ΑΝΑΦΟΡΑ ΕΡΓΑΣΙΑΣ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΟΛΓΑ ΚΑΛΤΣΑ AM: p3220242

ΣΤΥΛΙΑΝΗ ΣΠΥΡΟΓΙΑΝΝΗ AM: p3220189

ΕΥΑΓΓΕΛΙΑ ΧΑΣΙΩΤΗ AM: p3120203

Η παρούσα αναφορά δημιουργήθηκε με σκοπό την αναλυτικότερη περιγραφή και επεξήγηση του προγράμματος παραγγελιοληψίας και διανομής πίτσας, που κληθήκαμε να εκπωνήσουμε σε γλώσσα c, στα πλαίσια του μαθήματος των Λειτουργικών Συστημάτων.

ΠΕΡΙΓΡΑΦΗ

p3220242-p3220189-p3120203-pizza.h

Εκτός από την εισαγωγή τυπικών αρχείων κεφαλίδας που περιέχονται στο file.c (<stdio.h>,<stdlib.h>,<pthread.h> ,<unistd.h>,<time.h>) που μας βοηθούν στην υλοποίηση διάφορων λειτουργιών που αφορούν νήματα, διαχείριση μνήμης και συναρτήσεων, δημιουργήσαμε και ένα μη τυπικό αρχείο κεφαλίδας. Στο p3220242-p3220189-p3120203-pizza.h έχει οριστεί μια δομή δεδομένων, η order, που περιγράφει μια παραγγελία και τις παραμέτρους της (αριθμός πιτσών, αριθμός παραγγελίας και τύπος πίτσας).

Επίσης, δηλώθηκαν τόσο σταθερές (αριθμός, τηλεφωνητών, σεφ, φούρνων, διανομέων, χρόνοι εξυπηρέτησης και κρυώματος), όσο και συναρτήσεις(get_random() : παραγαγωγή τυχαίων αριθμών , process_order() : επεξεργασία παραγγελίας). Τέλος, με την εκτέλεση του file.c ο header εκτελείται στη main.

p3220242-p3220189-p3120203-pizza.c

main()

Δέχεται,αρχικά, για παραμέτρους τον αριθμό των πελατών και το seed(η τυχαία παραγωγή αριθμών βάσει του seed) και γίνεται έλεγχος ορθής σύνταξης(εδώ εισάγουμε αυστηρά 2 ορίσματα:αριθμός πελατών, σπόρος). Όποτε, συνολικά,με το όνομα της συνάρτησης είναι 3.

(γραμμές κώδικα : 170-174)

Έπειτα γίνεται η αρχικοποίηση των μεταβλητών, όπου έχουμε ανάθεση των τιμών των ορισμάτων στις μεταβλητές μας Ncust και seed και αρχικοποίηση της γεννήτριας τυχαίων

αριθμών με τον srand.Δημιουργούμε 2 πίνακες(threads, orders) για την αποθήκευση νημάτων και παραγγελιών αντίστοιχα. (γραμμές κώδικα : 176-181)

Κατόπιν, γίνεται η επεργασία των παραγγελιών. Όταν δημιουργείται μια παραγγελία για κάθε πελάτη, ξεκινάει ένα νήμα για την επεξεργασία της. Κάθε παραγγελία παίρνει το δικό της id,με όλα τα στοιχεία τυχαία .(αριθμό πιτσών, τύπο και χρόνο έναρξης). Με όρισμα την παραγγελία, δημιουργούμε το νήμα με την pthread_create και καλούμε την process_order.

Κάθε φορά υπάρχει μια μικρή καθυστέρηση στη δημιουργία μιας νέας παραγγελίας (sleep()).

Το πρόγραμμά μας περιμένει να ολοκληρώση την εργασία του κάθε νήμα, κάνοντας χρήση την pthread_join και τελικά με την free() απελευθερώνει τη μνήμη που καταλαμβάνεται από τον πίνακα τύπων πίτσας κάθε παραγγελίας.

(γραμμές κώδικα : 183-204)

Εκτυπώνει τα ζητούμενα αποτελέσματα και ολοκληρώνεται επιτυχώς με return 0.

