή τερματικών και γεγονός αναμονής μηνύματος, το οποίο δρομολογεί στην εξωτερική λίστα της προσομοίωσης. Το σημείο εισόδου σε περίπτωση επιτυχίας είναι η είσοδος 3, ενώ σε περίπτωση αποτυχίας η είσοδος 2. Τελικά έχουμε έξοδο στον πυρήνα. Η είσοδος 2, απλά επανεκδίδει την κλήση για αναμονή μηνύματος με τις ίδιες παραμέτρους. Όταν ληφθεί το μήνυμα, μπαίνουμε στην είσοδο 3. Εκεί, αν έχουμε αποτυχία μεταφοράς (mcommand=0), προχωράμε στην είσοδο abort, με χρόνο το χρόνο πρωτογενούς κλήσης και την αιτία αποτυχίας στο word6. Αλλιώς, δρομολογούμε την επόμενη αίτηση για ε/ε από την ίδια συσκευή

(word3), μειώνοντας τις απαιτήσεις που μένουν (ελέγχοντας πρώτα αν μένουν απαιτήσεις). Η αίτηση δρομολογείται όπως και οι πρώτες αιτήσεις (στην είσοδο 0), μόνο που τώρα βέβαια είναι γνωστοί οι μέσοι των κατανομών. Ανεξάρτητα από το αν δρομολογείται η όχι μία νέα αίτηση, το σημείο εισόδου γίνεται η κύρια είσοδος, στην οποία και επιστρέφουμε αμέσως.

Η υποείσοδος WAIT_FOR_CREATION καλείται όταν αναμένουμε τη δημιουργία ενός παιδιού και σ' αυτή δρομολογείται η λήψη ενός μηνύματος από το δημιουργό διεργασιών, το οποίο λαμβάνεται στην είσοδο 5 ή επαναδρομολογείται σαν κλήση στην είσοδο 4. Τελικά έχουμε έξοδο στον πυρήνα (kernel). Στην είσοδο 4 επανεκδίδεται η κλήση με τις ίδιες παραμέτρους, ενώ στην είσοδο 5 ελέγχεται αν η φόρτωση ήταν αποτυχημένη (mcommand=0), οπότε και τίθεται ως αιτία αποτυχίας η αποτυχία φόρτωσης στο word6, χρόνος γεγονότος ο χρόνος πρωτογενούς κλήσης και ο έλεγχος μεταφέρεται στην είσοδο abort. Αλλιώς, το παιδί δημιουργήθηκε επιτυχώς, οπότε φτιάχνουμε ένα μονοπάτι προς το παιδί μέσω της θύρας 4, για να μπορούμε να επικοινωνούμε με αυτό. Μετά τη δημιουργία προχωράμε στην είσοδο 6 όπου, έχοντας έτοιμο το μονοπάτι, περιμένουμε ένα γεγονός από αυτή τη θύρα (ενεργοποίηση ή μήνυμα). Όταν ληφθεί το γεγονός, προχωράμε στην είσοδο 8, από όπου δρομολογούμε το γεγονός διαγραφής του παιδιού για έναν τυχαίο χρόνο μεταξύ του παρόντος και του υπόλοιπου χρόνου εκτέλεσης. Αφού δρομολογήσουμε εσωτερικά το γεγονός, καταστρέφουμε το μονοπάτι προς το παιδί και επιστρέφουμε μετά την κλήση στην κύρια είσοδο.

Η υποείσοδος WAIT_FOR_DELETION καλείται για να δρομολογήσει την αναμονή για την διαγραφή ενός παιδιού της διεργασίας. Αυτό γίνεται με τη δρομολόγηση της αναμονής για ένα μήνυμα από το δημιουργό διεργασιών, με σημείο επιτυχούς λήψης την είσοδο 9 και σημείο εισόδου σε αποτυχία λήψης την είσοδο 7. Τελικά, έχουμε έξοδο προς τον πυρήνα (kernel). Στην είσοδο 7 απλά επανεκδίδουμε την κλήση, ενώ στην είσοδο 9 μειώνουμε τα παιδιά που πρέπει να δημιουργήσει ακόμη η διεργασία και αν απομένουν παιδιά, δρομολογούμε το εσωτερικό γεγονός της δημιουργίας του επόμενου παιδιού, με τον ίδιο τρόπο που δρομολογήσαμε στην είσοδο 0 την πρώτη δημιουργία παιδιού (πάλι, ο μέσος είναι εδώ γνωστός). Ανεξάρτητα από τη δημιουργία ή όχι παιδιών, το σημείο εισόδου γίνεται η κύρια είσοδος, στην οποία επιστρέφουμε άμεσα.

Τέλος, η είσοδος *abort* καλείται μόνο μέσα από goto από τη διεργασία χρήστη, και δρομολογεί το γεγονός αναστολής διεργασίας στη λίστα της εξωτερικής προσομοίωσης, με τελική έξοδο στον πυρήνα (kernel). Η αιτία της αναστολής πρέπει να έχει ήδη συμπληρωθεί από το σημείο που κάλεσε την είσοδο abort. Παρατηρήστε ότι η είσοδος αυτή καλείται από τα γεγονότα

αναμονής, όταν έχουμε αποτυχία σε κάποια λειτουργία μίας διεργασίας (όχι κατά τη διαγραφή βέβαια).

3.2. Διεργασίες διεπαφής προσομοίωσης και συστήματος

Οι διεργασίες που βρίσκονται σε αυτό το τμήμα, ασχολούνται με την υλοποίηση λειτουργιών του συστήματος οι οποίες επηρεάζονται σημαντικά από το ότι προσομοιώνουμε ένα πραγματικό σύστημα. Έτσι, η έλλειψη του πραγματικού φόρτου εργασίας και του υλικού, εξαναγκάζει αυτές τις διεργασίες να λαμβάνουν σοβαρά υπόψη το ότι κάνουμε προσομοίωση, με αποτέλεσμα σημαντικό (αν όχι το μεγαλύτερο) μέρος του κώδικά τους να ασχολείται με εργασίες που σχετίζονται με την προσομοίωση και δεν υπάρχουν σε ένα πραγματικό σύστημα. Το μεγαλύτερο τμήμα του κώδικα αυτών των διεργασιών είναι δύσκολο να τροποποιηθεί χωρίς να έχουμε λεπτομερή γνώση του τρόπου λειτουργίας του προγράμματος. Αυτό αντισταθμίζεται από το ότι οι διεργασίες του επόμενου κεφαλαίου αποδεσμεύονται από την προσομοίωση, ακριβώς χάρις σε αυτή τη διεπαφή, και αυτές είναι βέβαια οι διεργασίες που προσφέρονται για τροποποίηση και για την ευχερέστερη μελέτη των διαφορετικών πολιτικών διαχείρισης των πόρων του συστήματος.

3.2.1. Πυρήνας

3.2.1.1. Εισαγωγή

Βασικός σκοπός του πυρήνα είναι η παροχή της εικονικής μηχανής στις διεργασίες. Οι λειτουργίες του πυρήνα παρέχουν τους μηχανισμούς μέσω των οποίων υλοποιούνται (στα υψηλότερα επίπεδα) οι πολιτικές χρήσης των πόρων. Έτσι, ο πυρήνας διαχειρίζεται τις διακοπές, υλοποιεί τις πρωτογενείς κλήσεις και την επικοινωνία μεταξύ διεργασιών και διανέμει τον έλεγχο στις διεργασίες.

Στο σύστημά μας, οι διεργασίες ομαδοποιούνται σε δένδρα διεργασιών, τα οποία σχετίζονται με μία εργασία. Στη ρίζα του δένδρου, έχουμε τον οργανωτή της εργασίας. Η επικοινωνία μεταξύ διεργασιών είναι ελεύθερη και οι σχέσεις πατέρα παιδιού δεν περιορίζουν τους πόρους που καταχωρούνται, την επικοινωνία ή τις προτεραιότητες που δίνονται στα παιδιά. Η βασική χρησιμότητα της ομαδοποίησης είναι η λογιστική χρέωση και η ευκολία στη δημιουργία και διαγραφή των διεργασιών από τις διεργασίες που τις δημιουργούν. Το δένδρο που χρησιμοποιείται στην πραγματικότητα είναι περιορισμένο σε ένα επίπεδο παιδιών μόνο, αλλά μπορεί να επεκταθεί σε περισσότερα επίπεδα. Οι διεργασίες στο σύστημα μπορεί να είναι έτοιμες

(εκτελούμενες ή περιμένοντας για εκτέλεση), σταματημένες ή εμποδισμένες. Οι καταστάσεις αυτές παρουσιάζονται κατά την εκτέλεση της διεργασίας, ενώ έχουμε και ειδικές καταστάσεις κατά τη δημιουργία και αναμονή για διαγραφή της κάθε διεργασίας.

Οι πληροφορίες κατάστασης (περιβάλλον) της διεργασίας τηρούνται στον περιγραφητή της. Εκεί αποθηκεύονται όλα τα στοιχεία που σχετίζονται με αυτή. Όλοι οι περιγραφητές βρίσκονται σε έναν καθολικής χρήσης πίνακα, συνδεδεμένοι σε λίστα διπλής κατεύθυνσης. Οι διεργασίες ταυτοποιούνται από τη θέση τους στον πίνακα περιγραφητών και από έναν αριθμό δημιουργίας (στο σύστημα). Προσέξτε ότι για να κρατήσουμε τα στοιχεία των εκτιμήσεων της προσομοίωσης για τις απαιτήσεις των διεργασιών, χρησιμοποιούμε τον πίνακα σκιάς των διεργασιών.

Η επικοινωνία στο σύστημά μας γίνεται μέσω μεταβίβασης μηνυμάτων, τα οποία είναι ατομικά (δεν απαιτούν απαντήσεις) και στέλνονται μέσω θυρών. Κάθε διεργασία έχει ένα πλήθος θυρών στη διάθεσή της, τις οποίες συνδέει με θύρες άλλων διεργασιών. Έτσι, μέσω μίας θύρας έχουμε ένα μονοπάτι σύνδεσης με άλλες διεργασίες, μέσω του οποίου μπορεί να γίνει ανταλλαγή μηνυμάτων. Οι θύρες ορίζονται είτε κατά τη δημιουργία των διεργασιών, είτε με ρητή πρωτογενή κλήση, είτε όταν λαμβάνονται μηνύματα από διεργασίες που ζητούν επικοινωνία. Η ίδια η επικοινωνία γίνεται μέσω μίας κλήσης για αποστολή μηνύματος, η οποία δεν εμποδίζει τον αποστολέα, αλλά μπορεί να αφυπνίσει τον παραλήπτη, και μίας κλήσης για λήψη μηνύματος, η οποία σε περίπτωση αδυναμίας εύρεσης κατάλληλου μηνύματος, εμποδίζει τον παραλήπτη και γυρνάει πίσω το μετρητή του για να επαναληφθεί η κλήση, αν δεν έχει βρεθεί μήνυμα μέχρι την επόμενη ενεργοποίηση του παραλήπτη. Επιπλέον, η αφύπνιση μπορεί να γίνει μέσω ενός ιδιωτικού σηματοφορέα για κάθε διεργασία. Σε αυτή την περίπτωση, χρησιμοποιείται η κλήση αναμονής για γεγονός (receive_event), που αφυπνίζει τον παραλήπτη είτε όταν δεχθεί ενεργοποίηση στο σηματοφορέα, είτε όταν δεχθεί μήνυμα από μία από τις θύρες στις οποίες "ακούει".

Η διαχείριση των διακοπών, γίνεται από ένα τμήμα του κώδικα το οποίο φροντίζει για την κλήση των ρουτινών εξυπηρέτησης. Κατά την ολοκλήρωση της εξυπηρέτησης, μπορεί να έχει εμποδιστεί η τρέχουσα διεργασία ή να έχει αφυπνιστεί κάποια διεργασία με μεγαλύτερη προτεραιότητα, αφού οι διακοπές μπορεί να έχουν σαν αποτέλεσμα την έκδοση πρωτογενών κλήσεων. Στο σύστημά μας, έχουμε ομαδοποιήσει στο ίδιο επίπεδο τις εσωτερικές διακοπές (λογισμικού) και τις εξωτερικές διακοπές (υλικού). Η χρέωση κατά την εξυπηρέτηση των διακοπών, εξαρτάται από το αν η διακοπή οφείλεται στο χρήστη (εξυπηρέτηση) ή όχι, με αποτέλεσμα ο χρήστης να χρεώνεται μόνο για τις πραγματικές απαιτήσεις του και όχι για την επιβάρυνση του συστήματος.

Ο διανομέας, αποφασίζει σε κάθε ενεργοποίησή του ποια διεργασία θα εκτελεστεί στη συνέχεια. Για να το κάνει αυτό χρησιμοποιεί μία ταξινομημένη ουρά διεργασιών (λίστα διανομέα), την οποία προετοιμάζει (ταξινομεί) ο μεσοχρόνιος χρονοπρογραμματιστής της ΚΜΕ. Έτσι, ο διανομέας απλά διασχίζει τη λίστα από την κεφαλή, μέχρι να βρει μία διεργασία που να μπορεί να εκτελεστεί. Ο διανομέας καλείται όταν έχουμε αλλαγές στην κατάσταση των διεργασιών (εμποδισμός τρέχουσας ή αφύπνιση μίας διεργασίας με υψηλότερη προτεραιότητα), πράγμα που συμβαίνει μετά από εξωτερικές διακοπές που επηρεάζουν την κατάσταση των διεργασιών, μετά από πρωτογενείς κλήσεις που εμποδίζουν ή σταματούν τον καλούντα (ή ίσως απελευθερώνουν μία άλλη διεργασία) και μετά από διακοπές χρονομέτρου. Αν χρειαστεί, ο διανομέας μεταφέρει τον έλεγχο σε μία νέα διεργασία, αποθηκεύοντας την κατάσταση της προηγούμενης εκτελούμενης διεργασίας στον περιγραφητή της.

Ο κώδικας για τον πυρήνα βρίσκεται μέσα στο αρχείο *kernel.c*, στο τελευταίο τμήμα της συνάρτησης *main*. Στη συνάρτηση αυτή περιλαμβάνονται και άλλα τμήματα του συστήματος, για λόγους αποδοτικότητας, όπως οι διεργασίες χρήστη και ο κώδικας της αρχικής φόρτωσης του συστήματος.

3.2.1.2. Δομές Δεδομένων

Οι διεργασίες ταυτοποιούνται όπως είπαμε με δύο αριθμούς. Το αναγνωριστικό μίας διεργασίας είναι τύπου *tchr* (defs.h) με τα πεδία:

int pnumber: αριθμός δημιουργίας της διεργασίας (μοναδικός)
int pdslot: θέση της διεργασίας στον πίνακα περιγραφητών των διεργασιών

Η διπλή αρίθμηση είναι χρήσιμη για τον άμεσο εντοπισμό της διεργασίας στον πίνακα περιγραφητών και για την αποφυγή σύγχυσης διεργασιών που έτυχε να βρίσκονται στην ίδια θέση στον πίνακα αυτό σε διαφορετικές χρονικές στιγμές (πολύ χρήσιμο σε περίπτωση που μένουν ανεπίδοτα μηνύματα στο σύστημα).

Τα μηνύματα στο σύστημά μας είναι δομές τύπου *msg* (defs.h). Τα πεδία της δομής αυτή είναι:

int mport: θύρα αποστολής (επιθυμητή) ή θύρα λήψης (κατά τη λήψη παίρνει την τιμή της θύρας που χρησιμοποιήθηκε).
int mcommand: λειτουργία που θέλουμε να εκτελεστεί (εξαρτάται από τις διεργασίες που επικοινωνούν).
unsigned int portmask: χάρτης ψηφίων για τις θύρες από τις οποίες δεχόμαστε μηνύματα. Μετά τη λήψη μηνύματος παίρνει την τιμή 00...00 (δυαδικό), ενώ

μετά από λήψη ενεργοποίησης την τιμή 11...11 (δυαδικό). Για να ελέγχουμε αυτή την περίπτωση, έχουμε τη σταθερά ACTIVATED που είναι ίση με την τιμή της ενεργοποίησης (αρχείο defs.h). Επειδή έχουμε οκτώ θύρες ανά διεργασία, χρησιμοποιούνται μόνο τα οκτώ υψηλότερα bits.
tchr mpid: ταυτότητα (συνήθως αρχικού) αποστολέα του μηνύματος.

Τα υπόλοιπα πεδία εξαρτώνται πλήρως από την εφαρμογή και είναι:

int word3, tchr words45, int word6, int word7, void *word8.

Το πλήθος θυρών ανά διεργασία ορίζεται από τη σταθερά no_of_ports (defs.h), και οι θύρες αριθμούνται από το μηδέν. Επιπλέον, στο αρχείο vardefs.h ορίζονται συντομογραφίες των θυρών (για το χάρτη ψηφίων). Η ports[i] είναι ένας απρόσημος ακέραιος ίσος με τη μάσκα λήψης μηνυμάτων από τη θύρα i. Η ports0to[i] είναι το ίδιο, για όλες τις θύρες από 0 έως i. Τέλος, η μεταβλητή message τύπου msg είναι ένα καθολικό μήνυμα (γενικής χρήσης).

Η κάθε διεργασία έχει μία ουρά από μηνύματα, τα οποία είναι συνδεδεμένα σε λίστα διπλής κατεύθυνσης. Οι ουρές αυτές έχουν στοιχεία τύπου *queue* (defs.h), με πεδία:

msg mess: το μήνυμα της καταχώρησης.
int priority: προτεραιότητα στην ουρά (ίση με την προτεραιότητα της διεργασίας αποστολέα).
float tin: ώρα εισαγωγής στην ουρά.
tchr sender: αναγνωριστικό του αποστολέα.
queue* link, backlink: σύνδεσμοι δύο κατευθύνσεων στην ουρά.

Ο εντοπισμός των ουρών γίνεται από τα πεδία του περιγραφητή της διεργασίας, όπως θα δούμε παρακάτω.

Για την κατάσταση των διεργασιών, έχουμε έναν *πίνακα περιγραφητών των διεργασιών (process descriptor table, PDT)*, στον οποίο κρατάμε ένα στοιχείο για κάθε διεργασία που βρίσκεται στο σύστημα. Κάθε τέτοιο στοιχείο (περιγραφητής διεργασίας) είναι μία δομή τύπου *pdtstruct* (defs.h) με τα ακόλουθα πεδία:

tchr pdid: αναγνωριστικό της διεργασίας αυτής.
int pdmsec: θέση τεμαχίου μηδέν στον sdt (και κεφαλή της λίστας τεμαχίων της διεργασίας στον sdt).
user_process *process: δείκτης στον πίνακα σκιάς (στο στοιχείο για τη διεργασία αυτή).

int suspension point: σημείο εισόδου στη διεργασία (όταν ενεργοποιηθεί ξανά, το switch θα μεταφέρει τον έλεγχο στην ανάλογη ετικέτα).
int suspended: ψηφίο αναστολής κατά την εκτέλεση.
int page_accessing: τρέχον τεμάχιο της διεργασίας.
tchr port[no_of_ports+1]: περιγραφητές των διεργασιών που συνδέονται με την κάθε θύρα (-1 και -1 σε περίπτωση αόριστης θύρας).
unsigned int pdmam: χάρτης ψηφίων για τις θύρες από όπου αναμένουμε μηνύματα.
queue *message_head, *message_tail: κεφαλή και ουρά της λίστας μηνυμάτων της διεργασίας.
int blocked: κατάσταση εμποδισμού της διεργασίας, παίρνει τις τιμές

BL_READY	έτοιμη
BL_SECTION	εμποδισμένη για τεμάχιο (σφάλμα)
BL_KILLED	εμποδισμένη μέχρι να θανατωθεί
BL_MESSAGE	εμποδισμένη για μήνυμα
BL_EVENT	εμποδισμένη για μήνυμα ή ενεργοποίηση (γεγονός)
BL_CREATION	εμποδισμένη μέχρι να δημιουργηθεί
BL_TERMINAL	εμποδισμένη για αλληλεπίδραση
BL_SON	εμποδισμένη μέχρι θάνατο παιδιού

int stopped: κατάσταση σταματήματος της διεργασίας, παίρνει τις τιμές

ST_READY	έτοιμη
ST_WORKING_SET	δεν χωράει το σύνολο εργασίας της στη μνήμη
ST_SECTION_REMOVED	περιμένει απομάκρυνση ενός τεμαχίου της

double pdtrun, pdtrdy, pdtblk, pdtunb: χρόνοι παραμονής σε εκτέλεση, σε ετοιμότητα, σε εμποδισμένη κατάσταση, σε απελευθερωμένη κατάσταση.
double oldrun: χρόνος εκτέλεσης μέχρι την αρχή της πλέον πρόσφατης αλληλεπίδρασης της διεργασίας.
int in_core: ψηφίο παρουσίας στη μνήμη (τουλάχιστον ένα τεμάχιο της).
int activated: ιδιωτικός σηματοφορέας της διεργασίας.
long int ws_size: μέγεθος συνόλου εργασίας σε bytes.
float pddisp: κρίσιμος χρόνος διεργασίας.
double time_lastcs: χρόνος τελευταίας αλλαγής κατάστασης.
tchr pdsonptr, pdbrotherptr, pdbackptr, pddl, pdpid: αναγνωριστικό παιδιού, αδελφού, προηγούμενης και επόμενης διεργασίας στη λίστα του διανομέα και αναγνωριστικό πατέρα.
float resource: συνολική χρήση πόρων.
double start_interaction: έναρξη τελευταίας αλληλεπίδρασης.
double oldpdtrun: χρόνος εκτέλεσης μέχρι το προηγούμενο μεσοδιάστημα χρονοπρογραμματισμού.
int proctype: τύπος διεργασίας.

int sends[no_of_system_processes+1]: πλήθος μηνυμάτων από τις διεργασίες του συστήματος.
int no_of_sends[no_of_system_processes+1]: πλήθος μηνυμάτων για την τελευταία αλληλεπίδραση.

Όλοι οι περιγραφητές βρίσκονται στον πίνακα pdt, με δήλωση pdtstruct pdt[pdt_size+1] (vardefs.h), με το pdt_size να ορίζεται στο defs.h. Επίσης, έχουμε τον ακέραιο pdhead που δείχνει στην κεφαλή της λίστας του διανομέα και τον ακέραιο tail που δείχνει στην τελευταία θέση της λίστας του διανομέα.

Για το περιβάλλον του πυρήνα έχουμε τη μεταβλητή kernel_entry (vardefs.h), τύπου jmp_buf, η οποία αποθηκεύει το περιβάλλον για τις μη τοπικές κλήσεις (goto). Η ακέραια μεταβλητή flag χρησιμοποιείται μόνο για την πρώτη κλήση της εισόδου kernel από το ύψος της enter.

3.2.1.3. Αλγόριθμοι

3.2.1.3.1. Βοηθητικές συναρτήσεις

Οι συναρτήσεις *block*, *unblock*, *suspend* και *resume*, καλούνται αντίστοιχα κατά τον εμποδισμό, απελευθέρωση, αναστολή και επανεκκίνηση της διεργασίας που τους δίνεται σαν παράμετρος (int slot). Μαζί με την ενημέρωση των χρόνων στις κατάλληλες καταστάσεις, ενημερώνεται και ο χρόνος της τελευταίας αλλαγής κατάστασης της διεργασίας. Η συνάρτηση *activate* ενεργοποιεί τη διεργασία που της δίνεται σαν παράμετρος (int slot), ενεργοποιώντας τον ιδιωτικό της σηματοφορέα, αν δεν έχει ήδη ενεργοποιηθεί. Αν η διεργασία περιμένει γεγονός, απελευθερώνεται και σημειώνονται οι χρόνοι που παρέμεινε εμποδισμένη.

Η συνάρτηση *death_sentence* παίρνει σαν παράμετρο τη θέση μίας διεργασίας (int slot) την οποία θέλουμε να θανατώσουμε. Η διεργασία εμποδίζεται (αν δεν έχει εμποδιστεί ήδη) περιμένοντας να θανατωθεί. Αν ήταν αλληλεπιδραστική, ενημερώνουμε το μέσο χρόνο απόκρισης και αυξάνουμε το πλήθος αλληλεπιδράσεων στο τελευταίο μεσοδιάστημα. Ανεξαρτήτως τύπου, καταστρέφουμε την εσωτερική της λίστα γεγονότων, διασχίζοντάς την και απελευθερώνοντας το χώρο των στοιχείων της.

Η συνάρτηση *dummy_send* στέλνει ένα μήνυμα στη διεργασία με θέση slot (τύπου int) στον pdt. Η συνάρτηση αυτή υλοποιεί την πραγματική αποστολή μηνυμάτων μεταξύ διεργασιών και είναι αυτή που καλείται μέσα από τον πυρήνα (σε αντίθεση με το γεγονός / πρωτογενή κλήση send_message). Αρχικά, δημιουργούμε μία θέση σε ουρά μηνυμάτων, συνδέοντας το μήνυμα με την κατάλληλη ουρά και ορίζοντας τον αποστολέα

(τρέχουσα διεργασία), το χρόνο αποστολής (τρέχοντας) και την προτεραιότητα του μηνύματος. Αν πηγαίνει σε διεργασία συστήματος, καλούμε τη συνάρτηση *in_statistics* για παρακολούθηση στατιστικών. Παρατηρήστε ότι η προτεραιότητα είναι αντίστροφη του κρίσιμου χρόνου. Το μήνυμα εισάγεται στο τέλος της ουράς μηνυμάτων της διεργασίας και συνδέεται και προς τις δύο κατευθύνσεις.

Στη συνέχεια, αν το μήνυμα πηγαίνει στο σύστημα τερματικών, χωρίς να έρχεται από τον πυρήνα, έχουμε τέλος αλληλεπίδρασης με το τερματικό, οπότε ενημερώνουμε το χρόνο απόκρισης τερματικών και το πλήθος αλληλεπιδράσεων. Αν το μήνυμα έρχεται από το διαχειριστή τερματικών, έχουμε αρχή αλληλεπίδρασης με το τερματικό. Ο χρόνος της αρχής αλληλεπίδρασης ενημερώνεται και οι τρέχουσες αποστολές μηδενίζονται, ενώ ο χρόνος εκτέλεσης μέχρι την προηγούμενη αλληλεπίδραση γίνεται ο συνολικός χρόνος εκτέλεσης ως τώρα. Αν η διεργασία ήταν εμποδισμένη για αλληλεπίδραση, απελευθερώνεται και επιστρέφουμε στον καλούντα. Σε κάθε περίπτωση, αν δεν έχουμε φύγει από τη συνάρτηση, ελέγχουμε αν ο παραλήπτης περίμενε για μήνυμα, γεγονός, θάνατο παιδιού ή τερματικό. Αν ναι, τότε ελέγχουμε όλες τις θύρες του για να δούμε μήπως περιμένει μήνυμα από μία θύρα και η θύρα αυτή είναι είτε αόριστη (οπότε θα οριστεί στη συνέχεια), είτε δείχνει στον αποστολέα ήδη. Αν ισχύει ένα από τα παραπάνω, για τις διεργασίες αλληλεπίδρασης που περιμένουν θάνατο παιδιού αλλάζει ο χρόνος έναρξης της αλληλεπίδρασης. Τέλος, αν το μήνυμα μπορεί να γίνει δεκτό, η διεργασία απελευθερώνεται και επιστρέφουμε. Όλες οι συναρτήσεις αυτές βρίσκονται στο kernel.c.

3.2.1.3.2. Είσοδος στον πυρήνα

Ο κώδικας του πυρήνα αρχίζει από την ετικέτα *kernel*. Αρχικά, βεβαιωνόμαστε ότι έχουμε αποθηκευμένες τις πληροφορίες περιβάλλοντος του πυρήνα στο σημείο αυτό, καλώντας ξανά την setjmp αν επιστρέφουμε από μη τοπικό goto (το αν επιστρέφουμε από μη τοπικό goto ή μετά από την κλήση της setjmp φαίνεται από το αποτέλεσμα της setjmp). Στη συνέχεια, εξετάζουμε τη μεταβλητή flag για να δούμε αν πρέπει να επιστρέψουμε στην ετικέτα *enter* (την πρώτη φορά που καλούμε την kernel, απλά αποθηκεύουμε το περιβάλλον και επιστρέφουμε στην enter). Παρατηρήστε ότι η κάθε κλήση της setjmp επιστρέφει δύο φορές, την πρώτη φορά με το περιβάλλον αποθηκευμένο και τη δεύτερη μετά από το μη τοπικό goto (επιστρέφουμε δηλαδή στο περιβάλλον που αποθηκεύσαμε). Στη δεύτερη περίπτωση, πριν προχωρήσουμε, αποθηκεύουμε ξανά το περιβάλλον, για να επιστρέψουμε εκεί με το επόμενο μη τοπικό goto.

