ΟΜΑΔΑ 7
Μέμου Ελβίρα 2300
Κατσαρός Χρήστος 2240

HW2_2:

Main:

Worker Thread:

for each worker
-sem next = 1
-sem avl = 0
-sem term_sig = 0

```
while (argv[j] != NULL) {
    for (i=0; i<atoi(argv[1]); i++) {
        if(i >= argc-2 || jobs == 0) break;

        mysem_down(worker[i].next);

        worker[i].number = atoi(argv[j]);
        jobs--;
        j++;
        worker[i].has_job++;

        if(worker[i].has_job == 1){

            mysem_up(worker[i].avl);

        }
    }
}

term=1;

for(i=0; i<atoi(argv[1]); i++) {
    if (worker[i].has_job == 0) {

        mysem_up(worker[i].avl);

    }

    mysem_down(worker[i].term_sig);
}
```

```
void worker_thread (worker_arg_t *arg) {

    while(1) {
        if (term && !arg->has_job) break;

        mysem_down(arg->avl);

        if (arg->has_job == 0) break;

        if(primetest(arg->number)){
            printf("IS PRIME\n");
        }
        else{
            printf("IS NOT PRIME\n");
        }

        arg->has_job--;

        mysem_up(arg->next);

    }
    printf("Done Thread");

    mysem_up(arg->term_sig);
}
```

Στη main περνάμε δουλειές στο εκάστοτε worker thread. Αρχικά μπλοκάρουμε τον next έτσι ώστε να μην μπορεί να πάρει δουλειά ενώ έχει ήδη και στο worker thread με το που τελειώσει η δουλειά το ξεμπλοκάρουμε. Το worker thread από τη δημιουργία του είναι μπλοκαρισμένο στον σηματοφόρο available (avl) μέχρι να παρει δουλειά απο τη main ώστε να ξεμπλοκάρει. Τέλος έχουμε δύο συνθήκες τερματισμού για το κάθε worker thread. Η πρώτη (term = 0) είναι να μην έχουν ανατεθεί όλες οι δουλειές αλλά το συγκεκριμένο thread να μην έχει κάποια δουλειά να πάρει , τότε παραμένει κολλημένο στο προαναφερθέν σημείο και το has_job είναι ίσο με το μηδέν οπότε με το που πάρει σήμα από τη main θα τερματίσει. Η δεύτερη περίπτωση (term = 1) είναι να έχουν ανατεθεί από την main όλες οι δουλειές και να είναι μπλοκαρισμένο στο down αλλά να έχει δουλειά να κάνει. Τότε το has_job είναι ένα και μόλις τελειώσει τη δουλειά του θα γίνει μηδέν οπότε θα βγει και από το πρώτο break που υπάρχει στον κώδικα του worker thread. Τέλος, κάθε thread ενημερώνει την main ότι τελειώσε για να ξεμπλοκάρει και να τερματίσει.

HW2_3:

sem bridge = 1;
sem blue = 0;
sem red = 0;
sem finish = 0;

ENTER_BRIDGE:

down(bridge)

if (red_on_bridge == 0 and blue_on_bridge < MAX_CAPACITY)
    this blue car can pass the bridge
    up(blue)
else
    blue car waits

up(bridge)

down(blue)

-SAME CODE FOR BOTH COLORS-

EXIT_BRIDGE:

down(bridge)

blue car exits

if (there are red cars waiting and all blue cars passed)
    give priority to red cars now
    up(red)
else if (there are no red cars left and there are still blue cars)
    keep priority to blue cars
    up(blue)

up(bridge)

if (there are no other cars)
    signal main that we finished
    up(finish)

-SAME CODE FOR BOTH COLORS-

HW2_4:

Σε αυτήν την εργασία έπρεπε να συγχρονίσουμε το τρενάκι με τους επιβάτες, τους επιβάτες μεταξύ τους καθώς και την main με το τρενάκι ώστε να τερματίζει σωστά το πρόγραμμα.

Passenger:

```
void passenger_thread (int *id) {

    mysem_down(pass_semid);

    printf("%d thread riding\n", *id);
    passengers_riding++;
    passengers_left--;

    if (passengers_riding == N) {
        mysem_up(start_train);
    }
    else{
        mysem_up(pass_semid);
    }

    mysem_down(finish_route);

    printf("KATEBHKA THREAD: %d\n", *id);
    passengers_got_off++;

    if(passengers_got_off == N){
        passengers_got_off = 0;
        mysem_up(empty_train);
    }
}
```

Train:

```
void train_thread () {

    int i;

    while (1) {

        mysem_down(start_train);

        printf("\nTOY TOY TOY\n");
        sleep(TRIP_TIME);
        printf("TELOS DIADROMHS / KATEBEITE\n\n");

        for(i=0; i<N; i++){
            mysem_up(finish_route);
        }

        mysem_down(empty_train);

        if(passengers_left == 0){
            break;
        }
        else if (passengers_left < N) {
            printf("\nNot full train\n");
            break;
        }

        passengers_riding = 0;
        mysem_up(pass_semid);

    }
    mysem_up(term_train);
}
```

Συγχρονισμός: Το pass_semid αρχικοποιείται με 1 έτσι ώστε ο δεύτερος επιβάτης να μην μπει στο τρενάκι προτού ελέγξουμε αν χωράει. Αν χωράει τότε ξεμπλοκάρει ο επόμενος επιβάτης. Αν δεν χωράει το start_train γίνεται up για να ξεμπλοκάρει το τρενάκι που έχει ήδη μπλοκάρει στο start_train που αρχικοποιείται με 0 και να αρχίσει την διαδρομή. Στην συνέχεια όλοι οι επιβάτες που έχουν επιβιβαστεί μπλοκάρουν στο finish_route μέχρι να τελειώσει τη βόλτα το τρενάκι και να τους κάνει up όλους ώστε να ξεμπλοκάρουν και να κατεβούν. Αφού κατεβούν όλοι γίνεται up το empty_train ώστε να ενημερωθεί το τρενάκι ότι άδειασε και αυτό με την σειρά του να επιτρέψει στους επόμενους επιβάτες να εισέλθουν, κάνοντας up το pass_semid και 0 το passengers_riding. Επιπλέον, το τρενάκι κάθε φορά που τελειώνει την βόλτα ελέγχει αν το passengers_left είναι 0 ή αν δεν υπαρχουν οι απαραίτητοι επιβάτες ώστε να αρχίσει την επόμενη βόλτα ή να τερματίσει το train_thread και να κάνει up το term_train για να τερματίσει η main.