Λειτουργικά Συστήματα 2022-2023

Αναφορά Παράδοσης

Μητσάκης Νίκος : p3210122

Παντελίδης Ιπποκράτης : p3210150

Θέμα :

Το θέμα της εργασίας είναι η υλοποίηση ενός συστήματος παραγγελιών και διανομής πίτσας χρησιμοποιώντας νήματα POSIX(threads). Στο σύστημα οι παραγγελίες γίνονται ηλεκτρονικά και πληρώνονται, και στην συνέχεια κάθε μία παρασκευάζεται, ψήνεται, πακετάρεται και τελικά διανέμεται στον πελάτη. Λόγω του μεγάλου αριθμού των παραγγελιών και του περιορισμένου αριθμού πόρων είναι απαραίτητη η χρήση αμοιβαίου αποκλεισμού με mutexes και συγχρονισμό με condition variables.

Δηλώσεις :

Αρχικά έχουμε το header file “p3210150-p3210122.h” στο οποίο γίνονται οι δηλώσεις των σταθερών που μας δίνονται στην εκφώνηση, καθώς και οι δηλώσεις των συναρτήσεων που χρησιμοποιούμε στο κυρίως πρόγραμμα. Επίσης, έχουμε ορίσει ένα struct με όνομα Order το οποίο αναπαριστά μια παραγγελία πίτσας και έχει διάφορες σημαντικές πληροφορίες για αυτή(id, χρόνοι λειτουργιών και άλλα). Το c source file “p3210150-p3210122.c” κάνει include το παραπάνω header file και υλοποιεί όλες τις συναρτήσεις.

Στο πρόγραμμα χρησιμοποιούνται mutexes και condition variables των οποίων τα ονόματα και τα σχόλια δηλώνουν ακριβώς την χρήση τους. Ειδικότερα αυτά που χρησιμοποιούνται κυρίως στον συγχρονισμό των πόρων έτσι όπως περιγράφεται στο αντικείμενο της εργασίας είναι τα ακόλουθα :

1)mutexes

```
//resources mutexes
pthread_mutex_t cooks_count_mutex;          //locks cooks
pthread_mutex_t ovens_count_mutex;          //locks ovens
pthread_mutex_t packers_count_mutex;        //locks packers
pthread_mutex_t deliverers_count_mutex;     //locks deliverers
```

2)condition variables

```
//resources condition variables
pthread_cond_t cooks_cond;              //condition variable for cooks
pthread_cond_t ovens_cond;              //condition variable for ovens
pthread_cond_t packers_cond;            //condition variable for packers
pthread_cond_t deliverers_cond;         //condition variable for deliverers
```

Επίσης έχουμε τις ακόλουθες καθολικές μεταβλητές και μετρητές:

1. ΣΤΑΤΙΣΤΙΚΑ

- Συνολικά έσοδα (profit) και πλήθος απλών και σπέσιαλ πιτσών (plain & special)
- Πλήθος επιτυχημένων και αποτυχημένων παραγγελιών (succeeded & failed)
- Πλήθος ευχαριστημένων και δυσαρεστημένων πελατών (happy & disappointed) (EXTRA)

2. ΧΡΟΝΟΙ

- Συνολικός χρόνος ολοκλήρωσης όλων των παραγγελιών (total_waiting_time)
- Συνολικός χρόνος κρυώματος όλων των παραγγελιών (total_cooling_time)
- Μέγιστος χρόνος ολοκλήρωσης (max_waiting_time)
- Ελάχιστος χρόνος ολοκλήρωσης (min_waiting_time) (EXTRA)
- Μέγιστος χρόνος κρυώματος (max_cooling_time)
- Ελάχιστος χρόνος κρυώματος (min_cooling_time) (EXTRA)

Για κάθε bullet των κατηγοριών 1 και 2 έχουμε και το αντίστοιχο mutex εξασφαλίζοντας έτσι ότι μόνο ένα νήμα την φορά θα τροποποιεί τις μεταβλητές μας χωρίς να δημιουργούνται συνθήκες ανταγωνισμού. Τέλος, έχουμε τις μεταβλητές N_cust όπου είναι το πλήθος των πελατών και τον σπόρο seed, οι τιμές των οποίων μεταβιβάζονται ως ορίσματα, καθώς και ένα mutex για να εμφανίζουμε μηνύματα στην οθόνη.

Helper, Error Handling and other functions :

`void mutex_lock(pthread_mutex_t *lock);` : συνάρτηση που εσωτερικά καλεί την pthread_mutex_lock με ενσωματωμένο τον έλεγχο αποτυχίας κλειδώματος του mutex που παίρνει σαν όρισμα.