Στη συνέχεια, εξετάζουμε την κεφαλή της λίστας των γεγονότων του συστήματος. Αν έχουμε υπερβεί το χρόνο προσομοίωσης ή αν το σύστημα είναι άδειο (κενή λίστα διακοπών, ένα μόνο γεγονός και δεν υπάρχουν διεργασίες χρήστη), εκτυπώνουμε τα στατιστικά στοιχεία (κλήση της plot_statistics), καταστρέφουμε το γεγονός και εγκαταλείπουμε το πρόγραμμα (κανονικός τερματισμός, με κωδικό επιστροφής 0). Αλλιώς, ελέγχουμε αν πρέπει να γίνει η προγραμματισμένη εκτύπωση των στατιστικών (ο χρόνος της επόμενης εκτύπωσης στην plot_time), και αν ναι, τυπώνουμε τα στοιχεία και προχωράμε το χρόνο plot_time κατά statistics_period, ορίζοντας τον επόμενο χρόνο εκτύπωσης στατιστικών. Αποθηκεύουμε τα στοιχεία του τρέχοντος γεγονότος σε καθολικές μεταβλητές και προχωράμε το χρόνο στο χρόνο του γεγονότος. Απελευθερώνουμε το χώρο της κεφαλής της λίστας και αν η ΚΜΕ ενεργοποιείται μετά από αδράνεια, ενημερώνουμε το συνολικό χρόνο αδράνειας του συστήματος. Αν η ΚΜΕ ήταν σε χρήση, σε περίπτωση που το γεγονός είναι διακοπή του υλικού, αναστέλλουμε τη διεργασία που χρησιμοποιούσε την ΚΜΕ. Αυτό γίνεται με το να βρούμε το επόμενο γεγονός της στη λίστα γεγονότων και αν είναι διεργασία του συστήματος να το εισάγουμε στη λίστα διακοπών, δημιουργώντας ένα νέο τέτοιο στοιχείο (διακοπής), το οποίο έχει τα ίδια στοιχεία με το εξωτερικό γεγονός, εκτός από το χρόνο, ο οποίος είναι ο χρόνος που απομένει ως το γεγονός. Η λίστα είναι LIFO, για να έχουμε φωλιασμένες διακοπές. Αν ήταν διεργασία χρήστη, το γεγονός μεταφέρεται πάλι στη λίστα γεγονότων της διεργασίας (εσωτερική), σημειώνοντας αν το γεγονός αυτό προκάλεσε διακοπή (πεδίο suspension_marker, τίθεται αν η διεργασία δεν έχει ήδη ανασταλεί) και χρησιμοποιώντας τον μετασχηματισμό εξωτερικού σε εσωτερικό χρόνο της διεργασίας. Ο χρόνος εκτέλεσης που απομένει στη διεργασία αυξάνεται και το γεγονός επανεισάγεται στην εσωτερική λίστα της διεργασίας. Εδώ, η εισαγωγή είναι LIFO για στοιχεία με ίδιους χρόνους, έτσι ώστε κατά την νέα δρομολόγηση να εκτελούνται πρώτα τα γεγονότα που έχουν ανασταλεί (είχαν ξαναδρομολογηθεί). Τελικά, ανεξαρτήτως τύπου, η διεργασία σημειώνεται ως υπό αναστολή και ενημερώνονται οι σχετικοί χρόνοι. Το γεγονός που βρήκαμε διαγράφεται από την εξωτερική λίστα γεγονότων (έχει ήδη μεταφερθεί στην κατάλληλη λίστα) και προχωράμε στην εξυπηρέτηση της διακοπής, χρησιμοποιώντας την εντολή switch.

3.2.1.3.3. Εξωτερικές διακοπές

Γενικά αυτές οι είσοδοι χρεώνουν τον πυρήνα και επειδή μπορεί να αφυπνίσουν μία διεργασία υψηλής προτεραιότητας, καταλήγουν στο διανομέα. Η TIMER_INTERRUPT καλείται μετά από διακοπή χρονομέτρου και προγραμματίζει την επόμενη διακοπή, χρησιμοποιώντας το μεσοδιάστημα των διακοπών σαν χρόνο. Ενημερώνεται ο χρόνος επιβάρυνσης του πυρήνα και ο τρέχων χρόνος, ενεργοποιείται ο διανομέας και τελικά του μεταβιβάζεται ο

έλεγχος. Η ARRIVAL καλείται μετά από άφιξη χρήστη σε τερματικό ή, σε περίπτωση που έχουμε spooling, μετά από άφιξη μίας εργασίας δεσμίδων (η απουσία spooling σημαίνει ότι δεν μπορούμε να δεχόμαστε εργασίες δεσμίδων παρασκηνιακά μέσω του spooler εισόδου). Αφού ενημερωθεί ο χρόνος του συστήματος και ο χρόνος επιβάρυνσης, αν η εργασία είναι δεσμίδων, ενεργοποιείται ο χρονοπρογραμματιστής εργασιών, ενώ, αλλιώς, στέλνουμε ένα μήνυμα στο σύστημα τερματικών για να ενεργοποιηθεί (mcommand=3). Τελικά, πηγαίνουμε στο διανομέα. Η TRANSFER_COMPLETION_INTERRUPT καλείται όταν ολοκληρώνεται η ε/ε. Αν η μεταφορά είναι από/προς συσκευή, η συσκευή απελευθερώνεται και ενημερώνεται ο χρόνος τελευταίας χρήσης της. Ενημερώνεται ο χρόνος του συστήματος και ο χρόνος επιβάρυνσης και αποστέλλουμε ένα μήνυμα επιτυχούς ολοκλήρωσης (mcommand=1) στην κατάλληλη συσκευή (μπορεί να είναι και το σύστημα τερματικών). Τελικά καταλήγουμε στο διανομέα. Από εκεί θα ενεργοποιηθεί ο κατάλληλος διαχειριστής που θα δρομολογήσει και την επόμενη απαίτηση ε/ε χρησιμοποιώντας την πρωτογενή κλήση initiate_io, η οποία θα αντιμετωπιστεί τελικά πάλι από τον πυρήνα που εκδίδει ο ίδιος τις εντολές ε/ε.

3.2.1.3.4. Εσωτερικές διακοπές (παγίδες)

Οι εσωτερικές διακοπές οφείλονται στην εκτέλεση μίας διεργασίας χρήστη και έχουν σαν αποτέλεσμα τον εμποδισμό της. Έτσι, έχουν έξοδο προς το διανομέα. Η χρέωση γίνεται στη διεργασία, εκτός από την fault_sure η οποία οφείλεται στη συμπεριφορά του συστήματος. Η FAULT καλείται και άμεσα (σαν είσοδος fault_now) και ενημερώνει πρώτα το τρέχον τεμάχιο της διεργασίας, σημειώνοντας ότι χρησιμοποιείται και ανήκει στο σύνολο εργασίας της (το τεμάχιο στο word3). Αν το τεμάχιο βρίσκεται στη μνήμη, επιστρέφουμε στην enter1 (δεν ξαναμπαίνουμε στη διεργασία, η κλήση ήταν άσκοπη, λόγω δρομολόγησης που δεν χρειάστηκε τελικά). Αν υπάρχει πραγματικά σφάλμα, μπαίνουμε στην είσοδο fault_sure, η οποία επειδή οφείλεται στην πολιτική του συστήματος, χρεώνεται στον πυρήνα. Αφού ενημερώσουμε το χρόνο προσομοίωσης, εμποδίζουμε τη διεργασία για τεμάχιο και στέλνουμε μήνυμα ανάγνωσης τεμαχίου (mcommand=2) για τη διεργασία αυτή (στο mpid) στο διαχειριστή της μνήμης, πριν προχωρήσουμε στο διανομέα.

Η HALT και η ABORT1, καθώς και η είσοδος από άμεση κλήση abort_now, τερματίζουν τη διεργασία. Αφού ενημερώσουμε το χρόνο του συστήματος, καλούμε τη συνάρτηση death_sentence για τη διεργασία και θανατώνουμε και όλα τα παιδιά της (με τον ίδιο τρόπο). Τελικά, ενημερώνουμε το γονέα της για το θάνατό της (αναγνωριστικό στο mpid, θανάτωση στο mcommand=1) και στέλνουμε την απάντηση χρησιμοποιώντας

το αναγνωριστικό του πατέρα της από το πεδίο στον pdt (η αιτία θανάτου στα word3 και word6). Τελικά, βγαίνουμε προς το διανομέα.

3.2.1.3.5. Πρωτογενείς κλήσεις

Για όλες αυτές τις εισόδους, έχουμε εξυπηρέτηση του χρήστη, ο οποίος την χρεώνεται. Η έξοδος ποικίλλει (διανομέας ή είσοδος στην ίδια διεργασία). Η SEND, καλείται μετά από αποστολή μηνύματος. Αν ο παραλήπτης δεν υπάρχει πια, επιστρέφουμε στο σημείο enter της διεργασίας που έκανε την κλήση. Αλλιώς, ενημερώνουμε το χρόνο του συστήματος, αναστέλλουμε τον αποστολέα και στέλνουμε το μήνυμα στον παραλήπτη. Τελικά βγαίνουμε στο διανομέα. Η RECE καλείται για λήψη γεγονότος. Αν έχει ήδη γίνει ενεργοποίηση της διεργασίας, ορίζεται η portmask ώστε να δείχνει αφύπνιση λόγω ενεργοποίησης, καθαρίζεται ο σηματοφορέας και προχωράμε το χρόνο κατάλληλα (λαμβάνουμε υπόψη ότι δεν εξετάστηκε η ουρά μηνυμάτων), πριν επιστρέψουμε στη διεργασία που εκτελείται (enter). Αν ο σηματοφορέας δεν έχει ενεργοποιηθεί, η διεργασία εμποδίζεται για γεγονός και προχωράμε στην είσοδο RECM για να εξετάσουμε αν υπάρχει κατάλληλο μήνυμα για παραλαβή.

Η RECM καλείται μετά από κλήση για λήψη μηνύματος και αφού ενημερώσει το χρόνο του συστήματος, διασχίζει την ουρά μηνυμάτων της διεργασίας. Για κάθε μήνυμα, εξετάζει αν περιμένουμε μήνυμα από μία θύρα, η οποία να ταυτίζεται με τον αποστολέα του μηνύματος ή που να είναι αόριστη. Αν συμβαίνει το τελευταίο, ορίζουμε τη θύρα και το μήνυμα παραδίδεται στη διεργασία σημειώνοντας τη θύρα πηγής. Αν ο παραλήπτης είναι διεργασία του συστήματος, καλείται η *out_statistics*, ενώ αν είναι διεργασία χρήστη και ο αποστολέας είναι το σύστημα, ενημερώνουμε τους πίνακες λήψης μηνυμάτων από το σύστημα. Στη συνέχεια, ορίζουμε ότι η αφύπνιση προήλθε από λήψη μηνύματος και απομακρύνουμε το μήνυμα από την ουρά, καταστρέφουμε την καταχώρησή του και απελευθερώνουμε τη διεργασία, πριν επιστρέψουμε στη διεργασία (enter). Αν αφού διατρέξουμε τη λίστα δεν βρεθεί κατάλληλο μήνυμα, η διεργασία εμποδίζεται. Αν εμποδίζεται τώρα, ελέγχουμε αν περίμενε μήνυμα από το σύστημα τερματικών μέσω της θύρας 2 (αυτό γίνεται στις αλληλεπιδράσεις των χρηστών), οπότε εμποδίζεται για αλληλεπίδραση. Αλλιώς, εμποδίζεται για μήνυμα. Αν είναι ήδη εμποδισμένη και περιμένει μήνυμα από κάποια διεργασία χρήστη, εμποδίζεται για θάνατο παιδιού και αν είναι αλληλεπιδραστική αναστέλλεται προσωρινά η αλληλεπίδρασή της. Τελικά, αν η διεργασία έχει εμποδιστεί (πιθανά και για γεγονός) ο χάρτης ψηφίων των θυρών αποδοχής ορίζεται από το portmask. Το σημείο εισόδου ορίζεται ως η πρώτη παράμετρος της κλήσης receive_message (αποθηκεύεται στο mport του μηνύματος), έτσι ώστε η κλήση να επαναληφθεί στο μέλλον. Τελικά, βγαίνουμε στο διανομέα.

Η είσοδος START καλείται για να ξεκινήσει μία διεργασία. Το ψηφίο σταματημένης κατάστασης καθαρίζεται, ο χρόνος προχωράει, η τρέχουσα διεργασία ενημερώνεται για πιθανή αναστολή και βγαίνουμε στο διανομέα. Η είσοδος STOP σταματάει μία διεργασία, σημειώνοντας την αιτία της διακοπής (από το word3), προχωρώντας το χρόνο και επιστρέφοντας στην είσοδο της διεργασίας που εκτελείται (enter). Η είσοδος DESP καλείται για καταστροφή ενός μονοπατιού. Αφού προχωρήσει το χρόνο του συστήματος, κάνει τη θύρα αόριστη και επιστρέφει στην είσοδο της διεργασίας (enter). Η CREP δημιουργεί το μονοπάτι, προχωρώντας το χρόνο και ορίζοντας την κατάλληλη θύρα από το mpid. Τελικά, βγαίνει στην είσοδο της διεργασίας που εκτελούνταν (enter). Η ELSZ αλλάζει το μέγεθος του συνόλου καταλλήλων διεργασιών, σύμφωνα με το word3 και αφού προχωρήσει το χρόνο του συστήματος επιστρέφει στη διεργασία (enter).

Η τελευταία πρωτογενής κλήση είναι για την είσοδο INITIO, που καλείται για να ξεκινήσει μεταφορές ε/ε. Αρχικά προχωράμε το χρόνο και αν η συσκευή δεν είναι το τερματικό, τη δεσμεύουμε, ενημερώνουμε το χρόνο αδράνειάς της και υπολογίζουμε το χρόνο ολοκλήρωσης της μεταφοράς προσθέτοντας το χρόνο αναζήτησης, την περιστροφική καθυστέρηση και το χρόνο μεταφοράς επί το μέγεθος της μεταφοράς (στο word7). Παρατηρήστε ότι αυτό δουλεύει και για συσκευές χωρίς για παράδειγμα περιστροφική καθυστέρηση, αν μηδενίσουμε τα κατάλληλα πεδία του περιγραφητή της συσκευής. Αν η συσκευή είναι το τερματικό, εκτιμούμε το χρόνο ολοκλήρωσης χρησιμοποιώντας το χρόνο σκέψης της διεργασίας (χρόνος κατά τον οποίο ο χρήστης "σκέφτεται", δηλαδή μεταξύ των εντολών που δίνει από το τερματικό ή μεταξύ των αλληλεπιδράσεων). Σε κάθε περίπτωση, δρομολογούμε το γεγονός της ολοκλήρωσης ε/ε και επιστρέφουμε στη διεργασία (enter).

3.2.1.3.6. Διανομέας

Ο διανομέας ξεκινάει στην ετικέτα dispatcher και καλείται όποτε υπάρχει υπόνοια ή βεβαιότητα ότι πρέπει να αλλάξει ο κάτοχος της ΚΜΕ. Αυτό προκαλείται είτε από εμποδισμό της τρέχουσας διεργασίας, είτε επειδή αφυπνίζεται κάποια διεργασία με μεγαλύτερη προτεραιότητα. Το δεύτερο σημαίνει ότι έχουμε διακοπτόμενο χρονοπρογραμματισμό ή χρονοπρογραμματισμό με εκτόπιση (preemptive). Ο διανομέας αρχικά δεσμεύει την ΚΜΕ και αρχίζει τη διάσχιση της λίστας του από το pdhead φροντίζοντας να μην εξετάσει περισσότερες διεργασίες από το σύνολο καταλλήλων. Αν η διεργασία δεν είναι εμποδισμένη η σταματημένη, αν είναι διαφορετική από αυτήν που εκτελούνταν προηγουμένως, έχουμε μεταγωγή της κατάστασης του επεξεργαστή και αλλαγή της τρέχουσας διεργασίας. Η διεργασία ξαναρχίζει την εκτέλεση και ο χρόνος προχωράει σύμφωνα με το

χρόνο εισαγωγής σε διεργασία. Αν η διεργασία είναι μία διεργασία χρήστη που έχει ενεργοποιηθεί παλιότερα, σημειώνουμε το τρέχον τεμάχιό της ως χρησιμοποιημένο και μέλος του συνόλου εργασίας της. Αν δεν βρίσκεται στη μνήμη (το τεμάχιο), σημειώνουμε το τεμάχιο στο word3 και καλούμε τη fault_sure για άμεσο σφάλμα τεμαχίου (με βεβαιότητα). Αλλιώς, αν η διεργασία δεν έχει ανασταλεί, πηγαίνουμε στην είσοδο enter1, γλιτώνοντας το χρόνο νέας εισόδου στη διεργασία. Αν η διεργασία είχε ανασταλεί, πρέπει να δούμε τι γίνεται με τα γεγονότα που μετακινήθηκαν λόγω της διακοπής. Αν είναι διεργασία του συστήματος, σταματάει να είναι υπό αναστολή και διασχίζουμε τη λίστα διακοπών μέχρι να βρούμε το επόμενο γεγονός της. Αφού το βρούμε, το επαναφέρουμε στη λίστα γεγονότων της προσομοίωσης και το απομακρύνουμε από τη λίστα διακοπών. Αν η διεργασία ήταν διεργασία χρήστη, παίρνουμε το γεγονός στην κεφαλή της εσωτερικής της λίστας και το δρομολογούμε στη λίστα της προσομοίωσης. Αν είναι σφάλμα τεμαχίου, δρομολογείται ως εξωτερικό σφάλμα, ενώ διαφορετικά, αν είναι αυτό που προξένησε την αναστολή της διεργασίας, η διεργασία σταματάει να είναι υπό αναστολή. Ο χρόνος μετασχηματίζεται και η καταχώρηση στην εσωτερική λίστα καταστρέφεται. Σε κάθε περίπτωση, η επαναδρομολόγηση γίνεται και επιστρέφουμε στην είσοδο του πυρήνα. Αν μέχρι το τέλος της λίστας του διανομέα δεν βρεθεί καμία διεργασία, η ΚΜΕ απελευθερώνεται, σημειώνεται ο χρόνος αδρανοποίησης και επιστρέφουμε στον πυρήνα.

3.2.2. Χρονοπρογραμματιστής εργασιών

3.2.2.1. Εισαγωγή

Ο χρονοπρογραμματιστής εργασιών επιλέγει την επόμενη εργασία που θα ξεκινήσει στο σύστημα και δημιουργεί τη ρίζα του δένδρου (ομάδας) των διεργασιών που θα την εκτελέσουν (τον οργανωτή της εργασίας). Ο χρονοπρογραμματιστής αυτός είναι μακροχρόνιος, αφού επιλέγει τις εργασίες προς ενεργοποίηση από όλες τις εργασίες που περιμένουν για εξυπηρέτηση από το σύστημα. Στο παρακάτω επίπεδο, ο μεσοχρόνιος χρονοπρογραμματιστής αναλαμβάνει τον πολυπρογραμματισμό και τον καταμερισμό των πόρων στις διεργασίες πια. Οι αρμοδιότητες του χρονοπρογραμματιστή εργασιών σταματούν με την ενεργοποίηση της εργασίας, αφήνοντας στα χαμηλότερα επίπεδα να συντονίσουν την εκτέλεση των τεμαχίων της εργασίας (διεργασίες). Έτσι, ο χρονοπρογραμματιστής εργασιών εκτελείται όταν έχουμε άφιξη ή ολοκλήρωση εργασιών, αφού τότε αλλάζουν τα χαρακτηριστικά φόρτωσης του συστήματος. Η παρακολούθηση των εργασιών λοιπόν γίνεται με την ενεργοποίηση, δημιουργία και καταστροφή τους καθώς και με την παραχώρηση και επανάκτηση πόρων προς και από αυτές.

Επειδή γενικά μία εργασία υλοποιείται από πολλές διεργασίες, αρχικά δημιουργούμε μία διεργασία που διερμηνεύει τις εντολές από το τερματικό ή διαβάζει της εντολές της γλώσσας ελέγχου εργασιών και δημιουργεί τις διεργασίες που εκτελούν την επεξεργασία που ζητάει ο χρήστης. Αυτή η διεργασία, γνωστή ως οργανωτής της εργασίας ή φλοιός, αντίστοιχα, αναλαμβάνει τον έλεγχο των διεργασιών μετά τη δημιουργία της από το χρονοπρογραμματιστή εργασιών. Οι οργανωτές διαφέρουν ανάλογα με τον τύπο της εργασίας, όπως και η αντιμετώπιση των εργασιών από το χρονοπρογραμματιστή. Οι βασικές διαφορές είναι ότι οι εργασίες αλληλεπίδρασης γίνονται άμεσα δεκτές ή απορρίπτονται (λόγω φόρτου του συστήματος), ενώ οι εργασίες δεσμίδων εξετάζονται με τη σειρά και μπορεί να περιμένουν μέχρι να ενεργοποιηθούν, καθώς και ότι για τις εργασίες δεσμίδων έχουμε δύο γεγονότα που ενεργοποιούν το χρονοπρογραμματιστή: τέλος εργασίας και τέλος εκτύπωσης αποτελεσμάτων της (το δεύτερο μας ενδιαφέρει όταν θέλουμε να δούμε αν υπάρχουν ελεύθερες περιοχές εξόδου για εργασίες δεσμίδων).

Στο σύστημα, έχουμε μία δομή που κρατάει τα χαρακτηριστικά (περιγραφητές) των εργασιών που εκτελούνται ή περιμένουν ενεργοποίηση. Ο χρονοπρογραμματιστής των εργασιών επικοινωνεί με τους spoolers για την ανάγνωση και εκτύπωση αποτελεσμάτων των εργασιών δεσμίδων, έχοντας αποκλειστική ευθύνη για την κατανομή πόρων στις εργασίες. Έτσι, οι διεργασίες οργανωτές εργασίας δεν ασχολούνται με θέματα spooling και οι spoolers απλά μεταφέρουν εγγραφές από και προς τη μνήμη. Μία άλλη λειτουργία του χρονοπρογραμματιστή εργασιών, είναι να παρακολουθεί τη χρέωση των εργασιών για τους πόρους που χρησιμοποιούν. Επειδή κάθε διεργασία του συστήματος ενημερώνει τα δεδομένα χρέωσης, ο χρονοπρογραμματιστής εργασιών απλά αρχικοποιεί και ολοκληρώνει τις μετρήσεις αυτές.

Η επιλογή των εργασιών προς ενεργοποίηση, μπορεί να έχει στόχους όπως την καλή αξιοποίηση των πόρων του συστήματος, αλλά ταυτόχρονα πρέπει να λαμβάνει υπόψη και τη δυνατότητα του συστήματος να παρέχει εξυπηρέτηση στις εργασίες που εκτελούνται. Αντί να υλοποιηθεί μία περίπλοκη πολιτική επιλογής, απλά περιμένουμε να απελευθερωθούν οι πόροι που ζητάει η πρώτη εργασία από πλευράς προτεραιότητας, οπότε και την ενεργοποιούμε. Οι εκτιμήσεις των διαθέσιμων πόρων (εναλλακτικά, του φόρτου του συστήματος) προέρχονται από άλλες διεργασίες οι οποίες απαντούν σε αιτήσεις του χρονοπρογραμματιστή πάνω στην κατάσταση του συστήματος.

Για τις εργασίες αλληλεπίδρασης, η απόφαση για αποδοχή ή απόρριψη ενός νέου χρήστη λαμβάνεται συνεκτιμώντας την ικανότητα του συστήματος

να εξυπηρετήσει το χρήστη από πλευράς πλήθους χρηστών που χρησιμοποιούν το σύστημα (όριο το πλήθος τερματικών, no_of_terminals στο defs.h) αλλά και την απόδοση του συστήματος (χρόνο απόκρισης), η οποία όταν πέφτει κάτω από κάποιο όριο, απαγορεύει την είσοδο νέων χρηστών. Για τις εργασίες δεσμίδων, έχουμε επίπεδα προτεραιότητας (εξωτερικής) και αποδοχή FCFS σε κάθε επίπεδο. Αφού μία εργασία επιλεγεί, ελέγχεται η δυνατότητα αποδοχής της (αν υπάρχουν πόροι). Αν δεν μπορεί να γίνει δεκτή η εργασία, περιμένει να απελευθερωθούν πόροι, χωρίς να μπορεί να ενεργοποιηθεί μία άλλη εργασία πρώτα.

Τέλος, μία λειτουργία του χρονοπρογραμματιστή που δεν βρίσκεται στο πραγματικό σύστημα, είναι η εκτίμηση των χαρακτηριστικών των εργασιών και των διεργασιών που θα εκτελεστούν στο σύστημα, αφού προσομοιώνουμε το φόρτο εργασίας. Για κάθε νέα εργασία, έχουμε εκτίμηση των χαρακτηριστικών της και κατανομή των απαιτήσεών της στις διεργασίες που θα δημιουργήσει, στον πίνακα σκιάς που σχετίζεται με την εργασία. Με βάση τον πίνακα αυτό, δημιουργούνται και προσομοιώνονται οι πραγματικές διεργασίες στο σύστημα, αντλώντας τα στοιχεία που χρειάζονται για τις στατιστικές εκτιμήσεις από τον αντίστοιχο πίνακα σκιάς.

Ο κώδικας για το χρονοπρογραμματιστή εργασιών διαφέρει μεταξύ του συστήματος με σελιδοποίηση και του συστήματος με τεμαχισμό, λόγω της εκτίμησης του μεγέθους των τεμαχίων της κάθε εργασίας που δημιουργείται. Για το σελιδοποιημένο σύστημα, ο κώδικας βρίσκεται στο αρχείο *psvc.c* ενώ για το σύστημα με τεμαχισμό στο αρχείο *ssvc.c*. Αξίζει να σημειωθεί ότι, εκτός από την εκτίμηση, τα δύο τμήματα κώδικα είναι πανομοιότυπα, χρησιμοποιώντας τους ίδιους αλγόριθμους πολιτικής.

3.2.2.2. Δομές Δεδομένων

Για τις εργασίες στο σύστημα έχουμε τον *πίνακα περιγραφητών των εργασιών (job descriptor table, JDT)*, ο οποίος περιέχει στοιχεία τύπου *jdt* (defs.h). Κάθε δομή αυτού του τύπου, περιέχει τα ακόλουθα πεδία:

long int job_number: αριθμός εργασίας, όπως δίνεται από το σύστημα (κάθε νέα εργασία έχει αριθμό κατά μία μονάδα μεγαλύτερο από την προηγούμενη).
float think_time: χρόνος σκέψης των χρηστών στα τερματικά.
double cpu_time: χρόνος εκτέλεσης (εκτίμηση χρήστη ή μέγιστος επιτρεπτός χρόνος ολοκλήρωσης αλληλεπιδραστικής εργασίας).
double arrival_time: χρόνος άφιξης της εργασίας στο σύστημα (ολοκλήρωσης της ανάγνωσης ή της σύνδεσης).
int prty: εξωτερική προτεραιότητα της εργασίας.
int no_of_processes: πλήθος διεργασιών που θα δημιουργήσει η εργασία.

int jrnd: τυχαίος αριθμός για εκτιμήσεις που αφορούν την εργασία αυτή.
int jtype: τύπος εργασίας, μπορεί να είναι

JOB_MULTIACCESS	αλληλεπιδραστική
JOB_BATCH	δεσμίδων

int input_records: πλήθος εγγραφών εισόδου (δεσμίδες) ή πλήθος αλληλεπιδράσεων.
int backing_store_records: εγγραφές στο δίσκο (σε φυσικές ομάδες).
int output_records: πλήθος εγγραφών εξόδου (δεσμίδες) ή αριθμός τερματικού (αλληλεπιδράσεις).
int no_of_files: πλήθος αρχείων στο δίσκο (μέχρι 5), πέρα από τα αρχεία για spooling ή αλληλεπίδραση με το τερματικό.
long int cm_space: μέγιστος χώρος στη μνήμη για όλες τις σχετικές διεργασίες μαζί.
struct user_process *rootproc: δείκτης στον πίνακα σκιάς για τον οργανωτή της εργασίας.
float accounting: λογιστική χρέωση της εργασίας
int status: κατάσταση της εργασίας, μπορεί να είναι

JOB_ST_INPUT	διαβάζεται (μόνο για δεσμίδες)
JOB_ST_WAITING	περιμένει ενεργοποίηση
JOB_ST_ACTIVE	ενεργοποιήθηκε
JOB_ST_FAILED	απέτυχε να ενεργοποιηθεί, περιμένει

struct jdt *jlink: δείκτης στον επόμενο περιγραφητή εργασίας.

Οι περιγραφητές είναι συνδεδεμένοι σαν λίστα μίας κατεύθυνσης, η κεφαλή της οποία βρίσκεται στη μεταβλητή job_head (vardefs.h) τύπου jdt *.

Για την κατάσταση του συστήματος, έχουμε μεταβλητές για το πλήθος καταχωρήσεων στους πίνακες του συστήματος, το πλήθος των τερματικών που χρησιμοποιούνται σε κάθε στιγμή (int terminals_supported, το πλήθος τερματικών που έχει το σύστημα είναι no_of_terminals), την κατάσταση του spooler εισόδου (int input_spooler_free) και τους τρέχοντες αριθμούς της ψευδοτυχαίας σειράς για τις εργασίες δεσμίδων (int brnd) και αλληλεπίδρασης (int marnd). Εκτός από τους τυχαίους αριθμούς, κρατάμε και άλλα γενικά χαρακτηριστικά των εργασιών, τα οποία είναι παράμετροι της προσομοίωσης και χρησιμεύουν στην εκτίμηση των χαρακτηριστικών των πραγματικών εργασιών και διεργασιών του συστήματος. Οι μεταβλητές αυτές είναι τύπου float και δίνουν τους μέσους των κατανομών για τα χαρακτηριστικά των εργασιών, ανά κατηγορία (δεσμίδων ή αλληλεπίδρασης). Για όλες τις εργασίες κρατάμε μέσο για μνήμη, χρόνο ΚΜΕ, πλήθος διεργασιών, πλήθος εγγραφών στο δίσκο και πλήθος αρχείων. Για τις εργασίες δεσμίδων έχουμε και τις εγγραφές εισόδου και εξόδου, ενώ για τις εργασίες αλληλεπίδρασης έχουμε το χρόνο σκέψης των χρηστών και το πλήθος αλληλεπιδράσεων. Τέλος έχουμε τις χωρητικότητες των περιοχών (δεξαμενών ή πηγαδιών) εισόδου και εξόδου

(παράμετροι συστήματος). Όλα τα παραπάνω βρίσκονται στο αρχείο psvc.c ή ssvc.c.

