|  |
| --- |
| Κωδικός Εργασίας (4.2)  **Ομάδα** [13]  **Μέλος A**: [Γεώργιος, Σιδέρης , 1679,sideris@uth.gr]  **Μέλος B**:[Φώτιος , Τσώκος, 1622, [tsokos@inf.uth.gr](mailto:tsokos@inf.uth.gr)]  1.Μία γενική ιδέα για τη δομή της εικονικής μηχανής  Πριν προχωρήσουμε στη περιγραφή και ανάλυση του αλγόριθμου με βάση των οποίο κατασκευάσαμε την εικονική μηχανή μας πρέπει να αναφερθούμε στα δομικά της στοιχεία και το τι ρόλο διαδραματίζουν στη λειτουργία του προγράμματος.  1.1 Tasks:  Το συγκεκριμένο struct περιέχει όλες τις απαραίτητες πληροφορίες για τη λειτουργία του κάθε task (όπως τους καταχωρητές , την αντίστοιχη τοπική του μνήμη , τη κατάσταση του κα.). Κάθε στοιχείο του task έχει ένα ξεχωριστό διακριτικό ,το id (€(0-number of tasks)) το οποίο το διαφοροποιεί από τα όμοια του στοιχεία.  1.2 CPU:  Πρόκειται για μία δομή δεδομένων η οποία περιέχει πληροφορίες για τον κάθε εικονικό επεξεργαστή (τον αριθμό των tasks που θα εκτελέσει και ένα πίνακα με τα id αυτών ) καθώς και το προσωπικό του mutex ώστε να μπλοκάρει όποτε είναι απαραίτητο. Όπως και το struct tasks η δομή cpu περιέχει και έναν αριθμό id για κάθε επεξεργαστή.  1.3 Sorting and sharing tasks functions:  Για να πετύχουμε μια πιο “δίκαια” κατανομή εργασιών σε κάθε cpu προσπαθήσαμε κάθε επεξεργαστής να λαμβάνει εργασίες με όσο το δυνατόν λιγότερα locks (αυτό διότι η ανάθεση εργασιών που το μεγαλύτερο μέρος του χρόνου “κοιμούνται” είναι σπατάλη της επεξεργαστικής ισχύως). Έτσι :  1) Για κάθε κώδικα κρατάμε τον αριθμό των Block που εντοπίζουμε σε αυτόν  2)Στη αντιστοιχία του κάθε task με ένα κώδικα συνδέουμε το task με τον αυτόν τον αριθμό  3)Κάνουμε ταξινόμηση των tasks κατά αύξοντα αριθμό των locks περιέχουν  4)Με τη σειρά που τα διατάξαμε τα κατανέμουμε σε κάθε cpu αλλάζοντας παράλληλα τη προσπέλαση των cpu από την αρχή προς το τέλος και αντίθετα σε κάθε γύρο (πχ την πρώτη φορά θα μοιράσουμε εργασίες από τον cpu [0] προς τον cpu[n] και την επόμενη από τον cpu[n] προς τον cpu[0]).  Ο παραπάνω αλγόριθμος αποκτά πρακτικό νόημα αν φανταστούμε ένα τρίγωνο (ο διεταγμένος πίνακας απο tasks) που θέλουμε με αυτό να κατασκευάσουμε ένα ορθογώνιο. Ο κατάλληλος τρόπος είναι να τεμαχίσουμε το τρίγωνο σε N τμήματα (no of tasks/ no of cpuz) και να τα διατάξουμε το ένα πάνω στο άλλο εναλλάξ. Έτσι προσεγγίζουμε αρκετά το ορθογώνιο. Ανάλογα στο πνεύμα του προγράμματος μας μπορούμε να αντιστοιχήσουμε το ευρύτερο ορθογώνιο σε M (number of cpuz) κομμάτια τα οποία αναπαριστούν το φόρτο εργασίας κάθε επεξεργαστή και το ύψος τους είναι περίπου ίδιο |
