Übungsblatt 10

Übungsgruppe Pentium

Stefan Schmauch, Nina Cami, Anton Lydike

Donnerstag 09.07.2020

Aufgabe 1) __ /6p.

virt. Adresse	Zugriff	phys. Adresse	Schutzverl.?
0x000	Fetch	0x400	no
0x2a0	Lesen	0x1b0	size
0x1b0	Schreiben	0x130	read-only
0x330	Lesen	0x0d0	no
0x1c0	Fetch	0x140	data-only
0x304	Schreiben	0x0a4	no

Aufgabe 2) __ /10p.

```c
#include <stdio.h>
#include <stdlib.h>
#include <traffic.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#include <math.h>

#define NUM_CARS 12

/* Map for libtraffic, C-header structure */

/* First the dimension of the map */
#define map_width 11
#define map_height 11

/* Now the map data */
static map_shorts map_def[] =
{
AU,AU,AU,AU,AU,AU,AU,AU,AU,AU,AU,
AU,S2,SH,SH,SH,SH,SH,SH,SH,S3,AU,
AU,SV,AU,__,__,__,__,__,AU,SV,AU,
AU,SV,__,__,__,__,__,__,__,SV,AU,
AU,SV,__,__,__,__,__,__,__,SV,AU,
AU,SV,__,__,__,__,__,__,__,SV,AU,
AU,SV,__,__,__,__,__,__,__,SV,AU,
AU,SV,__,__,__,__,__,__,__,SV,AU,
AU,SV,AU,__,__,__,__,__,AU,SV,AU,
AU,S1,SH,SH,SH,SH,SH,SH,SH,S4,AU,
AU,AU,AU,AU,AU,AU,AU,AU,AU,AU,AU
};

typedef struct position {
    int x;
    int y;
} position;

typedef struct thread_parms {
    int id;             // car number
    sem_t* my_sem;      // this cars semaphore
    sem_t* next_sem;    // the semaphore for the car in front
    position pos;
    directions dir;
} thread_parms;

position get_pos(int dist);
void * car_thread(void *arg);
directions get_dir(position pos);
directions invert_dir(directions dir);

void main() {
    createMap (map_def, map_width, map_height, NULL);
    sem_t semaphores[NUM_CARS * 2];
    thread_parms parameters[NUM_CARS * 2];
    pthread_t threads[NUM_CARS * 2];

    // calculate total number of street tiles
    int street_tile_count = 2 * (map_width - 2) + 2 * (map_height - 2);

    // we take this as float and round later to place cars evenly along the course
    float distance = street_tile_count / ((float) NUM_CARS + 1.0);

    printf("running course with %d cars, distance is %f\n", NUM_CARS, distance);

    // initialize thread parameters (clockwise)
    for (int i = 0; i < NUM_CARS; i++) {
        // calc distance from prev car to this car
        int curr_dist = floor(distance * i) - floor(distance * (i - 1));
        // first car's distance to prev car has rounding error
        if (i == 0) curr_dist--;

        sem_init(semaphores + i, 0, curr_dist - 1);
        parameters[i].id = i;
        parameters[i].pos = get_pos(floor(distance * i));
        parameters[i].my_sem = semaphores + i;
        parameters[i].dir = get_dir(parameters[i].pos);

        // link car semaphores together
        if (i > 0) {
            parameters[i - 1].next_sem = semaphores + i;
        }
    }

    // and anti clockwise
    for (int j = NUM_CARS; j < NUM_CARS * 2; j++) {
        int i = j - NUM_CARS;
        // calc distance to next car
        int curr_dist = floor(distance * (i + 1)) - floor(distance * i);

        sem_init(semaphores + j, 0, curr_dist - 1);
        parameters[j].id = j;
        parameters[j].pos = get_pos(floor(distance * i));
        parameters[j].my_sem = semaphores + j;
        parameters[j].dir = invert_dir(get_dir(parameters[j].pos));

        // link car semaphores together
        if (i < NUM_CARS - 1) {
            parameters[j + 1].next_sem = semaphores + j;
        }
    }

    // link to first semaphore in last car
    parameters[NUM_CARS - 1].next_sem = semaphores;
    // and also for other direction
    parameters[NUM_CARS].next_sem = semaphores + (NUM_CARS * 2) - 1;

    // start all threads
    for (int i = 0; i < NUM_CARS * 2; i++) {
        pthread_create(threads + i, NULL, car_thread, parameters + i);
    }

    // wait for threads
    for (int i = 0; i < NUM_CARS * 2; i++) {
        pthread_join(threads[i], NULL);
    }

    // destroy semaphores
    for (int i = 0; i < NUM_CARS * 2; i++) {
        sem_destroy(semaphores + i);
    }

    destroyMap(0);
}

// get position moved x clockwise from (1,1) (or thereabouts)
position get_pos(int dist) {
    // define an initial position
    position pos;
    pos.x = 1;
    pos.y = 2;

    // honestly, I don't really know what the final thnking behind this was
    // old code did NOT work, so I made this.
    pos.y -= dist;

    if (pos.y < 1) {
        pos.x += 1 - pos.y;
        pos.y = 1;
    }

    if (pos.x > 9) {
        pos.y += pos.x - 9;
        pos.x = 9;
    }

    if (pos.y > 9) {
        pos.x -= pos.y - 9;
        pos.y = 9;
    }

    if (pos.x < 1) {
        pos.y -= 1 - pos.x;
        pos.x = 1;
    }

    return pos;
}

directions get_dir(position pos) {
    // left side
    if (pos.x == 1) {
        if (pos.y == 1) return O;
        return N;
    }
    // bottom side
    if (pos.y == 9)  return W;
    // right side
    if (pos.x == 9) return S;
    // must be top side
    return O;
}
directions invert_dir(directions dir) {
    if (dir == W) return O;
    if (dir == O) return W;
    if (dir == S) return N;
    if (dir == N) return S;
    return O; // ???
}

void * car_thread(void *arg) {
    thread_parms *pars = ((thread_parms *) arg);
    thread_parms par = *pars;

    int car_id = putCar(par.pos.x, par.pos.y, par.dir, 0);
    //printf("car %d at pos (%d,%d,%s)\n", par.id, par.pos.x, par.pos.y, dirToString(
        par.dir));

    int dist_to_travel = 60;

    while(dist_to_travel > 0) {
        if (par.id % NUM_CARS == 0 && dist_to_travel == 55) {
            sleep(10);
        }

        // wait for next car to be at least 1 away
        sem_wait(par.next_sem);
        // move that one place
        moveCar(car_id, routeCar(car_id).dir);
        // notify waiting cars that you moved 1 forward
        sem_post(par.my_sem);
        // make it stop eventually
        dist_to_travel--;
    }

    return NULL;
}
```

Gesamtpunkte: **__ /16p.**