`void mutex_unlock(pthread_mutex_t *lock);` : συνάρτηση που αντίστοιχα εσωτερικά καλεί την pthread_mutex_unlock με ενσωματωμένο τον έλεγχο αποτυχίας ξεκλειδώματος.

`void mutex_init();` : συνάρτηση που καλώντας εσωτερικά την pthread_mutex_init αρχικοποιεί με τους απαραίτητους ελέγχους όλα τα mutexes που έχουμε δηλώσει.

`void mutex_destroy();` : συνάρτηση που καλώντας εσωτερικά την pthread_mutex_destroy καταστρέφει με τους απαραίτητους ελέγχους όλα τα mutexes που έχουμε δηλώσει.

`void cond_init();` : συνάρτηση που καλώντας εσωτερικά την pthread_cond_init αρχικοποιεί με τους απαραίτητους ελέγχους όλα τα condition variables που έχουμε δηλώσει.

`void cond_wait(pthread_cond_t *cond, pthread_mutex_t *lock);` : συνάρτηση που καλώντας εσωτερικά την pthread_cond_wait και έχοντας τους απαραίτητους ελέγχους σφάλματος, περιμένει κάποιο σήμα να φτάσει.

`void cond_destroy();` : συνάρτηση που καλώντας εσωτερικά την pthread_cond_destroy καταστρέφει με τους απαραίτητους ελέγχους όλα τα condition variables που έχουμε δηλώσει.

`void clock_get_time_check(struct timespec *time);` : συνάρτηση που καλεί εσωτερικά την clock_gettime με πρώτο όρισμα το CLOCK_REALTIME για να περάσει το τρέχοντα χρόνο της κλήσης στο δεύτερο όρισμα, κάνοντας τον απαραίτητο έλεγχο σφάλματος.

`double get_time_difference(struct timespec *start_time, struct timespec *stop_time);`
: συνάρτηση που υπολογίζει και επιστρέφει την διαφορά μεταξύ δύο χρονικών στιγμών (timespec) λαμβάνοντας υπόψη τα πεδία των δευτερολέπτων (tv_sec) αλλά και αυτά των νανοδευτερολέπτων (tv_nsec).

`uint rand_generator(uint *seed, uint min, uint max);` : συνάρτηση που περνάει το seed σαν όρισμα στην συνάρτηση rand_r, παράγοντας έτσι ένα τυχαίο αριθμό και έπειτα χρησιμοποιώντας τον τελεστή % τον μετατρέπει ώστε να δείχνει στο διάστημα [min, max]

`void update_max_time(double new_time, pthread_mutex_t *max_mutex, double *cur_max);`
: συνάρτηση που παίρνει σαν όρισμα τον τρέχοντα χρόνο και τον μέγιστο χρόνο μέχρι τώρα και τον ενημερώνει στην περίπτωση που ο τρέχοντας χρόνος > μέγιστο χρόνο. Αυτό συμβαίνει σε μία περιοχή κλειδώματος για τον λόγο που αναφέραμε παραπάνω.

`void update_min_time(double new_time, pthread_mutex_t *min_mutex, double *cur_min);`

: συνάρτηση που παίρνει σαν όρισμα τον τρέχοντα χρόνο και τον ελάχιστο μέχρι τώρα χρόνο και τον ενημερώνει στην περίπτωση που ο τρέχοντας χρόνος < ελάχιστο χρόνο. Όμοια μέσα σε κλείδωμα.

