ΙΩΑΝΝΗΣ ΚΑΠΕΤΑΝΓΕΩΡΓΗΣ **Α.Μ. : 1115201800061**

Κ22: Λειτουργικά Συστήματα (Περιττοί) – Χειμερινό '20

Project 3

Γενικές παρατηρήσεις:

- Όλα τα αρχεία κώδικα είναι πλήρως σχολιασμένα και εξηγείται η κάθε λειτουργία που γίνεται στο κάθε σημείο. Στο documentation αυτό, οι λειτουργιές που γίνονται σε κάθε σημείο εξηγούνται περιγραφικά. Οι προγραμματιστικές λεπτομέρειες εξηγούνται στα σχόλια του κώδικα έτσι ώστε αν γίνει πλήρως κατανοητός.
- Όση μνήμη έχει δεσμευθεί δυναμικά κατά την εκτέλεση του προγράμματος, αποδεσμεύεται πριν το τέλος αυτού με ελεγχόμενο τρόπο και χωρίς να υπάρχουν απώλειες μνήμης η errors.
- Έχουν υλοποιηθεί όλα όσα ζητούνται στην εκφώνηση αλλά και αναφέρθηκαν στο piazza
- Έχουν ελεγχθεί όλες οι πιθανές περιπτώσεις έτσι ώστε να μην συμβαίνει ποτέ deadlock και πάντα οι διεργασίες να ολοκληρώνονται κανονικά.

Αρχεία που περιέχονται στο directory

- chef.c : αφορά την διεργασία του chef, η οποία εκτελεί τις λειτουργίες που αναφέρονται στην εκφώνηση και εξηγούνται παρακάτω.
- saladmaker.c : στο αρχείο αυτό περιέχεται η λειτουργία των saladmakers. Μέσω αυτού δημιουργείται το εκτελέσιμο το οποίο εκτελούμε 3 φορές με τα αντίστοιχα ορίσματα έτσι ώστε να δημιουργήσουμε τους 3 saladmakers.
- header.h: στο αρχείο αυτό περιέχονται όλα τα include τα οποία χρειάζεται να κάνουν τα chef.c και saladmaker.c, κάποια define σταθερών που χρησιμοποιούν τα δύο προγράμματα καθώς και την δήλωση της δομής (ένα struct) που αποθηκεύεται στο shared memory.
- findConcurrentTimes.c: στο αρχείο αυτό περιέχονται όλες οι συναρτήσεις οι οποίες χρειάζονται για τον υπολογισμό των διαστημάτων στα οποία παραπάνω από μια διεργασίες δρουν ταυτόχρονα. Για να βρεθούν τα διαστήματα αυτά καλείται η συνάρτηση printConcurrentTimes η οποία με την σειρά της καλεί τις υπόλοιπες συναρτήσεις του αρχείου. Η δήλωση της printConcurrentTimes υπάρχει στο αρχείο findConcurrentTimes.h έτσι το οποίο γίνεται include από το chef.c έτσι ώστε να μπορεί να την καλέσει.
- Makefile: εξηγείται αμέσως παρακάτω.

Makefile

- Εφαρμόζεται separate compilation. Αρχικά δημιουργούνται όλα τα .o αρχεία από τα αντίστοιχα .c αρχεία και έπειτα συνδέονται κατάλληλα έτσι ώστε να παραχθούν τα αντίστοιχα εκτελέσιμα.
- Παράγονται 2 εκτελέσιμα προγράμματα. Το chef το οποίο είναι το εκτελέσιμο για τον chef. Επίσης δημιουργείται το εκτελέσιμο saladmaker το οποίο αφορά την διεργασία των saladmakers.
- Έχει προστεθεί και η λειτουργία make clean έτσι ώστε να διαγράφονται όλα τα object files και τα εκτελέσιμα αρχεία.

Εκτέλεση προγραμμάτων

- Για την εκτέλεση του προγράμματος αρχικά “τρέχουμε” το εκτελέσιμο του chef δηλαδή το ./chef δίνοντας του τα κατάλληλα ορίσματα. Κατά την εκτέλεση του πρέπει να δοθούν οι παράμετροι που ζητούνται από την εκφώνηση, δηλαδή:

./chef -n numOfSlds -m mantime

όπου numOfSlds είναι ο αριθμός των σαλατών που πρέπει να κατασκευαστούν και mantime είναι η χρονική διάρκεια που έχει στην διάθεση του ο chef για να κάνει εάν διάλειμμα μεταξύ εναποθέσεων υλικών. Βέβαια γίνεται ο κατάλληλος έλεγχος σε περίπτωση που δοθούν λάθος οι παράμετροι. Στην περίπτωση αυτή τυπώνεται το αντίστοιχο μήνυμα και τερματίζεται η εκτέλεση.
Τέλος μόλις εκτελέσουμε τον chef, εκτυπώνεται το id του shared memory το οποίο πρέπει να δοθεί σαν παράμετρος στα εκτελέσιμα των saladmakers όπως εξηγείται παρακάτω.