Τέλος, εσωτερικά στο χρονοπρογραμματιστή εργασιών έχουμε μετρητές για το πλήθος εργασιών στην είσοδο και την έξοδο, το πλήθος εργασιών που περιμένουν και την κατάσταση του συστήματος από πλευράς φόρτωσης.

3.2.2.3. Αλγόριθμοι

3.2.2.3.1. Διαγραφή εργασιών

Η συνάρτηση *delete_job* καλείται κατά τη διαγραφή μίας εργασίας από το σύστημα και διαγράφει τον περιγραφητή της από τον jdt και τις καταχωρήσεις της από τον πίνακα σκιάς της. Οι θέσεις σε πίνακα σκιάς και jdt δίνονται από καθολικές μεταβλητές. Αρχικά, καθαρίζουμε τον πίνακα σκιάς, διαγράφοντας πρώτα τον οργανωτή της εργασίας και μετά διασχίζοντας τη λίστα των παιδιών του (η αρχή της οποίας βρίσκεται στο πεδίο son_pointer του οργανωτή, ενώ η σύνδεση γίνεται μέσω των πεδίων brother_pointer για κάθε παιδί). Μαζί με την απελευθέρωση των στοιχείων, μειώνεται και το πλήθος καταχωρήσεων στον πίνακα αυτό. Στη συνέχεια, ανάλογα με τον τύπο της εργασίας ενημερώνουμε είτε το μέσο χρόνο ανακύκλωσης των εργασιών (turnaround time) είτε το μέσο χρόνο σύνδεσης των χρηστών, ανάλογα με το αν η εργασία είναι η πρώτη του είδους της ή όχι, όπου και υπολογίζουμε έναν εξομαλυμένο μέσο. Σε κάθε περίπτωση, ενημερώνουμε το πλήθος εργασιών που ολοκληρώθηκαν για τον τύπο αυτό και, για τις εργασίες αλληλεπίδρασης, απελευθερώνουμε το τερματικό (ο αριθμός του οποίου βρίσκεται στο word7), μειώνουμε τα τερματικά σε χρήση και υπολογίζουμε το συσσωρευμένο χρόνο σύνδεσης χρηστών με τα τερματικά. Τελικά, διατρέχουμε τον πίνακα των εργασιών ψάχνοντας την εργασία αυτή και την αποσυνδέουμε από τη λίστα, μειώνοντας τις εργασίες στο σύστημα και απελευθερώνοντας το χώρο του περιγραφητή της.

3.2.2.3.2. Δημιουργία επόμενης εργασίας δεσμίδων

Η συνάρτηση *next_batch_job* δημιουργεί τον περιγραφητή εργασίας για την επόμενη εργασία δεσμίδων που θα διαβαστεί και ετοιμάζει το ανάλογο μήνυμα ανάγνωσης για αποστολή στο spooler εισόδου. Η συνάρτηση δεσμεύει το spooler εισόδου και μειώνει το πλήθος των εργασιών που απομένουν για δημιουργία. Στη συνέχεια, δημιουργεί τη θέση του περιγραφητή αυξάνοντας και το πλήθος εργασιών στο σύστημα. Τα πεδία του περιγραφητή αρχικοποιούνται, με τύπο εργασίας δεσμίδων, μηδέν χρέωση, χρόνο άφιξης τον τρέχοντα, τυχαίο αριθμό εργασίας τον επόμενο της σειράς (χρήση του brnd) και προτεραιότητα που υπολογίζεται από αυτό τον αριθμό. Ο χρόνος

εκτέλεσης είναι τουλάχιστον ο χρόνος εισόδου, ενώ ο χώρος στη μνήμη φτάνει μέχρι δύο φορές το μέγεθος της μνήμης. Το πλήθος διεργασιών είναι μέχρι διπλάσιο του μέσου (ο οργανωτής δεν περιλαμβάνεται), ο χρόνος σκέψης είναι φυσικά μηδέν, οι εγγραφές εισόδου, εξόδου και δίσκου εκτιμώνται με όριο το 32766 και το πλήθος αρχείων είναι από 1 έως 5. Η εργασία σημειώνεται σαν αναμένουσα είσοδο και ετοιμάζεται τέλος ένα μήνυμα προς τον spooler εισόδου με το πλήθος των εγγραφών εισόδου στο word3 και έναν δείκτη προς την εργασία στο word8.

3.2.2.3.3. Κύρια διαδικασία

Ο κώδικας του χρονοπρογραμματιστή εργασιών βρίσκεται στη συνάρτηση *job_scheduler*, η οποία κατά την πρώτη της είσοδο διαβάζει τα χαρακτηριστικά των εργασιών δεσμίδων και αλληλεπίδρασης καθώς και τις χωρητικότητες των περιοχών εισόδου και εξόδου, όπως δίνονται στην παράγραφο για τις δομές δεδομένων. Στη συνέχεια αρχικοποιούνται μεταβλητές για το πλήθος εργασιών που εκτελέστηκαν, τους χρήστες σε σύνδεση, το πλήθος εργασιών που περιμένουν ενεργοποίηση (όλα 0), οι μετρητές και τα μέγιστα των καταχωρήσεων στις σχετικές δομές, η κατάσταση των περιοχών και το πλήθος εργασιών που έτυχαν επεξεργασίας. Τέλος, αν απομένουν εργασίες για δημιουργία (μεταβλητή int job_count, από το vardefs.h), προγραμματίζουμε την άφιξη ορισμένων χρηστών (εργασίες αλληλεπίδρασης, στο word3). Αν έχουμε ένα μικρό πλήθος εργασιών (χωράνε όλες ή στα τερματικά ή στο πηγάδι εισόδου), μοιράζουμε τις εργασίες μεταξύ τερματικών και δεσμίδων, κατανέμοντας στα τερματικά το πολύ τις μισές εργασίες. Αλλιώς, θα δρομολογήσουμε τόσες εργασίες όσα και τα τερματικά. Μειώνουμε το πλήθος εργασιών που μένουν κατά το πλήθος αυτών που θα προγραμματίσουμε και δρομολογούμε το γεγονός άφιξης χρήστη επαναληπτικά σε χρόνο που αρχίζει από τον τρέχοντα (πρώτος χρήστης), και αυξάνεται σύμφωνα με μία κατανομή Poisson του χρόνου μεταξύ των αφίξεων (σε κάθε μήνυμα αλλάζει το word7 που περιέχει τον αριθμό τερματικού). Παρατηρήστε ότι στο μέλλον οι προβλέψεις άφιξης δεν θα είναι καθαρά σύμφωνα με την κατανομή Poisson, επειδή η δρομολόγηση θα γίνεται μόνο όταν ένας χρήστης φεύγει από το σύστημα και όχι συνεχώς. Αφού μηδενίσουμε τα διαθέσιμα τερματικά και το χρόνο σύνδεσης, ελέγχουμε αν υπάρχει χώρος στο πηγάδι εισόδου και, αν μπορούμε, καλούμε τη next_batch_job για την κατασκευή του περιγραφητή μίας νέας εργασίας δεσμίδων, και στέλνουμε το μήνυμα που ετοιμάζεται εκεί, προχωρώντας τελικά σε κάθε περίπτωση στην κύρια είσοδο.

Στην κύρια είσοδο, περιμένουμε ένα μήνυμα από το δημιουργό διεργασιών (θύρα 1), το διαχειριστή της ΚΜΕ (θύρα 2), το σύστημα τερματικών (θύρα 3), το spooler εξόδου (θύρα 4), το spooler εισόδου (θύρα 5)

ή μία άλλη διεργασία. Το μήνυμα λαμβάνεται στην είσοδο 2, όπου ελέγχεται ο αποστολέας του. Αν είναι το σύστημα τερματικών, προχωρούμε στην είσοδο accept_multiaccess_job. Στην είσοδο αυτή, εξετάζουμε την αίτηση ενός χρήστη για σύνδεση με το σύστημα. Αρχικά, αυξάνουμε το πλήθος των χρηστών που ήρθαν στο σύστημα και βρίσκουμε έναν νέο τυχαίο αριθμό (από τη σειρά του marnd). Ελέγχουμε την εγκυρότητα του μηνύματος και την κατάσταση του συστήματος. Αν το σύστημα είναι υπερφορτωμένο προχωράμε στην είσοδο abort_multiaccess_job, με αιτία αναστολής τον ανεπαρκή χώρο. Αλλιώς, στέλνουμε μήνυμα αίτησης για αναφορά (mcommand=4) στο διαχειριστή της ΚΜΕ και προχωράμε στην είσοδο 12. Εκεί περιμένουμε την απάντηση από το διαχειριστή της ΚΜΕ, η οποία λαμβάνεται στην είσοδο 13. Σε περίπτωση που ο φόρτος του συστήματος υπερβαίνει τον επιτρεπόμενο (ο φόρτος στο word3, το μέγιστο στο maxresp) και έχουμε επεξεργαστεί αρκετές εργασίες αλληλεπίδρασης, προχωράμε πάλι στην είσοδο abort_multiaccess_job με κωδικό λάθους την χαμηλή απόκριση του συστήματος. Στην είσοδο αυτή, ελέγχουμε αν απομένουν εργασίες προς δημιουργία, και αν αυτό συμβαίνει προβλέπουμε την επόμενη άφιξη χρήστη σε τερματικό, χρησιμοποιώντας την κατανομή Poisson για το χρόνο μέχρι την άφιξη. Σε κάθε περίπτωση, τελικά στέλνουμε ένα μήνυμα στο σύστημα τερματικών με την αιτία της αποτυχίας στο word3 και επιστρέφουμε στην κύρια είσοδο. Παρατηρήστε ότι χρησιμοποιούμε τη σειρά τυχαίων αριθμών για τις προβλέψεις που θα σχετίζονται με το νέο χρήστη, ανεξάρτητα από το αν τελικά αυτός θα γίνει αποδεκτός ή όχι, για να έχουμε επαναληψιμότητα στα αποτελέσματα.

Στην περίπτωση που ο χρήστης γίνει αποδεκτός από το σύστημα, ενημερώνουμε το πλήθος συνδεδεμένων χρηστών και το συσσωρευμένο χρόνο σύνδεσης καθώς και τον πίνακα τερματικών για την κατάληψη του τερματικού. Στη συνέχεια, δημιουργούμε τον περιγραφητή της εργασίας ενημερώνοντας τους σχετικούς μετρητές και τον συνδέουμε στην κεφαλή της λίστας των εργασιών (η σειρά των αλληλεπιδραστικών εργασιών δεν έχει σημασία). Η εργασία σημειώνεται ως αλληλεπίδρασης και ως αναμένουσα ενεργοποίηση και ο τυχαίος της αριθμός σημειώνεται. Ορίζεται η χρέωση (0) και ο χρόνος άφιξης (από τον pdt). Η προτεραιότητα είναι η υψηλότερη. Ο χώρος μνήμης, ο χρόνος της ΚΜΕ, το πλήθος διεργασιών, το πλήθος αρχείων και εγγραφών στο δίσκο υπολογίζεται όπως και στην next_batch_job, με τη διαφορά ότι χρησιμοποιούμε τους μέσους για τις εργασίες αλληλεπίδρασης. Το πεδίο για το χρόνο σκέψης του χρήστη παίρνει ελάχιστη τιμή το 1 sec, στις εγγραφές εισόδου βάζουμε το πλήθος αλληλεπιδράσεων και στις εγγραφές εξόδου τον αριθμό του τερματικού. Αμέσως μετά, προχωράμε στην είσοδο create_root_process.

Η είσοδος create_root_process καλείται όταν θέλουμε να δημιουργήσουμε τον πίνακα σκιάς μίας οποιασδήποτε εργασίας. Αρχικά

ενημερώνουμε το πεδίο αριθμού εργασίας στον jdt και προχωράμε το μετρητή εργασιών. Υπολογίζουμε το μέσο χρόνο εκτέλεσης, το μέσο πλήθος εγγραφών εισόδου ή αλληλεπιδράσεων και το μέσο πλήθος εγγραφών στο δίσκο για κάθε διεργασία, διαιρώντας τις απαιτήσεις τις εργασίες με το πλήθος των διεργασιών που θα δημιουργηθούν (υπολογίζοντας και τον οργανωτή της εργασίας). Το ίδιο γίνεται και για τις εγγραφές εξόδου, μόνο όμως για τις εργασίες δεσμίδων. Στη συνέχεια, σημειώνουμε τις συνολικές απαιτήσεις των εργασιών σε μετρητές πόρων και δημιουργούμε τις διεργασίες μία προς μία, αρχίζοντας με τη ρίζα. Για κάθε διεργασία, εκτιμούμε το μέγεθός της και δημιουργούμε την καταχώρησή της στον πίνακα σκιάς. Υπολογίζουμε το πλήθος των τεμαχίων της (μέχρι no_of_pages) και ενημερώνουμε τους μετρητές για τον πίνακα σκιάς. Το στοιχείο συνδέεται με τον jdt και τα αρχεία γίνονται δύο παραπάνω από τα αρχεία της εργασίας (τα μόνιμα αρχεία). Υπολογίζεται ο τυχαίος αριθμός της διεργασίας και προχωρούμε στον υπολογισμό των μεγεθών των τεμαχίων. Στο σύστημα σελιδοποίησης όλα τα τεμάχια είναι ίσα με page_size, ενώ στο σύστημα τεμαχισμού εκτιμώνται τυχαία, με το τελευταίο να είναι το υπόλοιπο του συνόλου. Παρατηρήστε ότι και στο σύστημα σελιδοποίησης προχωράμε τη σειρά των ψευδοτυχαίων αριθμών για επαναληψιμότητα. Έτσι, η μεταβλητή που δέχεται αυτή την τιμή στο σύστημα σελιδοποίησης δεν χρησιμοποιείται ποτέ (με αποτέλεσμα ένα warning σε ορισμένους μεταγλωττιστές). Ο χρόνος εκτέλεσης και οι απαιτήσεις ορίζονται μέσω εκτιμήσεων, με ελάχιστο του χρόνου εκτέλεσης το χρόνο εισόδου στη διεργασία και μέγιστο το συνολικό χρόνο εκτέλεσης που απομένει στην εργασία. Για τις εργασίες δεσμίδων, οι απαιτήσεις από το τερματικό είναι μηδέν, ενώ η είσοδος και η έξοδος εκτιμώνται με όριο το συνολικό πλήθος που απομένει για την εργασία. Για τις εργασίες αλληλεπίδρασης, η είσοδος και η έξοδος γίνονται μηδέν ενώ οι αλληλεπιδράσεις εκτιμώνται με όριο τις αλληλεπιδράσεις που απομένουν για την εργασία. Τα κανονικά αρχεία τέλος, μοιράζονται τις απαιτήσεις σχεδόν εξίσου, κατ' εκτίμηση, με όριο τις απαιτήσεις που απομένουν στην εργασία. Αν η διεργασία είναι η τελευταία, δεν έχουμε εκτίμηση των παραμέτρων αλλά εκχώρηση των υπολοίπων απαιτήσεων στα κατάλληλα πεδία. Τέλος, συνδέουμε το στοιχείο με τη ρίζα και τα αδέλφια του αν είναι παιδί, σημειώνοντας ότι δεν έχει παιδιά (μία γενιά παιδιών μόνο) ή σημειώνουμε το πλήθος παιδιών και την έλλειψη αδελφού για τη ρίζα (αν δεν έχει παιδιά, ο δείκτης παιδιών γίνεται NULL, αλλιώς ορίζεται με το πρώτο παιδί). Ο δείκτης του jdt συνδέεται με τη ρίζα του πίνακα σκιάς. Παρατηρήστε, ότι τα δύο μόνιμα αρχεία συνδέονται με το τερματικό (αλληλεπιδράσεις, αρχείο 0) ή με την είσοδο (αρχείο 0) και την έξοδο (αρχείο 2) των spoolers για τις εργασίες δεσμίδων. Αμέσως, προχωράμε στην είσοδο 11.

Στην είσοδο 11 εξυπηρετούμε τις αιτήσεις για φόρτωση της διεργασίας ρίζας (οργανωτή εργασίας ή φλοιού). Αρχικά, στέλνουμε μήνυμα για τη

δημιουργία της διεργασίας (mcommand=0) στο δημιουργό διεργασιών, προχωρώντας στην είσοδο 3. Εκεί, περιμένουμε τη σχετική απάντηση, την οποία λαμβάνουμε στην είσοδο 4. Αν η απάντηση είναι για επιτυχή φόρτωση (mcommand=1), η εργασία σημειώνεται ως ενεργή και αν είναι εργασία δεσμίδων, αυξάνουμε τις κατειλημμένες θέσεις στην έξοδο και ελαττώνουμε τις εργασίες που περιμένουν ενεργοποίηση (οι αλληλεπιδραστικές εργασίες δεν περιμένουν ποτέ). Αν δεν υπάρχουν άλλες διεργασίες που περιμένουν ή αν το σύστημα είναι φορτωμένο ή η περιοχή εξόδου είναι γεμάτη, επιστρέφουμε στην κύρια είσοδο. Αλλιώς, προχωράμε στην επόμενη εργασία του jdt και καλούμε την είσοδο try_to_select_another_batch_job_for_activation. Αν έχουμε αποτυχία φόρτωσης, αν η εργασία είναι δεσμίδων και η αιτία αποτυχίας είναι έλλειψη χώρου στους πίνακες του συστήματος (αιτία στο word3), το σύστημα θεωρείται υπερφορτωμένο, η εργασία σημειώνεται ως αποτυχούσα να φορτωθεί και επιστρέφουμε στην κύρια είσοδο. Αν έχουμε άλλη αιτία αποτυχίας, διαγράφουμε την εργασία από το σύστημα καλώντας τη συνάρτηση delete_job, τυπώνουμε ένα μήνυμα λάθους και επιστρέφουμε στην κύρια είσοδο (η περίπτωση αυτή είναι παράλογη αφού δεν έχουμε σφάλματα ε/ε στο σύστημά μας). Αν η εργασία ήταν πολλαπλής πρόσβασης, ελέγχουμε πάλι για υπερφόρτωση όπως πριν και αν απομένουν εργασίες προς δημιουργία, προγραμματίζουμε την άφιξη του επόμενου χρήστη (στο ίδιο τερματικό), με τον ίδιο τρόπο που είδαμε παραπάνω. Τελικά, καλούμε την delete_job για να διαγράψουμε την εργασία από το σύστημα και στέλνουμε μήνυμα αποτυχίας στο χρήστη (word3=1), μέσω του συστήματος τερματικών, επιστρέφοντας τελικά στην κύρια είσοδο.

Στην try_to_select_another_batch_job_for_activation, ψάχνουμε πάντα μία εργασία δεσμίδων για ενεργοποίηση. Έτσι, διασχίζουμε τον jdt ψάχνοντας για μία τέτοια εργασία (αυτό μπορεί να ισχύει μόνο για τις εργασίες δεσμίδων). Αν βρεθεί μία τέτοια εργασία, πηγαίνουμε στην είσοδο create_root_process που έχουμε ήδη δει. Εναλλακτικά, αν βρεθεί μία εργασία που έχει αποτύχει να φορτωθεί, πηγαίνουμε στην είσοδο 11 για να προσπαθήσουμε να την φορτώσουμε ξανά. Παρατηρήστε ότι η μετάβαση στην είσοδο αυτή γίνεται μόνο όταν υπάρχουν εργασίες που περιμένουν, έτσι η αποτυχία εύρεσης μίας τέτοιας εργασίας είναι δείγμα λάθους.

Επανερχόμαστε στην κύρια είσοδο, όπου αν ο αποστολέας του αρχικού μηνύματος είναι μία διεργασία χρήστη, προχωράμε στην είσοδο delete_root_process. Η είσοδος αυτή διαγράφει τη διεργασία ρίζα μίας εργασίας (τα παιδιά, δημιουργούνται και διαγράφονται με ευθύνη της ρίζας). Αφού βρούμε τον πίνακα σκιάς της εργασίας, στέλνουμε μήνυμα στο δημιουργό διεργασιών να διαγράψει τη διεργασία (mcommand=1), διατηρώντας τις αιτίες διαγραφής στα word6 και word3 και την ταυτότητά της στο mpid. Προχωράμε στην είσοδο 5, όπου αναμένουμε την απάντηση του

δημιουργού διεργασιών, την οποία λαμβάνουμε στην είσοδο 6, όπου καταστρέφουμε το μονοπάτι επικοινωνίας με τη διεργασία. Προχωράμε στην είσοδο 9, όπου ολοκληρώνουμε την εργασία, στέλνοντας ένα μήνυμα στον spooler εξόδου να τυπώσει τα αποτελέσματα της εργασίας (mcommand=0), αν η εργασία ήταν δεσμίδων, έχοντας το αναγνωριστικό της εργασίας στο word8 και το πλήθος εγγραφών εξόδου στο word3. Μετά το μήνυμα προχωράμε στην είσοδο 8. Εναλλακτικά, αν η εργασία ήταν αλληλεπίδρασης, βρίσκουμε τον αριθμό του τερματικού της και αν υπάρχουν εργασίες προς δημιουργία, δρομολογούμε την επόμενη άφιξη χρήστη, με το γνωστό τρόπο, και στη συνέχεια προχωράμε στην είσοδο 14.

Στην είσοδο 8, έχουμε έλθει για εργασία δεσμίδων, οπότε απελευθερώνουμε τη θέση στην περιοχή εισόδου και αν υπάρχουν εργασίες προς ενεργοποίηση και ο spooler εισόδου είναι ελεύθερος, καλούμε τη διαδικασία *next_batch_job* για να δημιουργήσει μία νέα εργασία δεσμίδων και να ετοιμάσει το μήνυμα προς το spooler εισόδου, για να τη διαβάσει, το οποίο στέλνουμε και προχωράμε στην είσοδο 15. Ακόμη και αν δεν έχουμε νέα εργασία, προχωράμε πάλι στην είσοδο 15, όπου ελέγχουμε τη δυνατότητα ενεργοποίησης μίας υπάρχουσας εργασίας. Αυτό είναι δυνατόν αν υπάρχουν εργασίες που περιμένουν και υπάρχει χώρος στο πηγάδι εξόδου, οπότε βρίσκουμε την κεφαλή του jdt και προχωράμε στην try_to_select_another_batch_job_for_activation, που είδαμε παραπάνω. Αλλιώς, επιστρέφουμε στην κύρια είσοδο.

Για τις εργασίες αλληλεπίδρασης, ερχόμαστε στην είσοδο 14, όπου διαγράφουμε την εργασία καλώντας την delete_job και σημειώνουμε ότι το σύστημα δεν είναι υπερφορτωμένο, αφού μόλις έφυγε ένας χρήστης. Τελικά, προχωράμε στην είσοδο 15 για να δούμε αν μπορούμε να ενεργοποιήσουμε κάποια εργασία δεσμίδων. Αν η ενεργοποίηση του χρονοπρογραμματιστή εργασιών οφείλεται σε μήνυμα από το spooler εξόδου, σημαίνει ότι ολοκληρώθηκε η εκτύπωση αποτελεσμάτων μίας εργασίας δεσμίδων. Έτσι, απελευθερώνουμε μία θέση στο πηγάδι εξόδου και προχωράμε στην είσοδο 14, για να διαγράψουμε την εργασία και να ελέγξουμε αν μπορεί να ενεργοποιηθεί κάποια εργασία δεσμίδων. Αν το μήνυμα δεν έρχεται ούτε από το spooler εξόδου, έρχεται από το spooler εισόδου και σημαίνει ότι διαβάστηκε μία εργασία δεσμίδων. Τότε, δεσμεύουμε μία θέση στην είσοδο και σημειώνουμε την εργασία ως αναμένουσα ενεργοποίηση. Η εργασία τοποθετείται στην ουρά του jdt σύμφωνα με την προτεραιότητά της (πεδίο prty). Τέλος, ελέγχουμε αν γέμισε η περιοχή εισόδου, οπότε απελευθερώνουμε το spooler εισόδου. Αλλιώς, αν απομένουν εργασίες για δημιουργία, καλούμε τη διαδικασία *next_batch_job* να προετοιμάσει την επόμενη εργασία δεσμίδων και στέλνουμε το μήνυμα που κατασκευάστηκε στον spooler εισόδου, προχωρώντας στην είσοδο 10. Εκεί φτάνουμε και όταν δεν δημιουργούμε νέα εργασία. Στην

είσοδο 10 αυξάνουμε το πλήθος των εργασιών που περιμένουν ενεργοποίηση και αν υπάρχει ελεύθερη περιοχή εξόδου και το σύστημα δεν είναι υπερφορτωμένο, καλούμε την είσοδο create_root_process για να προσπαθήσουμε να δημιουργήσουμε έναν οργανωτή εργασίας για την εργασία που διαβάστηκε. Αλλιώς, επιστρέφουμε άμεσα στην κύρια είσοδο.

3.2.3. Δημιουργός διεργασιών

3.2.3.1. Εισαγωγή

Γενικά ο δημιουργός διεργασιών έχει σαν καθήκον να δημιουργεί και να διαγράφει διεργασίες μέσα στο σύστημα. Αυτό σημαίνει ότι ικανοποιεί απαιτήσεις διεργασιών για δημιουργία και διαγραφή των παιδιών τους. Κατά τη δημιουργία, διαβάζουμε από το δίσκο τις πληροφορίες για τη διεργασία που θα δημιουργηθεί και ανάλογα ειδοποιούμε τους διαχειριστές μνήμης και ΚΜΕ για τη δέσμευση θέσεων στους πίνακες του συστήματος για τη διεργασία. Κατά την καταστροφή διεργασιών, όλα αυτά τα στοιχεία αποδεσμεύονται. Επειδή η δημιουργία μίας διεργασίας δεν είναι πάντα δυνατή, στην αίτηση μπορούμε να απαντήσουμε με αποτυχία, δίνοντας έναν κωδικό ανάλογα με την αιτία της αποτυχίας. Οι κωδικοί αυτοί είναι:

PC_NO_PID	δεν υπάρχουν θέσεις στον pdt
PC_NO_SECS	δεν υπάρχουν θέσεις στον sdt
PC_NO_DISK	αποτυχία κατά τη μεταφορά πληροφοριών από το δίσκο

Παρατηρήστε ότι οι πρώτοι δύο κωδικοί αναφέρονται σε πληρότητα, δηλαδή υπερφόρτωση, των πινάκων του συστήματος.

Επειδή στο σύστημά μας χρησιμοποιούμε τις απαιτήσεις μνήμης των διεργασιών για να υπολογίσουμε το μέγεθος του συνόλου καταλλήλων, κατά τη δημιουργία μίας διεργασίας πρέπει να υπολογίζουμε και το αρχικό σύνολο εργασίας της, έτσι ώστε να μην παραπλανήσουμε το μεσοχρόνιο χρονοπρογραμματιστή. Αυτή η πρόβλεψη βέβαια είναι θέμα της προσομοίωσης και δεν γίνεται σε ένα πραγματικό σύστημα, όπου οι πληροφορίες αυτές διαβάζονται και υπολογίζονται αντί να εκτιμώνται στοχαστικά.

Ο κώδικας για το δημιουργό διεργασιών βρίσκεται στο αρχείο *procr.c*. Οι δομές δεδομένων που χρησιμοποιούνται από το δημιουργό διεργασιών δεν είναι κάτω από την κυριαρχία του, αλλά συνδέονται περισσότερο με άλλες διεργασίες, έτσι δεν περιγράφονται ειδικά σε αυτή την ενότητα. Τα μόνα ιδιωτικά δεδομένα που χρησιμοποιεί ο διαχειριστής διεργασιών είναι στατικές τοπικές μεταβλητές στο αρχείο procr.c.

3.2.3.2. Δομές Δεδομένων

Πέρα από τις μεταβλητές υπενθύμισης για το ποια διεργασία επικοινωνεί με το δημιουργό διεργασιών, έχουμε και δύο αριθμούς οι οποίοι επηρεάζουν τις εκτιμήσεις που κάνει ο δημιουργός διεργασιών για τα αρχικά σύνολα εργασίας των διεργασιών που δημιουργεί. Οι μεταβλητές αυτές είναι οι:

float f1: ποσοστό συνολικού μεγέθους διεργασίας
float f2: ποσοστό συνολικού μεγέθους μνήμης

Και οι δύο αυτοί αριθμοί είναι ποσοστά (από 0 έως 1), σύμφωνα με τα οποία υπολογίζουμε το αρχικό μέγεθος του συνόλου εργασίας για τις νέες διεργασίες, όπως θα δούμε παρακάτω.

