Εργασία στα Λειτουργικά Συστήματα

Η εργασία αυτή αφορά την προσομοίωση μιας υποθετικής πιτσαρίας στην οποία υλοποιούμε τον παραλληλισμό κάθε παραγγελίας που πραγματοποιούμε με την χρήση mutex της βιβλιοθήκης pthread.

Το πρόγραμμά ξεκινάει με την εκτέλεση της main, η main είναι υπεύθυνη για τον έλεγχο εγκυρότητας των παρακάτω μεταβλητών:

- numberOfCustomers: Το πλήθος των πελάτων(παραγγελιών) θα εξυπηρετηθούν κατα την διάρκεια της προσομοίωσης
- seed: Η τιμή αυτή χρησιμοποιείται για να προσφέρει παραπάνω τυχαιότητα για την κατασκευή των τυχαίων φυσικών αριθμών.

Οι μεταβλητές αυτές πρέπει να είναι θετικές αλλιώς το πρόγραμμα επιστρέφει το κατάλληλο ενημερωτικό μήνυμα και τερματίζει.

Πρώτού ξεκινήσουμε οποιοδήποτε από τα threads(πελάτες) το πρόγραμμα αρχικοποίησή τις εξής λειτουργίες:

1. Τις καθολικές μεταβλητές τις πιτσαρίας.
2. Τη γεννήτρια τυχαίων φυσικών αριθμών.

Ή πιτσαρία

Για καλύτερη οργάνωση και διευκόλυνση, τα στοιχεία της προσομοίωσης της πιτσαρίας βρίσκονται όλα στο αντικείμενο **struct Pizzaria**,εκεί αποθηκεύονται οι μεταβλητές **,**δηλαδή το διαθέσιμο προσωπικό καθώς και στατιστικά δεδομένα τα οποία αξιοποιούμε. Εκεί αποθηκεύουμε τα mutex και τα conditions που θα χρησιμοποιήσουμε.

Συναρτήσεις που χρησιμοποιούνται:

- Void setup_customer_service(): Αρχικοποίησή των τιμών των μεταβλητών ,των mutex και των conditions.
- close_Pizzaria(): Καταστρέφει τα mutex και τα conditions

Για την αξιοποίηση των μεταβλητών εφαρμόζουμε μεθοδολογία που θα αναφερθούμε παρακάτω.

Γεννήτρια φυσικών αριθμών

Η γεννήτρια φυσικών αριθμών χρησιμοποιείται τακτικά για την παραγωγή τυχαίων αποτελεσμάτων και τυχαίων χρόνων αναμονής .Οι αριθμοί αυτοί αξιοποιούνται ως εξής:

- void initialize_random_numbers(unsigned int seed,int numbers_to_make):Η συνάρτηση ξεκινάει στην main, με την χρήση του seed αρχίσουμε numbers_to_make τυχαίους αριθμούς τους οποίους τοποθετεί σε έναν πίνακα random_numbers.Εγγυόμαστε ότι για n threads, το πλήθος των αριθμών είναι επαρκές για να αποφύγουμε κάποιο memory leak .
- int get_random_number_between(int start,int end):Από ένα στοιχείο του πίνακα, παίρνει το στοιχείο που ανήκει στο σύνολο [0, seed] και το περιφράζουμε στο σύνολο [start,end].Τέλος επιστρέφουμε τον αριθμό.
  Σημείωση: Η πρόσβαση στην γεννήτρια τυχαίων αριθμών απαιτεί την χρήση mutex lock,αλλιώς διατρέχουμε των κίνδυνο να χρησιμοποιήσουμε την ίδια τιμή πολλές φορές.
- void destroy_random_numbers(): Απελευθερώνει την μνήμη και διαγράφει τον πίνακα με τους τυχαίους αριθμούς.

ΥΛΟΠΟΙΗΣΗ ΤΩΝ MUTEX ΚΑΙ ΤΩΝ CONDITION

Η ανάλυση της μεθοδολογίας που ακολουθούμε είναι απαραίτητη για την κατανόηση των thread.Στην υλοποίηση μας πολλά νήματα ζητούν παράλληλα τους ίδιους πόρους. Κάθε παραγγελία θέλει πρόσβαση στον τηλεφωνητή, τον μάγειρα ,τον φούρνο κτλ. Η αποδέσμευση είναι απλή διαδικασία, δεδομένου ότι ελέγχουμε την πρόσβαση στους διαθέσιμους πόρους, η απελευθέρωση π.χ. ενός τηλεφώνου δεν απαιτεί κάποιον έλεγχο(πέραν από mutex lock).Η δέσμευση όμως ενός πόρου εφόσον δεν είναι διαθέσιμος απαιτεί την αναμονή αυτού του πόρου.

Αποδέσμευση πόρων / αλλαγή χωρίς περιορισμό

Όπως αναφέραμε η αποδέσμευση πόρων είναι απλή , έστω ένα mutex **resource_lock** και μια μεταβλητή **int balance**. Για την τροποποίηση του τρέχοντος λογαριασμό χρησιμοποιούμε το mutex για να περιορίσουμε την πρόσβαση πάντα σε ένα thread την φορά.

```
Pthread_mutex_lock(&resource_lock);

Balance = balance + money;

Pthread_mutex_lock(&resource_lock);
```

Αυτή η υλοποίηση χρησιμοποιείται για την αποδέσμευση πόρων(τηλέφωνα για παραγγελίες, μάγειρες, διανομής κτ.λ.).Χρησιμοποιείται επίσης και για την ενημέρωση στατιστικών δεδομένων και τη μοναδική χρήση κάθε τυχαίου αριθμού στη γεννήτρια κώδικα.