- Ένα εκτελέσιμο ./saladmaker χρειάζεται τις εξής παραμέτρους:

./saladmaker -t1 lb -t2 ub -s shmid -id saladmakerId

Οι παράμετροι lb, ub, shmid είναι αυτές που ορίζονται από την εκφώνηση. Για να βρούμε το shmid εκτελούμε πρώτα τον chef ο οποίος εκτυπώνει το κλειδί. Έτσι μέσω της εκτύπωσης αυτής, παίρνουμε το κλειδί και το δίνουμε σαν παράμετρο στους saladmakers. Έχει προστεθεί το επιπλέον όρισμα saladmakerId, το οποίο προσδιορίζει το ποιον από τους 3 saladmakers θέλουμε να εκτελέσουμε (δηλαδή ορίζουμε το ποια υλικά θα έχει διαθέσιμα). Το id αυτό μπορεί να είναι είτε 1 για τον saladmaker1 είτε 2 για τον saladmaker2 είτε 3 για τον saladmaker3. Δηλαδή μετά από την σημαία -id ακολουθεί ένας αριθμός από τους (1,2,3).
Άρα για να εκτελέσουμε τους τρεις saladmakers, χρειάζεται να εκτελέσουμε 3 ξεχωριστές διεργασίες όπου στην κάθε μία θα βάλουμε ένα από τα 3 id. Δηλαδή ένα εκτελέσιμο με -id 1, ένα εκτελέσιμο με -id 2 και ένα με -id 3.

Shared Memory

Το shared memory segment δημιουργείται αρχικά από τον chef (πριν εκτελεστούν οι saladmakers) καθώς και αρχικοποιεί όλα τα δεδομένα τα οποία περιέχονται σε αυτό. Έπειτα ο chef εκτυπώνει το κλειδί του shared memory segment έτσι ώστε να δοθεί σαν παράμετρος στους saladmakers και αυτοί να το κάνουν attach.
Στο shared segment αποθηκεύεται μια δομή τύπου smData (ένα struct) το οποίο ορίζεται στο header file header.h.
Η δομή αυτή περιέχει:

- Έναν ακέραιο με όνομα saladsToBeMade ο οποίος αντιπροσωπεύει τον αριθμό σαλατών που πρέπει να κατασκευαστούν. Οπότε, αρχικοποιείται με το numOfSlds το οποίο δόθηκε σαν παράμετρος στον chef. Κάθε φορά που κατασκευάζεται μια σαλάτα από έναν saladmaker, ο saladmaker αυτός μειώνει το saladsToBeMade κατά ένα έως ότου γίνει μηδέν, δηλαδή κατασκευαστούν όλες οι σαλάτες
- Έναν πίνακα τριών θέσεων (μία για κάθε saladmaker) με όνομα finished. Ο πίνακας αυτός περιέχει την πληροφορία για το αν μια διεργασία saladmaker έχει ολοκληρωθεί. Κάθε θέση αρχικοποιείται με την τιμή FALSE (η οποία γίνεται defined στο αρχείο header.h). Κάθε διεργασία saladmaker, λίγο πριν ολοκληρωθεί αλλάζει την τιμή του πίνακα (στην αντίστοιχη θέση) σε True δηλώνοντας έτσι ότι ολοκληρώθηκε.
- Έναν πίνακα τριών θέσεων (μία για κάθε saladmaker) με όνομα saladsMade. Ο πίνακας αυτός περιέχει το πόσες σαλάτες έχει φτιάξει ο κάθε saladmaker. Έτσι, αρχικοποιείται με 0 και κάθε φορά που ένας saladmaker φτιάχνει μια σαλάτα αυξάνει την τιμή της αντίστοιχης θέσης κατά ένα. Άρα τελικά ο πίνακας αυτός θα περιέχει το πόσες σαλάτες έφτιαξε συνολικά ο κάθε saladmaker.
- Έναν πίνακα τριών θέσεων (μία για κάθε saladmaker) με όνομα pid. Στον πίνακα αυτόν περιέχονται τα process ids των saladmakers. Έτσι όταν εκτελείται ο κάθε saladmaker αναθέτει το pid του στην αντίστοιχη θέση του πίνακα. Η πληροφορία αυτή χρησιμοποιείται έτσι ώστε να γίνουν οι κατάλληλες εκτυπώσεις από τον chef όπως αυτές περιγράφονται από την μορφοποίηση εξόδου στην σελίδα του μαθήματος.

Πριν ολοκληρωθεί η εκτέλεση του κάθε saladmaker ο κάθε ένας κάνει detach to shared memory segment.
Τέλος, ο chef “μαρκάρει” το segment σαν to be destroyed μέσω της shmctl με όρισμα IPC_RMID και τελικά αποδεσμεύεται το shared memory segment.

Semaphores - Συνεργασία μεταξύ διεργασιών

Χρησιμοποιώ named semaphores. Τα ονόματα του κάθε semaphore γίνονται defined στο αρχείο header.h .
Οι semaphores δημιουργούνται από τον chef (πριν ακόμα εκτελεστούν οι saladmakers) ενώ οι saladmakers τους “ανοίγουν” μέσω της sem_open και χρησιμοποιώντας το προκαθορισμένο όνομα το οποίο έχει γίνει defined.
Τέλος, οι semaphores “κλείνουν” μέσω της sem_close πριν το τέλος της εκτέλεσης του chef (και αφού έχουν ήδη ολοκληρωθεί οι διεργασίες των saladmakers).
Οι semaphores που χρησιμοποιώ είναι οι εξής:

- **mutex**: είναι ο κλασικός semaphore ο οποίος χρησιμοποιείται έτσι ώστε να πετύχουμε mutual exclusion. Δηλαδή μέσω αυτού μόνο μια διεργασία την φορά μπορεί να κάνει access μια πληροφορία του shared memory. Έτσι κάθε διεργασία κάθε φορά πριν κάνει access μια πληροφορία του shared memory εκτελεί την εντολή sem_wait(mutex). Δηλαδή, σε περίπτωση που κάποια άλλη διεργασία κάνει access την τρέχουσα στιγμή κάποια πληροφορία της shared memory, τότε γίνεται suspended πάνω στον semaphore έως ότου η άλλη διεργασία τελειώσει με την όποια λειτουργία έκανε πάνω στο shared memory. Όταν μια διεργασία ολοκληρώσει την λειτουργία στα δεδομένα του shared memory (π.χ. αλλαγή της τιμής μιας μεταβλητής του shared memory) εκτελεί την εντολή sem_post(mutex) έτσι ώστε κάποια άλλη διεργασία να μπορέσει να έχει access στο shared memory.
- **joinLogSemaphore**: Όλες οι διεργασίες (chef,saladmaker1,saladmaker2,salamaker3) γράφουν σε ένα κοινό logFile (το οποίο εξηγείται και παρακάτω). Πρέπει να διαφυλάξουμε ότι μόνο μία διεργασία γράφει κάθε φορά στο αρχείο αυτό. Δηλαδή να μην γράψουν πότε 2 διεργασίες ακριβώς την ίδια στιγμή στο αρχείο αυτό. Έτσι χρησιμοποιούμε τον joinLogSemaphore ο οποίος έχει ακριβώς την ίδια λογική με τον mutex semaphore δηλαδή εξασφαλίζει το mutual exclusion όσον αφορά το γράψιμο το ενιαίο logFile. Έτσι κάθε φορά που μια διεργασία θέλει να γράψει στο αρχείο εκτελεί την εντολή sem_wait(joinLogSemaphore) (και γίνεται suspended σε περίπτωση που κάποια άλλη γράφει τη ίδια χρονική στιγμή) και αφού γράψει εκτελεί την εντολή sem_post(joinLogSemaphore).

Όσον αφορά την “συνεργασία” μεταξύ chef και saladmakers ακολουθείται η λογική του γνωστού προβλήματος producer-consumer το οποίο χρησιμοποιεί έναν empty semaphore και έναν full. Η μόνη διαφορά είναι ότι έχω αντικαταστήσει τον έναν full semaphore με 3 ξεχωριστούς (έναν για κάθε saladmaker). Πιο αναλυτικά:

- **tableEmptySemaphore**: Ο semaphore αυτός υποδηλώνει το αν υπάρχουν υλικά στο τραπέζι (δηλαδή αν το τραπέζι είναι γεμάτο) ή αν το τραπέζι είναι άδειο (δηλαδή ο κάποιος saladmaker πήρε τα υλικά από το τραπέζι). Αρχικά ο tableEmptySemaphore αρχικοποιείται με 1 καθώς το τραπέζι είναι άδειο. Ο chef κάθε φορά πριν επιλέξει τα 2 υλικά εκτελεί την εντολή sem_wait(tableEmptySemaphore) δηλαδή ελέγχει αν το τραπέζι είναι άδειο. Σε περίπτωση που το τραπέζι δεν είναι άδειο τότε γίνεται suspended πάνω στο semaphore έως ότου το τραπέζι αδειάσει (δηλαδή ο αντίστοιχος saladmaker πάρει τα υλικά). Κάθε φορά που ένας saladmaker παίρνει τα υλικά από το τραπέζι ειδοποιεί στον chef ότι το τραπέζι άδειασε (δηλαδή εκτελεί την εντολή sem_post(tableEmptySemaphore)) έτσι ώστε ο chef να ειδοποιηθεί και να συνεχίσει με τα επόμενα βήματα του (δηλαδή να κάνει το διάλειμμα του και να επιλέξει τα επόμενα δύο υλικά).

- **tableFullSemaphore:** Υπάρχουν τρεις τέτοιοι semaphores, ένας για κάθε saladmaker. Ο tableFullSemaphore1 για τον saladmaker1, ο tableFullSemaphore2 για τον saladmaker 2 και ο tableFullSemaphore3 για τον saladmaker3. Στον chef είναι ανοιχτοί και οι 3 semaphores ενώ στους saladmakers είναι ανοιχτός μόνο ο αντίστοιχος. Ο tableFullSemaphore δηλώνει το αν υπάρχουν στο τραπέζι τα υλικά που χρειάζεται ο αντίστοιχος saladmaker. Δηλαδή ο κάθε saladmaker εκτελεί την εντολή sem_wait στο αντίστοιχο tableFullSemaphore και γίνεται suspended έως ότου ο chef τον ειδοποιήσει. Αντίθετα, ο chef διαλέγει τυχαία 2 υλικά και ανάλογα με το ποια είναι αυτά τα δύο υλικά πρέπει να ειδοποιήσει τον αντίστοιχο saladmaker. Έτσι εκτελεί την εντολή sem_post στον tableFullSemaphore του αντίστοιχου saladmaker και ειδοποιεί τον saladmaker αυτόν. Μόλις γίνει αυτό γίνεται wake up ο συγκεκριμένος salad maker έτσι ώστε να πάρει τα υλικά και να συνεχίσει με την αντίστοιχη διαδικασία που ακολουθεί ενώ οι υπόλοιποι saladmakers παραμένουν suspended έως ότου έρθει η σειρά τους (δηλαδή έως ότου ο chef επιλέξει τα 2 υλικά που χρειάζονται και τους ειδοποιήσει).