3.2.3.3. Αλγόριθμοι

Ο δημιουργός διεργασιών υλοποιείται από τον κώδικα της συνάρτησης *process_creator*. Στην πρώτη του είσοδο, ο δημιουργός διεργασιών διαβάζει το ποσοστό του μεγέθους της διεργασίας το οποίο θα δίνει το μέγεθος του αρχικού συνόλου εργασίας για κάθε νέα διεργασία, καθώς και το ποσοστό αυτό σε σχέση με το συνολικό μέγεθος της μνήμης του συστήματος (που είναι διαθέσιμη στους χρήστες). Στην κύρια είσοδο, προετοιμάζουμε την αποδοχή ενός μηνύματος από το χρονοπρογραμματιστή εργασιών (θύρα 0), το διαχειριστή της ΚΜΕ (θύρα 1), το διαχειριστή της μνήμης (θύρα 2), το διαχειριστή του δίσκου (θύρα 3) ή μία διεργασία (που θα ζητάει δημιουργία ή διαγραφή ενός παιδιού της). Στην είσοδο 2, λαμβάνουμε το μήνυμα και αφού αποθηκεύσουμε την θύρα αποστολής του (έτσι ώστε να μπορούμε να απαντήσουμε αργότερα), εξετάζουμε το περιεχόμενο του πεδίου mcommand, ανάλογα με το οποίο μπορεί να κινηθούμε σε μία από τις ακόλουθες (υπο)εισόδους:

PR_CREATE_IT	δημιουργία διεργασίας
PR_DELETE_IT	διαγραφή διεργασίας

Η υποείσοδος PR_CREATE_IT, καλείται όταν ζητείται η δημιουργία μίας νέας διεργασίας. Στο word8 έχουμε ένα δείκτη προς τον πίνακα σκιάς της σχετικής εργασίας, από όπου μπορούμε να πάρουμε τα χαρακτηριστικά της διεργασίας. Το πρώτο βήμα, είναι να στείλουμε ένα μήνυμα στο διαχειριστή της ΚΜΕ με εντολή δημιουργίας διεργασίας (mcommand=0), έτσι ώστε να δεσμευθεί χώρος στον pdt για τον περιγραφητή της διεργασίας. Μετά το μήνυμα, προχωράμε στην είσοδο 3, όπου προετοιμάζουμε την λήψη της απάντησης από το διαχειριστή της ΚΜΕ και προχωράμε στην είσοδο 4. Εκεί, ελέγχουμε το περιεχόμενο της απάντησης και, σε περίπτωση αποτυχίας

δημιουργίας, προχωράμε στην είσοδο 13, με κωδικό αποτυχίας PC_NO_PID. Στην είσοδο 13, ετοιμάζουμε μία απάντηση προς τον αιτούντα, με την αιτία της αποτυχίας δημιουργίας στο word3 και τον κωδικό αποτυχίας (mcommand=0). Αν ο γονέας (που ζήτησε τη δημιουργία) είναι ο χρονοπρογραμματιστής εργασιών, μετά την αποστολή του μηνύματος γυρνάμε στην κύρια είσοδο, αλλιώς προχωράμε στην είσοδο 10 για καταστροφή του μονοπατιού προς τον αιτούντα (δεν χρειάζεται πια, είναι προς διεργασία χρήστη), απ' όπου τελικά επιστρέφουμε στην κύρια είσοδο.

Αν μετά την αίτηση για δέσμευση θέσης στον pdt έχουμε επιτυχία, στην απάντηση από το διαχειριστή της ΚΜΕ παίρνουμε τη θέση του περιγραφητή της νέας διεργασίας στον pdt. Έτσι, συνδέουμε τη θέση αυτή με τον πίνακα σκιάς της διεργασίας (όπου περιέχονται τα χαρακτηριστικά της προσομοίωσης) και προχωράμε στέλνοντας ένα μήνυμα στο διαχειριστή του δίσκου για να διαβάσουμε το αρχείο περιγραφής της διεργασίας (όπου περιέχεται και το μέγεθος του προγράμματος), με εντολή μεταφοράς από το δίσκο (mcommand=0) και το μέγεθος ενός μέσου τεμαχίου (με την αυθαίρετη υπόθεση ότι όλοι οι "περιγραφητές" προγραμμάτων έχουν αυτό το μέγεθος) στο word7. Μετά την αποστολή, προχωράμε στην είσοδο 5, όπου περιμένουμε την απάντηση από το διαχειριστή του δίσκου. Η απάντηση λαμβάνεται στην είσοδο 6, όπου αν έχουμε κωδικό αποτυχίας ανάγνωσης (mcommand=0), ορίζουμε την αιτία αποτυχίας δημιουργίας ως PC_NO_DISK και πηγαίνουμε στην είσοδο fail. Στην είσοδο fail, στέλνουμε πρώτα ένα μήνυμα στο διαχειριστή της ΚΜΕ με κωδικό αποτυχίας φόρτωσης (mcommand=3) και αναφέροντας στο mpid το αναγνωριστικό της διεργασίας που δεν μπόρεσε να δημιουργηθεί και μετά την αποστολή προχωράμε στην είσοδο 13, συνεχίζοντας όπως είδαμε παραπάνω. Εδώ, χρειάζεται το επιπλέον μήνυμα για να αποδεσμευθεί η θέση του pdt που δεσμεύσαμε ήδη, ενώ πριν, η αποτυχία εύρεσης θέσης στον pdt δεν άφηνε κάποια υπολείμματα της διεργασίας στο σύστημα.

Στην περίπτωση που η μεταφορά των πληροφοριών ήταν επιτυχής, ετοιμάζουμε ένα μήνυμα προς το διαχειριστή της μνήμης για να ορίσει τα τεμάχια της διεργασίας, παραχωρώντας θέσεις στον sdt για αυτά, με την εντολή ορισμού τεμαχίων (mcommand=0) και με τον περιγραφητή της διεργασίας στο mpid (από εκεί βρίσκονται τα χαρακτηριστικά των τεμαχίων, μέσω του πίνακα σκιάς). Μετά την αποστολή προχωράμε στην είσοδο 7, όπου περιμένουμε τη σχετική απάντηση, την οποία εξετάζουμε στην είσοδο 8. Σε περίπτωση αποτυχίας (mcommand=0), ο κωδικός αποτυχίας δημιουργίας γίνεται PC_NO_SECS και προχωράμε στην είσοδο fail, για τους ίδιους λόγους που είδαμε νωρίτερα. Αλλιώς, η δημιουργία έχει πετύχει, και μπορούμε να γεμίσουμε τα κατάλληλα πεδία του pdt. Έτσι, ορίζουμε τις αρχικές θύρες της διεργασίας (0 προς τον πατέρα της, ο οποίος είναι και ο αιτούντας, 1 προς το

δημιουργό διεργασιών, 2 προς το σύστημα τερματικών και 3 προς το σύστημα αρχείων), και αφήνουμε αόριστες τις άλλες θύρες. Στη συνέχεια, υπολογίζουμε το αρχικό μέγεθος του συνόλου εργασίας της διεργασίας, βρίσκοντας από τον πίνακα σκιάς της το συνολικό της μέγεθος (διασχίζουμε τη λίστα των τεμαχίων της και αθροίζουμε τα μεγέθη τους, που βρίσκονται στο πεδίο sections) και θέτοντας το πεδίο ws_size του pdt στο ελάχιστο των αριθμών f1 επί μέγεθος διεργασίας και f2 επί μέγεθος μνήμης. Τελικά, στέλνουμε ένα μήνυμα στο διαχειριστή της ΚΜΕ, ενημερώνοντάς τον για την τελική επιτυχία της δημιουργίας της διεργασίας (mcommand=2) με αναγνωριστικό που δίνεται στο mpid, γονέα που δίνεται στο words45 και τεμάχιο 0 που δίνεται στο word3 (έχει τεθεί από το διαχειριστή της μνήμης). Επιπλέον, έχουμε και την προτεραιότητα της εργασίας στο word6 (βρίσκεται από τον πίνακα περιγραφητών των εργασιών στο πεδίο prty). Μετά από το μήνυμα, προχωράμε στην είσοδο 9, όπου στέλνουμε την τελική απάντηση στον γονέα για επιτυχή δημιουργία (mcommand=1) με το αναγνωριστικό της διεργασίας που δημιουργήθηκε στο mpid. Αν ο γονέας είναι ο χρονοπρογραμματιστής, επιστρέφουμε στην κύρια είσοδο, αλλιώς, προχωράμε πρώτα στην είσοδο 10 για καταστροφή του μονοπατιού επικοινωνίας, όπως αναφέραμε παραπάνω.

Η υποείσοδος PR_DELETE_IT, καλείται όταν ζητείται διαγραφή μίας διεργασίας από το γονέα της. Στο mpid έχουμε τη ζητούμενη διεργασία και από εκεί βρίσκουμε τον πίνακα σκιάς της διεργασίας. Αρχικά, στέλνουμε ένα μήνυμα στο διαχειριστή της ΚΜΕ για διαγραφή της διεργασίας από τον pdt (mcommand=1) και προχωράμε στην είσοδο 11. Εκεί, περιμένουμε την απάντηση από το διαχειριστή της ΚΜΕ, την οποία λαμβάνουμε στην είσοδο 12. Τέλος, απαντάμε στον γονέα για την επιτυχή διαγραφή της διεργασίας που σχετίζεται με την θέση στον πίνακα σκιάς η οποία αναφέρεται στο word8. Αν ο γονέας είναι ο χρονοπρογραμματιστής εργασιών, επιστρέφουμε στην κύρια είσοδο, ενώ σε κάθε άλλη περίπτωση, προχωράμε πρώτα στην είσοδο 10 για καταστροφή του μονοπατιού επικοινωνίας, όπως είδαμε παραπάνω.

3.3. Διεργασίες του πραγματικού τμήματος του συστήματος

Οι διεργασίες που θα δούμε παρακάτω, είναι αυτές που συμπεριφέρονται σχεδόν ακριβώς με τον ίδιο τρόπο με τις διεργασίες ενός πραγματικού συστήματος. Οι αποκλίσεις από αυτόν τον κανόνα οφείλονται στην απουσία του πραγματικού υλικού και του πραγματικού φόρτου εργασίας του συστήματος. Σαν αποτέλεσμα, οι διεργασίες αυτές έχουν μικρή ως καθόλου σχέση με το τμήμα της προσομοίωσης και υλοποιούν πολιτικές, ο πειραματισμός με τις οποίες μπορεί να έχει μεγάλο ενδιαφέρον για τους χρήστες του προγράμματος. Ακριβώς λόγω της αποδέσμευσης αυτού του επιπέδου από την προσομοίωση, τέτοιος πειραματισμός είναι εφικτός χωρίς να

απαιτείται λεπτομερής γνώσης του τρόπου με τον οποίο λειτουργεί το υπόλοιπο πρόγραμμα της προσομοίωσης.

3.3.1. Διαχειριστής Μνήμης

3.3.1.1. Εισαγωγή

Γενικά ένας διαχειριστής μνήμης ασχολείται με την υλοποίηση της εικονικής μνήμης, προσπαθώντας να μοιράσει τη φυσική μνήμη μεταξύ των διεργασιών που βρίσκονται στο σύστημα. Η υλοποίηση (παροχή) της εικονικής μνήμης, μπορεί να γίνεται μέσω σελιδοποίησης, τεμαχισμού ή συνδυασμού των δύο αυτών τεχνικών. Ο διαχειριστής που θα δούμε εδώ έχει μία μόνο έκδοση και για απλή σελιδοποίηση και για απλό τεμαχισμό (όχι και τα δύο μαζί), εκμεταλλευόμενος τις παρόμοιες ιδιότητες των δύο αυτών τεχνικών από πλευράς υλοποίησης. Επειδή τα μέρη της μνήμης που καταχωρούνται στις διεργασίες μπορεί να είναι είτε σελίδες είτε τμήματα, χρησιμοποιούμε τον όρο *τεμάχια* (sections) για να αναφερόμαστε χωρίς διάκριση και στα δύο είδη.

Οι βασικές λειτουργίες του διαχειριστή της μνήμης είναι να παρακολουθεί την κατάσταση της μνήμης, γνωρίζοντας που βρίσκονται τα τεμάχια των διεργασιών και γνωρίζοντας σε ποια κατάσταση βρίσκεται κάθε ομάδα θέσεων της μνήμης, να αποφασίζει τι θα έρχεται στην κύρια μνήμη, να αποφασίζει που θα τοποθετούνται οι πληροφορίες που έρχονται στη μνήμη και τέλος να αποφασίζει ποιες πληροφορίες θα απομακρύνονται από την κύρια μνήμη, όταν αυτό είναι απαραίτητο (για την προσκόμιση νέων τεμαχίων όταν η μνήμη είναι γεμάτη).

Το πιο ενδιαφέρον σημείο στο διαχειριστή μνήμης, είναι η ικανότητά του να χειρίζεται και συστήματα με σελιδοποίηση και συστήματα με τεμαχισμό. Αυτό γίνεται με το να θεωρούμε τη σελιδοποίηση ως ειδική περίπτωση του τεμαχισμού (όλα τα τεμάχια είναι ισομεγέθη). Αυτό έχει σαν αποτέλεσμα το σύστημα να είναι λιγότερο ταχύ από ότι θα μπορούσε να ήταν αν χρησιμοποιούσαμε ειδικές δομές δεδομένων και αλγόριθμους για το σελιδοποιημένο σύστημα (για παράδειγμα, ένα χάρτη ψηφίων για τις ελεύθερες σελίδες και όχι μία λίστα οπών). Βέβαια, το σύστημα δεν είναι πολύ βραδύτερο απ' ότι αν ήτανε βελτιστοποιημένο για σελιδοποίηση, αφού, κατά τη σελιδοποίηση, στο σύστημά μας οι αναζητήσεις μέσα στις λίστες σταματούν συνήθως στο πρώτο στοιχείο (ίσα μεγέθη τεμαχίων). Από την άλλη μεριά, το σύστημα σελιδοποίησης πάσχει από σχετικά αυξημένη πολυπλοκότητα. Έτσι, μία πολύ ενδιαφέρουσα επέκταση θα ήταν να δημιουργηθεί ένας ειδικός διαχειριστής μνήμης για σελιδοποίηση.

Ο κώδικας του διαχειριστή της μνήμης βρίσκεται στο αρχείο cmman.c, μαζί με τις συναρτήσεις για τοποθέτηση και αντικατάσταση (στις παλιότερες εκδόσεις υπήρχε το αρχείο plrepl.c για αυτές τις συναρτήσεις, αλλά εδώ κρίθηκε ότι υπήρχε μεγάλη εννοιολογική συγγένεια και αρκετά κοινά δεδομένα μεταξύ των συναρτήσεων αυτών τα οποία δε χρειάζονται αλλού, με αποτέλεσμα την ενοποίηση αυτή).

3.3.1.2. Δομές Δεδομένων

Είναι προφανές ότι η διαφορά μεταξύ εικονικών και πραγματικών διευθύνσεων καθώς και ο καταμερισμός της μνήμης από πολλές διεργασίες, οι οποίες μπορεί να βρίσκονται μόνο εν μέρει στην κύρια μνήμη, σημαίνουν ότι πρέπει να έχουμε κάποιον τρόπο να βλέπουμε που βρίσκονται τα τεμάχια των διεργασιών, ποια μέρη της μνήμης είναι κατειλημμένα και ποια ελεύθερα, να παρακολουθούμε γενικά δηλαδή την κατάσταση της μνήμης.

Μία ιδιαιτερότητα του συστήματος, είναι ότι ασχολούμαστε μόνο με τις διεργασίες του χρήστη στο διαχειριστή της μνήμης, αφού υποθέτουμε ότι οι διεργασίες του συστήματος είναι μόνιμα φορτωμένες στη μνήμη, με αποτέλεσμα να υπάρχει κάποια μορφή μόνιμης (σταθερής) καταχώρησης για αυτές. Οι διεργασίες του χρήστη από την πλευρά τους, μπορεί να βρίσκονται στη μνήμη, στο δίσκο ή σε μία συσκευή ανταλλαγής και γι' αυτό πρέπει να παρακολουθούνται. Παρατηρήστε ότι συνεπές με αυτή την υλοποίηση είναι και το ότι η μνήμη του συστήματος (παράμετρος του υλικού) αναφέρεται στη μνήμη που είναι διαθέσιμη στις διεργασίες των χρηστών.

Η κατανομή των τεμαχίων των διεργασιών σε θέσεις της κύριας μνήμης ή σε συσκευές δευτερεύουσας μνήμης μπορεί να παρακολουθηθεί μέσω του *πίνακα περιγραφητών των τεμαχίων (Section Descriptor Table, SDT)*, ο οποίος περιέχει ένα μεταβαλλόμενο πλήθος καταχωρήσεων, κάθε μία από τις οποίες περιγράφει ένα τεμάχιο μίας διεργασίας. Για κάθε τέτοιο τεμάχιο έχουμε μία δομή με το όνομα *sdtstruct* (defs.h), με μία ποικιλία πληροφοριών, οι οποίες για το σύστημα μας είναι οι ακόλουθες:

long int sdbase: διεύθυνση βάσης του τεμαχίου στη μνήμη.
long int sdrange: μέγεθος (μήκος) του τεμαχίου σε bytes.
tchr sdproc: πλήρες αναγνωριστικό της διεργασίας στην οποία ανήκει αυτό το τεμάχιο.
int in_working_set: ψηφίο που δείχνει κατά πόσο το τεμάχιο αυτό ανήκει στο τρέχον σύνολο εργασίας της διεργασίας.
int present_in_core: ψηφίο που δείχνει κατά πόσο το τεμάχιο αυτό βρίσκεται στην κύρια μνήμη.

int on_drum: ψηφίο που δείχνει κατά πόσο το τεμάχιο αυτό βρίσκεται στη συσκευή ανταλλαγής.
int sdacess: δύο ψηφία που δείχνουν τον τρόπο προσπέλασης του τεμαχίου. Οι τιμές που μπορεί να παίρνει το πεδίο αυτό είναι 0 για εκτελέσιμο τεμάχιο, 1 για τεμάχιο μόνο εγγραφής, 2 για τεμάχιο μόνο ανάγνωσης και 3 για τεμάχιο ανάγνωσης / εγγραφής.
int page_acessing_referenced: ψηφίο χρήσης, δείχνει αν έχει χρησιμοποιηθεί το τεμάχιο.
int sdnlink: δείκτης στο επόμενο τεμάχιο της διεργασίας (το τελευταίο τεμάχιο έχει δείκτη -1).

Παρατηρήστε ότι υπάρχει ένα ψηφίο για την παρουσία του τεμαχίου στη συσκευή ανταλλαγής, ενώ δεν υπάρχει κάτι τέτοιο για το δίσκο, αφού εννοείται ότι πάντα ένα τεμάχιο βρίσκεται και στο δίσκο. Όλες αυτές οι καταχωρήσεις βρίσκονται σε έναν πίνακα με το όνομα sdt (vardefs.h), ο οποίος έχει μέγεθος ίσο με τη σταθερά sdtsize (defs.h), συν μία θέση. Οι καταχωρήσεις για κάθε διεργασία συνδέονται μέσω του πεδίου sdnlink το οποίο δείχνει σε μία άλλη θέση στον sdt. Τέλος, οι αχρησιμοποίητες καταχωρήσεις είναι και αυτές συνδεδεμένες σε μία λίστα με κεφαλή τη μεταβλητή sdflptr (vardefs.h), τα στοιχεία της οποίας συνδέονται μέσω του πεδίου sdnlink. Υπάρχει και μία μεταβλητή με το όνομα sdhead (vardefs.h), η οποία δείχνει στο πρώτο τεμάχιο των διεργασιών του χρήστη, αλλά προς το παρόν δεν έχει αξία, αφού οι διεργασίες του συστήματος δεν περιέχονται στον sdt, με αποτέλεσμα η τιμή της να μην μεταβάλλεται καθόλου κατά τη διάρκεια της εκτέλεσης του προγράμματος.

Για να μπορούμε άμεσα να δούμε την κατάσταση της μνήμης, έχουμε δύο ακόμη δομές: τη *λίστα κατειλημμένων (occupied list)* και τη *λίστα οπών (hole list)*. Για τη σελιδοποίηση θα μπορούσαμε να έχουμε έναν μόνο κοινό χάρτη ψηφίων, αλλά αυτή η τακτική έχει το πλεονέκτημα της ευκολίας χρήσης και για τεμαχισμό. Και οι δύο λίστες περιγράφουν τεμάχια, με τη λίστα κατειλημμένων να περιέχει την επιπλέον πληροφορία για τον ιδιοκτήτη του τεμαχίου. Τα στοιχεία της λίστας οπών περιγράφουν περιοχές της μνήμης οι οποίες είναι διαθέσιμες και είναι δομές τύπου *hole_table* (defs.h), με τα ακόλουθα πεδία:

long int htbase: η διεύθυνση βάσης της οπής.
long int htrange: το μέγεθος της οπής.
struct hole_table *htlink: δείκτης προς την επόμενη οπή της λίστας.

Τα στοιχεία της λίστας είναι συνδεδεμένα προς τα εμπρός και καταστρέφονται και δημιουργούνται δυναμικά. Η κεφαλή της λίστας είναι ο δείκτης hthead (vardefs.h), ίδιου τύπου με το πεδίο htlink. Επιπλέον, έχουμε

έναν ακόμη δείκτη προς τη λίστα οπών, το cycle, ο οποίος χρησιμοποιείται από την πολιτική τοποθέτησης που χρησιμοποιεί μία κυκλική μέθοδο ανεύρεσης οπών. Η λίστα κατειλημμένων είναι πολύ παρόμοια, αφού δημιουργείται δυναμικά και είναι συνδεδεμένη προς τα εμπρός με κεφαλή το δείκτη ochead (vardefs.h). Τα στοιχεία της είναι τύπου *oc_list* (defs.h) με πεδία:

int ocsection: θέση του τεμαχίου στον sdt (από εκεί έρχονται όλες οι υπόλοιπες πληροφορίες).
tchr ocproc: αναγνωριστικό της διεργασίας που κατέχει το τεμάχιο (αφού μπορούμε να δούμε κατ' ευθείαν στον sdt που περιέχει αυτή την πληροφορία, εδώ είναι περιττή, αλλά θα μπορούσε να είναι χρήσιμη αν υλοποιούσαμε καταμεριζόμενα τεμάχια, αν κρατούσε ένα ειδικό όνομα).
struct oc_list *oclink: δείκτης προς το επόμενο κατειλημμένο στοιχείο.

Και για τις δύο παραπάνω δομές, κρατάμε μετρητές τρέχοντος πλήθους στοιχείων. Προσέξτε ότι ο sdt περιέχει πληροφορίες για όλα τα τεμάχια όλων των διεργασιών χρήστη που βρίσκονται στο σύστημα (δηλαδή για όλους τους εικονικούς χώρους διευθύνσεων μαζί), ενώ συνολικά η λίστα οπών και η λίστα κατειλημμένων περιέχουν πληροφορίες για όλες τις θέσεις της φυσικής μνήμης του συστήματος (δηλαδή για τον πραγματικό χώρο διευθύνσεων). Παρατηρήστε επίσης ότι τα στοιχεία του sdt μπορούν να περιέχουν και άλλες πληροφορίες, ενώ τα στοιχεία των δύο λιστών μπορεί να είναι συνδεδεμένα με οποιονδήποτε τρόπο (κατά διεύθυνση βάσης, κατά μέγεθος, τυχαία, κλπ).

3.3.1.3. Αλγόριθμοι

3.3.1.3.1. Τοποθέτηση

Οι πολιτικές τοποθέτησης ασχολούνται με το πρόβλημα του που θα πρέπει να φορτώνονται τα νέα τεμάχια στη μνήμη. Μία βασική υπόθεση στο σύστημά μας είναι ότι οι διεργασίες του συστήματος είναι μόνιμα στη μνήμη (σε κάποιο σημείο). Έτσι, το πρόβλημα περιορίζεται στη διαχείριση των απαιτήσεων των διεργασιών των χρηστών για μνήμη και την ικανοποίησή τους με επιλογή κατάλληλων ελεύθερων περιοχών. Η πολιτική που ακολουθούμε είναι το κυκλικό πρώτο ταίριασμα, δηλαδή να διασχίζουμε τη λίστα κυκλικά, ξεκινώντας από έναν δείκτη που κινείται σε κάθε πέρασμα, και να επιλέγουμε το πρώτο τεμάχιο με επαρκές μέγεθος (cyclic first fit).

Η υλοποίηση της πολιτικής τοποθέτησης γίνεται μέσω της συνάρτησης *place* η οποία παίρνει σαν παραμέτρους την θέση του sdt (int pos) για το τεμάχιο της οποίας θέλουμε να βρούμε μία επαρκώς μεγάλη οπή (εκεί περιέχεται και το μέγεθος του τεμαχίου), καθώς και δύο δείκτες προς τη λίστα των οπών, οι οποίοι κατά την έξοδο περιέχουν την οπή που βρέθηκε

(hole_table **h_temp) και την προηγούμενή της (hole_table **h_prev). Παρατηρήστε ότι αν η οπή είναι στην κεφαλή της λίστας, ο δεύτερος δείκτης δείχνει στην τελευταία οπή της λίστας. Αν στον δείκτη της επιλεγμένης οπής επιστραφεί NULL, η τοποθέτηση έχει αποτύχει. Για την επιλογή τμήματος της μνήμης, διασχίζουμε τη λίστα κυκλικά, αρχίζοντας μία θέση μετά από τον δείκτη cycle και σταματώντας είτε όταν βρούμε ένα κατάλληλο τμήμα της μνήμης είτε όταν ο τρέχων δείκτης φτάσει στον cycle χωρίς να βρεθεί τίποτα, οπότε επιστρέφεται η τιμή NULL στο δείκτη. Χρειάζεται προσοχή κατά τη διάσχιση στο ότι η λίστα δεν είναι κυκλικά συνδεδεμένη, έτσι όταν φτάνουμε στο τέλος πρέπει να ξαναξεκινάμε από την κεφαλή. Στο τέλος της (επιτυχούς) αναζήτησης, προχωράμε κατά μία θέση τον cycle, προσέχοντας πάντα μήπως έχει φτάσει στο τέλος της λίστας, με σκοπό την επόμενη φορά να ξεκινήσουμε από την επόμενη οπή.

3.3.1.3.2. Αντικατάσταση

Το πρόβλημα της αντικατάστασης είναι η επιλογή τεμαχίων τα οποία μπορούν να απομακρυνθούν από την κύρια μνήμη (έτσι ώστε να δημιουργηθεί χώρος για άλλα τεμάχια). Βασικά στοιχεία της πολιτικής είναι ο τρόπος επιλογής τεμαχίου και η στρατηγική αναζήτησης (τοπική ή καθολική πολιτική). Η πολιτική που υλοποιήθηκε χρησιμοποιεί μία σειρά από κανόνες / κριτήρια αντικατάστασης για να επιλέξει ένα θύμα (τεμάχιο), τα οποία δίνονται με φθίνουσα σειρά χρήσης:

(1) τεμάχια ανενεργών διεργασιών (που θανατώνονται),
(2) τεμάχια διεργασιών που περιμένουν αλληλεπίδραση από το τερματικό,
(3) τεμάχια διεργασιών που περιμένουν τερματισμό παιδιών τους,
(4) τεμάχια διεργασιών χαμηλότερης προτεραιότητας (από αυτήν που έσφαλλε), και αν είναι δυνατόν που να μην ανήκουν στο σύνολο καταλλήλων,
(5) τεμάχια της διεργασίας που έσφαλλε τα οποία δεν ανήκουν στο τρέχον σύνολο εργασίας της.

Επιπλέον, σε κάθε επίπεδο έχουμε προτίμηση σε τεμάχια τα οποία δεν έχουν χρησιμοποιηθεί (αν δεν υπάρχουν διαλέγουμε ένα τυχαίο χρησιμοποιημένο). Αν κανένας από τους κανόνες δεν μπορεί να χρησιμοποιηθεί, δε διαλέγουμε τεμάχια που χρησιμοποιούνται από τη διεργασία (για να αποφύγουμε το αλώνισμα), αλλά σταματάμε τη διεργασία μέχρι να βρεθεί χώρος. Παρατηρήστε ότι οι παραπάνω κανόνες υλοποιούν αρχικά μία καθολική πολιτική και μόνο αν δεν βρεθεί κάποια διεργασία που να μην χρειάζεται αμέσως τα τεμάχιά της η πολιτική γίνεται τοπική.

Για να υλοποιηθούν τα παραπάνω, χρησιμοποιούμε τη συνάρτηση *replace* η οποία επιστρέφει μέσω δύο παραμέτρων τη θέση του τεμαχίου που

διάλεξε στον sdt (int *sdt_slot) και τη θέση της διεργασίας ιδιοκτήτη στον pdt (int *pdt_slot). Αν δεν βρεθεί κάποιο τεμάχιο, η θέση του sdt παίρνει την τιμή -1. Θα μπορούσαμε να χρησιμοποιήσουμε και τη λίστα κατειλημμένων, αλλά μέσω του πίνακα διεργασιών μπορούμε εύκολα να αντιμετωπίσουμε το θέμα της προτεραιότητας και της κατάστασης των διεργασιών, μπαίνοντας στον sdt μόνο όποτε βρίσκουμε υποψήφιες διεργασίες (σύμφωνα με τα παραπάνω κριτήρια). Η αναζήτηση γίνεται σε τρία περάσματα. Στο πρώτο, διασχίζουμε τον pdt από το τέλος προς την αρχή (αύξουσες προτεραιότητες) με στόχο να βρούμε διεργασίες που περιμένουν να θανατωθούν ή που περιμένουν αλληλεπίδραση, οι οποίες να έχουν τουλάχιστον ένα τεμάχιο στη μνήμη. Αν βρεθεί μία τέτοια διεργασία, είναι αυτή με τη χαμηλότερη δυνατή προτεραιότητα της κατηγορίας της (προσέξτε ότι η έρευνα σταματάει μόλις φτάσουμε στις διεργασίες του συστήματος). Αν δε βρεθεί τίποτα, προχωράμε σε ένα δεύτερο πέρασμα με τον ίδιο τρόπο, με στόχο τον εντοπισμό διεργασιών που βρίσκονται εν μέρει στη μνήμη και περιμένουν να θανατωθεί ένα παιδί τους. Αν δε βρεθεί πάλι τίποτα, γίνεται τρίτο όμοιο πέρασμα (προς τα πίσω), με στόχο μία οποιαδήποτε διεργασία με χαμηλότερη προτεραιότητα από αυτήν που έσφαλλε (το αναγνωριστικό της οποίας βρίσκεται στη μεταβλητή 1 processid). Παρατηρήστε ότι εδώ η έρευνα δεν σταματάει όπως πριν στις διεργασίες του συστήματος, αλλά στη διεργασία που έσφαλλε, αφού η λίστα είναι ταξινομημένη σύμφωνα με την προτεραιότητα των διεργασιών. Αν και πάλι δεν βρεθεί τίποτα, ελέγχουμε αν η διεργασία που έσφαλλε έχει τεμάχια στη μνήμη. Αν όχι, επιστρέφουμε με τον κωδικό αποτυχίας. Αν ναι, τότε θα πάρουμε ένα δικό της τμήμα μνήμης.

