OS Project 3

Βησσαρίων Μουτάφης sdi1800119@di.uoa.gr
1115201800119

Notes για τους διορθωτές:

- Στα log files πέρα από το format που έδωσε ο κ. Δελής παρέχω και ένα επιπλέον πεδίο για διευκόλυνση κατά το parsing.
- Ο αλγόριθμος που χρησιμοποιώ για την εύρεση των κοινών διαστημάτων ήταν θέμα της 1ης εργασίας στο μάθημα *"Αλγόριθμοι και Πολυπλοκότητα"*, του προηγούμενου εξαμήνου. Ο αλγόριθμος για το merge των διαστημάτων αυτών αναφέρεται στις πηγές στο τέλος του doc.
- Γίνεται χρήση του PQ module, που χρησιμοποιήθηκε στην προηγούμενη εργασία. Επίσης γίνεται χρήση ενός stack, υλοποιημένου με pointers. Τα προγράμματα αυτά δεσμεύουν και αποδεσμεύουν κατάλληλα την μνήμη, κάνοντας χρήση συναρτήσεων που παρέχει ο χρήστης.
- Παρέχω κοινό Logfile, στο οποίο για να γράψει κανείς καλεί την συνάρτηση *print_log*, οποία γράφει πάντα στο *τέλος* του αρχείου και κάνει fflush το file pointer ώστε να αποφευχθεί το write buffering.

Compilation and Run

Για το compilation και run της εφαρμογής παρέχω ένα Makefile στο root directory. Οι πιθανές εντολές είναι:

```
~$ make (all)# compile all the parts of the app
~$ make clean # uninstall the app
~$ make clean-logs #clean the log files directory
~$ make chef # compile and link ONLY the chef app
~$ make salad-maker # compile and link ONLY the salad-maker app
```

Χρησιμοποιούμε separate compilation με objective files ώστε να μην χρειάζεται να ξαναφτιάχνουμε τα παντα όταν αλλάζουμε ένα μόνο αρχείο. Παράλληλα για το modulation του make παρέχω .mk αρχεία στο ανάλογο directory, ώστε με ένα total Makefile να μπορούμε να κάνουμε compile όλα τα απαραίτητα αρχεία

Abstract

Η εφαρμογή βασίζεται στα προγράμματα chef και salad-maker, τα οποία επικοινωνούν μεταξύ τους με την χρήση ***named semaphores*** και ενός ***shared memory segment***, ώστε να μπορούν να συγχρονίζονται και να παρέχου/λαμβάνουν τους αναγκαίους πόρους αναμεταξύ τους.

Utilities

Στο αρχείο *Utilities.c*, πέρα από τις συναρτήσεις που παρείχα και στις προηγούμενες 2 εργασίες περιλαμβάνω και τις συναρτήσεις για ένα καινούργιο τύπο, ονόματι MyTime, που χρησιμοποιείται για να παράγω το output στα logs όπως το ήθελε ο διδάσκοντας και είναι βάση για τον τύπο MyTimeInterval, ο οποίος υλοποιεί την έννοια του χρονικού διαστήματος έχοντας start και end timestamps, τύπου MyTime.

Shared Memory Segment

Για την χρήση των shared memory συναρτήσεων του UNIX, παρέχω ένα δικό μου API το οποίο απλά μαζεύει το error checking και κάνει πιο κατανοητές τις κλήσεις και των κώδικα, προσφέροντας επιπλέον abstraction.

Στο *shared memory segment*, παρέχουμε

- ένα counter, για την καταμέτρηση των σαλατών που δημιουργήθηκαν,
- ενα counter board, για την καταμέτρηση των σαλατών ανά σαλατοποιό,
- ένα counter, για την καταμέτρηση των σαλατοποιών που έχουν τελειώσει την δουλειά τους και έχουν κάνει dettach από το shared memory

Το read/write σε shared memory segments γίνεται πάντα ατομικά χρησιμοποιώντας ένα semaphore στην μορφή ενός mutex.

Semaphores

Για την χρήση των POSIX semaphore συναρτήσεων, παρέχω ένα δικό μου API το οποίο απλά μαζεύει το error checking και κάνει πιο κατανοητές τις κλήσεις και των κώδικα, προσφέροντας επιπλέον abstraction.

Παρέχω semaphores για τις εξής λειτουργίες:

- 3 semaphores, για την ειδοποιήση των σαλατοποιών ότι τα υλικά τους είναι στο τραπέζι,
- 1 semaphore, χρησιμοποιώντας τον ως mutex για το shared memory write/read,
- 1 semaphore, χρησιμοποιώντας τον ως mutex για την εγγραφή στο κοινό log,
- 1 semaphore, για να ειδοποιούν οι σαλατοποιοί τον chef, όταν παίρνουν τα υλικα.

Chef

Ο σεφ ξεκινάει δημιουργώντας και αρχικοποιώντας το shared memory και τους named semaphores και εκτυπώνοντας στο terminal screen, μία εντολή για να τρέξουμε τους workers μονομιάς. Ο αλγόριθμος του πάει ως εξής:

```
Shared Memory:
    salads, salads-per-saladmaker, done;
    mutex, table-mutex, w1, w2, w3;

    while (check(salads) > 0) {
        P(table-mutex);
        pick random i from {1, 2, 3} (not the same as previous time)
        V(w_i);
        sleep(mantime);
    }

    if (check(done) == 3)
        clean shared memory and semaphores

    print-stats();
    exit();
```

Listing 1: Exercise 7.25

Το *check*, γίνεται ***ατομικά με την χρήση του semaphore mutex***. Κάτι σαν αυτό:

```
bool check(salads){
    P(mutex);
    int n = salads;
    V(mutex);
    return n;
}
```

Listing 2: Exercise 7.25

Salad Makers

Οι saladmakers ξεκινάνε κάνοντας attatch το shared memory και τους named semaphores και αρχικοποιώντας σημαντικές πληροφόρίες για το logging όπως το name τους (i.e. Saladmaker1) και τα λοιπά. Ο αλγόριθμος του πάει ως εξής:

```
Shared Memory:
    salads, salads-per-saladmaker, done;
    mutex, table-mutex, w1, w2, w3;

    do {
        P(w_i); // the proper semaphore name that the relative worker block to
     is given in the cmd args
        V(table-mutex);
        if (check(salads) > 0) {
            make salad in [x] secs
            deliver(salads);
        }
    } while(check(salads) > 0)

```

```
14      check_out(done); // increase the 'done' shared variable in an ATOMIC way
15      exit();
```

Listing 3: Exercise 7.25

Η *deliver*, γίνεται ***ατομικά με την χρήση του semaphore mutex***. Κάτι σαν αυτό:

```
1 void deliver(salads){
2     P(mutex);
3     if (salads > 0) // double-check (you never know)
4         salads -= 1;
5         salads-per-saladmaker[i] += 1
6     V(mutex);
7 }
```

Listing 4: Exercise 7.25

Logging

Το logging των διεργασίων περιέχει 4 φακέλους, ένα για κάθε process και ένα κοινό log file στο οποίο όλα τα processes γραφουν ατομικά. Το τελευταίο είναι και αυτό που χρησιμοποιούμε για να βγάλουμε τα στατιστικά για τα common work intervals των salad makers.

References

- Merge Overlapping Intervals
- Χρήση του υλικού που έδωσε ο καθηγητής για τους named semaphores, καθώς και μερικά post στο stack overflow για την κατανόηση της σωστής αποδεύσμευσης και cleaning των semaphores και του shared memory segment.