  Παρατήρηση: η επιλογή να χρησιμοποιηθούν 3 ξεχωριστοί τέτοιοι semaphores για κάθε saladmaker και όχι ένας ενιαίος έγινε καθώς με αυτόν τον τρόπο γλιτώνουμε burst cycles. Πιο συγκεκριμένα εάν είχαμε έναν τέτοιο semaphore τότε ό chef θα ειδοποιούσε όλους τους saladmakers οι οποίοι θα “ξυπνάγαν” για να ελέγξουν αν τα 2 υλικά που επιλέχθηκαν είναι αυτά που χρειάζονται. Έτσι οι 2 saladmakers θα έκαναν ένα άσκοπο burst cycle έτσι ώστε να κάνουν τον έλεγχο αυτό και έπειτα να ξαναγίνουν suspended. Κάτι που δεν συμβαίνει στην υλοποίηση μου με τους 3 ξεχωριστούς semaphores.
- **endSemaphore:** Κατά την ολοκλήρωση της τελευταίας σαλάτας (όχι απαραίτητα την υπ’ αριθμό numOfSlds αλλά την σαλάτα η οποία ολοκληρώνεται τελευταία) οι άλλοι δύο saladmakers είναι suspended στον tableFullSemaphore περιμένοντας να τους “ξυπνήσει” ο chef για να τους δώσει υλικά καθώς δεν γνωρίζουν πως οι ζητούμενες σαλάτες έχουν ολοκληρωθεί. Το πρόβλημα αυτό το λύνουμε με τον endSemaphore. Πιο συγκεκριμένα μόλις ο chef επιλέξει τα τελευταία 2 υλικά (και κάνει το διάλειμμα του) εκτελεί την εντολή sem_wait(endSemaphore) και γίνεται suspended περιμένοντας να ολοκληρωθεί η τελευταία σαλάτα. Μόλις ο αντίστοιχος saladmaker ολοκληρώσει την τελευταία σαλάτα ελέγχει κατευθείαν αν η σαλάτα που μόλις έφτιαξε ήταν η τελευταία. Όταν αυτό ισχύει τότε εκτελεί την εντολή sem_wait(endSemaphore) ειδοποιώντας τον chef ότι ολοκληρώθηκε η τελευταία σαλάτα. Έτσι ο chef ειδοποιεί και τους υπόλοιπους saladmakers, “ξυπνώντας” τους από τον tableFullSemaphore οι οποίοι με την σειρά τους ελέγχουν αν οι σαλάτες έχουν ολοκληρωθεί και στην συνέχεια ολοκληρώνονται. (η διαδικασία αυτή εξηγείται και πιο αναλυτικά παρακάτω)

Chef

Τα “βήματα”/λειτουργίες που ακολουθεί ο chef είναι τα εξής:
- Δημιουργεί το shared memory segment και αρχικοποιεί όλα τα δεδομένα που περιέχονται σε αυτό
- Δημιουργεί (και ανοίγει) το ενιαίο logFile στο οποίο γράφουν τα μηνύματα τους chef και saladmakers.
- Δημιουργεί όλους τους semaphores που περιγράφονται παραπάνω
- Ξεκινάει μια επαναληπτική διαδικασία για την δημιουργία των σαλατών. Ο αριθμός των επαναλήψεων είναι όσος και ο αριθμός των σαλατών που πρέπει να δημιουργηθούν. Εσωτερικά της επανάληψης αυτής γίνονται οι εξής λειτουργίες:
  - Ο chef ελέγχει αν το τραπέζι είναι άδειο, δηλαδή αν ο αντίστοιχος saladmaker έχει παραλάβει τα υλικά από το τραπέζι. Αν το τραπέζι δεν είναι άδειο περιμένει (γίνεται suspended) μέχρι ο saladmaker να παραλάβει τα υλικά. Στην πρώτη επανάληψη το τραπέζι είναι άδειο, οπότε δεν γίνεται suspended και συνεχίζει με την επόμενη εντολή.
  - Από την δεύτερη επανάληψη και μετά (καθώς στην πρώτη επανάληψη δεν έχει επιλέξει ακόμα υλικά), ο chef κάνει το διάλειμμα του (διάρκειας mantime). Το διάλειμμα (εφόσον έχει εκτελεστεί η παραπάνω εντολή) ξεκινάει από την στιγμή που τα υλικά παραλειφθούν, όπως δηλαδή περιγράφεται στην εκφώνηση.
  - Αφού τελειώσει το διάλειμμα επιλέγει τυχαία 2 υλικά, τα οποία είναι διαφορετικά από τε τελευταία δύο που επέλεξε στην προηγούμενη επανάληψη. Προφανώς δεν υπάρχει κάποιος περιορισμός στο ποια υλικά θα επιλέξει στην πρώτη επανάληψη.
  - Αφού επιλέξει τα 2 υλικά, ειδοποιεί τον αντίστοιχο saladmaker μέσω του αντίστοιχου semaphore. Δηλαδή κάνοντας post τον αντίστοιχο tableFullSemaphore.
  - Συνεχίζει με την επόμενη επανάληψη.
- Μόλις ολοκληρωθεί η επανάληψη ο chef περιμένει να παραληφθούν τα τελευταία δύο υλικά από τον αντίστοιχο saladmaker, δηλαδή γίνεται suspended στον tableEmptySemaphore μέχρι τα υλικά να παραλειφθούν
- Μόλις παραληφθούν κάνει και το τελευταίο του διάλειμμα, το οποίο αντιστοιχεί στα τελευταία 2 υλικά που επέλεξε.
- Όταν τελειώσει το διάλειμμα γίνεται suspended στον endSemaphore περιμένοντας να ολοκληρωθεί η τελευταία σαλάτα. Ο saladmaker ο οποίος θα φτιάξει την τελευταία σαλάτα, μόλις την ολοκληρώσει, κάνει post τον endSemaphore ειδοποιώντας τον chef ότι η τελευταία σαλάτα ολοκληρώθηκε, “ξυπνώντας” τον από τον endSemaphore.
- Στην συνέχεια ο chef ειδοποιεί τους υπόλοιπους saladmakers ότι οι σαλάτες ολοκληρώθηκαν, ξυπνώντας τους μέσω των αντίστοιχων tableFullSemaphores.
- Έπειτα ο chef περιμένει να τερματίσουν οι διεργασίες των saladmakers. Τον έλεγχο αυτό επέλεξα να τον κάνω για λόγους ασφάλειας έτσι ώστε όλοι οι saladmakers να έχουν προλάβει να γράψουν όλες τις πληροφορίες στα logFiles, να τα έχουν κλείσει καθώς και να έχουν γίνει όλες οι απαραίτητες αλλαγές στα δεδομένα του shared memory από μεριάς saladmakers. Αυτό γιατί όλα τα δεδομένα αυτά τα χρησιμοποιούμε στην συνέχεια και σε περίπτωση που για κάποιο λόγο κάποιος saladmaker αργήσει δεν θα ήταν καλό να κάνουμε access αυτά τα δεδομένα χωρίς αυτά να είναι ολοκληρωμένα.

