Ταυτόχρονος Προγραμματισμός Εργασία 2η

Κυριακίδης Χρήστος 3029
Κεστελίδης Φίλιππος 3060

2.1 Library

- Η άσκηση 2.1 αποτελέι απλά την υλοποίηση της βιβλιοθήκης των σηματοφόρων.

2.2 Αναγνώριση πρώτων αριθμών

- Race conditions
  - Να γίνει ο έλεγχος εάν ένας αριθμός είναι πρώτος, πριν διαβαστεί η επόμενη τιμή
- Πώς το επιλύσαμε
  - Χρήση προσωρινής μεταβλητής

Worker Thread

Algorithm calcPrime(struct args)

    while (down(sem to wake up) && main is not Done) {
        save input to temp var
        “signal” main that you’ve read input using sem
        calculate Prime
    }

    “signal” main that you’ve exited using sem

    freeze all other child from signaling that they’re done
    using a sem
    “signal” that you’re done using sem

Main Thread

```
main
    initialize sem, args struct and create threads

    while(there is input) {
        “signal” the first child available to wake up
        up(sem to wake up child)
        wait for them to receive input using a sem
    }
    set a variable to show that there is no input left

[…]
```

```
[…]

    for ( i < number of workers) {
        wake up each child using sem
        wait for them to signal back that they’re
        awake and out of the loop
    }

    for ( i < number of workers) {
        check if a child exited
        wake up next child using sem
    }
```

2.3 Cars on Bridge

- Στο πρόγραμμα καλούμαστε να συγχρόνισουμε αυτοκίνητα που θέλουν να περάσουν μια γέφυρα διπλής κατεύθυνσης (αλλα μιας λωρίδας)
- Ο συγχρονισμός αυτός γίνεται με την χρήση δύο σηματοφόρων, έναν για κάθε πλευρά (left, right).
- Επειδή γίνεται αυξομείωση ακέραιων τιμών από όλα τα threads, και μπορεί κάποια να τρέχουν ταυτόχρονα, χρησιμοποιούμε «mutex lock» (με σηματοφόρο) για να αποφευχθούν τυχόν Race Conditions.
- Και η main με την σειρά της περιμένει (με χρήση σηματοφόρου) να περάσουν όλα τα αυτοκίνητα για να τερματίσει το πρόγραμμα

Απλή περιγραφή προγράμματος

- Η main κανει generate αυτοκίνητα από τις 2 πλευρές με διάφορα delay
- Όποιο αυτοκίνητο έρθει πρώτο, μπαίνει στην γέφυρα και το ακολοθούν αυτοκίνητα της ίδιας κατεύθυνσης μέχρι να γεμίσει η γέφυρα ή να μην περιμένουν άλλα αυτοκίνητα απο την συγκερκιμένη κατέυθυνση **.
- Μόλις περάσει και το τελευταίο αυτοκίνητο, δίνεται η προτεραιότητα στην απέναντι μεριά (αν υπαρχουν αυτοκίνητα που περιμένουν) και επαναλαμβάνουμε μέχρι να τελειώσει το input και έχουν περάσει όλα τα αυτοκίνητα
- ** στην συγκερκιμένη περίπτωση, άμα καταφθάσουν άλλα αυτοκίνητα της ίδιας κατέυθυνσης, περιμένουν, αφού απο την απέναντι μερία τα αυτοκίνητα περιμένουν περισσότερη ώρα.

```
mysem mtx;
mysem left, right;
int carsOnBridge = 0; int waitingLeft = 0; int WaitingRight = 0;
int direction = -1; \\ no direction
```

Car_left:

```
waitingLeft++;
down(left);
down(mtx);
if (not my direction) {
        up(mtx);
        down(left);
}
if (bridge full) down(left);
if (bridge not full && waitingLeft) up(left);

waitingLeft--;
carsOnBridge++;
up(mtx);
sleep(time_to_pass_bridge);
```

Car_right:

```
waitingRight++;
down(right);
down(mtx);
if (not my direction) {
        up(mtx);
        down(right);
}
if (bridge full) down(right);
if (bridge not full && waitingRight) up(right);

waitingRight--;
carsOnBridge++;
up(mtx);
sleep(time_to_pass_bridge);
```

```
down(mtx);
carsOnBridge--;
if (bridge empty && waitingRight) {
        give direction to other side;
        up(right);
}
else if (bridge empty && waitingLeft) up(left);
else if (bridge empty && no cars waiting) {
        set bridge to no direction;
        up(right);
        up(left);
}
up(mtx);
if (no waitingLeft && no waitingRight && input ended) {
        up(main);
}
```

```
down(mtx);
carsOnBridge--;
if (bridge empty && waitingLeft) {
        give direction to other side;
        up(left);
}
else if (bridge empty && waitingRight) up(right);
else if (bridge empty && no cars waiting) {
        set bridge to no direction;
        up(right);
        up(left);
}
up(mtx);
if (no waitingLeft && no waitingRight && input ended) {
        up(main);
}
```

2.4 Τραινάκι

- Στο συγκεκριμένο πρόγραμμα καλούμαστε να συγχρονίσουμε τα threads-επιβάτες με το thread-τραινάκι, προσομοιώνοντας την λογική του τραίνου του λούνα πάρκ
- Με την χρήση κατάλληλων διαδικών σηματοφόρων (passenger, train) οι επιβάτες «ανεβαίνουν» στο τραινάκι και όταν γεμίσει τρέχει (οι υπόλοιποι επιβάτες περιμένουν)
- Δεν δημιουργούνται race conditions γιατί σε κάθε στιγμή εκτέλεσης του προγράμματος, εκτελείται ο κώδικας είτε ενός απο τους επιβάτες είτε του τραίνου
- Ο σηματοφόρος train αρχικοποιείται σε 0 (κοιμάται) ένω των επιβατών σε 1 (για να ξεκινήσει ο πρώτος επιβάτης)

Απλή περιγραφή προγράμματος

- Όταν εκτελείται ο κώδικας ενός νήματος-επιβάτη, ανεβαίνει στο τραίνο και «ξυπνάει» έναν άλλον επιβάτη (αν υπάρχει)
- Αν το τραίνο γεμίσει, ο επιβάτης δεν «ξυπνάει» άλλον επιβάτη, αλλα ξυπνάει το νήμα του τραίνου, το οποίο τρέχει για κάποιο προκαθορισμένο χρόνο, και μετα κατεβαίνουν οι επιβάτες, και το τραίνο ξυπνάει τον επόμενο επιβάτη (αν υπάρχει)
- Αυτό επαναλαμβάνεται για πάντα (σύμφωνα με οδηγίες)

```
Struct args {
        int numPassengers;
        int maxPassengers;
        mysem train;
        mysem passenger;
};
```

```
Train (struct args):
        while (down(train)) {
                sleep(time_to_run_train);
                numPassengers = 0;
                up(passenger);
        }
```

```
Passenger (struct args):
        down(passenger);
        numPassengers++;

        if (train full) up(train);
        else up(passenger);
```