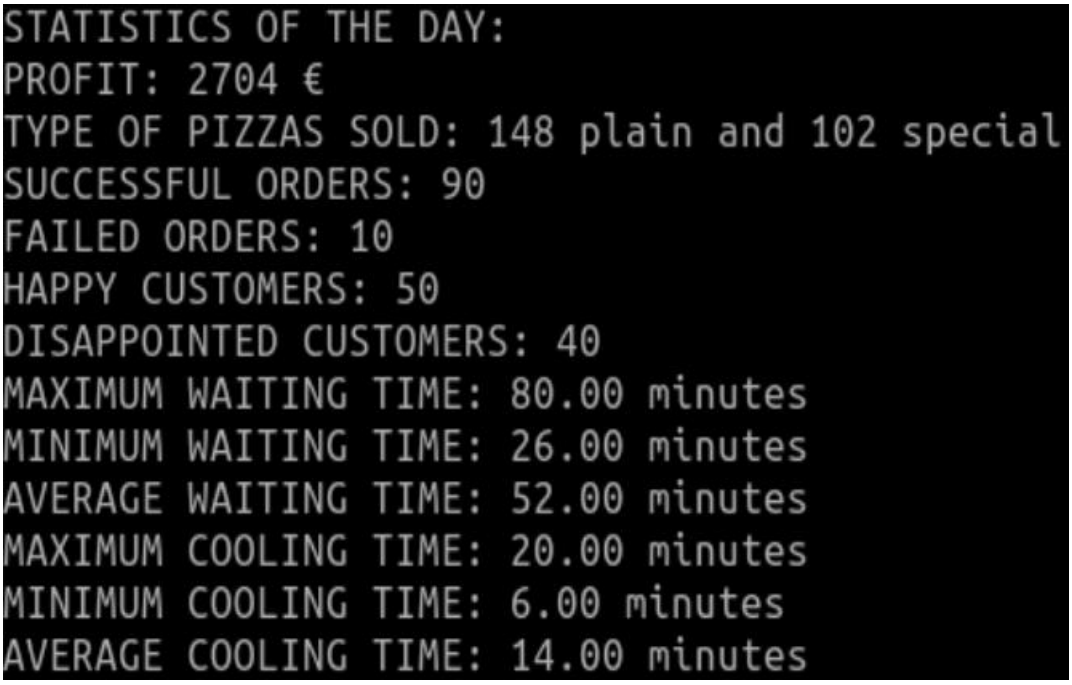
`void wait_for_resource(uint *resource, uint value, pthread_mutex_t *resource_mutex, pthread_cond_t *resource_cond);`
: συνάρτηση που έχει σχεδιαστεί για να κάνει τον πόρο που της περνάμε σαν όρισμα να περιμένει μέχρις ότου να αποκτήσει μια συγκεκριμένη ποσότητα από αυτόν και να συνεχίσει. Αρχικά κλειδώνει τον πόρο για να εξασφαλίσει μοναδική πρόσβαση και περιμένει με την cond_wait όσο η απαιτούμενη ποσότητα πόρου δεν υπάρχει. Όταν το νήμα λάβει σήμα από την resource_signal, θα γίνει αναφορά παρακάτω, σημαίνει ότι η ποσότητα έγινε διαθέσιμη και επομένως αφαιρούμε όσες μονάδες του πόρου χρειαζόμαστε, απελευθερώνουμε το κλείδωμα και συνεχίζουμε την εκτέλεση.

`void signal_resource(uint *resource, uint value, pthread_mutex_t *resource_mutex, pthread_cond_t *resource_cond);`
: συνάρτηση που χρησιμοποιείται από ένα νήμα που έχει τελειώσει την χρήση του κοινού πόρου και ειδοποιεί άλλα νήματα ότι ο πόρος έχει γίνει διαθέσιμος. Αυξάνει τον πόρο κατά τις μονάδες που είχε δεσμεύσει(value) και στέλνει μήνυμα στο condition variable του πόρου, ξυπνώντας έτσι ένα από τα νήματα που περιμένουν. Έτσι όταν το νήμα που μόλις ξυπνάει αποκτήσει το κλείδωμα του η εκτέλεση συνεχίζεται κανονικά.

- Οι δύο τελευταίες μέθοδοι είναι πάρα πολύ σημαντικές για τον συγχρονισμό και την επικοινωνία νημάτων σε ένα πολυνηματικό πρόγραμμα.

Main :