- Στην συνέχεια ο chef μέσω του shared memory και συγκεκριμένα του πεδίου saladsMade εκτυπώνει το πόσες σαλάτες έφτιαξε ο κάθε saladmaker και το πόσες σαλάτες φτιάχτηκαν συνολικά.
- Έπειτα καλεί την συνάρτηση printConcurrentTimes η οποία εκτυπώνει τα κοινά διαστήματα στα οποία δύο ή παραπάνω διεργασίες ήταν ενεργές ταυτόχρονα. Η ακριβής λειτουργία της printConcurrentTimes εξηγείται παρακάτω.
- Τέλος ο chef κλείνει όλους τους semaphores και "μαρκάρει" το shared memory segment προς διαγραφή μέσω της shmctl με παράμετρο IPC_RMID. Αυτό σημαίνει ότι όταν όλες οι διεργασίες κάνουν detach το shared memory τότε αυτό θα διαγραφεί. Στην περίπτωση μας όλες οι υπόλοιπες διεργασίες το έχουν κάνει ήδη detach όποτε θα διαγραφεί κατευθείαν.

Παρατήρηση: κατά την εκτέλεση των παραπάνω λειτουργιών, ο chef γράφει τα αντίστοιχα μηνύματα στο ενιαίο logFile όπως αυτά περιγράφονται μορφοποίηση της εξόδου της εργασίας. Παραπάνω λεπτομέρειες εξηγούνται παρακάτω στην παράγραφο για τα logFiles.