Δέσμευση Πόρων / Αλλαγή υπό περιορισμό

Η αύξηση είναι απλή αλλά η αφαίρεση απαιτεί έλεγχο ειδικά αν πρέπει να γίνει. Στην περίπτωση της πιτσαρίας πρέπει να ενημερώνεται κάποιο thread το όποιο αναμμένη ένα τηλέφωνο για να κάνει μία παραγγελία. Δεν μπορούμε απλά να του πούμε να περιμένει επ αορίστων, για να λύσουμε αυτό το πρόβλημα χρησιμοποιούμε τα conditions.Έστω ότι μιλάμε για την δέσμευση ενός τηλέφωνου για την λήψη μιας παραγγελίας.

```
//reserve a call
  pthread_mutex_lock(&Pizzaria.telephone_mutex);
  while (1) {
    if (Pizzaria.available_phones) {
      Pizzaria.available_phones--;
      break;
    }else {
     pthread_cond_wait(&Pizzaria.answer_telephone, &Pizzaria.telephone_mutex );
    }
  }
  //Phone secured, close mutex
  pthread_mutex_unlock(&Pizzaria.telephone_mutex);
//Take Order
```

Στο παραπάνω παράδειγμα υλοποιούμε το signal . Αν δεν υπάρχει κάποιος διαθέσιμος τηλεφωνητής, περιμένουμε το σήμα **Pizzaria.answer_telephone**. Η **pthread_cond_wait** θα αποδέσμευση το mutex lock και θα περιμένει, έως να αποδεσμευτεί ένας πόρος. Στην συγκεκριμένη περίπτωση ,όταν κάποιος άλλος πελάτης ολοκλήρωση την παραγγελία του.

```
//release phone
pthread_mutex_lock(&Pizzaria.telephone_mutex);
Pizzaria.available_phones++;
pthread_mutex_unlock(&Pizzaria.telephone_mutex);
pthread_cond_signal(&Pizzaria.answer_telephone);
```

Η pthread_cond_signal ενημερώνει 1 ακριβώς thread ότι υπάρχει διαθέσιμος πόρος που μπόρει να δέσμευση, το thread παίρνει το mutex και δεσμευή το τηλέφωνο.

Ειδική Περίπτωση για τους φούρνους

Η κάθε παραγγελία διαφέρει σε πλήθος. Στην προσομοίωση μας απαιτούμε τις πίτσες μίας παραγγελίας να ψήνονται όλες μαζί. Αυτή η διαφορά απαιτεί κάθε φορά που αποδεσμεύουμε x φούρνους , να ελέγχουμε κάθε παραγγελία που περιμένει και

ενδεχόμενος μπορεί να χρησιμοποίηση τους ελευθέρους φούρνους σε σχέση με μια άλλη.Σε αυτή την περίπτωση καλούμαι όλα τα threads που περιμένουν τους φούρνους με την κλήση της **pthread_cond_broadcast**.

Προσομοίωση threads

Κάθε thread του προγράμματος αναπαριστά μία παραγγελία. Η κάθε παραγγελία αποτελείται από την συνάρτηση void *customer_service(void *id). Η συνάρτηση αυτή προσομοιώνει την διαδικασία μιας παραγγελίας από την έναρξη μέχρι την ολοκλήρωση της.

Δέσμευση κοινόχρηστων πόρων

Η παραγγελία δεσμεύει το προσωπικό της πιτσαρίας, την πρόσβαση στην γεννήτρια αριθμών ,το τερματικό και την τροποποίηση στατιστικών κάθε παραγγελίας.

Πόρος	**Mutexes**	**Conditions**
Τηλεφωνητές	Telephone_mutex	Answer_telephone
Μάγειρες	Prep_cook_mutex	Look_for_cook
Φούρνοι	Oven_mutex	Check_for_ovens
Συσκευαστές	Packing_mutex	Available_packer
Διανομείς	Delivery_mutex	Make_delivery
Πρόσβαση στο τερματικο	Write_on_terminal	-
Συνολικά έσοδα	revenue_mutex	-
Στατιστικά παραγγελίας	Pizza_delivery_statistics_mutex	-

Υπολογισμός χρόνου

Για τον υπολογισμού χρόνου χρησιμοποιούμε την struct timespec previous,current για να προσδιορίσουμε την διάρκεια της παραγγελίας και να εμφανίζουμε και τα ζητούμενα μηνύματα. Χρησιμοποιούμε και την pizza_cooked_timestamp για τον λόγο που θα δούμε παρακάτω.

Συλλογή στατιστικών δεδομένων

Το revenue_mutex αφορά την πρόσβαση(ενημέρωση) στατιστικών ως προς τα έσοδα και τις πώλησης κάθε πίτσας.

Το Pizza_delivery_statistics_mutex έχει να κάνει με την πραγματοποίηση της παραγγελίας(χρόνος παράδοσης, χρόνος που η πίτσα μένει έξω από τον φούρνο ώσπου να φτάσει κτλ.),εδώ αξιοποιούμε και το pizza_cooked_timestamp.

Εμφάνιση δεδομένων

Το Write_on_terminal mutex χρησιμοποιείται στον παραλληλισμό για να εγγυηθούμε κανένα μήνυμα δεν θα γραφτεί όσο γράφεται ένα άλλο.