| **2. Το Πρόγραμμα**  2.1 Βασική λειτουργία  Μπορούμε να συνοψίσουμε τη βασική λειτουργία του αλγορίθμου στα παρακάτω:  1)Άνοιξε το binary αρχείο  2)Δίαβασε το binary αρχείο κάνοντας ελέγχους για την ορθότητα του και αποθηκεύοντας τα στοιχεία στις αντίστοιχες μεταβλητές, δημιούργησε τα tasks, δέσμευσε την αντίστοιχη μνήμη κλπ.  3)Μοίρασε τις εργασίες κατάλληλα στο κάθε επεξεργαστή  4)Άρχισε την εκτέλεση των tasks διαβάζοντας από τον κώδικα που αντιστοιχεί στο καθένα κάθε εντολή  5) Κάθε φορά που όλα τα tasks σε ένα cpu έχουν μπλοκάρει “κλείδωσε” τον  6)Κάθε φορά που ξεκλειδώνουμε ένα task γίνεται έλεγχος αν το cpu του είναι κλειδωμένο ξεκλειδώνοντας το αν είναι απαραίτητο  7)Συνέχισε μέχρι όλα τα άλλα cpu έχουν μπλοκάρει και το τελευταίο πρόκειται και αυτό να μπλοκάρει.  2.2 Σημεία συγχρονισμού  Όπως μπορούμε να φανταστούμε η διαχείριση της κοινής μνήμης (globalMem) για κάθε task αποτελεί ένα επικίνδυνο σημείο για το κώδικα. Με τη χρήση των mutexes ελέγχουμε ότι μόνο ένα νήμα θα εισέλθει σε αυτές τις περιοχές:  lock (&mutex check);  ...  interfereWith (globalMem);  ...  unlock (&mutex check);  Άλλο ένα κρίσιμο σημείο είναι το μπλοκάρισμα των cpu. Έτσι όταν όλα τα tasks είναι σε διαφορετική κατάσταση από ready τότε η cpu μπλοκάρει εφόσων το τελευταίο από αυτά στείλει σήμα σε αυτά.  Ο παρακάτω ψευτοκώδικας εξηγεί εν συντομία τη λειτουργία του κλειδώματος και της αφύπνισης των cpu.  Lock:  if (IsTimeToBlock()){    //state that this cpu is locked  blockedFlag[i]=1;    //lock  lock (mtx[i])    //when cpu is unlocked find and switch to the appropriate task      curr=NextCurr;      }    unlock:  ...after unlock-command  //check if the thread unlocked belonges to a blocked cpu  if (blockedFlag[i]!=0){    blockedFlag[tasks[i].slaveTO]=0;  //next curr for that cpu is this task  nextCurr =tasks[i].id;  //unlock cpu containing mutex  unlock (&mutex[i]);  } |
| Σημειώσεις:  1)Διαχωρίσαμε τη λειτουργία του Print integer (0x19) απο αυτή του print String (0x1b). Με το print integer εκτυπώνει απλά έναν ακέραιο με την print String εκτυπώνει string (χαρακτήρες μέχρι να βρει 0)  2)Δημιουργήσαμε 5 binary αρχεία για να δοκιμάσουμε το πρόγραμμα μας  factorial : βρίσκει το factorial ενός αριθμού  prime : επιστρέφει τον ίδιο τον αριθμό που του δίνεται αν είναι πρώτος ή το διεραίτη του αν υπάρχει  train: μοιάζει με το train του 2.3  min : από ένα σύνολο αριθμών βρίσκει το μικρότερο  max : το αντίθετο  3)Το πρόγραμμα διαβάζει τις μεταβλητές σε little endian μορφή. Για αυτό το λόγο εισάγουμε τα δυο bits του global memory ανάποδα  4)Ο binary κώδικας γράφτηκε μέσω του προγράμματος Okteta |