Saladmakers

Τα "βήματα"/λειτουργίες που ακολουθούν οι saladmakers είναι τα εξής:
- Αρχικά ανοίγουν τα logFiles στα οποία θα γράψουν τα μυνήματα τους. Κάθε saladmakers ανοίγει 3 logFiles: το ενιαίο logFile, το ατομικό logFile και ένα κοινό logFile μεταξύ των saladmakers το οποίο χρησιμοποιείται για την εύρεση των κοινών διαστημάτων λειτουργίας. Παραπάνω λεπτομέρειες στην παράγραφο για τα logFiles.
- Κάνει attach το shared memory segment το οποίο δημιουργήθηκε από τον chef και του οποίου το κλειδί δόθηκε σαν παράμετρος κατά την εκτέλεση του saladmaker.
- Ανοίγει τους semaphores που περιγράφηκαν παραπάνω. Προφανώς ο κάθε saladmaker ανοίγει μόνο τον tableFullSemaphore που το αντιστοιχεί και όχι όλους.
- Αναθέτει το pid του στην αντίστοιχη δομή pid του shared memory έτσι ώστε αργότερα να το γνωρίζει ο chef έτσι ώστε να το γνωρίζει ο chef για να κάνει τις εκτυπώσεις όπως αυτές περιγράφονται στην μορφοποίηση εξόδου.
- Στην συνέχεια ξεκινάει μια επαναληπτική διαδικασία όπου γίνονται οι εξής λειτουργίες:
  - Ο saladmaker ελέγχει μέσω του shared memory (προφανώς με την χρήση του semaphore mutex), δηλαδή της μεταβλητής saladsToBeMade, αν έχουν ολοκληρωθεί όλες οι σαλάτες. Αν έχουν ολοκληρωθεί τότε αναθέτει την αντίστοιχη τιμή στο πεδίο finished του shared memory και τέλος διακόπτεται η επανάληψη. Αν δεν έχουν ολοκληρωθεί οι σαλάτες (δηλαδή αν saladsToBeMade>0) τότε συνεχίζει με το επόμενο βήμα.
  - Ελέγχει (μέσω του tableFullSemaphore) αν τον έχει ειδοποιήσει ο chef, δηλαδή αν στο τραπέζι υπάρχουν τα δύο υλικά που χρειάζεται. Αν όχι, τότε γίνεται suspended έως ότου ειδοποιηθεί από τον chef. Αν ναι τότε συνεχίζει το επόμενο βήμα.
  - Μόλις ειδοποιηθεί από τον chef, αρχικά ελέγχει αν οι σαλάτες έχουν ολοκληρωθεί. Αυτό συμβαίνει καθώς όταν ολοκληρωθεί η τελευταία σαλάτα από έναν saladmaker οι υπόλοιπο είναι suspended στο βήμα 2 της επανάληψης. Έτσι ο chef τους ξυπνάει έτσι ώστε να τερματίσουν. Άρα όταν "ξυπνάει" ένας saladmaker ο saladmaker πρέπει να γνωρίζει αν ο chef τον ξύπνησε για να φτιάξει μια σαλάτα η αν τον ξύπνησε επειδή οι σαλάτες ολοκληρώθηκαν και πρέπει να τερματίσει. Για αυτόν τον λόγο ξαναγίνεται αυτός ο έλεγχος.
  - Έπειτα από τον έλεγχο αυτό, ο saladmaker ειδοποιεί τον chef ότι παρέλαβε τα υλικά κάνοντας post τον tableEmptySemaphore.
  - Στην συνέχεια ξεκινάει να φτιάχνει την σαλάτα η οποία χρειάζεται έναν τυχαίο χρόνο μέσα στο εύρος που δόθηκε από τις παραμέτρους κατά την εκτέλεση.
  - Μόλις ολοκληρωθεί η σαλάτα, ο saladmaker μειώνει κατά ένα το saladsToBeMade και αυξάνει κατά ένα το saladsMade που αντιστοιχεί σε αυτόν (οι πληροφορίες αυτές βρίσκονται στο shared memory).
  - Συνεχίζει με την επόμενη επανάληψη.
- Η επανάληψη ολοκληρώνεται αρχικά από τον saladmaker ο οποίος έφτιαξε την τελευταία σαλάτα ενώ οι άλλοι είναι suspended στο βήμα 2 της επανάληψης. Για τον λόγο αυτό, ο saladmaker αυτός ειδοποιεί τον chef ότι ολοκληρώθηκε η τελευταία σαλάτα κάνοντας post τον endSemaphore. Ο chef με την σειρά του ειδοποιεί τους υπόλοιπους saladmakers κάνοντας post τους αντίστοιχους tableFullSemaphores. Έτσι οι άλλοι 2 saladmakers "ξυπνάνε" κάνουν τον έλεγχο στο βήμα 3 της επανάληψης και έτσι τερματίζεται η επανάληψη.
- Στην συνέχεια ο κάθε saladmakers κάνει detach το shared memory segment, κλείνει όλα τα ανοιχτά logFiles και τελικά τερματίζεται.

Παρατήρηση: Λέγοντας για τον saladmaker που θα φτιάξει την τελευταία σαλάτα δεν εννοώ απαραίτητα τον saladmaker που θα φτιάξει την σαλάτα υπ' αριθμό NumOfSlds, άλλα την σαλάτα η οποία θα ολοκληρωθεί τελευταία (π.χ. αν ο saladmaker που φτιάχνει την NumOfSlds σαλάτα την ολοκληρώσει πολύ γρήγορα και ένας άλλος φτιάχνει ακόμα την NumOfSlds-1 σαλάτα).

Παρατήρηση 2: Προφανώς κατά την εκτέλεση των παραπάνω λειτουργιών μεσολαβεί και το γράψιμο των μηνυμάτων στα αντίστοιχα logFiles.

LogFiles

Όλα τα μηνύματα των διεργασιών γράφονται σε logFiles αντί να εκτυπώνονται στην οθόνη.
Το όνομα του κάθε logFile έχει την μορφή logFileName_smid.txt όπου smid είναι το id του shared memory segment. Έτσι τα logFiles χαρακτηρίζονται μοναδικά και σε πολλαπλές εκτελέσεις τα προηγούμενα logFiles δεν χάνονται.
Τα logFiles που δημιουργούνται είναι: (με bold είναι τα logFileNames)