Στην αρχή της εκτέλεσης της main ελέγχουμε τα ορίσματα μας με τον περιορισμό ότι πρέπει να λάβουμε από τον χρήστη ακριβώς δύο ορίσματα και μάλιστα το πρώτο που είναι το πλήθος των καταναλωτών να είναι θετικός αριθμός. Έπειτα δεσμεύουμε δυναμικά μνήμη για τα νήματα των παραγγελιών και κατασκευάζουμε δυναμικά ένα πίνακα παραγγελιών στον οποίο αποθηκεύουμε διάφορα στοιχεία τους. Καλούμε τις συναρτήσεις mutex_init και cond_init που αναφέραμε παραπάνω και είμαστε έτοιμοι να προχωρήσουμε στην δημιουργία των νημάτων. Φτιάχνουμε ένα for loop μέχρι N_cust, όσα και οι παραγγελίες μας, και σε κάθε επανάληψη κατασκευάζουμε ένα νήμα με την χρήση της pthread_create που παίρνει με την σειρά τα ακόλουθα ορίσματα : 1) διεύθυνση της $i^{-οστής}$ θέσης της δυναμικά δεσμευμένης μνήμης μας, 2) NULL, 3) την διεύθυνση της ρουτίνας που θα καλεί το κάθε νήμα, δηλαδή την συνάρτηση order για στην οποία θα αναφερθούμε παρακάτω, 4)void * στο struct των παραγγελιών μας ώστε να το περάσουμε σαν όρισμα στην order. Το κάθε νήμα δημιουργείται (= παραγγελία φτάνει) μετά από έναν τυχαίο αριθμό στο διάστημα [1,3] και αυτό επιτυγχάνεται με την χρήση της μεθόδου rand_generator που είδαμε παραπάνω. Μόλις τα νήματα δημιουργηθούν και ολοκληρώσουν το έργο τους έχουμε πάλι ένα for loop πάλι μέχρι N_cust ώστε ουσιαστικά να τα καταστρέψουμε χρησιμοποιώντας την pthread_join. Τέλος κάνουμε μερικούς τελευταίους υπολογισμούς και τυπώνουμε τα στατιστικά μας. Σε κάθε περίπτωση, δηλαδή αν το πρόγραμμα ολοκληρωθεί κανονικά ή κάποια από τις pthread_create ή pthread_join αποτύχει τερματίζουμε αφού πρώτα καταστρέψουμε τα mutexes και τα condition variables(mutex_destroy & cond_destroy) και κυρίως αφού αποδεσμεύσουμε την δυναμική μας μνήμη μέσω της free.



Order function :

Η order είναι μία συνάρτηση ρουτίνας που παίρνει σαν όρισμα το struct που υλοποιήσαμε, επιτελεί διάφορες λειτουργίες όπως η παρασκευή της, η πληρωμή της, το ψήσιμο της, το πακετάρισμα της και η διανομή της και υπολογίζει ορισμένους χρόνους.

```
struct timespec start_timespec;         //arrived
struct timespec payment_timespec;       //approved or cancelled
struct timespec cooked_timespec;        //cooked
struct timespec baked_timespec;         //baked
struct timespec packed_timespec;        //packed
struct timespec delivered_timespec;     //delivered
```

Αρχικά σημειώνει το χρόνο άφιξης της παραγγελίας και παράγει με την rand_generator έναν τυχαίο αριθμό στο διάστημα [1,3] ο οποίος αντιπροσωπεύει τον χρόνο μέσα στον οποίο γίνεται η δοκιμή πληρωμής της παραγγελίας και αυτή είτε γίνεται αποδεκτή είτε όχι. Σημείωση : σε όλες τις κλήσεις αυτής της μεθόδου της περνιέται σαν όρισμα ένας τοπικός σπόρος ο οποίος προκύπτει για κάθε παραγγελία από τον σπόρο όρισμα πολλαπλασιασμένο με το id της παραγγελίας. Με τον ίδιο τρόπο προκύπτει και ο τοπικός σπόρος στην main. Υπολογίζουμε μέσω της πιθανότητας αποτυχίας που μας δίνεται τι από τα δύο συμβαίνει και σε κάθε περίπτωση σημειώνουμε το χρόνο που ολοκληρώνεται ο έλεγχος και χρησιμοποιώντας τα κατάλληλα κλειδώματα αυξάνουμε τον αντίστοιχο μετρητή και τυπώνουμε το αντίστοιχο μήνυμα. Σε περίπτωση αποτυχίας το νήμα τερματίζεται με την pthread_exit. Εφόσον γίνει δεκτή η παραγγελία πάλι με την χρήση της rand_generator παίρνουμε το πλήθος των πιτσών της κάθε παραγγελίας και υπολογίζουμε με την πιθανότητα του να είναι απλή, πόσες πίτσες είναι απλές και πόσες σπέσιαλ, αυξάνοντας ανάλογα τους μετρητές και συναθροίζοντας το συνολικό κέρδος(οι τροποποιήσεις των καθολικών αυτών μεταβλητών γίνονται με κλειδώματα) . Στην συνέχεια έχουμε για κάθε νήμα μια ακολουθία κλήσεων των wait_for_resource και signal_resource όπου κάθε παραγγελία περιμένει σειριακά τον κάθε πόρο (παρασκευαστές, φούρνοι, packers και διανομείς) να γίνει διαθέσιμος. Όταν φτάσει κάποιο σήμα και ο πόρος υπάρχει στην ποσότητα που θέλουμε εκτελείται ο αναφερόμενος χρόνος με την sleep είτε για κάθε πίτσα είτε ταυτόχρονα για όλες ανάλογα με την λειτουργία (π.χ. ο χρόνος ψησίματος τις αφορά όλες). Αξίζει να αναφερθεί ότι η μέθοδος signal δεν αρκεί στην περίπτωση με τους φούρνους, καθώς υπάρχει η πιθανότητα να ξυπνήσουμε μια παραγγελία οι πίτσες της οποίας είναι περισσότερες από τους διαθέσιμους φούρνους εκείνη την στιγμή με αποτέλεσμα το νήμα να μπλοκάρεται ξανά. Για να το αντιμετωπίσουμε αυτό χρησιμοποιούμε την μέθοδο pthread_cond_broadcast που ξυπνάει όλα τα νήματα που περιμένουν ώστε να μειώσουμε τις πιθανότητες χασίματος χρόνου από πιθανά μπλοκαρίσματα. Επίσης για τον χρόνο διανομής κρατά έναν τυχαίο χρόνο ο οποίος με την χρήση της rand_generator τον προσαρμόζεται ώστε να βρίσκεται στο διάστημα [5,15]. Κάθε φόρα που ξεκινάει και μία καινούργια λειτουργία