(γραμμές κώδικα : 206-223)

Έξω από την main(), έχουμε ορίσει την συνάρτηση process_order που επεξεργάζεται μια παραγγελία. Έχει υλοποιηθεί με τέτοιο τρόπο ώστε να προστατεύεται από κλειδώματα για να υπάρχει συγχρονισμός και να αποφύγουμε τον ανταγωνισμό των νημάτων για κοινούς πόρους. (available_cook, available_oven , available_tel).

Η process_order παίρνει την παραγγελία ως όρισμα και αποθηκεύει όλα τα στοιχεία που την αποτελούν. Σε όλη την έκτασή του γίνεται έλεγχος αν είναι διαθέσιμοι οι παραπάνω πόροι. Στην περίπτωση που δεν είναι, χρησιμοποιούμε την pthread_cond_wait(). Όταν ολοκληρωθεί η παράδοση της παραγγελίας , ενημερώνονται τα στατιστικά που καλούμαστε να τυπώσουμε με το πέρας του προγράμματος.

(γραμμές κώδικα : 27-162)

Έχουμε ορίσει επίσης τη συνάρτηση get_random() που δέχεται δύο ορίσματα min και max, όπου στο τέλος της πράξης προσθέτει ξανά το min για να βρίσκεται σίγουρα μέσα στο εύρος τιμών που θέλουμε.

(γραμμές κώδικα : 165-167)

Αξίζει να σημειωθεί ότι επιλέξαμε στατική αρχικοποίηση των νημάτων (γραμμές κώδικα : 8-25). /* Αρχικά είχαμε βρει μια υλοποίηση στο stackoverflow που έκανε δυναμική αρχικοποίηση, αλλά εν τέλει μας φάνηκε πιο “βολική” η στατική υλοποίηση, καθώς δηλώνεται στην αρχή του προγράμματος,απλοποιεί αρκετά το πρόγραμμα και μας διασφαλίζει ότι οι πόροι θα είναι διαθέσιμοι εξαρχής. Αυτό βέβαια αντικρούει το γεγονός ότι στη δυναμική παραχώρηση δεν γίνεται “σπατάλη μνήμης”, όμως επιλέξαμε την πρώτη για χάρην ευκολίας στην υλοποίηση και τη διόρθωση του προγράμματος.*/

Εδώ, δημιουργούμε mutexes(order_mutex, driver_mutex κλπ) και condition variables(order_cond,chef_cond κλπ) για να ελέγξουμε και να συγχρονίσουμε τις εργασίες του προγράμματος.

Ας αναλύσουμε τον τρόπο που διαχειριζόμαστε τους διαθέσιμους σεφ.

Κλειδώνουμε το chef_mutex με την pthread_mutex_lock(&chef_mutex) για να διασφαλίσουμε ότι μόνο το τρέχον νήμα θα μπορεί να επηρεάσει την available_cook κατά την εκτέλεση. Έπειτα γίνεται έλεγχος αν υπάρχει διαθέσιμος σεφ (available_cook == 0). Αν δεν υπαρχει,εκτελείται η pthread_cond_wait(&chef_cond,&chef_mutex), δηλαδή το νήμα απελευθερώνει το mutex μέχρι να υπάρχει διαθέσιμος σεφ. Αν υπάρχει διαθέσιμος σεφ τότε μειώνουμε τον αριθμό διαθεσιμότητας τους (available_cook--).

Στη συνέχεια ξεκλειδώνουμε το chef_mutex με την pthread_mutex_unlock(&chef_mutex).

Για την προετοιμασία της παραγγελίας το νήμα περιμένει Tprep*pizzas χρόνο.

Όπου Tprep ο χρόνος προετοιμασίας ανα πίτσα και pizzas ο αριθμός των πιτσών της παραγγελίας.

Όταν ολοκληρωθεί η προετοιμασία της παραγγελίας, ξανακλειδώνουμε το mutex για να ενημερώσουμε την available_cook. (available_cook++). Τέλος με την pthread_cond_signal(&chef_cond) ενημερώνουν άλλα νήματα που περιμένουν την κατάσταση του σεφ.

ΣΗΜΕΙΩΣΗ: Είτε αναφερόμαστε στους διαθέσιμους σεφ, τηλεφωνητές ή στους διανομείς η διαδικασία υλοποίησης γίνεται με τον ίδιο τρόπο με μικρές παραλλαγές, αναλόγως τον πόρο που θέλουμε να περιγράψουμε.