- **logFile_saladmaker[id]:** Σε κάθε εκτέλεση δημιουργούνται 3 τέτοια αρχεία, ένα για κάθε saladmaker, δηλαδή δημιουργούνται τα αρχεία: logFile_saladmaker1_smid, logFile_saladmaker2_smid, logFile_saladmaker3_smid. Κάθε ένα αρχείο περιέχει τα μηνύματα του αντίστοιχου saladmaker σχετικά με τις λειτουργίες που κάνει. Δηλαδή περιέχονται τα χρονικά timelines της δραστηριότητας του κάθε saladmaker όπως αυτό ζητείται από την εκφώνηση της άσκησης. Τα μηνύματα έχουν την μορφή η οποία περιγράφεται στην μορφοποίηση εξόδου που δόθηκε. Σε κάθε ένα από αυτά τα αρχεία γράφει μόνο μία διεργασία την φορά (δηλαδή ο αντίστοιχος saladmaker) επομένως δεν χρειάζεται κάποιος έλεγχος κάθε φορά που γράφεται κάτι.
- **joinedLogFile:** Το αρχείο αυτό περιέχει τα χρονικά timelines όλων των διεργασιών (δηλαδή του chef και των τριών saladmakers). Δηλαδή αποτυπώνεται η λειτουργία όλων των διεργασιών με την σειρά που αυτές συμβαίνουν. Μέσω αυτού μπορούμε εύκολα να δούμε με ποια σειρά εκτελούνται οι διεργασίες και η κάθε εντολή και να ελέγξουμε αν λειτουργεί σωστά και σύμφωνα με τα πρότυπα της εργασίας η ταυτόχρονη εκτέλεση των διεργασιών. Το αρχείο αυτό είναι ανοιχτό ταυτόχρονα και στις 4 διεργασίες, έτσι μπορεί 2 η παραπάνω διεργασίες να προσπαθήσουν να γράψουν ταυτόχρονα στο αρχείο αυτό. Έτσι χρειάζεται κάποιος έλεγχος κάθε φορά που μια διεργασία πρέπει να γράψει ένα δεδομένο έτσι ώστε να διασφαλίσουμε ότι μόνο μία διεργασία γράφει στο αρχείο κάθε φορά. Για τον έλεγχο αυτό χρησιμοποιώ έναν semaphore και συγκεκριμένα τον joinLogSemaphore (ο οποίος περιγράφεται και στην παράγραφο των semaphores). Ο semaphore αυτός λειτουργεί όπως ο κλασικός semaphore mutex (mutual exclusion). Έτσι, κάθε φορά που μια διεργασία θέλει να γράψει στο αρχείο εκτελεί την εντολή sem_wait(joinLogSemaphore) (και γίνεται suspended σε περίπτωση που κάποια άλλη γράφει τη ίδια χρονική στιγμή μέχρι η τελευταία να τελειώσει με το γράψιμο) και αφού γράψει εκτελεί την εντολή sem_post(joinLogSemaphore).
- **activeTimesLogFile:** Το αρχείο αυτό χρησιμοποιείται για την εύρεση των κοινών διαστημάτων εκτέλεσης μεταξύ των saladmakers. Στο αρχείο αυτό γραφούν και οι 3 saladmakers. Κάθε φορά που ολοκληρώνουν την ετοιμασία μιας σαλάτας γράφουν στο αρχείο το ποιος saladmaker είναι (π.χ. SALADMAKER1) ακολουθούμενο από το διάστημα στο οποίο ήταν ενεργοί. Το διάστημα που ένας saladmaker είναι ενεργός είναι από την στιγμή που πήρε τα υλικά από το τραπέζι μέχρι την στιγμή που ολοκλήρωσε την σαλάτα. Έτσι τελικά στο αρχείο έχουμε όλα τα διαστήματα κατά τα οποία ήταν ενεργοί οι saladmakers και τα οποία μας χρειάζονται για να βρούμε τα κοινά time intervals. Επειδή και οι 3 saladmakers γράφουν ταυτόχρονα στο αρχείο αυτό χρειάζεται ένας μηχανισμός (όπως και στο joinedLogFile) έτσι ώστε να διασφαλίζουμε ότι μόνο ένας saladmaker γράφει στο αρχείο κάθε φορά. Για αυτό, ομοίως με παραπάνω, χρησιμοποιούμε τον mutex semaphore.

Time intervals - findConcurrentTimes.c

Όπως αναφέρθηκε και παραπάνω, ο chef έπειτα από την ολοκλήρωση των σαλατών πρέπει να βρει τα διαστήματα κατά τα οποία 2 ή περισσότεροι saladmakers έδρασαν ταυτόχρονα. Αυτό γίνεται μέσω της κλήσης της συνάρτησης printConcurrentTimes από τον chef η οποία περιέχεται στο αρχείο findConcurrentTimes.c.
Η printConcurrentTimes καλεί εσωτερικά άλλες συναρτήσεις του αρχείου μέσω των οποίων τελικά βρίσκουμε τα κοινά διαστήματα. Επίσης όπως αναφέρθηκε και αμέσως παραπάνω για την εύρεση των διαστημάτων αυτών χρησιμοποιούμε το αρχείο activeTimesLogFile μέσω του οποίου αντλούμε τα διαστήματα κατά τα οποία ο κάθε saladmaker ήταν ενεργός.
Επίσης επειδή τα διαστήματα αυτά ζητείται να εκτυπώνονται ταξινομημένα χρησιμοποιώ μια ταξινομημένη λίστα. Η υλοποίηση της είναι πολύ απλή και περιέχεται επίσης στο αρχείο findConcurrentTimes.c μαζί με τα κατάλληλα σχόλια έτσι ώστε να γίνει πλήρως κατανοητή. Έτσι κάθε φορά που βρίσκουμε ένα κοινό διάστημα το προσθέτουμε στην ταξινομημένη λίστα. Αφού ολοκληρωθεί η διαδικασία (ή οποία εξηγείται αμέσως μετά) και βρεθούν όλα τα κοινά διαστήματα, τα διαστήματα είναι ταξινομημένα στην λίστα. Οπότε αρκεί να εκτυπώσουμε την λίστα αυτή και έτσι έχουμε το επιθυμητό αποτέλεσμα.
Η διαδικασία που ακολουθείται είναι η εξής:

- Συγκρίνουμε κάθε ένα ενεργό διάστημα του saladmaker1 με κάθε ένα του saladmaker2. Σε κάθε σύγκριση ελέγχουμε εάν έχουν κάποιο κοινό διάστημα μεταξύ τους και αν ναι τότε εξάγουμε το κοινό διάστημα αυτό. Εισάγουμε το κοινό διάστημα αυτό στην λίστα διαστημάτων.
  - Για κάθε κοινό διάστημα που βρίσουμε (μεταξύ των saladmaker1 και saladmaker2) το συγκρίνουμε με κάθε ένα ενεργό διάστημα του saladmaker3. Σε κάθε σύγκριση ελέγχουμε εάν έχουν κάποιο κοινό διάστημα μεταξύ τους και αν ναι τότε έχουμε βρει ένα διάστημα κατά το οποίο και οι 3 saladmakers ήταν ενεργοί. Εισάγουμε το διάστημα αυτό στην λίστα των διαστημάτων.

Μόλις ολοκληρωθεί το πρώτο bullet έχουμε καταφέρει να βρούμε όλα τα κοινά διαστήματα στα οποία ο saladmaker1 και ο saladmaker2 ήταν ενεργοί ταυτόχρονα. Επίσης έχουμε καταφέρει να βρούμε όλα τα διαστήματα στα οποία και οι 3 saladmakers ήταν ενεργοί ταυτόχρονα.

- Συγκρίνουμε κάθε ένα ενεργό διάστημα του saladmaker1 με κάθε ένα του saladmaker3. Σε κάθε σύγκριση ελέγχουμε εάν έχουν κάποιο κοινό διάστημα μεταξύ τους και αν ναι τότε εξάγουμε το κοινό διάστημα αυτό. Εισάγουμε το κοινό διάστημα αυτό στην λίστα διαστημάτων.

Μόλις ολοκληρωθεί το δεύτερο bullet έχουμε καταφέρει να βρούμε επιπλέον όλα τα κοινά διαστήματα στα οποία ο saladmaker1 και ο saladmaker3 ήταν ενεργοί ταυτόχρονα.

- Τέλος, συγκρίνουμε κάθε ένα ενεργό διάστημα του saladmaker2 με κάθε ένα του saladmaker3. Σε κάθε σύγκριση ελέγχουμε εάν έχουν κάποιο κοινό διάστημα μεταξύ τους και αν ναι τότε εξάγουμε το κοινό διάστημα αυτό. Εισάγουμε το κοινό διάστημα αυτό στην λίστα διαστημάτων.

Μόλις ολοκληρωθεί το τρίτο bullet έχουμε καταφέρει να βρούμε επιπλέον όλα τα κοινά διαστήματα στα οποία ο saladmaker2 και ο saladmaker3 ήταν ενεργοί ταυτόχρονα.

Άρα συνολικά έχουμε βρει:

- Όλα τα διαστήματα στα οποία ήταν ταυτόχρονα ενεργοί ο saladmaker1 και ο saladmaker2 (bullet 1)
- Όλα τα διαστήματα στα οποία ήταν ταυτόχρονα ενεργοί ο saladmaker1 και ο saladmaker3 (bullet 2)
- Όλα τα διαστήματα στα οποία ήταν ταυτόχρονα ενεργοί ο saladmaker2 και ο saladmaker3 (bullet 1)
- Όλα τα διαστήματα στα οποία και οι 3 διεργασίες ήταν ενεργές ταυτόχρονα (bullet 1)

Δηλαδή έχουμε βρει όλα τα διαστήματα που ζητούνται.

Άρα το μόνο που απομένει είναι να εκτυπωθεί η λίστα, δηλαδή τα διαστήματα αυτά.

Παρατήρηση: Όπως αναφέρθηκε στο piazza: αν την στιγμή που ένας σαλατοποιός τελειώνει την σαλάτα, ο chef περιμένει ήδη να δώσει τα επόμενα υλικά τότε ο σαλατοποιός δεν σταματάει, παραμένει ενεργός και συνεχίζει με την επόμενη σαλάτα.

Το παραπάνω έχει ληφθεί υπόψιν και έτσι ένα τέτοιο διάστημα παραγωγής 2 σαλατών θεωρείται σαν ένα ενιαίο διάστημα.

Παρατήρηση 2: Κατά την εκτύπωση των κοινών διαστημάτων, κάθε ένα διάστημα ακολουθείται από ένα μήνυμα το οποίο προσδιορίζει το σε ποιες διεργασίες αναφέρεται το διάστημα αυτό. Το μήνυμα αυτό δεν υπάρχει στου κανόνες μορφοποίησης αλλά το πρόσθεσα εγώ καθώς θεωρώ ότι κάνει πολύ ευκολότερο τον έλεγχο των αποτελεσμάτων.

Παρατήρηση 3: Οι λεπτομέρειες της υλοποίησης των παραπάνω μεθόδων στο αρχείο findConcurrentTimes.c εξηγούνται πλήρως στα σχόλια του αρχείου.