σημειώνεται και ο χρόνος της ώστε είτε να μας βοηθήσει στον υπολογισμό του χρόνου αναμονής και χρόνου κρυώματος είτε να γεμίσουμε το struct μας με πληροφορία. Στο σημείο που και το πακετάρισμα της παραγγελίας έχει ολοκληρωθεί υπολογίζουμε τον μέχρι τότε χρόνο και κλειδώνοντας την οθόνη για να γράψουμε εμφανίζουμε το ανάλογο μήνυμα. Το ίδιο συμβαίνει και όταν η παραγγελία παραδοθεί, όπου επιστρέφεται στην οθόνη ο χρόνος που χρειάστηκε από την ώρα που έφτασε στο κατάστημα μέχρι που παραδόθηκε στον πελάτη. Τέλος χρησιμοποιώντας τους σημειωμένους χρόνους από τα διάφορα στάδια υπολογίζουμε τον χρόνο κρυώματος και αναμονής της κάθε παραγγελίας και εν συνεχεία με τα κατάλληλα mutexes τον συνολικό χρόνο και των δύο καθώς και με τις προαναφερθείσες συναρτήσεις το max και το min και των δύο και το κάθε νήμα τερματίζει με την pthread_exit.

EXTRA:

Εκτός από τις έξτρα υλοποιήσεις που δείξαμε παραπάνω(υπολογισμός ελάχιστου χρόνου κρυώματος και αναμονής, πλήθος χαρούμενων και δυσαρεστημένων πελατών ανάλογα με τον χρόνο κρυώματος) έχουμε επίσης υλοποιήσει την ακόλουθη μέθοδο:

`void print_order(Order order);` : Η μέθοδος αυτή καλείται επαναληπτικά για κάθε παραγγελία στην main και επιστρέφει χρήσιμες και ακριβής πληροφορίες(κυρίως χρόνους λειτουργιών) για κάθε παραγγελία τις οποίες κρατήσαμε με την χρήση των timespecs. Παρακάτω φαίνεται τι εκτυπώνει στα αριστερά για παραγγελίες που έχουν γίνει αποδεκτές και στα δεξιά για παραγγελίες που ακυρώθηκαν.

```
========================
ID: 100
Pizza quantity: 2
Is Canceled: 0
Payment Time: 2.000810
Cooking Time: 50.977556
Baking Time: 10.010307
Packing Time: 2.000554
Delivery Time: 14.002829
Preparation Time: 64.989227
Cooling Time 16.003383
Waiting Time 78.992057
========================
```

```
========================
ID: 71
Pizza quantity: 0
Is Canceled: 1
Payment Time: 2.000488
Cooking Time: 0.000000
Baking Time: 0.000000
Packing Time: 0.000000
Delivery Time: 0.000000
Preparation Time: 0.000000
Cooling Time 0.000000
Waiting Time 0.000000
========================
```

Παρατηρούμε ότι ο χρόνος παρασκευής αυξάνεται αρκετά με την πάροδο του χρόνου και αυτό συμβαίνει επειδή οι παραγγελίες φτάνουν με πολύ γρηγορότερο ρυθμό από αυτόν της ολοκλήρωσης της μιας με αποτέλεσμα να δημιουργείται μετά από κάποιο σημείο αυτή η συμφόρηση.