Αν με κάποιον από τους παραπάνω τρόπους βρεθεί μία κατάλληλη διεργασία, από τον περιγραφητή της βρίσκουμε το πρώτο της τεμάχιο (τεμάχιο 0), απ' όπου ξεκινάει η λίστα των τεμαχίων της. Από εκεί, διασχίζουμε τη λίστα αυτής της διεργασίας, ψάχνοντας για ένα τεμάχιο το οποίο να βρίσκεται στη μνήμη και να μην έχει χρησιμοποιηθεί (να μην ανήκει στο τρέχον σύνολο εργασίας της). Αν βρεθεί κάτι τέτοιο, το επιστρέφουμε. Αν όμως βρεθεί ένα τεμάχιο που είναι στη μνήμη αλλά έχει χρησιμοποιηθεί πρόσφατα, το σημειώνουμε ως υποψήφιο θύμα, και το επιστρέφουμε μόνο αν δεν βρεθεί άλλο τεμάχιο, με αποτέλεσμα ουσιαστικά να επιστρέφουμε το τελευταίο τεμάχιο της λίστας αν δεν υπάρχουν τεμάχια εκτός συνόλου εργασίας (τυχαία απομάκρυνση). Παρατηρήστε ότι ειδικά για την περίπτωση που η διεργασία που έσφαλλε είναι το θύμα, αν δεν βρεθεί τεμάχιο που να μην ανήκει στο σύνολο εργασίας της, δεν επιστρέφεται ένα χρησιμοποιούμενο τεμάχιο, αλλά η τιμή -1, με στόχο να αποφευχθεί το αλώνισμα.

3.3.1.3.3. Συμπύκνωση

Η συμπύκνωση αντιμετωπίζει το πρόβλημα του κατακερματισμού της μνήμης, που παρουσιάζεται στα συστήματα τεμαχισμού. Το πρόβλημα αυτό έχει σαν αποτέλεσμα να μην μπορεί να τοποθετηθεί ένα τεμάχιο στη μνήμη, παρά το ότι υπάρχει η απαιτούμενη ελεύθερη μνήμη, λόγω του ότι είναι κατακερματισμένη σε οπές. Κατά τη συμπύκνωση, μετακινούμε τα κατειλημμένα τεμάχια προς το ένα άκρο της μνήμης, δημιουργώντας μία μόνο μεγάλη οπή στο άλλο άκρο. Στο σύστημά μας, για αν υποβοηθηθεί η συμπίεση, έχουμε διατάξει τις λίστες οπών και κατειλημμένων σύμφωνα με τη διεύθυνση βάσης των στοιχείων τους, έτσι ώστε η διάσχισή τους να δίνει την εικόνα ότι "ανεβαίνουμε" στη μνήμη. Επιπλέον, έχουμε αποφασίσει να κάνουμε συμπίεση μόνο όταν είναι αναγκαίο και όχι σε τακτά διαστήματα ή σε κάθε στιγμή απελευθέρωσης μνήμης.

Η συμπύκνωση υλοποιείται μέσω της συνάρτησης *shuffle*, η οποία δεν παίρνει παραμέτρους. Η shuffle τοποθετεί όλα τα κενά τεμάχια της μνήμης έτσι ώστε να είναι γειτονικά στο ένα άκρο της μνήμης (στο επάνω), δημιουργώντας μία μόνο ελεύθερη οπή στο άλλο άκρο της μνήμης. Προφανώς επηρεάζονται και η λίστα οπών και η λίστα κατειλημμένων. Επειδή και οι δύο λίστες είναι ταξινομημένες κατά αύξουσες διευθύνσεις βάσης τεμαχίων, τις διατρέχουμε και τις δύο ταυτόχρονα, με βάση τη λίστα κατειλημμένων. Για κάθε οπή που βρίσκουμε, εξετάζουμε αν είναι η τελευταία, ελέγχοντας αν το τέλος της συμπίπτει με το τέλος της μνήμης. Αν αυτό συμβαίνει, η συμπύκνωση έχει ολοκληρωθεί και έχουμε μία μόνο οπή. Αν όχι, τότε προχωράμε από την τρέχουσα θέση στη λίστα κατειλημμένων μέχρι να βρούμε το κατειλημμένο τεμάχιο το οποίο βρίσκεται ακριβώς πάνω από την οπή. Παρατηρήστε ότι λόγω του τρόπου με τον οποίο απελευθερώνονται τα τεμάχια (συνάρτηση release_section), δεν υπάρχει περίπτωση να έχουμε γειτονικές οπές. Έτσι, αφού η οπή δεν είναι στο άκρο της μνήμης θα πρέπει να υπάρχει οπωσδήποτε γειτονικό (προς τα επάνω) κατειλημμένο τεμάχιο. Μόλις βρεθεί το γειτονικό τεμάχιο, αντιγράφεται από το σημείο της μνήμης από όπου αρχίζει η οπή, καθυστερώντας το σύστημα τόσο χρόνο όσος απαιτείται για να αντιγραφούν τα bytes του κατειλημμένου τεμαχίου από θέση σε θέση της μνήμης. Περνώντας στον sdt, αλλάζουμε τη διεύθυνση βάσης του τεμαχίου που αντιγράψαμε (το μήκος, αλλά και ο δείκτης σύνδεσης στη λίστα κατειλημμένων μένουν ως έχουν). Η οπή κρατάει το μήκος της αλλά αλλάζει διεύθυνση βάσης. Η τελευταία κίνηση είναι να εξετάσουμε αν πάνω από την κατειλημμένη θέση υπήρχε οπή, δηλαδή αν η επόμενη οπή της λίστας συνορεύει με την τρέχουσα, μετά τις τροποποιήσεις. Αν δεν υπάρχει άλλη οπή, σταματάμε (αυτό σημαίνει ότι δεν υπάρχει εγγύηση ότι η τελική οπή, που είναι βέβαια πάντοτε η μοναδική που μένει, θα είναι στο άκρο της μνήμης, αφού μπορεί η οπή που μένει να είναι η τελευταία στη λίστα και να έχει πάνω από

αυτήν κατειλημμένα τεμάχια). Αν η οπή δεν είναι γειτονική, επαναλαμβάνουμε την ανακύκλωση χωρίς να προχωρήσουμε τους δείκτες. Αν όμως είναι γειτονική, συνενώνουμε τις δύο οπές στην πρώτη καταχώρηση και αποσυνδέουμε και απελευθερώνουμε την καταχώρηση της δεύτερης οπής, ενημερώνοντας και τους μετρητές. Επειδή η οπή χάνεται, πρέπει να προσέξουμε μήπως έδειχνε σε αυτήν ο δείκτης cycle και, αν χρειάζεται, να τον προχωρήσουμε, προσέχοντας αν έχει φτάσει στο τέλος της λίστας (οπότε και τον βάζουμε πάλι στην αρχή).

3.3.1.3.4. Απελευθέρωση

Κατά την απομάκρυνση τεμαχίων από την κύρια μνήμη, πρέπει να ενημερωθούν οι λίστες κατειλημμένων και οπών, καθώς και ο sdt. Αυτά γίνονται με την κλήση της συνάρτησης *release_section*, η οποία παίρνει σαν παράμετρο τη θέση του τεμαχίου που απομακρύνεται στον sdt (int slot). Αρχικά ενημερώνει την κατάσταση του τεμαχίου και το μέγεθος της διαθέσιμης μνήμης (καθολική μεταβλητή). Η απελευθέρωση σημαίνει αρχικά τη δημιουργία μίας νέας οπής, η οποία σχετίζεται με ένα νέο στοιχείο στη λίστα οπών αυξάνοντας το πλήθος τους, που παίρνει τη βάση και τη διεύθυνση του τεμαχίου και εισάγεται στη λίστα οπών κατά τη διεύθυνση της βάσης του, μέσω απλής διάσχισης. Κατά την εισαγωγή, προσέχουμε πάντα μήπως πρέπει να ενημερωθεί ο δείκτης cycle, πράγμα που χρειάζεται μόνο αν ήταν πριν NULL, δηλαδή αν η λίστα ήταν κενή και αυτό είναι το πρώτο στοιχείο (αυτό, με τη σειρά του, μπορεί να συμβαίνει μόνο όταν το στοιχείο αυτό γίνεται νέα κεφαλή της λίστας και όχι στην περίπτωση που βρίσκεται στο μέσο της).

Στη συνέχεια, ελέγχουμε τη λίστα οπών για γειτονικές οπές, σε όλο της το μήκος, τις οποίες αν βρούμε τις συνενώνουμε (αυτό έχει σαν αποτέλεσμα την αδυναμία ύπαρξης γειτονικών οπών κατά τη διάρκεια της συμπύκνωσης). Για να γίνει αυτό, διασχίζουμε τη λίστα εξετάζοντας κάθε γειτονικό ζεύγος στοιχείων, ως προς το αν είναι γειτονικά. Αν συνορεύουν, το τεμάχιο στην υψηλότερη θέση αποσυνδέεται από τη λίστα και το κατώτερο τεμάχιο παίρνει το συνολικό μέγεθος των δύο οπών. Το στοιχείο για το δεύτερο τεμάχιο καταστρέφεται και οι καταχωρήσεις στη λίστα οπών μειώνονται. Λόγω της καταστροφής, ελέγχουμε αν ο δείκτης cycle έχει έγκυρη (υπαρκτή) τιμή μέσα στη λίστα και τον ενημερώνουμε κατάλληλα. Σε κάθε περίπτωση, το παράθυρο εξέτασης μετατοπίζεται στις επόμενες θέσεις (με τη διαφορά ότι αν έγινε συνένωση ο πρώτος δείκτης δεν προχωράει, γιατί η θέση αυτή δεν έχει συγκριθεί με την καινούρια επόμενή της).

Τέλος, διασχίζεται και η λίστα κατειλημμένων για να βρεθεί η θέση του τεμαχίου που φεύγει από τη μνήμη, να διαγραφεί το αντίστοιχο στοιχείο από τη λίστα, και να μειωθούν οι καταχωρήσεις στη λίστα κατειλημμένων.

3.3.1.3.5. Κύρια διαδικασία

Η κύρια συνάρτηση *store_manager* του διαχειριστή της μνήμης, είναι το μόνο τμήμα του προγράμματος που μεταβάλλει το πλήθος καταχωρήσεων στον sdt, με αποτέλεσμα να περιέχει τοπικές στατικές μεταβλητές για το πλήθος των καταχωρήσεων και των ελεύθερων θέσεων σε αυτόν. Και η συνάρτησή αυτή είναι μία ρουτίνα με πολλά σημεία ενεργοποίησης. Επιπλέον, συνολικά ο διαχειριστής της μνήμης μπορεί να δεχτεί τρεις εντολές από τις ενδιαφερόμενες διεργασίες:

DEFINE_SECTIONS	παραχώρηση θέσεων (του sdt) για τεμάχια μίας διεργασίας
UNDEFINE_SECTIONS	αποδέσμευση θέσεων (του sdt) από τεμάχια μίας διεργασίας
GET_SECTION	μεταφορά ενός τεμαχίου στη μνήμη

Στην πρώτη είσοδο (είσοδο 0), ο διαχειριστής της μνήμης αρχικοποιεί τα πλήθη καταχωρήσεων σε λίστες οπών (σε 1), κατειλημμένων και sdt (σε 0), καθώς και το πλήθος ελεύθερων θέσεων στον sdt (όλες οι θέσεις). Από πλευράς sdt, χτίζεται η λίστα ελευθέρων θέσεων, με όλες τις θέσεις παρούσες και κεφαλή το δείκτη sdflptr. Ο πίνακας οπών περιέχει αρχικά όλη τη διαθέσιμη μνήμη σαν μία οπή, ενώ η λίστα κατειλημμένων είναι κενή. Αρχικά όλη η μνήμη είναι διαθέσιμη, αφού όπως είπαμε οι διεργασίες του συστήματος θεωρείται ότι είναι αποθηκευμένες μόνιμα κάπου αλλού και δεν περιέχονται στον sdt.

Στην κύρια είσοδο (είσοδο 1), ο διαχειριστής περιμένει ένα μήνυμα από το γονέα (θύρα 0), το δημιουργό διεργασιών (θύρα 1), το διαχειριστή της συσκευής ανταλλαγής (θύρα 2), το διαχειριστή της ΚΜΕ (θύρα 3), το διαχειριστή του δίσκου (θύρα 4) ή από μία άλλη διεργασία μέσω μίας αόριστης θύρας. Όταν λάβει το μήνυμα (είσοδος 2), αποθηκεύει τη θύρα αποστολής και το αναγνωριστικό του αποστολέα σε στατικές μεταβλητές και βρίσκει τη θέση του αποστολέα στον pdt και την αρχή του πίνακα "σκιάς" του. Στη συνέχεια, ανάλογα με την εντολή που έστειλε ο αποστολέας, γίνεται διακλάδωση σε μία από τις τρεις υποεισόδους, που περιγράφονται χωριστά παρακάτω.

Η υποείσοδος DEFINE_SECTIONS καλείται όταν ο δημιουργός διεργασιών ζητάει την παραχώρηση θέσεων του sdt για τα τεμάχια μίας διεργασίας που δημιουργεί. Αν τα τεμάχια της διεργασίας είναι περισσότερα από τις ελεύθερες θέσεις στον sdt, τότε επιστρέφουμε mcommand=0 (υπερχείλιση πινάκων) και στέλνουμε το μήνυμα απάντησης στον αποστολέα, επιστρέφοντας στην κύρια είσοδο για την επόμενη αίτηση. Αλλιώς, ξεκινώντας από το δείκτη στις ελεύθερες θέσεις του sdt και ακολουθώντας τη λίστα

ελευθέρων θέσεων, ορίζουμε τόσα τεμάχια όσα χρειάζεται η διεργασία, χρησιμοποιώντας στοιχεία του πίνακα "σκιάς" για το μέγεθος του τεμαχίου (η βάση ορίζεται κατά την προσκόμιση του τεμαχίου στη μνήμη). Η προσπέλαση ορίζεται μέσω του τυχαίου αριθμού της διεργασίας, ο ιδιοκτήτης από το αναγνωριστικό του μηνύματος, και το τεμάχιο σημειώνεται ως απών από μνήμη και συσκευή ανταλλαγής και εκτός συνόλου εργασίας. Προχωρώντας, τα τεμάχια συνδέονται μεταξύ τους σε λίστα (μέσω sdnlink) και το τελευταίο παίρνει την τιμή δείκτη -1. Ο δείκτης sdflptr δείχνει στο επόμενο ελεύθερο στοιχείο και οι ελεύθερες και οι κατειλημμένες θέσεις προσαρμόζονται κατάλληλα. Η διεργασία σημειώνεται ως εκτός μνήμης (στον pdt) και στέλνεται απάντηση για τη σωστή ολοκλήρωση (mcommand=1) στον αποστολέα, η οποία περιέχει το θέση του τεμαχίου 0 στο word3, με επιστροφή στην κύρια είσοδο. Παρατηρήστε ότι δεν έχουμε προσκόμιση τεμαχίων, γιατί τα τεμάχια τοποθετούνται στη μνήμη όταν η διεργασία σφάλλει για αυτά (σελιδοποίηση με αίτηση, demand paging).

Η είσοδος UNDEFINE_SECTIONS καλείται όταν ο διαχειριστής της ΚΜΕ ζητάει τη διαγραφή των τεμαχίων μίας διεργασίας από τον sdt. Το μήνυμα περιέχει και τη θέση του τεμαχίου 0 της διεργασίας, την κεφαλή δηλαδή της λίστας των τεμαχίων της. Έτσι, διασχίζουμε τη λίστα αυτή μέσω του πεδίου sdnlink και, για όσα τεμάχια βρίσκονται στη μνήμη, καλούμε τη συνάρτηση release_section, η οποία ενημερώνει τις λίστες οπών και κατειλημμένων περιοχών. Η διεργασία σημειώνεται ως εκτός μνήμης και το πλήθος κατειλημμένων και ελευθέρων περιοχών προσαρμόζεται ανάλογα. Επειδή τα τεμάχια είναι ήδη συνδεδεμένα μεταξύ τους, αρκεί να τα συνδέσουμε στην αρχή της λίστας ελευθέρων και να βάλουμε την κεφαλή να δείχνει στο παλιό τεμάχιο 0. Τέλος, απαντάμε με αποστολή μηνύματος στο διαχειριστή της ΚΜΕ και επιστρέφουμε στην κύρια είσοδο. Παρατηρήστε ότι δε γράφουμε τίποτα στο δίσκο, ούτε απελευθερώνουμε χώρο στην περιοχή ανταλλαγής (αν και θα έπρεπε).

Η είσοδος GET_SECTION καλείται όταν συμβεί ένα σφάλμα τεμαχίου το οποίο έχει σαν αποτέλεσμα την αποστολή μηνύματος στο διαχειριστή μνήμης για να προσκομίσει το τεμάχιο στη μνήμη. Το μήνυμα φαίνεται να έρχεται από τη διεργασία που έσφαλλε και περιέχει τη θέση του τεμαχίου στον sdt. Η εγκυρότητα του μηνύματος ελέγχεται (θετικό μέγεθος τεμαχίου, απουσία τεμαχίου από τη μνήμη) και προχωράμε στην είσοδο 3. Η είσοδος 3 σημειώνει ότι δεν έχει γίνει συμπύκνωση μνήμης (κρατάμε τέτοια μεταβλητή ώστε η συμπύκνωση να μην γίνεται περισσότερες από μία φορές σε κάθε αναζήτηση για χώρο στη μνήμη, αν δεν έχει παρεμβληθεί απελευθέρωση μνήμης) και προχωράει στην είσοδο try_again, όπου ξεκινούν όλες οι προσπάθειες για την τοποθέτηση των τεμαχίων. Ελέγχουμε αν υπάρχει ελεύθερη μνήμη για το τεμάχιο. Αν όχι, προχωράμε στην είσοδο throw για να

απελευθερώσουμε μνήμη. Αν ναι, καλούμε τη συνάρτηση place για να τοποθετήσει το τεμάχιο. Αν η place καταφέρει να βρει χώρο, ενημερώνουμε τον sdt για τη θέση του τεμαχίου στη μνήμη και στη συνέχεια ενημερώνουμε τον πίνακα οπών, προσαρμόζοντας το μέγεθος της οπής (αφαιρούμε το χώρο του νέου τεμαχίου) ή αποσυνδέοντας την οπή αν έχει ίδιο μέγεθος με το τεμάχιο (σε σύστημα σελιδοποίησης, η κάθε οπή είναι είτε ακριβώς ίση με μία σελίδα ή έχει μέγεθος ένα ακέραιο πολλαπλάσιο της σελίδας), όπου και απελευθερώνουμε το σχετικό στοιχείο της οπής και προσέχουμε να μην μείνει αόριστος ο δείκτης cycle ή εκτός λίστας. Στη συνέχεια, ενημερώνουμε τη λίστα κατειλημμένων, δημιουργώντας ένα νέο στοιχείο με τα χαρακτηριστικά του νέου τεμαχίου, το οποίο εισάγουμε στη λίστα κατειλημμένων σύμφωνα με τη διεύθυνση της βάσης του. Τελικά, ανάλογα με το αν το τεμάχιο βρίσκεται στην περιοχή ανταλλαγής ή μόνο στο δίσκο, ετοιμάζουμε ένα μήνυμα για ανάγνωση, συμπεριλαμβάνοντας σε αυτό το μέγεθος του τεμαχίου προς ανάγνωση, και τελικά το στέλνουμε στην κατάλληλη διεργασία (διαχειριστή δίσκου ή συσκευής ανταλλαγής), προχωρώντας στην είσοδο 6.

Αν μετά από την κλήση της place δεν βρέθηκε χώρος τοποθέτησης, αυτό σημαίνει ότι δεν υπάρχει κατάλληλη οπή, οπότε αν δεν έχει γίνει συμπύκνωση, την κάνουμε τώρα και επιστρέφουμε στην είσοδο try_again για μία νέα προσπάθεια. Αν έχει γίνει ήδη η συμπίεση, θα έπρεπε να έχει γίνει η τοποθέτηση γιατί υπάρχει αρκετός χώρος στη μνήμη και είναι συνεχόμενος. Αν από την αρχή η μνήμη ήταν ανεπαρκής, είδαμε ότι καλείται η είσοδος throw, η οποία καλεί τη συνάρτηση replace για να βρει ένα τεμάχιο θύμα. Αν δεν βρεθεί τέτοιο τεμάχιο, η διεργασία σημειώνεται ως σταματημένη για σύνολο εργασίας και απελευθερώνεται, με ταυτόχρονη ενημέρωση των στατιστικών του συστήματος, και προχωρούμε στην είσοδο 8 για καταστροφή του μονοπατιού από τη διεργασία αυτή. Αν όμως βρεθεί τεμάχιο και η διεργασία που το κατέχει δεν περιμένει να θανατωθεί, ελέγχουμε αν το τεμάχιο μπορεί να γραφτεί ή δεν βρίσκεται στην περιοχή ανταλλαγής και έχει χρησιμοποιηθεί πρόσφατα, οπότε πρέπει να το γράψουμε στη συσκευή ανταλλαγής. Αν ισχύει αυτό, η διεργασία σημειώνεται ως σταματημένη για απομάκρυνση τεμαχίου και στέλνουμε ένα μήνυμα στο διαχειριστή της συσκευής ανταλλαγής να αποθηκεύσει το τεμάχιο, προχωρώντας μετά στην είσοδο 4. Αν δεν χρειάζεται αποθήκευση, προχωράμε κατ' ευθείαν στην ενημέρωση των πινάκων (είσοδος update_tables). Η είσοδος 4 περιμένει την απάντηση από το διαχειριστή της συσκευής ανταλλαγής και προχωράει στην είσοδο 5 όπου το τεμάχιο σημειώνεται ως παρόν στη συσκευή ανταλλαγής. Στη συνέχεια, προχωράμε άμεσα στην είσοδο update_tables, όπως και παραπάνω.

Η είσοδος update_tables καλείται όταν έχουν γίνει οι μεταφορές και μένει μόνο η ενημέρωση των πινάκων του συστήματος. Έτσι, καλούμε τη

συνάρτηση release_section για το θύμα και αυξάνουμε τα τεμάχιά του εκτός μνήμης, προσέχοντας μήπως έχει απομακρυνθεί πλήρως η διεργασία από τη μνήμη. Αν το τεμάχιο ανήκει σε μία ζωντανή διεργασία η οποία το χρειάζεται, αυξάνουμε τα τεμάχια του συνόλου εργασίας της που βρίσκονται εκτός μνήμης και αν το τεμάχιο δεν χρησιμοποιείται εκείνη τη στιγμή, δρομολογούμε ένα σφάλμα τεμαχίου για αυτό πριν από την επόμενη αλλαγή του συνόλου εργασίας της. Για να γίνει αυτό, διασχίζουμε τη λίστα γεγονότων της διεργασίας μέχρι να βρούμε το γεγονός αλλαγής συνόλου εργασίας και ορίζουμε ως κάτω όριο χρόνου το χρόνο που ήδη έχει χρησιμοποιήσει η διεργασία και ως όριο μεσοδιαστήματος μέχρι το σφάλμα τον υπόλοιπο χρόνο εκτέλεσης ή το χρόνο μέχρι το επόμενο γεγονός αλλαγής συνόλου εργασίας. Τελικά το γεγονός ορίζεται να συμβεί σε ένα τυχαίο μεσοδιάστημα εντός των ορίων και δρομολογείται. Σε κάθε περίπτωση, η διεργασία σημειώνεται ως μη σταματημένη και επιστρέφουμε στην είσοδο 3, και όχι στην try_again, για να ξαναεπιχειρήσουμε την τοποθέτηση (λόγω της απελευθέρωσης, δεν έχει γίνει συμπύκνωση).

Στην είσοδο 6 ερχόμαστε μετά από αίτηση προσκόμισης προς το δίσκο ή τη συσκευή ανταλλαγής. Έτσι περιμένουμε το μήνυμα απάντησης και προχωράμε στην είσοδο 7, όπου έχοντας λάβει την απάντηση, ενημερώνουμε τη διαθέσιμη μνήμη, σημειώνουμε ότι το τεμάχιο και η διεργασία βρίσκονται στη μνήμη και απελευθερώνουμε τη διεργασία, μειώνοντας το τεμάχιά της που είναι εκτός μνήμης. Τελικά προχωράμε στην είσοδο 8, η οποία καλείται σε κάθε περίπτωση που θέλουμε να καταστρέψουμε ένα μονοπάτι. Μετά την κατάλληλη κλήση, επιστρέφουμε στην κύρια είσοδο, περιμένοντας την επόμενη αίτηση.

3.3.2. Διαχειριστής ΚΜΕ

3.3.2.1. Εισαγωγή

Η βασική λειτουργία του διαχειριστή της ΚΜΕ είναι να καταμερίζει το χρόνο της ΚΜΕ μεταξύ των διεργασιών στο σύστημα, έτσι ώστε να επιτυγχάνεται καλή αξιοποίηση των πόρων του συστήματος και αποδεκτή απόδοση για τις διεργασίες. Για να γίνει ο καταμερισμός αποδοτικά, ο διαχειριστής της ΚΜΕ αποφασίζει όχι μόνο για το ποιες διεργασίες θα εκτελούνται (σύνολο καταλλήλων), αλλά και για την προτεραιότητα εκτέλεσής τους, προετοιμάζοντας την ουρά του διανομέα, έτσι ώστε, όποτε εκτελείται ο διανομέας, να μπορεί να αποφασίζει ποια διεργασία θα πάρει τον έλεγχο της ΚΜΕ.

Τα βασικά γεγονότα που προκαλούν αφύπνιση του διαχειριστή της ΚΜΕ είναι η δημιουργία και η διαγραφή διεργασιών καθώς και οι περιοδικές

διακοπές του χρονομέτρου (κατά τις οποίες γίνεται η προετοιμασία της λίστας του διανομέα και ο προσδιορισμός του συνόλου των καταλλήλων, δηλαδή ο μεσοχρόνιος χρονοπρογραμματισμός). Στο σύστημά μας, ο βραχυχρόνιος χρονοπρογραμματισμός γίνεται από το διανομέα (μέσα στον πυρήνα), ενώ ο μακροχρόνιος χρονοπρογραμματισμός γίνεται από το χρονοπρογραμματιστή των εργασιών.

Ο χρονοπρογραμματιστής που χρησιμοποιείται στο σύστημά μας, οδηγείται από συναρτήσεις πολιτικής, με αποτέλεσμα να είναι πολύ ευέλικτος, αφού οι συναρτήσεις πολιτικής μπορεί να κατασκευάζονται με τέτοιον τρόπο, ώστε να αντανακλούν τις απαιτήσεις κάθε συγκεκριμένης εγκατάστασης. Επειδή οι παράμετροι των συναρτήσεων διαβάζονται κατά την αρχική φόρτωση, ο πειραματισμός πάνω στην αποτελεσματικότητά τους είναι πολύ εύκολος. Χωρίς να προχωρήσουμε σε λεπτομερή ανάλυση του θέματος των συναρτήσεων πολιτικής, αναφέρουμε απλά ότι χρειάζεται κάποια μέθοδος μέτρησης των πόρων που έχει καταναλώσει κάθε εργασία κατά την εκτέλεσή της, πράγμα που στο σύστημά μας γίνεται με την μετατροπή όλων των πόρων σε μονάδες χρόνου.

Για να πετύχουμε την εκτίμηση του φόρτου που επιβάλλουν στο σύστημα οι διεργασίες, χρησιμοποιούμε το σύνολο εργασίας τους, θεωρώντας σαν περιοριστικό παράγοντα τη μνήμη. Κάθε φορά που μπαίνουμε στο χρονοπρογραμματιστή, εκτιμούμε το σύνολο εργασίας της κάθε διεργασίας, με αποτέλεσμα όταν προσδιορίζουμε το σύνολο καταλλήλων να αρκούμαστε στις διεργασίες που υπερβαίνουν κατά λίγο τη διαθέσιμη μνήμη (όλες μαζί). Συνδυάζοντας τις δύο εργασίες του χρονοπρογραμματιστή, κάθε φορά που αυτός εκτελείται υπολογίζει τους κρίσιμους χρόνους των διεργασιών που εκτελέστηκαν στο προηγούμενο διάστημα, διατάσσει τη λίστα του διανομέα και υπολογίζει τα σύνολα εργασίας και το μέγεθος του συνόλου καταλλήλων.

Μία τελευταία αρμοδιότητα του διαχειριστή της ΚΜΕ, είναι να απαντά σε ερωτήσεις του χρονοπρογραμματιστή εργασιών σχετικά με την κατάσταση του συστήματος. Αυτή η εργασία συνδυάζεται με τη λογιστική χρέωση των διεργασιών που γίνεται κατά τη διαγραφή τους από το σύστημα. Οι πληροφορίες από το διαχειριστή της ΚΜΕ (μεσοχρόνιο προγραμματιστή) επιτρέπουν στο (μακροχρόνιο) χρονοπρογραμματιστή εργασιών να εκτιμήσει το αποτέλεσμα της εισαγωγής μίας νέας εργασίας στο σύστημα. Στο χαμηλότερο επίπεδο βέβαια, οι πληροφορίες προέρχονται από την ενημέρωση των πινάκων του συστήματος που γίνονται σε κάθε κλήση του διανομέα (βραχυχρόνιου χρονοπρογραμματιστή).

Ο κύριος όγκος του κώδικα του διαχειριστή της ΚΜΕ (κύρια και βοηθητικές συναρτήσεις) βρίσκεται στο αρχείο *cpuman.c*, ενώ ο κώδικας για

τις συναρτήσεις πολιτικής (αρχικοποίηση, μέτρηση πόρων και εύρεση κρίσιμων χρόνων) βρίσκεται στο αρχείο *polfn.c*.

3.3.2.2. Δομές Δεδομένων

Επειδή στο σύστημά μας έχουμε πολλά επίπεδα προτεραιότητας διεργασιών, οι συναρτήσεις πολιτικής έχουν διαφορετικούς συντελεστές για το κάθε ένα. Οι συναρτήσεις που χρησιμοποιούνται έχουν τη μορφή:

$R*m_H$ για $0<=R<=t_H$
$t_c(R)=R*m_{INF}+t_H*(m_H-m_{INF})$ για $t_H<=R<=t_{INF}$
$t_{INF}*m_{INF}+t_H*(m_H-m_{INF})$ για $R>=t_{INF}$

Στα παραπάνω, t_c είναι ο κρίσιμος χρόνος της διεργασίας για R μονάδες εξυπηρέτησης. Αυτή η μορφή συναρτήσεων δίνεται για όλα τα επίπεδα προτεραιότητας, με αποτέλεσμα να έχουμε έναν πίνακα για τον κάθε συντελεστή, με επτά ενεργά στοιχεία (για επτά επίπεδα προτεραιότητας, αφού οι διεργασίες του συστήματος δεν μπαίνουν σε αυτό το σχήμα, αν και αυτή η έκδοση χρησιμοποιεί μόνο τρία) τύπου float: mh, minf, th, tinf (αρχείο polfn.c). Το υψηλότερο επίπεδο αναφέρεται στις διεργασίες αλληλεπίδρασης.

Επιπλέον, έχουμε μεταβλητές για το πλήθος καταχωρήσεων στον pdt (int cpdtentries), για τις κλήσεις του χρονοπρογραμματιστή (int n), το πλήθος κλήσεων μεταξύ διαδοχικών ερωτήσεων από το χρονοπρογραμματιστή εργασιών (int no_of_intervals), τη λίστα ελευθέρων θέσεων στον pdt (int flptr), το χρόνο της τελευταίας κλήσης του χρονοπρογραμματιστή (double oldtime) και τους πόρους ανά μεσοδιάστημα (float resource_per_interval). Οι χρόνοι είναι σημαντικοί, γιατί ο χρονοπρογραμματιστής κάνει εκτιμήσεις των συνόλων εργασίας και των προσφερθέντων πόρων χρησιμοποιώντας εξομαλυμένες μετρήσεις από τα προηγούμενα χρονικά διαστήματα. Όλες οι παραπάνω μεταβλητές περιέχονται στο αρχείο cpuman.c.

3.3.2.3. Αλγόριθμοι

3.3.2.3.1. Συναρτήσεις πολιτικής

Η πρώτη από τις συναρτήσεις είναι η *initialize_policy_functions* η οποία παίρνει σαν παράμετρο τα επίπεδα προτεραιότητας τα οποία χρησιμοποιεί το σύστημα (int levels) και διαβάζει τους συντελεστές των συναρτήσεων πολιτικής για τα επίπεδα αυτά, ξεκινώντας από το επίπεδο 1 (αλληλεπιδραστικές διεργασίες). Η δεύτερη συνάρτηση είναι η *resource_since_time*, η οποία υπολογίζει την εξυπηρέτηση που έλαβε μία διεργασία από τη χρονική στιγμή old_time (int) μέχρι τη στιγμή της κλήσης.

Επιπλέον, δίνονται σαν παράμετροι ο χρόνος εκτέλεσης από τη στιγμή εκείνη (int *r), η τρέχουσα εκτίμηση του συνόλου εργασίας της διεργασίας (long int ws) και το πλήθος των μηνυμάτων από τις διεργασίες του συστήματος (int cur_sends[]), πάντα από το χρόνο old_time. Η συνάρτηση υπολογίζει το διάστημα για το οποίο γίνεται η εκτίμηση και υπολογίζει τους πόρους χρησιμοποιώντας:

(1) το χρόνο εκτέλεσης (στην *r)
(2) το μνήμη που καταλαμβάνεται για το διάστημα αυτό, σαν ποσοστό της συνολικής μνήμης (βγαίνει από το ws)
(3) τις απαντήσεις από το σύστημα, σταθμίζοντας με το συνολικό χρόνο εκτέλεσης (από τον pdt), την απασχόληση των ουρών του συστήματος και το πλήθος αποστολών (από τον πίνακα cur_sends).

Τελικά, το σύνολο πόρων επιστρέφεται μέσω της r στον καλούντα. Η τελευταία συνάρτηση είναι η *critical_time* η οποία υπολογίζει τον κρίσιμο χρόνο μίας διεργασίας, η θέση της οποίας στον pdt δίνεται από την παράμετρο slot (int). Η συνάρτηση υπολογίζει τον χρόνο εκτέλεσης στο τελευταίο διάστημα (πιο πρόσφατη αλληλεπίδραση) και καλεί τη resource_since_time για το χρόνο από την τελευταία αλληλεπίδραση με τις τρέχουσες εκτιμήσεις για τα μηνύματα, το χρόνο εκτέλεσης και το μέγεθος του συνόλου εργασίας (πρέπει να έχει ήδη ενημερωθεί) και ανάλογα με τον τύπο της διεργασίας, υπολογίζει τον κρίσιμο χρόνο της σύμφωνα με τις συναρτήσεις πολιτικής που αναφέραμε και το r που δίνει η resource_since_time. Τελικά, ενημερώνει το κατάλληλο πεδίο στον pdt με το νέο κρίσιμο χρόνο της διεργασίας.

3.3.2.3.2. Διαγραφή διεργασιών

Η συνάρτηση *delete_process* καλείται με παράμετρο τη θέση της διεργασίας που θα διαγραφεί στον pdt (int slot) και ενημερώνει τους πίνακες του συστήματος για το γεγονός. Η διεργασία υποτίθεται ότι έχει ήδη εμποδιστεί (έτσι, ενημερώνει τους σχετικούς χρόνους). Αν χρειάζεται, τυπώνεται το γεγονός και οι χρόνοι που πέρασε η διεργασία σε κάθε κατάσταση. Ο λόγος για τον οποίο τερματίστηκε η διεργασία δίνεται στο τρέχον μήνυμα, σε δύο επίπεδα. Ο κύριος λόγος, δίνεται στο word6, ως εξής:

0 Κανονικός τερματισμός
1 Υπέρβαση χρονικού ορίου
2 Σφάλμα ε/ε
3 Αποτυχία δημιουργίας παιδιού
4 Διαγραφή γονέα

Στην περίπτωση 3, έχουμε και τον δευτερεύοντα λόγο στο word3 (την αιτία δηλαδή της αποτυχίας δημιουργίας του παιδιού) ως εξής:

1 Υπερχείλιση pdt
2 Υπερχείλιση sdt
3 Σφάλμα ε/ε κατά τη δημιουργία

Αν έχουμε ανώμαλο τερματισμό, ενημερώνουμε τους μετρητές χαμένων διεργασιών, αναλόγως της αιτίας. Στη συνέχεια, χρεώνουμε την εργασία που δημιούργησε τη διεργασία, ανάλογα με τους πόρους που χρησιμοποίησε στο τελευταίο διάστημα. Αυτό σημαίνει υπολογισμό του πιο πρόσφατου χρόνου εκτέλεσης και του πιο πρόσφατου συνόλου τεμαχίων που χρησιμοποιήθηκαν (από τον sdt). Το σύνολο εργασίας για αυτή την περίοδο υπολογίζεται με τον τύπο:

νέο σύνολο = a * συνολικό μέγεθος σε χρήση + (1-a) * παλιό σύνολο

Το a βρίσκεται διαιρώντας τον πρόσφατο χρόνο εκτέλεσης με το συνολικό χρόνο εκτέλεσης. Με την εκτίμηση αυτή, καλούμε την resource_since_time για να βρούμε τους πόρους που χρησιμοποιήθηκαν και να ενημερώσουμε έτσι το αντίστοιχο πεδίο χρέωσης στον pdt.

Στη συνέχεια, βρίσκουμε την κατηγορία της διεργασίας και ενημερώνουμε τους συνολικούς μετρητές του συστήματος για τους χρόνους σε κάθε κατάσταση, το χρόνο στο σύστημα, το σύνολο εργασίας, το πλήθος αποστολών και τους πόρους, διακρίνοντας ανάμεσα στην περίπτωση που αυτή είναι η πρώτη διεργασία που ολοκληρώνεται ή μία άλλη διεργασία. Αυξάνουμε το πλήθος των διεργασιών που εκτελέστηκαν και χρεώνουμε την εργασία για τους πόρους της διεργασίας. Τέλος, διαγράφουμε τα μηνύματα που εκκρεμούν στις ουρές της, ενημερώνοντας και τους μετρητές του συστήματος, και αποσυνδέουμε τη διεργασία από τη λίστα του διανομέα (προσέχοντας μήπως βρίσκεται στο τέλος της), προσθέτοντας τη θέση του περιγραφητή της στη λίστα ελευθέρων (μονή σύνδεση, όχι διπλή όπως στο διανομέα) και ενημερώνοντας το πλήθος ελεύθερων θέσεων του pdt. Παρατηρήστε ότι επειδή η διεργασία έχει σαν αναγνωριστικό και τη θέση της στον pdt και έναν μοναδικό ακέραιο, αρκεί να κάνουμε τον ακέραιο αυτό -1 για να βεβαιωθούμε ότι οι μελλοντικές αποστολές μηνύματος θα θεωρούνται άκυρες, μέχρι τουλάχιστον να καταληφθεί ξανά η θέση (οπότε θα πάρει και έναν νέο αριθμό).

3.3.2.3.3. Μεσοχρόνιος χρονοπρογραμματιστής

Η συνάρτηση *manager* καλείται όποτε χρειάζεται να αναθεωρηθεί η λίστα του διανομέα, την οποία αναδιατάσσει σύμφωνα με τους κρίσιμους χρόνους των διεργασιών στο σύστημα. Σε κάθε μεσοδιάστημα, αφού μηδενίσουμε τους πόρους που προσφέρθηκαν πρόσφατα σε όλες τις διεργασίες, αρχικά διασχίζουμε ολόκληρη τη λίστα του διανομέα, ελευθερώνοντας πρώτα τις διεργασίες που έχουν σταματήσει λόγω συνόλου εργασίας. Κατά τη διάσχιση, οι εργασίες που εκτελέστηκαν στο τελευταίο μεσοδιάστημα εξετάζονται ως προς τους πόρους που κατανάλωσαν, όπως ακριβώς γίνεται στη διαδικασία delete_process (εκτίμηση νέου συνόλου εργασίας, κλήση της resource_since_time και ενημέρωση των πόρων στον pdt και των συνολικών πόρων στο μεσοδιάστημα). Τελικά, καλείται η συνάρτηση *critical_time* για να υπολογιστεί ο νέος κρίσιμος χρόνος της διεργασίας και μηδενίζεται ο χρόνος πρόσφατης εκτέλεσης και οι μετρητές μηνυμάτων, ετοιμάζοντας τη διεργασία για το επόμενο διάστημα. Παρατηρήστε ότι οι διεργασίες που δεν εκτελέστηκαν δεν εξετάζονται καθόλου.

Στη συνέχεια, έχοντας τους νέους κρίσιμους χρόνους, αναδιατάσσουμε τη λίστα του διανομέα χρησιμοποιώντας μία τεχνική παρόμοια με bubble sort. Αυτό σημαίνει ότι διασχίζουμε τη λίστα από την πρώτη διεργασία χρήστη και σε κάθε σημείο, διασχίζουμε τη λίστα ανάποδα όσο χρειάζεται για να φέρουμε την τρέχουσα διεργασία στο σωστό σημείο προτεραιότητας. Η σύνδεση γίνεται σε δύο κατευθύνσεις και πρέπει πάντοτε να γίνεται έλεγχος για το αν βρισκόμαστε στο τέλος της λίστας. Κατά τη διάρκεια της αναταξινόμησης, γίνεται και έλεγχος συνέπειας της λίστας. Η ορθότητα της μεθόδου οφείλεται στο ότι με τη μέθοδο του κρίσιμου χρόνου, η λίστα παραμένει μερικώς ταξινομημένη πάντοτε, με αποτέλεσμα σε κάθε στοιχείο να αρκεί η προώθησή του στην ουρά προτεραιότητας μέχρι το στοιχείο με την αμέσως μεγαλύτερη προτεραιότητα.

Έχοντας ταξινομήσει τη λίστα του διανομέα, ενημερώνουμε το χρόνο τελευταίας κλήσης και τους πόρους ανά μεσοδιάστημα, καθώς και το πλήθος μεσοδιαστημάτων. Τέλος, υπολογίζουμε το μέγεθος του συνόλου καταλλήλων, διασχίζοντας πάλι τη λίστα του διανομέα (τώρα ταξινομημένη) και υπολογίζοντας πόσες από τις διεργασίες χωράνε στη μνήμη, αγνοώντας τις διεργασίες που περιμένουν να θανατωθούν, αλληλεπίδραση ή για τερματισμό παιδιού (για την ακρίβεια, υπολογίζουμε μία διεργασία παραπάνω από αυτές που χωράνε ακριβώς στη μνήμη, για να μην μείνει αναξιοποίητο ένα τμήμα της μνήμης). Ενημερώνουμε το πλήθος κλήσεων του χρονοπρογραμματιστή και υπολογίζουμε το μέσο μέγεθος του συνόλου καταλλήλων, χρησιμοποιώντας εξομάλυνση. Αν έχει αλλάξει το μέγεθος του συνόλου καταλλήλων, τότε κάνουμε την αντίστοιχη κλήση του πυρήνα (change_eligible_set_number) και

επιστρέφουμε στην κύρια είσοδο του διαχειριστή της ΚΜΕ, ενώ διαφορετικά επιστρέφουμε στον καλούντα.

3.3.2.3.4. Κύρια διαδικασία

Στην πρώτη είσοδο του διαχειριστή (συνάρτηση *cpu_manager*), ορίζουμε το πλήθος επιπέδων προτεραιότητας και αρχικοποιούμε τις συναρτήσεις πολιτικής (μέσω της initialize_policy_functions) και στη συνέχεια δίνουμε προτεραιότητες στις διεργασίες του συστήματος (αρχικά 0), καθώς και αριθμούς στα αναγνωριστικά τους (η ίδια η λίστα δημιουργείται κατά την αρχική φόρτωση του συστήματος). Κατά τη διάσχιση, γίνεται και έλεγχος συνέπειας καθώς και μέτρηση του πλήθους διεργασιών που έχουν δημιουργηθεί. Στη συνέχεια θέτουμε την ουρά της λίστας του διανομέα (tail, από το vardefs.h), συνδέουμε τις ελεύθερες θέσεις του pdt σε μία λίστα ελευθέρων θέσεων με κεφαλή τη μεταβλητή flptr και ενημερώνουμε τις μεταβλητές που σχετίζονται με τις καταχωρήσεις στον pdt. Μηδενίζουμε όλα τα στατιστικά στοιχεία για τους μέσους χρόνους σε κάθε κατάσταση για κάθε επίπεδο προτεραιότητας, τους μετρητές διεργασιών και χαμένων διεργασιών (δηλαδή διεργασιών που δεν μπόρεσαν να τύχουν επεξεργασίας, για τους λόγους που έχουμε ήδη αναφέρει) καθώς και τα μέτρα εξυπηρέτησης και πόρων. Τέλος ορίζουμε το μέγεθος του αρχικού συνόλου καταλλήλων (διεργασίες του συστήματος) και το πλήθος κλήσεων του χρονοπρογραμματιστή καθώς και το χρόνο της τελευταίας κλήσης του χρονοπρογραμματιστή και της τελευταίας κλήσης για αναφορά κατάστασης.

Στην κύρια είσοδο, περιμένουμε ένα γεγονός (μήνυμα ή ενεργοποίηση) από το χρονοπρογραμματιστή εργασιών (θύρα 0), το διαχειριστή μνήμης (θύρα 1) ή το δημιουργό διεργασιών (θύρα 2). Στην είσοδο 2, εξετάζουμε την αιτία της αφύπνισης. Αν ήταν ενεργοποίηση, τότε προέρχεται από το χρονόμετρο, οπότε καλούμε το χρονοπρογραμματιστή και επιστρέφουμε στην κύρια είσοδο. Αλλιώς, έχουμε ένα μήνυμα, το οποίο μπορεί να περιέχει μία από πέντε εντολές στο πεδίο mcommand:

CREATE	δημιουργία διεργασίας (δέσμευση pdt)
DELETE	διαγραφή διεργασίας (από τον pdt)
LOADED	η διεργασία φορτώθηκε
LOAD_FAILED	η διεργασία δεν φορτώθηκε
REPORT	αίτηση για αναφορά από το μακροχρόνιο χρονοπρογραμματιστή (δηλαδή από το χρονοπρογραμματιστή εργασιών)

Ανάλογα με την εντολή, διακλαδωνόμαστε σε κάποια από τις παρακάτω υποεισόδους.

Η υποείσοδος CREATE καλείται μετά από αίτηση του δημιουργού διεργασιών για δέσμευση ενός αναγνωριστικού στον pdt, προκειμένου να δημιουργήσει μία διεργασία. Αν η λίστα ελευθέρων θέσεων είναι κενή, απαντάμε με αποτυχία (mcommand=0) και στέλνουμε ένα μήνυμα απάντησης, επιστρέφοντας στην κύρια είσοδο. Αλλιώς, δεσμεύουμε τη θέση στην κεφαλή της λίστας ελευθέρων, σημειώνοντάς την ως εμποδισμένη για δημιουργία και δίνοντας τον επόμενο ακέραιο ως αριθμό διεργασίας. Τα στοιχεία αυτά επιστρέφονται στο mpid και ενημερώνουμε το πλήθος διεργασιών και τις καταχωρήσεις στον pdt. Αφού ενημερώσουμε και τη λίστα ελευθέρων, απαντάμε στο δημιουργό διεργασιών με επιτυχία (mcommand=1) και επιστρέφουμε στην κύρια είσοδο.

Η υποείσοδος DELETE καλείται όταν ο δημιουργός διεργασιών ζητάει τη διαγραφή της διεργασίας που δίνεται στο mpid. Οι λόγοι δίνονται όπως είδαμε παραπάνω στα word6 και word3. Σημειώνουμε τους λόγους αυτούς και τους αντικαθιστούμε με τον κωδικό για διαγραφή γονέα, με σκοπό να διαγράψουμε πρώτα όλα τα παιδιά της διεργασίας. Αυτό γίνεται προχωρώντας στην είσοδο 3, με την οποία διαγράφουμε ένα παιδί σε κάθε ενεργοποίησή της. Η διαγραφή γίνεται με κλήση της delete_process για το τρέχον παιδί και την αποστολή ενός μηνύματος στο διαχειριστή της μνήμης για απελευθέρωση της μνήμης της διεργασίας (mcommand=1), με αναγνωριστικό την τρέχουσα διεργασία (παιδί). Κάθε φορά, βρίσκουμε το επόμενο παιδί (σε static μεταβλητή, γιατί μεσολαβεί αποστολή μηνύματος μέχρι την επόμενη χρήση αυτής της τιμής) και στέλνουμε το μήνυμα, προχωρώντας στην είσοδο 4. Εκεί, δεχόμαστε την απάντηση και επιστρέφουμε στην είσοδο 3. Όταν τελειώσουν τα παιδιά, επαναφέρουμε τους αρχικούς λόγους διαγραφής και αποσυνδέουμε την ίδια τη διεργασία από τη λίστα των παιδιών του πατέρα της, ακολουθώντας τη λίστα αδελφών από το μεγαλύτερο μέχρι την τρέχουσα διεργασία και συνδέοντας τα δύο γειτονικά αδέλφια μεταξύ τους. Τελικά, καλείται η delete_process για τη διαγραφή της διεργασίας αυτής και στέλνεται μήνυμα στο διαχειριστή της μνήμης, όπως και για τα παιδιά, προχωρώντας αυτή τη φορά στην είσοδο 5. Η είσοδος 5 περιμένει την απάντηση και προχωράει μετά στην είσοδο 6, απ' όπου και στέλνεται το τελικό μήνυμα απάντησης στο δημιουργό διεργασιών και επιστρέφουμε στην κύρια είσοδο.

Η υποείσοδος LOADED καλείται μετά από τη δημιουργία μίας διεργασίας, σαν αποτέλεσμα της παραλαβής ενός μηνύματος από το δημιουργό διεργασιών, το οποίο περιέχει το αναγνωριστικό της διεργασίας (mpid) τη θέση του τεμαχίου 0 (word3) και το αναγνωριστικό του γονέα (words45), τα οποία μπαίνουν ως έχουν στα αντίστοιχα πεδία του pdt. Επιπλέον, ο τύπος της διεργασίας υπολογίζεται από το επίπεδο προτεραιότητας της διεργασίας (word6) και η διεργασία σημειώνεται ως έτοιμη και απελευθερωμένη. Το σημείο εισόδου και οι δείκτες ενεργοποίησης και αναστολής μηδενίζονται.

Όλοι οι χρόνοι μηδενίζονται και η τελευταία αλλαγή κατάστασης ορίζεται ότι έγινε τώρα. Τέλος ενημερώνουμε τους τρέχοντες μετρητές μηνυμάτων και πόρων καθώς και την αρχή της αλληλεπίδρασης. Αφού μηδενίσουμε και τους συνολικούς μετρητές μηνυμάτων και εκτέλεσης, βρίσκουμε την προτεραιότητα της διεργασίας, ανάλογα με τον τύπο της, τη συνδέουμε στη λίστα αδελφών και σαν πρώτο παιδί του γονέα της και τελικά την βάζουμε στην κατάλληλη θέση στη λίστα του διανομέα (γραμμική παρεμβολή, αναλόγως προτεραιότητας). Στο τέλος, επιστρέφουμε άμεσα στην κύρια είσοδο.

Η υποείσοδος LOAD_FAILED καλείται όταν έχουμε αποτυχία δημιουργίας της διεργασίας που ορίζεται από το mpid. Έτσι, η θέση της στον pdt προστίθεται στη λίστα ελευθέρων και μειώνονται οι καταχωρήσεις στον pdt (παρατηρήστε ότι δεν έχουμε κάνει συνδέσεις ή άλλες μεταβολές στις διεργασίες, αφού αυτές γίνονται από την είσοδο LOADED). Τελικά, επιστρέφουμε άμεσα στην κύρια είσοδο.

Η υποείσοδος REPORT καλείται μετά από μία αίτηση του χρονοπρογραμματιστή εργασιών να πληροφορηθεί την κατάσταση του συστήματος. Υπολογίζεται αρχικά ένας παράγοντας εξομάλυνσης (με τον τύπο α=διάστημα από προηγούμενη κλήση / τρέχων χρόνος), και υπολογίζεται η τρέχουσα απόκριση και οι τρέχοντες πόροι με συνυπολογισμό των προηγούμενων και των μέσων τιμών τους. Ο χρόνος τελευταίας αναφοράς αναπροσαρμόζεται και τα στοιχεία πλήθους διαστημάτων και αλληλεπιδράσεων από την τελευταία αίτηση για αναφορά μηδενίζονται. Τελικά, στέλνουμε την απάντηση με τα word3 να έχει την απόκριση, το word6 τη μέγιστη απόκριση και το word7 τους πόρους. Μετά από την αποστολή της απάντησης, επιστρέφουμε στην κύρια είσοδο. Παρατηρήστε ότι όλες οι μετρήσεις δεν πρέπει να υπερβαίνουν ένα όριο (MAXINT) και ότι οι μετρήσεις απόκρισης αφορούν μόνο τις διεργασίες αλληλεπίδρασης, αφού για αυτές γίνεται η κλήση (αυτό δεν συμβαίνει για τους πόρους όμως).

3.3.3. Διαχειριστές εισόδου και εξόδου

3.3.3.1. Εισαγωγή

Στο σύστημά μας, οι διαχειριστές εισόδου και εξόδου έχουν σαν στόχο να επιτυγχάνουν υψηλό βαθμό αξιοποίησης των συσκευών, φροντίζοντας για ανεξαρτησία και ομοιομορφία στη διαχείρισή τους. Αυτό σε κάποιο βαθμό το πετυχαίνουμε με το να χειριζόμαστε εικονικές συσκευές, οι οποίες φαίνεται να καταμερίζονται από πολλές διεργασίες. Έτσι, έχουμε τους κανονικούς διαχειριστές συσκευών και τους διαχειριστές των εικονικών συσκευών ή spoolers. Στα παρακάτω θα δούμε και τα δύο είδη διαχειριστών.

Η βασική εργασία ενός διαχειριστή συσκευής είναι να λαμβάνει μηνύματα απαιτήσεων για ε/ε και να δημιουργεί αντίστοιχα προγράμματα διαύλων για να τις εκτελέσουν. Αφού ξεκινήσει τη λειτουργία του προγράμματος, ο διαχειριστής θα πρέπει να περιμένει την απάντηση, να χειριστεί αν χρειαστεί τις περιπτώσεις λαθών και τέλος να ενημερώσει τη διεργασία που ζήτησε τη μεταφορά. Ειδική περίπτωση διαχειριστή συσκευών στο σύστημά μας είναι ο διαχειριστής τερματικών, ο οποίος χειρίζεται πολλά τερματικά ταυτόχρονα. Για να χειριστεί αυτή την πολλαπλή εργασία, χρησιμοποιεί μία χωριστή δομή δεδομένων για κάθε τερματικό, όπως θα δούμε στη συνέχεια.

Οι spoolers, λειτουργούν με το να λαμβάνουν μηνύματα απαιτήσεων για εργασία και να φροντίζουν να διεκπεραιώνουν την εργασία, μεταφέροντας αρχεία μεταξύ συσκευών, πριν να απαντήσουν στον αιτούντα. Οι spoolers ασχολούνται με την ανάγνωση εργασιών στο σύστημα (μεταφορά από διάτρητα δελτία στο δίσκο) και την εκτύπωση αποτελεσμάτων εργασιών (μεταφορά από δίσκο σε εκτυπωτή). Για να μην περιορίζονται οι spoolers από τον πραγματικό χρονοπρογραμματισμό του συστήματος, έχουμε πολλαπλές περιοχές εισόδου και εξόδου εργασιών, οι οποίες γεμίζουν (κατά την είσοδο) και αδειάζουν (κατά την έξοδο), μία κάθε φορά, αλλά μπορεί να αδειάζουν και να γεμίζουν, αντίστοιχα, με διαφορετική σειρά και ταχύτητα.

Ο κώδικας για τους διαχειριστές συσκευών βρίσκεται στο αρχείο *devman.c* ή στο *sdevman.c*, ανάλογα με την έκδοση, και στο αρχείο *terman.c* για το διαχειριστή τερματικών. Οι spoolers βρίσκονται στο *insp.c* για την είσοδο και στο *outsp.c* για την έξοδο.

3.3.3.2. Δομές Δεδομένων

Ένας τρόπος επίτευξης της ανεξαρτησίας των συσκευών είναι η τοποθέτηση των βασικών τους πληροφοριών σε πίνακες. Για κάθε συσκευή λοιπόν, έχουμε έναν περιγραφητή συσκευής, με το σύνολο των περιγραφητών να αποθηκεύεται σε έναν πίνακα, με το όνομα *πίνακας περιγραφητών συσκευών (device descriptor table, DDT)*. Ο πίνακας αυτός περιέχει ένα στοιχείο για κάθε συσκευή (τύπου *ddtstruct*, στο vardefs.h), με τα ακόλουθα πεδία:

char device name[20]: όνομα της συσκευής.
int record_size: μέγεθος φυσικής εγγραφής.
int device_busy: ψηφίο κατάστασης απασχόλησης της συσκευής.
double device_idle_time: συνολικός χρόνος αδράνειας.
double devlbt: χρόνος τελευταίας χρήσης της συσκευής.

float latency_time: λανθάνων χρόνος (περιστροφική καθυστέρηση) της συσκευής.
float positioning_time: χρόνος αναζήτησης.
float transfer_time: χρόνος μεταφοράς ενός byte.

Παρατηρήστε ότι τα πεδία για τους χαρακτηριστικούς χρόνους της συσκευής μπορεί να περιέχουν την τιμή 0, με αποτέλεσμα να χρησιμοποιούμε τους ίδιους τύπους για τον υπολογισμό του χρόνου ολοκλήρωσης μίας απαίτησης, ανεξάρτητα από το είδος της συσκευής (ανεξαρτησία συσκευών). Ο πίνακας περιέχει no_of_devices+1 περιγραφητές (δε χρησιμοποιείται ο πρώτος, έτσι έχουμε 4 συσκευές), με τα ονόματα και κωδικούς:

DEV_DRUM_DRIVE	1
DEV_DISK_DRIVE	2
DEV_LINE_PRINTER	3
DEV_CARD_READER	4

Οι ορισμοί αυτοί δίνονται στο defs.h και ο ορισμός του πίνακα ddt δίνεται στο vardefs.h.

Για τα τερματικά, έχουμε τον πίνακα ttt (vardefs.h) ο οποίος περιέχει no_of_terminals+1 στοιχεία (ορίζεται ως 32 στο defs.h) τύπου *tttstruct* (defs.h) με ένα μόνο πεδίο, τον ακέραιο terminal_busy, που είναι ένα ψηφίο απασχόλησης του αντίστοιχου τερματικού.

Παρατηρήστε ότι για κάθε συσκευή έχουμε ένα μόνο περιγραφητή, αφού το σύστημά μας υποστηρίζει μόνο μία συσκευή ενός τύπου. Αν χρησιμοποιούσαμε πολλές συσκευές, θα έπρεπε αντί να επεκτείνουμε το μέγεθος του πίνακα, να χρησιμοποιήσουμε κύριους και δευτερεύοντες αριθμούς συσκευών (όπως στο UNIX), με την επιλογή του δευτερεύοντα αριθμού να γίνεται από το σύστημα (για τον εκτυπωτή και τη συσκευή ανταλλαγής) ή από τη διεργασία (για το δίσκο και τον αναγνώστη δελτίων).

3.3.3.3. Αλγόριθμοι

3.3.3.3.1. Διαχειριστής συσκευών

Ο (γενικός) διαχειριστής συσκευών υλοποιείται από τον κώδικα της συνάρτησης *device_managers* (αρχείο devman.c ή sdevman.c), που είναι κοινός για όλες τις συσκευές (η συμπεριφορά του αλλάζει ανάλογα με την τρέχουσα συσκευή, σύμφωνα με τον περιγραφητή της). Η αρχική είσοδος τυπώνει μόνο ένα μήνυμα ενεργοποίησης. Η κύρια είσοδος του κώδικα περιμένει ένα μήνυμα ή ενεργοποίηση (γεγονός) από τον πυρήνα (θύρα 1) ή

από μία τυχαία διεργασία χρήστη. Στην είσοδο 2, αντιμετωπίζουμε το μήνυμα (αγνοούμε την περίπτωση ενεργοποίησης, που γενικά σημαίνει πάτημα του πλήκτρου reset της συσκευής), ελέγχοντας το μήκος της μεταφοράς που ζητείται (στο word7). Εκχωρούμε τη συσκευή (σαν να είναι αφοσιωμένη) δίνοντας στο word3 τον αριθμό της συσκευής (είναι ο αριθμός του διαχειριστή της μείον 3) και καλούμε την είσοδο initiate_io του πυρήνα για να εκτελεστεί η πραγματική εντολή ε/ε, προχωρώντας στην είσοδο 3. Παρατηρήστε ότι δεν ασχολούμαστε με το αν η αίτηση είναι για είσοδο ή έξοδο, αφού εκεί που αυτό έχει νόημα (δίσκος και συσκευή ανταλλαγής), ο χρόνος διεκπεραίωσης είναι ο ίδιος.

Στην είσοδο 3, περιμένουμε την απάντηση του πυρήνα, η οποία εξετάζεται στην είσοδο 4. Χρησιμοποιώντας τη θύρα 2 (θύρα διεργασιών που ζητούν τη συσκευή) απαντάμε και είτε επιστρέφουμε στην κύρια είσοδο (για μη καταμεριζόμενες συσκευές) είτε προχωράμε στην είσοδο 5 μετά την απάντηση, όπου καταστρέφουμε τη σύνδεση (κλήση destroy_path), και μετά επανερχόμαστε στην κύρια είσοδο. Οι καταμεριζόμενες συσκευές απαιτούν καταστροφή της σύνδεσης, για να μπορούν να ξαναχρησιμοποιηθούν αμέσως. Καταμεριζόμενες συσκευές είναι ο δίσκος και η συσκευή ανταλλαγής, ενώ ο εκτυπωτής και ο αναγνώστης δελτίων είναι αφιερωμένες συσκευές.

Η έκδοση που περιγράψαμε είναι αυτή που περιέχεται στο devman.c. Υπάρχει και μία άλλη έκδοση της συνάρτησης device_managers, στο αρχείο sdevman.c, στην οποία θεωρούμε όλες τις συσκευές καταμεριζόμενες. Αυτό σημαίνει ότι πάντοτε στην είσοδο 4 προχωράμε στην είσοδο 5 για καταστροφή του μονοπατιού σύνδεσης μετά από την ολοκλήρωση της μεταφοράς. Επίσης, στα μηνύματα προς τον πυρήνα περιέχεται το αναγνωριστικό της διεργασίας που ζήτησε τη μεταφορά, ενώ στα μηνύματα απάντησης περιέχεται πάντα και ο αριθμός της συσκευής από την οποία έρχεται η απάντηση.

3.3.3.3.2. Διαχειριστής τερματικών

Ο διαχειριστής τερματικών υλοποιείται από τη συνάρτηση *terminal_system_manager* (terman.c), η οποία στην πρώτη της είσοδο αρχικοποιεί τους μετρητές χαμένων χρηστών, λόγω ανεπάρκειας χώρου σε πίνακες του συστήματος και λόγω χαμηλής απόκρισης του συστήματος. Επίσης, σημειώνει τον αριθμό συσκευής στον οποίο αναφερόμαστε (αν και είναι κοινός για όλα τα τερματικά, και θα μπορούσε να θεωρηθεί και ως σταθερά). Στην κύρια είσοδο, περιμένουμε ένα μήνυμα από το χρονοπρογραμματιστή εργασιών (θύρα 0), τον πυρήνα (θύρα 1) ή από μία διεργασία χρήστη. Στην είσοδο 2, εξετάζουμε τον αποστολέα του μηνύματος. Αν είναι ο πυρήνας, κοιτάμε αν έχουμε αίτηση για σύνδεση (mcommand=3), οπότε στέλνουμε ένα μήνυμα στο χρονοπρογραμματιστή εργασιών, ζητώντας

τη σύνδεση ενός χρήστη (ενεργοποίηση εργασίας αλληλεπίδρασης) και επιστρέφουμε στην κύρια είσοδο. Αν, αντίθετα, ο πυρήνας ζητάει την εξυπηρέτηση μίας διεργασίας, συνδεόμαστε με τη διεργασία αυτή μέσω της θύρας 2, καλώντας την είσοδο create_path του πυρήνα και προχωρώντας στην είσοδο 4. Από εκεί, στέλνουμε ένα μήνυμα στη διεργασία που ζήτησε την εξυπηρέτηση χρησιμοποιώντας τον αριθμό συσκευής του τερματικού και κινούμαστε στην είσοδο 3. Στην είσοδο 3, καταστρέφουμε το μονοπάτι προς τη διεργασία (κλήση destroy_path) και επιστρέφουμε στη κύρια είσοδο.

Αν από την κύρια είσοδο διαπιστώσουμε ότι ο αποστολέας του μηνύματος ήταν ο χρονοπρογραμματιστής των εργασιών, αυτό σημαίνει ότι έχουμε ειδοποιηθεί για αποτυχημένη σύνδεση, οπότε ενημερώνουμε τον κατάλληλο πίνακα και επιστρέφουμε στην κύρια είσοδο. Σε κάθε άλλη περίπτωση, έχουμε μήνυμα από μία διεργασία, η οποία τελείωσε την αλληλεπίδρασή της και θέλει να χρησιμοποιήσει το τερματικό (για έξοδο). Για να γίνει η μεταφορά, βρίσκουμε το αναγνωριστικό της διεργασίας και το βάζουμε στο mpid, μεταφέρουμε από τον πίνακα σκιάς τον αριθμό του τερματικού αυτού στο word7 και τον αριθμό του συστήματος τερματικών στο word3, και ξεκινάμε την μεταφορά προχωρώντας στην είσοδο 3, από όπου και καταστρέφουμε το μονοπάτι σύνδεσης, επιστρέφοντας τελικά στην κύρια είσοδο.

3.3.3.3.3. Spooler εισόδου

Ο spooler εισόδου υλοποιείται από τη συνάρτηση *input_spooler* (insp.c), η οποία στην πρώτη της είσοδο υπολογίζει τον παράγοντα ομαδοποίησης εγγραφών για την είσοδο (int in_block, στο vardefs.h), δηλαδή το πλήθος εγγραφών εισόδου (δελτίων) που χωράνε σε μία εγγραφή στο δίσκο. Η κύρια είσοδος, περιμένει ένα μήνυμα από το χρονοπρογραμματιστή εργασιών (θύρα 0), το σύστημα αρχείων (θύρα 1) και το διαχειριστή του αναγνώστη δελτίων (θύρα 2). Στην είσοδο 2, αποθηκεύουμε το πλήθος των εγγραφών της εργασίας (από το word3) και ένα δείκτη προς τον περιγραφητή της εργασίας αυτής (από το word8) και σημειώνουμε ότι αυτή είναι η πρώτη κλήση για την εργασία αυτή.

Προχωρούμε άμεσα στην είσοδο 3, από όπου ξεκινάει πάντα το διάβασμα μίας σειράς εγγραφών εισόδου (όσες ο παράγοντας ομαδοποίησης). Αρχίζουμε να μετράμε τόσες εγγραφές όσες χωράνε σε μία εγγραφή στο δίσκο, ή όσες μένουν (αν είναι λιγότερες). Στην είσοδο 4, μπαίνουμε με σκοπό να διαβάσουμε μία εγγραφή εισόδου. Αυτό γίνεται με αποστολή μηνύματος στον αναγνώστη δελτίων, χρησιμοποιώντας το μέγεθος ενός δελτίου στο πεδίο word7. Μετά την αποστολή μηνύματος προχωράμε στην είσοδο 5. Εκεί, αυξάνουμε το πλήθος των μηνυμάτων που στείλαμε για την τρέχουσα ομάδα

αναγνώσεων και αν υπάρχουν κι άλλα, επιστρέφουμε στην είσοδο 4 για το επόμενο. Όταν διαβάσουμε όλη την ομάδα προχωράμε στην είσοδο 6.

Στην είσοδο 6, είμαστε έτοιμοι να αρχίσουμε να λαμβάνουμε τις απαντήσεις από το διαχειριστή του αναγνώστη δελτίων. Περιμένοντας την πρώτη απάντηση προχωράμε στην είσοδο 7. Εκεί, μετράμε τις απαντήσεις, και αν δεν έχουν φτάσει όλες επιστρέφουμε στην είσοδο 6 για να περιμένουμε την επόμενη απάντηση. Αλλιώς (αν λάβαμε όλες τις απαντήσεις), στέλνουμε ένα μόνο μήνυμα στο σύστημα αρχείων να γράψει το σύνολο των εγγραφών εισόδου που διαβάσαμε (μία εγγραφή δίσκου), με το κατάλληλο μέγεθος (όχι το μέγεθος εγγραφής πάντα, αφού η τελευταία ομάδα ανάγνωσης δε θα εξαντλεί πάντα τη φυσική εγγραφή). Μετά την αποστολή του μηνύματος προχωράμε στην είσοδο 8, όπου μειώνουμε το σύνολο εγγραφών που μένουν να μεταφερθούν και ελέγχουμε αν αυτή είναι η πρώτη μεταφορά. Αν ναι, τότε σε περίπτωση που υπάρχουν κι άλλες εγγραφές, επιστρέφουμε στην είσοδο 3 για να διαβάσουμε την επόμενη ομάδα, ενώ, αν τελειώσαμε, προχωράμε στην είσοδο 11. Αν δεν είμαστε στην πρώτη κλήση, εκκρεμεί μία απάντηση από το σύστημα αρχείων, επειδή έχουμε διπλή ενταμίευση (δηλαδή, κάθε φορά που διαβάζουμε μία ομάδα δελτίων, στέλνουμε την αίτηση εγγραφής στο σύστημα αρχείων και προχωράμε στην ανάγνωση της επόμενης ομάδας χωρίς να περιμένουμε αμέσως την απάντηση). Έτσι, προχωρούμε στην είσοδο 9, η οποία περιμένει αυτή την απάντηση και συνεχίζει στην είσοδο 10, όπου, αν υπάρχουν κι άλλες εγγραφές, επιστρέφουμε στην είσοδο 3 για την επόμενη ομάδα. Αλλιώς, προχωράμε πάλι στην είσοδο 11, όπου περιμένουμε την τελευταία απάντηση από το σύστημα αρχείων, και μετά στην είσοδο 12 όπου επιστρέφουμε σαν απάντηση στο χρονοπρογραμματιστή εργασιών το ότι διαβάστηκε η εργασία (με το αναγνωριστικό εργασίας στο word8) και γυρνάμε τελικά στην κύρια είσοδο.

Παρατηρήστε ότι λόγω της διπλής ενταμίευσης, την πρώτη φορά που ολοκληρώνεται η ανάγνωση μίας ομάδας δελτίων, δεν περιμένουμε άμεσα την απάντηση του συστήματος αρχείων, αλλά προχωράμε να γεμίσουμε και τον άλλον ενταμιευτή εισόδου. Όταν όμως είμαστε στις επόμενες εισόδους, πρέπει να περιμένουμε το άδειασμα του άλλου ενταμιευτή εισόδου (όχι αυτού που μόλις γεμίσαμε) πριν προχωρήσουμε να τον ξαναγεμίσουμε. Η χρησιμότητα της τελευταίας απάντησης έγκειται στο ότι μετά από κάθε ανάγνωση περιμένουμε απάντηση για την προηγούμενη ομάδα. Έτσι, στο τέλος πρέπει να περιμένουμε μία επιπλέον απάντηση, για την τελευταία ομάδα που πραγματικά μεταφέρθηκε.

3.3.3.3.4. Spooler εξόδου

Ο spooler εξόδου υλοποιείται από τη συνάρτηση *output_spooler* (outsp.c), η οποία στην πρώτη της είσοδο υπολογίζει τον παράγοντα ομαδοποίησης της εξόδου (int out_block, στο vardefs.h), δηλαδή το πλήθος των εγγραφών εξόδου (γραμμών εκτυπωτή) που απαιτούνται για μία εγγραφή από το δίσκο. Στην κύρια είσοδο, περιμένουμε ένα μήνυμα από το χρονοπρογραμματιστή εργασιών (θύρα 0), το σύστημα αρχείων (θύρα 1) και το διαχειριστή του εκτυπωτή (θύρα 2). Στην είσοδο 2 κρατάμε το πλήθος των εγγραφών εξόδου (word3) και την εργασία στην οποία ανήκουν (word8) και ετοιμάζουμε το πρώτο μήνυμα προς το διαχειριστή αρχείων για να διαβάσουμε την εγγραφή για την έξοδο. Το μέγεθος της εγγραφής είναι είτε όσο το μέγεθος της φυσικής εγγραφής, είτε (για το τελευταίο μήνυμα) το μέγεθος των υπόλοιπων εγγραφών (γραμμών) που μένουν. Τελικά, προχωράμε στην είσοδο 3.

Στην είσοδο 3, περιμένουμε απάντηση από το σύστημα αρχείων και προχωράμε στην είσοδο 4. Εκεί, υπολογίζουμε πόσες εγγραφές εξόδου μένουν ακόμη, και αν υπάρχουν κι άλλες, στέλνουμε ένα μήνυμα στο σύστημα αρχείων για να γεμίσουμε τον άλλο ενταμιευτή (διπλή ενταμίευση) με τις εγγραφές που μένουν (όπως στην είσοδο 3). Σε κάθε περίπτωση, είτε μετά από το μήνυμα είτε άμεσα (όταν τελείωσαν οι εγγραφές για διάβασμα), προχωράμε στην είσοδο 5. Εκεί, ορίζουμε την αρχή της εκτύπωσης της τρέχουσας ομάδας και προχωράμε στην είσοδο 6, όπου στέλνουμε ένα μήνυμα για την εκτύπωση μίας γραμμής στον εκτυπωτή γραμμών και προχωράμε στην είσοδο 7. Εκεί, ενημερώνουμε το πλήθος των τυπωμένων γραμμών και αν απομένουν κι άλλες στην τρέχουσα ομάδα, γυρνάμε στην είσοδο 6, αλλιώς αρχίζουμε να μετράμε απαντήσεις και προχωράμε στην είσοδο 8. Στην είσοδο 8 περιμένουμε μία απάντηση από τον εκτυπωτή και προχωράμε στην είσοδο 9, όπου μετράμε τις απαντήσεις, και αν περιμένουμε κι άλλες, επιστρέφουμε στην είσοδο 8. Αλλιώς, ελέγχουμε αν τελείωσε η έξοδος. Αν όχι, ξαναγυρνάμε στην είσοδο 3 για να πάρουμε απάντηση από το σύστημα αρχείων (έχουμε ήδη στείλει αίτηση για ανάγνωση της επόμενης εγγραφής). Αν ναι, στέλνουμε την τελική απάντηση στο χρονοπρογραμματιστή εργασιών ότι τελείωσε η έξοδος της διεργασίας που δίνεται στο word8, και επιστρέφουμε στην κύρια είσοδο.

Παρατηρήστε ότι εδώ, η διπλή ενταμίευση είναι απλούστερη από ότι στην είσοδο, αφού στέλνουμε μαζί τα πρώτα μηνύματα για ανάγνωση εγγραφών και στη συνέχεια η κάθε απάντηση συμπίπτει με την έναρξη της εξόδου στον εκτυπωτή για την εγγραφή αυτή. Το επόμενο μήνυμα στέλνεται τότε, αν υπάρχει λόγος, και δε χρειάζεται να περιμένουμε την τελευταία απάντηση χωριστά.

3.3.4. Σύστημα αρχείων

3.3.4.1. Εισαγωγή

Το (υπο)σύστημα αρχείων στο σύστημά μας είναι πολύ απλό, αφού δεν ασχολείται με "πραγματικά" αρχεία (τα οποία, όπως και οι χρήστες, δεν υπάρχουν) αλλά με την μεταφορά των απαιτήσεων των διεργασιών στο διαχειριστή συσκευής του δίσκου. Ουσιαστικά, το σύστημα αρχείων δεν κάνει τίποτα άλλο από το να επαναδρομολογεί μηνύματα. Από αυτή την άποψη, η μόνη του χρησιμότητα είναι ότι ανεξαρτητοποιεί τις άλλες διεργασίες από τις ανάγκες της προσομοίωσης και ότι βοηθάει στο να έχουμε το σωστό πλήθος μηνυμάτων στο σύστημα. Αφού δεν έχουμε πραγματικά αρχεία, το σύστημα αρχείων δε χρησιμοποιεί ειδικές δομές δεδομένων. Ο κώδικας για το σύστημα αρχείων βρίσκεται ολόκληρος στο αρχείο *fsys.c*.

3.3.4.2. Αλγόριθμοι

Η συνάρτηση που υλοποιεί το διαχειριστή αρχείων είναι η *file_system_manager*. Στην πρώτη είσοδο, υπολογίζεται ο παράγοντας ομαδοποίησης μεταξύ φυσικών εγγραφών και εγγραφών που μεταφέρονται με μία προσπέλαση στο δίσκο. Στην κύρια είσοδο προετοιμάζουμε τη λήψη ενός μηνύματος από μία οποιαδήποτε διεργασία από μία αόριστη θύρα, έχοντας ορισμένη μόνο τη θύρα 1, προς το διαχειριστή του δίσκου. Στην είσοδο 2, κρατάμε τον αριθμό του αρχείου, όπως τον παρέχει η διεργασία (στο word3) και αρχίζουμε να μετράμε τις εγγραφές που θα διαβάσουμε. Επειδή πάντοτε διαβάζουμε μία ολόκληρη φυσική εγγραφή (μεγέθους physical_blocksize) ο μετρητής θα φτάσει τελικά μέχρι την τιμή του παράγοντα ομαδοποίησης που υπολογίστηκε αρχικά. Στην είσοδο 3, στέλνουμε μήνυμα μεταφοράς (mcommand=0) στο δίσκο και προχωράμε στην είσοδο 4. Εκεί, ενημερώνουμε το πλήθος των εγγραφών που διαβάστηκαν και ελέγχουμε αν ολοκληρώθηκε η μεταφορά. Αν όχι, επιστρέφουμε άμεσα στην είσοδο 3 για την επόμενη ανάγνωση. Αλλιώς, αρχίζουμε να μετράμε τις απαντήσεις από το διαχειριστή του δίσκου και προχωράμε στην είσοδο 5. Στην είσοδο 5 ετοιμαζόμαστε για τη λήψη ενός μηνύματος απάντησης από το δίσκο. Το μήνυμα λαμβάνεται στην είσοδο 6, όπου ενημερώνεται το πλήθος των απαντήσεων που λάβαμε και αν υπολείπονται απαντήσεις, επιστρέφουμε άμεσα στην είσοδο 5. Αλλιώς, στέλνουμε ένα μήνυμα απάντησης στη διεργασία που ζήτησε τη μεταφορά, χρησιμοποιώντας τον αριθμό του αρχείου (στο word3) και προχωράμε στην είσοδο 7, όπου καταστρέφουμε το μονοπάτι προς τη διεργασία αυτή και επιστρέφουμε στην κύρια είσοδο.

3.4. Συμπληρωματικές συναρτήσεις και δηλώσεις

Στο αρχείο defs.h έχουμε ορίσει τις μακροαντικαταστάσεις *ENTRY(i)* και *SYSTEM_PROCESS(i),* έτσι ώστε να αντικαθίστανται με ετικέτες της μορφής case i. Αυτό βοηθάει στο να φαίνεται η σχέση μεταξύ των εντολών switch και των σημείων εισόδου ή των διεργασιών του συστήματος. Ο εντοπισμός λοιπόν του σημείου εισόδου i (που αναφέρεται στο κείμενο αυτό) γίνεται με την αναζήτηση της ετικέτας ENTRY(i) στον κώδικα της κατάλληλης συνάρτησης. Στο αρχείο vardefs.h έχουμε μία σειρά από βοηθητικές μεταβλητές (μετρητές στατιστικών και μεγίστων καταχωρήσεων στο σύστημα) καθώς και πίνακες με τις αιτίες απόρριψης χρηστών και ανώμαλου τερματισμού διεργασιών. Τέλος, στο αρχείο utils.c έχουμε μία σειρά από συναρτήσεις που εξομαλύνουν τις διαφορές μεταξύ των συστημάτων για τα οποία γράφτηκε το πρόγραμμα κατά καιρούς.

Η συνάρτηση *cputim* επιστρέφει το χρόνο σε δευτερόλεπτα από την πρώτη της κλήση. Αυτό γίνεται μέσω της παραμέτρου double *tim. Η συνάρτηση επιστρέφει τον πραγματικό χρόνο εκτέλεσης του προγράμματος από την πρώτη της κλήση και διαφέρει από το ANSI C σύστημα στο UNIX σύστημα. Στο σύστημα UNIX η κλήση times επιστρέφει (μέσω παραμέτρων, όχι σαν αποτέλεσμα) ακριβώς το χρόνο αυτό σε χτύπους ρολογιού, ενώ στο σύστημα ANSI η κλήση clock επιστρέφει την ώρα της ημέρας, με αποτέλεσμα να κρατάμε την ώρα που άρχισε να εκτελείται το πρόγραμμα σε μία στατική τοπική μεταβλητή. Στην PL/1 η cputim είναι ενσωματωμένη συνάρτηση. Εδώ πρέπει να παρατηρήσουμε ότι σε ένα σύστημα πολλών χρηστών όπως το UNIX, ο χρόνος εκτέλεσης αναφέρεται στο χρόνο που διέθεσε ο επεξεργαστής στη διεργασία (στο χρήστη) και όχι στον πραγματικό χρόνο που πέρασε από την αρχή εκτέλεσης της διεργασίας. Αντίθετα, στο MS-DOS, όπου έγινε η αρχική υλοποίηση σε ANSI C, ο χρόνος που επιστρέφει η clock είναι ο πραγματικός χρόνος εκτέλεσης, αφού έχουμε ένα χρήστη μόνο. Αν θέλουμε τον πραγματικό χρόνο από την στιγμή έναρξης μέχρι την στιγμή κλήσης στα συστήματα UNIX, μπορούμε να χρησιμοποιήσουμε είτε την κλήση times είτε την επιστρεφόμενη τιμή της clock που σχετίζονται με τον πραγματικό χρόνο στο σύστημα. Αυτό θα έχει σαν αποτέλεσμα οι δείκτες απόδοσης του συστήματος να ποικίλλουν ανάλογα με το φόρτο του συστήματος.

Οι συναρτήσεις *isum* και *lsum* αθροίζουν τα στοιχεία ενός πίνακα με όρια int l (κάτω) και int r (άνω) και επιστρέφουν μία τιμή long int στο όνομά τους. Η lsum παίρνει σαν παράμετρο έναν πίνακα τύπου long int a[], ενώ η isum έναν πίνακα τύπου int a[]. Αυτό γίνεται αυτόματα στην PL/1. Τέλος, η συνάρτηση *substr* επιστρέφει μία τιμή unsigned int η οποία περιέχει n (δυαδικά) ψηφία από τα ψηφία ενός αριθμού x, αρχίζοντας από τη θέση του ψηφίου p μετρώντας από αριστερά προς τα δεξιά. Ουσιαστικά απομονώνουμε

τις τιμές των n ψηφίων του x στο σημείο αυτό, για λόγους σύγκρισης πεδίων τύπου bit. Όλες οι παράμετροι (x, p, n) είναι unsigned int, με την υπόθεση ότι ο τύπος int είναι 16 bit. Η συνάρτηση αυτή υπάρχει ενσωματωμένη στην PL/1.

4. Διαφορές μεταξύ των εκδόσεων και γενικές παρατηρήσεις

Εκτός από τη διαφορά στη συνάρτηση *cputim*, λόγω της διαφορετικής σημασιολογίας των κλήσεων *clock* και *times* (που επηρεάζουν και τη σημασιολογία των στατιστικών του προγράμματος), η μόνη άλλη διαφορά της έκδοσης σε UNIX C (υλοποίηση σε σύστημα ULTRIX 32 της DEC, τροποποίηση 4.3 BSD) και της έκδοσης σε ANSI C (υλοποίηση σε Turbo C++ 1.01 της Borland για MS-DOS) είναι ο τρόπος με τον οποίο δηλώνονται οι παράμετροι των συναρτήσεων. Επίσης οι δηλώσεις των συναρτήσεων διαφέρουν και στο αρχείο defs.h (η ANSI C χρησιμοποιεί πρωτότυπα συναρτήσεων, ενώ η C του UNIX ενδιαφέρεται μόνο για την επιστρεφόμενη τιμή). Στο θέμα των δηλώσεων, παρατηρήστε ότι στη UNIX C δεν υπάρχει το αρχείο alloc.h για τη διαχείριση μνήμης (οι σχετικές συναρτήσεις δηλώνονται στο stdlib.h), που υπάρχει στην ANSI C. Για να μπορέσει τελικά να υπάρξει μόνο μία έκδοση των αρχείων, χρησιμοποιήθηκαν ευρέως οι εντολές υπό συνθήκη μεταγλώττισης που παρέχει ο προεπεξεργαστής της γλώσσας C. Τυχόν μελλοντικές εκδόσεις θα πρέπει να ακολουθήσουν τις ίδιες μεθόδους για την διασφάλιση της συμβατότητας. Για μεταφορά σε άλλα συστήματα UNIX που δεν παρέχουν μεταγλωττιστή ANSI C, μπορεί να χρειαστεί και αλλαγή στις εκχωρήσεις, αφού οι παλιοί μεταγλωττιστές δεν επιτρέπουν εκχώρηση δομών (αυτό ισχύει και για την παλιά έκδοση για UNIX Version 7). Αυτό γενικά θα επιβραδύνει το σύστημα, αν και όχι σημαντικά. Έχει γίνει προσπάθεια να διατηρηθεί συμβατότητα μεταξύ των συστημάτων, με αποτέλεσμα να μην εκμεταλλευόμαστε για παράδειγμα όλα τα σήματα (signals) του UNIX, ούτε άλλες ιδιαιτερότητές του. Τελικά, η συμπεριφορά των συστημάτων και τα αποτελέσματα θα πρέπει να είναι ίδια, εφόσον χρησιμοποιούμε αριθμητικούς τύπους με τα ίδια μεγέθη και οργάνωση. Αυτό είναι μάλλον απίθανο, όταν κινούμαστε από έναν προσωπικό υπολογιστή σε ένα μίνι υπολογιστή, αλλά κρίθηκε καλύτερο τα όρια των αριθμών να επιβάλλονται μέσω τυποποιημένων αρχείων και να χρησιμοποιούνται όσο περισσότερα ψηφία μπορεί να διαθέσει το κάθε σύστημα. Για να γίνει, πάντως, έλεγχος εγκυρότητας κατά τη μεταφορά του προγράμματος από σύστημα σε σύστημα, μπορούν να αντικατασταθούν οι σταθερές MAXINT και MAXFLOAT με κάποιον αριθμό που να είναι αποδεκτός και από τα δύο συστήματα μεταξύ των οποίων γίνεται η σύγκριση (αν και οι διαφορές μεταξύ των υλοποιήσεων των αριθμητικών πράξεων σε πραγματικούς αριθμούς θα έχουν σαν αποτέλεσμα αποκλίσεις στη συμπεριφορά των συστημάτων).

Σε σχέση με την προηγούμενη έκδοση (για Version 7 UNIX) η πιο εμφανής μεταβολή είναι η μεταφορά στα ελληνικά και ο πλήρης εσωτερικός σχολιασμός κώδικα. Διορθώθηκαν δύο λάθη (ένας έλεγχος υπέρβασης των ορίων ενός πίνακα και μία παράλειψη της ενημέρωσης του πλήθους των τερματικών που χρησιμοποιούνται) και πιθανότατα εισήχθηκαν αρκετά άλλα (μέχρι τώρα δεν έχει βρεθεί κάποιο που να μην έχει διορθωθεί, παρά το ότι οι εκδόσεις έχουν εκτελεστεί για 8 ώρες στο MS-DOS και για 24 ώρες στο UNIX). Η άλλη βασική διαφορά είναι ότι έγινε μία προσπάθεια για βελτιστοποίηση του κώδικα και προσκόλληση στα πρότυπα της ANSI, με στόχο το πρόγραμμα να γίνει πιο συντηρήσιμο και εύκολα μεταφέρσιμο (το δεύτερο το πετύχαμε, το πρώτο όχι απόλυτα). Επίσης, το πρόγραμμα διασπάστηκε σε ανεξάρτητες ενότητες οι οποίες μπορούν να μεταγλωττίζονται χωριστά και να συνδέονται όποτε χρειάζεται, χωρίς να περνάει όλο το σύστημα από το μεταγλωττιστή. Συνίσταται η χρήση ενός προγράμματος τύπου *make* για διευκόλυνση στη μεταγλώττιση και σύνδεση του συστήματος (με τον τρόπο αυτό ρυθμίζεται και το θέμα της επιλογής της κατάλληλης έκδοσης για το μεταγλωττιστή που χρησιμοποιούμε). Ταυτόχρονα, έγινε μία προσπάθεια να μεταφερθούν οι μεταβλητές από τα καθολικά αρχεία δηλώσεων στις διαδικασίες ή τουλάχιστον στα αρχεία που τις χρειάζονται, πράγμα που οδήγησε σε αρκετά λάθη (μερικά από τα οποία μπορεί να μην έχουν βρεθεί ακόμη), αλλά ξεκαθάρισε τις λειτουργίες του κώδικα. Ευτυχώς, στη C παρέχονται πολλά είδη μεταβλητών, με πιο ενδιαφέρουσες τις μεταβλητές *static* που δεν παρέχονται σε πολλές άλλες γλώσσες. Οι μεταβλητές τύπου static μέσα σε συναρτήσεις έχουν μόνιμες τιμές, αλλά είναι ιδιωτικές για τη συνάρτηση (τέτοιες υπάρχουν και στην PL/1). Οι μεταβλητές τύπου static εκτός συναρτήσεων είναι ορατές σε όλες τις συναρτήσεις ενός αρχείου και είναι επίσης μόνιμες. Αυτές είναι ιδιαίτερα χρήσιμες για την υλοποίηση κοινών μεταβλητών μεταξύ μίας ομάδας μόνο συναρτήσεων που θα μπουν στο ίδιο αρχείο κώδικα (για παράδειγμα τις συναρτήσεις πολιτικής).

Δυστυχώς, η αρχική δομή του προγράμματος σε PL/1 (η οποία αντανακλάται πλήρως στην έκδοση για C στο Version 7 UNIX) δεν έδινε πολλά περιθώρια ξεκαθαρίσματος, με αποτέλεσμα το μεγάλο όγκο των κοινών δηλώσεων μεταβλητών. Κάπου, πάντως, ορισμένες αλλαγές που θα μπορούσαν να γίνουν αποφεύχθηκαν λόγω προσπάθειας για μείωση του υπολογιστικού κόστους (που είναι και η αιτία για την μάλλον άναρχη αρχική μορφή του προγράμματος), πράγμα που θα συνέβαινε αν αντικαθιστούσαμε όλες τις μεταβιβάσεις τιμών με καθολικές μεταβλητές με μεταβίβαση μέσω παραμέτρων. Πολλές αλλαγές κρίθηκαν επίσης επικίνδυνες από άποψης διατήρησης της ορθότητας του προγράμματος και αποφεύχθηκαν. Τέλος, άλλες αλλαγές "κόλλησαν" στην αύξηση της πολυπλοκότητας του κώδικα. Συνολικά, έγιναν οι αλλαγές αυτές που κρίθηκε ότι συνεισφέρουν στην

επαύξηση της κατανοησιμότητας και σταθερότητας του προγράμματος, χωρίς να επηρεάζουν δραστικά την αρχική του μορφή.

Σε πολλά σημεία χρησιμοποιήθηκαν μακροορισμοί αντί για τις "κωδικές" τιμές, αν και αυτό θα πρέπει να επεκταθεί σε πολύ μεγαλύτερο βαθμό για να γίνει το πρόγραμμα πραγματικά κατανοητό, αν βέβαια λυθεί το πρόβλημα της επιλογής μεταξύ της ονομασίας των αριθμητικών τιμών και των ετικετών με *case*. Επίσης, δε θα έβλαπτε μία μεγαλύτερη μείωση των εντολών *goto*, αν και αυτό μπορεί να γίνει μόνο με πολύ μεγάλη δυσκολία, αφού διασταυρώνονται προς όλες τις κατευθύνσεις και πολλές φορές είναι αναπόφευκτες λόγω της φύσης των επανεισαγόμενων διαδικασιών. Προσπάθησα πάντως να τους δώσω όσο γίνεται πιο κατανοητά ονόματα, για να μην μπερδεύουν τον αναγνώστη του κώδικα. Δυστυχώς, πάντως, μόνο με αρκετή επανασχεδίαση του κώδικα μπορούν να φύγουν τα περισσότερα από αυτά τα goto που έμειναν. Φυσικά όταν έχουμε βρόγχο που διακόπτεται και ο έλεγχος μεταφέρεται μέσα και έξω από αυτόν, πολύ λίγα μπορούμε να κάνουμε πέρα από το να χαιρόμαστε που δεν χρειάζεται να γράψουμε κάτι τέτοιο σε assembly.

Μία τελευταία παρατήρηση είναι ότι η εκφραστική δύναμη της C μπόρεσε με ευκολία να αντιμετωπίσει τον όγκο της PL/1, η οποία ήταν (και είναι) μία τερατώδης γλώσσα. Βέβαια, λείπουν οι πίνακες δυναμικού μεγέθους και ορισμένες ρουτίνες, αλλά όλα αντιμετωπίζονται με παρόμοιο τρόπο, χωρίς μεγάλο κόστος ή καταστροφικές συνέπειες. Το βασικό πρόβλημα της C είναι ότι όχι μόνο επιτρέπει σχεδόν τα πάντα (όπως για παράδειγμα έξοδο από δομή switch και είσοδο σε μία άλλη δομή switch λίγο παραπέρα), αλλά δεν ελέγχει κι αν έχουν νόημα όλα αυτά που επιτρέπονται, με αποτέλεσμα να είναι πολύ εύκολο το σύστημα να καταστραφεί από ένα δείκτη εκτός ορίων. Αυτό βέβαια γίνεται σε όλες τις γλώσσες, αλλά στη C όπου οι πίνακες προσπελάζονται με δείκτες, μπορεί να φέρει τον προγραμματιστή σε καταστάσεις προχωρημένης παράνοιας. Το μόνο που σώζει την κατάσταση είναι η επαναληψιμότητα των αποτελεσμάτων λόγω των τυχαίων σειρών, αν και αυτό έχει μικρή αξία όταν τα λάθη εμφανίζονται μετά από ώρες εκτέλεσης. Ο συνδυασμός της δυσκολίας εντοπισμού λαθών, λόγω όγκου του κώδικα, με τη δυσκολία ανεύρεσης των σημείων που χρησιμοποιούνται και ορίζονται οι μεταβλητές, είναι ο κύριος λόγος που οι μεταβλητές μετακινήθηκαν σε τοπικά αρχεία. Τελικά βέβαια, ο αρχικός σχεδιασμός κερδίζει τη μάχη, πράγμα αναμενόμενο, αφού ένα σύστημα προσομοίωσης λειτουργικών συστημάτων είναι μάλλον απίθανο να έχει λιγότερα bugs από ένα λειτουργικό σύστημα. Ίσως φταίει που τα γράφουμε σε C (αν κάποιος θέλει να γράψει το σύστημα σε C++, ας το ξανασκεφτεί πρώτα, και αν επιμένει, ας το γράψει από την αρχή αντί να το μεταφέρει - το καλό πρόγραμμα φαίνεται από τις κοινές του δηλώσεις). Έχετε πάντως υπόψη σας ότι το μόνο (άλλο) λειτουργικό σύστημα που γράφτηκε σε

PL/1 ήταν το MULTICS (καλή ιδέα, κακή εκτέλεση). Ο κώδικας σε PL/1 για το σύστημα αυτό, είναι στη διάθεση παντός ενδιαφερομένου. Έχει, πάντως, μεγάλη σημασία το ότι ένα τέτοιο πρόγραμμα μπόρεσε να υλοποιηθεί, με την ίδια σχεδόν πολυπλοκότητα με το αρχικό, σε μία γλώσσα τόσο λιτή όσο η C, ενώ η αρχική έκδοση ήταν για μία γλώσσα η οποία ποτέ δεν κυκλοφόρησε σε πλήρη μορφή σε έναν μόνο μεταγλωττιστή. Η αποδοτικότητα της εκτέλεσης είναι επίσης πολύ εντυπωσιακή σε σχέση με τα παλιότερα δεδομένα, συνυπολογίζοντας και την αύξηση του χρόνου εκτέλεσης λόγω της διάσπασης του προγράμματος.

Θα πρέπει επιπλέον να σημειώσουμε ότι για την τυποποιημένη από την ANSI έκδοση της C χρησιμοποίησα ως οδηγό αναφοράς τα παραρτήματα του [KER88], όπου μπορούν να βρεθούν οι περιγραφές των πιο "ασυνήθιστων" συναρτήσεων βιβλιοθήκης (όπως την *longjmp* και την *setjmp*). Αν και δε χρησιμοποιήθηκαν ιδιαίτερα χαρακτηριστικά της UNIX C, οι αποκλίσεις μπορούν εύκολα να εξηγηθούν με χρήση των on-line ή των τυπωμένων εγχειριδίων αναφοράς που είναι διαθέσιμα σε κάθε σύστημα UNIX ή με αναδρομή σε κάποιο βιβλίο που να περιέχει τις κλήσεις του UNIX (προσωπικά, χρησιμοποίησα το [BOU87], το οποίο θεωρητικά περιγράφει το System V, αλλά στην πραγματικότητα περιορίζεται στον κοινό παρονομαστή μεταξύ System V, BSD 4.2 και Version 7, για να αποφύγω προβλήματα συμβατότητας μεταξύ των εκδόσεων του UNIX). Για τις παλιότερες εκδόσεις της C (κυρίως για UNIX συστήματα, συμπεριλαμβανομένου του Version 7), συμβουλευτείτε την παλιά έκδοση του βιβλίου των Kernighan και Ritchie. Για την PL/1, αναζητήστε σχετικά συγγράμματα σε κάποιο μουσείο ή ρωτήστε το συγγραφέα του αρχικού προγράμματος.

Αξίζει, τελειώνοντας αυτή την παράγραφο, να αναφέρω ένα πρόβλημα που δυσκόλεψε ιδιαίτερα τη μεταφορά του συστήματος από μηχανή σε μηχανή (όταν γράφτηκε η πρώτη έκδοση αυτού του κειμένου, το πρόγραμμα "κόλλαγε" στο UNIX εξ' αιτίας του προβλήματος αυτού). Η αρχική έκδοση της συνάρτησης *cputim* για το UNIX, χρησιμοποιούσε μία συνάρτηση η οποία είχε τη συνήθεια να παίρνει κυκλικά τις τιμές από το πεδίο τιμών της, δηλαδή μετά από κάποιο χρόνο, οι τιμές που επέστρεφε άρχιζαν πάλι από το μηδέν, λόγω ελλιπούς χώρου στα επιστρεφόμενα αποτελέσματα. Το αποτέλεσμα αυτής της συμπεριφοράς ήταν να παίρνουμε μηδενική τιμή για το χρόνο εκτέλεσης του προγράμματος και τελικά να έχουμε διαίρεση με το μηδέν, μετά από αρκετές (2 για την ακρίβεια) ώρες εκτέλεσης. Όπως γίνεται συνήθως, το πρόβλημα λύθηκε μετά από αρκετές ώρες με το dbx (το πρόγραμμα εκσφαλμάτωσης του UNIX) και μετά από πολλές προσπάθειες να βρεθεί μία λογική εξήγηση για τα παρατηρούμενα προβλήματα (παρεμπιπτόντως, η δεύτερη τεχνική έδωσε το ποθητό αποτέλεσμα). Φυσικά, το πρόβλημα δεν θα είχε εμφανιστεί καθόλου, αν η ανάγνωση των εγχειριδίων του UNIX στο θέμα των κλήσεων χρόνου είχε

γίνει με περισσότερη επιμέλεια, οπότε και θα είχε αποκαλυφθεί η ιδιομορφία αυτή. Το δίδαγμα εδώ είναι ότι τα σφάλματα που εισάγονται από απροσεξία, σε ένα μεγάλο πρόγραμμα έχουν τεράστιο κόστος διόρθωσης, αν βέβαια παρατηρηθούν έγκαιρα. Γι' αυτό, απαιτείται να επενδύσουμε πολύ χρόνο στην κατανόηση του συστήματος που χρησιμοποιούμε και ιδιαίτερα στα σημεία που διαφέρουν από τα τυποποιημένα μέρη του (όπως διαφέρουν οι κλήσεις συστήματος του UNIX και η βιβλιοθήκη της γλώσσας C για το UNIX από τη βιβλιοθήκη της ANSI C).

5. Προτεινόμενες βελτιώσεις, επεκτάσεις και εργασίες

Αν και πολύ θα θέλαμε αυτή η εργασία να έχει σαν παραδοτέο προϊόν ένα πρόγραμμα που να επιδέχεται ελάχιστες βελτιώσεις, εν μέρει λόγω του ήδη μεγάλου φόρτου για τη μετατροπή στα ελληνικά και στην τυποποιημένη διάλεκτο της C, εν μέρει λόγω της αρχικής επιλογής να μη γίνουν σημαντικές μετατροπές στη λογική ροή του προγράμματος (για να μπορεί να συγκριθεί και να επικυρωθεί μέσω των προηγούμενων εκδόσεων) και εν μέρει λόγω του ίδιου του όγκου του προγράμματος, που κάνει μάλλον αδύνατη τη συνολική βελτιστοποίησή του (με όποια έννοια και αν χρησιμοποιήσουμε τη λέξη βελτιστοποίηση), το τελικό αποτέλεσμα δεν αντέχει ιδιαίτερα σε κριτική πάνω στην ποιότητα του κώδικα. Αν και η βασική ιδέα της υλοποίησης μέσω των διακοπτόμενων συναρτήσεων επιβάλλει κάποια όρια στις μεταβολές που μπορούν να γίνουν (βλέπε και [CAV83-1]), πολλές δομές ελέγχου μπορούν να μεταβληθούν, όπως για παράδειγμα τα τοπικά *goto* μέσα στις διεργασίες του συστήματος, τα οποία θα μπορούσαν μέσω αναδιάταξης του κώδικα να γίνουν *break* ή *continue*, με πολύ ευεργετικά αποτελέσματα πάνω στην ποιότητα του κώδικα. Οι ίδιοι οι αλγόριθμοι επεξεργασίας, στο μεγαλύτερο μέρος τους, δεν επιδέχονται μεγάλες βελτιώσεις, αφού εκφράζουν κυρίως πολιτικές και η υλοποίησή τους γίνεται με ένα σχετικά περιορισμένο τρόπο. Ορισμένα, πάντως, σημεία όπου γίνεται επαναληπτική σάρωση λιστών (όπως, για παράδειγμα, όταν ψάχνουμε τη λίστα κατειλημμένων τεμαχίων για να εκλέξουμε ένα τεμάχιο θύμα) μπορούν να εκφραστούν με πιο αποδοτικό τρόπο (μία σάρωση με χρήση προσωρινών μεταβλητών για τα ενδιάμεσα αποτελέσματα), βελτιώνοντας έτσι τις επιδόσεις του συστήματος.

Μία αρκετά σημαντική ποιοτική βελτίωση θα ήταν ο τεμαχισμός του αρχείου δηλώσεων τύπων (defs.h) και του αρχείου δηλώσεων μεταβλητών (vardefs.h) σε τμήματα που να σχετίζονται με απομονωμένες μονάδες της μεταγλώττισης (για παράδειγμα, μία χωριστή μονάδα δηλώσεων για τις συναρτήσεις του διαχειριστή της μνήμης), έτσι ώστε να έχουμε τις κοινές μεταβλητές γνωστές μόνο στις μονάδες που τις χρειάζονται (συνήθως, αυτές

είναι ο πυρήνας, οι ρουτίνες λαθών και στατιστικών και τα τμήματα κώδικα που εκτελούν τις σχετικές με τις δηλώσεις λειτουργίες). Με το να τεμαχίσουμε τις δηλώσεις των μεταβλητών μπορούμε να χρησιμοποιούμε τη δήλωση *#include* του προεπεξεργαστή της C για να συμπεριλάβουμε σε κάθε μονάδα της μεταγλώττισης μόνο τις δηλώσεις που χρειάζεται να γνωρίζει αυτή, πετυχαίνοντας έτσι μεγαλύτερη ασφάλεια και στεγανότητα μεταξύ των τμημάτων του κώδικα. Πάνω στο θέμα αυτό, πρέπει να σημειώσω ότι παρά το ότι έχει γίνει προσπάθεια να μεταφερθούν οι μεταβλητές μόνο στα σημεία που χρειάζονται, δηλαδή οι καθολικές μεταβλητές να γίνουν τοπικές σε ένα αρχείο ή τοπικές (*static* ή *auto*) μέσα σε συναρτήσεις, και να μη χρησιμοποιούνται για διαφορετικά καθήκοντα μέσα στον κώδικα, υπάρχουν ακόμη μεγάλα περιθώρια βελτίωσης. Μία τελευταία βελτίωση, που θα έχει όμως σημαντικό κόστος χρόνου, είναι να αποφευχθεί το πέρασμα παραμέτρων μέσω καθολικών μεταβλητών (για παράδειγμα, στις συναρτήσεις που δρομολογούν γεγονότα) και να χρησιμοποιηθούν κανονικές τυπικές και πραγματικές παράμετροι για την επικοινωνία μεταξύ των συναρτήσεων.

Μία βελτίωση ύφους, που έχει όμως σημαντικό αντίκτυπο στην αναγνωσιμότητα του κώδικα (χωρίς μάλιστα επίδραση στο χρόνο εκτέλεσης), είναι η συνολική αντικατάσταση των (κωδικών) αριθμητικών τιμών που χρησιμοποιούνται στο σύστημα για να δίνονται εντολές στις διεργασίες (μέσω του πεδίου *mcommand* στα μηνύματα) με συμβολικές τιμές (χρησιμοποιώντας τη δήλωση *#define* του προεπεξεργαστή της C). Το πρόβλημα είναι να συγκεντρωθούν οι τιμές, να διασταυρωθούν και να οριστούν σαφή και περιγραφικά ονόματα. Αυτό δεν έγινε στην έκδοση αυτή, επειδή προτιμήθηκε να κρατηθεί η αρχική μορφή της υλοποίησης, όπου αντί για τις αριθμητικές τιμές κωδικοποιούσαμε με συμβολικές σταθερές τις ετικέτες, δηλαδή, αντί να κωδικοποιήσουμε την τιμή i κωδικοποιούσαμε την *case i*, με αποτέλεσμα να μην μπορούμε να κάνουμε εκχώρηση της σταθεράς αυτής στο πεδίο mcommand, αφού αυτή δεν παριστάνει έναν ακέραιο αριθμό. Με την αλλαγή σε κωδικοποίηση των αριθμητικών σταθερών, θα έχουμε ετικέτες της μορφής case CM_CREATE και οι εντολές θα δίνονται με εκχωρήσεις της μορφής mcommand=CM_CREATE. Επί του ύφους, σημειώστε ότι δεν είναι μόνο η απουσία ετικετών που δυσχεραίνει την κατανόηση ορισμένων ενεργειών, αλλά και η απουσία λογικών ονομάτων σε ορισμένες βασικές μεταβλητές (για παράδειγμα, ποιος θα σκεφτόταν τι χρησιμεύει η μεταβλητή p3 μόνο βλέποντας το όνομά της;).

Από πλευράς επεκτάσεων, παρατηρούμε ότι το σύστημα έχει σχεδιαστεί με τέτοιον τρόπο, ώστε να μπορεί να εξελιχθεί εύκολα με τη δημιουργία νέων διεργασιών ή με την αντικατάσταση των ήδη υπαρχουσών. Μπορούμε, έτσι, να αντικαταστήσουμε το διαχειριστή αρχείων με ένα "πραγματικό" σύστημα αρχείων, το οποίο να προσομοιώνει με ακρίβεια τις λειτουργίες ενός

πραγματικού συστήματος αρχείων (να παρέχει δηλαδή προσομοιωμένες υπηρεσίες αρχείων και ευρετηρίων), αντί να είναι ενδιάμεση διεργασία μεταξύ των διεργασιών των χρηστών και του διαχειριστή του δίσκου. Ανάλογα, μπορούμε να δημιουργήσουμε νέους διαχειριστές συσκευών ή να τροποποιήσουμε τους παλιούς, για παράδειγμα χρησιμοποιώντας έναν αλγόριθμο βελτιστοποίησης των κινήσεων του βραχίονα της κεφαλής του δίσκου. Μπορούμε επίσης να επεκτείνουμε το σύστημα προσθέτοντας πολλές μονάδες από κάθε είδος συσκευής (όπως πολλούς δίσκους) για να μοιραστεί ο φόρτος εργασίας. Έχουμε ήδη αναφέρει ότι θα μπορούσε να δημιουργηθεί ένας νέος βελτιστοποιημένος διαχειριστής μνήμης, ειδικά για το σύστημα σελιδοποίησης. Ανάλογα, μπορεί να δημιουργηθεί ένας διαχειριστής μνήμης πολλών επιπέδων (σελιδοποίησης με τεμαχισμό). Μία επέκταση άλλης μορφής, μπορεί να είναι η γενίκευση της μορφής των ομάδων διεργασιών, από ρίζα και ένα επίπεδο παιδιών, σε γενικευμένα δένδρα (όπως στο UNIX). Ανάλογα, δανειζόμενοι κι άλλες ιδέες από υπάρχοντα λειτουργικά συστήματα, μπορούμε να επεκτείνουμε το σύστημα δημιουργώντας νέες πρωτογενείς κλήσεις και νέες μεθόδους επικοινωνίας, χρησιμοποιώντας τις ήδη υπάρχουσες πρωτογενείς κλήσεις. Πρακτικά, οι επεκτάσεις που μπορούν να γίνουν είναι απεριόριστες, αφού το σύστημα θεωρητικά μπορεί να εξελιχθεί μέχρι να φτάσει στη μορφή ενός πραγματικού λειτουργικού συστήματος, όπως αναφέρεται στο [CAV78]. Θα ήταν επίσης πολύ ενδιαφέρον να δοκιμαστεί η εγκυρότητα αυτού του τελευταίου ισχυρισμού (δείτε το [TAN87] για την υλοποίηση ενός UNIX-like συστήματος σε C, του MINIX, για σύγκριση με την υλοποίηση αυτή και για να πάρετε ιδέες και κατευθυντήριες γραμμές για τη δημιουργία πρόσθετου ή νέου κώδικα).

Πέρα από τις παραπάνω εργασίες που πρέπει ή μπορεί να γίνουν για να βελτιωθεί ή να τροποποιηθεί το σύστημα, δε θα πρέπει να ξεχνάμε το βασικό στόχο του συστήματος, που είναι να παρέχει στοιχεία για την αποτελεσματικότητα της υλοποίησης ενός λειτουργικού συστήματος. Έτσι, μεταβάλλοντας τις παραμέτρους λειτουργίας του συστήματος, μπορούμε να δούμε την επιρροή που έχουν στην απόδοσή του. Οι παράμετροι του υλικού μπορούν να μεταβληθούν για να μελετήσουμε την αποτελεσματικότητα συσκευών διαφορετικής ποιότητας και οι παράμετροι του λογισμικού (για παράδειγμα, οι συναρτήσεις πολιτικής) μπορούν να τροποποιηθούν, έτσι ώστε το σύστημα να προσομοιώνει διαφορετικές πολιτικές καταχώρησης πόρων. Φυσικά, μπορούμε να πειραματιστούμε και με πιο δραστικές αλλαγές, αλλάζοντας για παράδειγμα τον κώδικα ολόκληρου του διαχειριστή μνήμης, ή κάποιων τμημάτων του (όπως τον κώδικα της πολιτικής αντικατάστασης ή τοποθέτησης). Λόγω πάντως της ευελιξίας του συστήματος, μπορούμε να πειραματιστούμε με ριζικά διαφορετικές πολιτικές, μεταβάλλοντας μόνο παραμέτρους, πράγμα που οφείλεται κυρίως στη γενικότητα της αντιμετώπισης ορισμένων βασικών θεμάτων. Ένα παράδειγμα αυτής της γενικότητας είναι η

χρήση παραμετρικών συναρτήσεων πολιτικής, που με διαφορετικές παραμέτρους μπορούν να προσομοιώσουν διάφορες γενικές πολιτικές διαχείρισης της ΚΜΕ (τέτοια αποτελέσματα με χρήση διαφορετικών παραμέτρων για το σύστημα παρουσιάζονται στο [MOH84]). Ένα ακόμη σημείο που μπορεί να έχει αρκετό ενδιαφέρον (και δεν έχει καλυφθεί ως τώρα) είναι η μελέτη του συστήματος με χρήση εμπειρικών δεδομένων για τη συμπεριφορά των χρηστών.

Κλείνοντας, θέλω να ευχαριστήσω το Θ. Μπεγέλη που μου διέθεσε την υπολογιστική ισχύ που μου χρειαζόταν για τον έλεγχο λειτουργίας του προγράμματος σε μακρά χρονικά διαστήματα στο MS-DOS, και να μοιραστώ τις ευθύνες για τα σφάλματα στην παρουσίαση αυτή (και τη σχετική δόξα!) με τον επιβλέποντα της εργασίας Ι. Κάβουρα που διάβασε το αρχικό κείμενο και εισηγήθηκε (επέβαλλε) σωρεία αλλαγών, διορθώσεων και βελτιώσεων. Εύχομαι καλή τύχη σε όσους θέλουν να πειραματιστούν τροποποιώντας ή βελτιώνοντας το πρόγραμμα αυτό, είτε για πειραματισμό είτε για να διορθώσουν ή να βελτιστοποιήσουν τη λειτουργία του. Οπλιστείτε με προσοχή και υπομονή, γιατί ακόμη και οι απλές επεμβάσεις σε τόσες χιλιάδες γραμμές κώδικα είναι σημαντικές προγραμματιστικές ασκήσεις. Μάλλον λοιπόν η τύχη θα σας φανεί χρησιμότερη από τις γνώσεις. Για όσους εξαναγκαστούν να ασχοληθούν με το πρόγραμμα, ζητάω προκαταβολικά συγγνώμη για την πολύ άναρχη μορφή του και τις ελλείψεις στην τεκμηρίωση, αλλά ακόμη και η δική μου υπομονή ήταν περιορισμένη και οι δυνατότητες αντοχής μου πεπερασμένες.

6. Βιβλιογραφία

Ακολουθεί η λίστα των συγγραμμάτων και δημοσιεύσεων στα οποία γίνεται αναφορά στις προηγούμενες σελίδες, διατεταγμένη κατά χρονολογική σειρά. Πέρα από τις δημοσιεύσεις που σχετίζονται άμεσα με το σύστημα, τα υπόλοιπα προτεινόμενα βιβλία είναι ενδεικτικά των συμπληρωματικών κειμένων που μπορούν να χρησιμοποιηθούν για ευχερέστερη κατανόηση του παρόντος και έχουν επιλεγεί με βάση την ευκολία ανεύρεσής τους από όποιον ενδιαφέρεται και το βαθμό στον οποίο χρησίμευσαν στο συγγραφέα του παρόντος κατά την επεξεργασία του προγράμματος. Φυσικά, από εμάς μπορείτε να βρείτε και τις παλιότερες εκδόσεις του προγράμματος (σε PL/I για IBM 370 και C για Unix Version 7). Μπορείτε να επικοινωνήσετε μαζί μας είτε για να βρείτε τις παλιότερες εκδόσεις του συστήματος ή τις σχετικές δημοσιεύσεις, είτε για να μας υποδείξετε προβλήματα ή λάθη στο πρόγραμμα, μέσω ηλεκτρονικού ταχυδρομείου (e-mail) στις διευθύνσεις *jc@aueb.gr* (Ι.Κ. Κάβουρας) και *xgeorge@aueb.gr* (Γ.Β. Ξυλωμένος).

[CAV78]: Cavouras J.C., The development and application of a method for producing software tools for computer systems design. Ph.D. Thesis, Department of Computing Science, University of Glasgow, 1978.

[CAV81-1]: Cavouras J.C. and Davis R.H., Simulation tools in computer system design methodologies. The Computer Journal, Vol. 24, No. 1, 1981.

[CAV81-2]: Cavouras J.C., Simulation of user processes. Computer Performance, Vol. 2, No. 4, Dec 1981.

[CAV83-1]: Cavouras J.C., Implementing a simulation tool in a high-level language with no multitasking facilities. Software Practice and Experience, Vol. 13, pp. 809-815, 1983.

[CAV83-2]: Cavouras J.C., Computer system evaluation through supervisor replication. The Computer Journal, Vol. 26, No. 2, 1983.

[MOH84]: Mohamad S.M.A. and Cavouras J.C., Performance study of descriptor-oriented architectures. Computer Performance, Vol. 5, No. 1, Mar 1984.

[BOU87]: Bourne S.R., The UNIX System V Environment, Addison-Wesley, 1987.

[TAN87]: Tanenbaum A.S., Operating Systems: Design and Implementation. Prentice Hall, 1987.

[KER88]: Kernighan B.W. and Ritchie D.M., The C programming Language. 2nd Edition, Prentice Hall, 1988 (υπάρχει και στα ελληνικά, με τον τίτλο "Η γλώσσα προγραμματισμού C", 2η έκδοση, Κλειδάριθμος, 1990).

[KAB94]: Κάβουρα Ι.Κ., Λειτουργικά Συστήματα / Συστήματα Υπολογιστών Τόμος ΙΙ. 3η έκδοση, Αθήνα 1994.

Πίνακας Περιεχομένων