Министерство цифрового развития

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра прикладной математики и кибернетики

Отчёт

по лабораторной работе № 4 «Система ПВО и летающие тарелки»

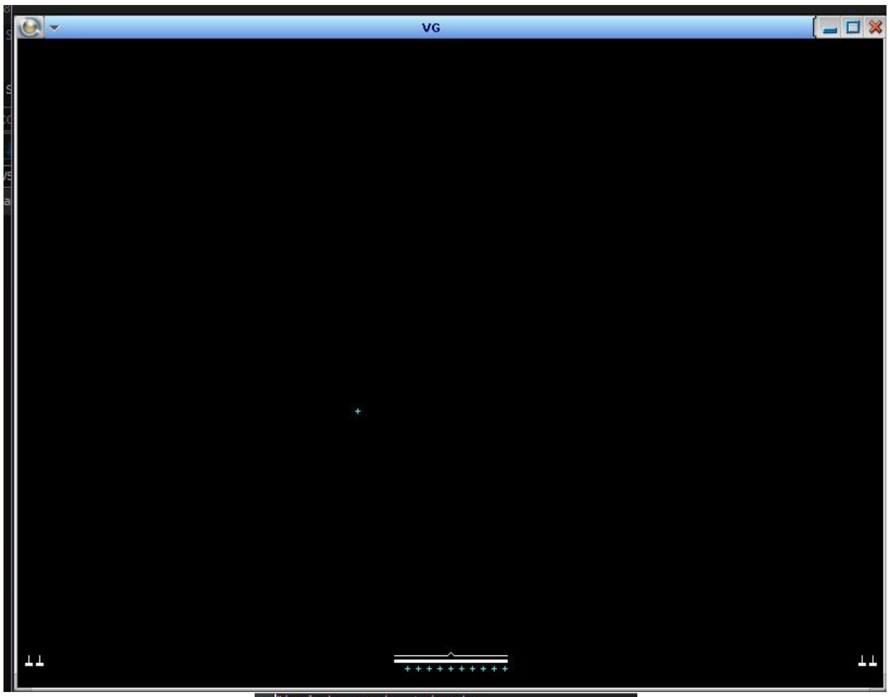
Выполнил:
студент группы ИП-216
Русецкий А.С.

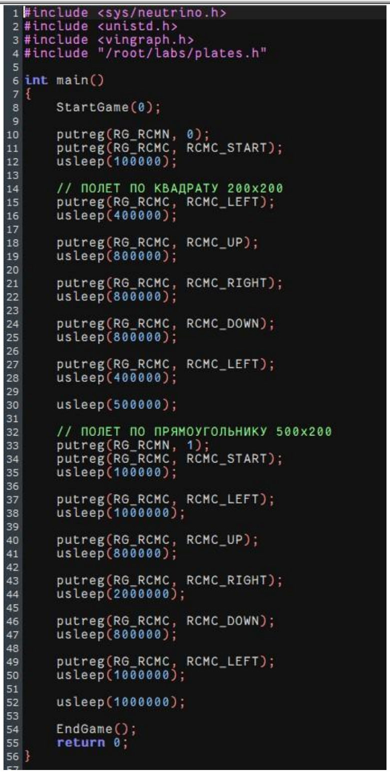
Работу проверил:
Ассистент
Шевелькова В.Ю.

Новосибирск 2025 г.

Ступень 0

Написать программу, осуществляющую полёт управляемого снаряда по квадрату 200x200, а затем по прямоугольнику 500x200 точек. Тарелок нет.

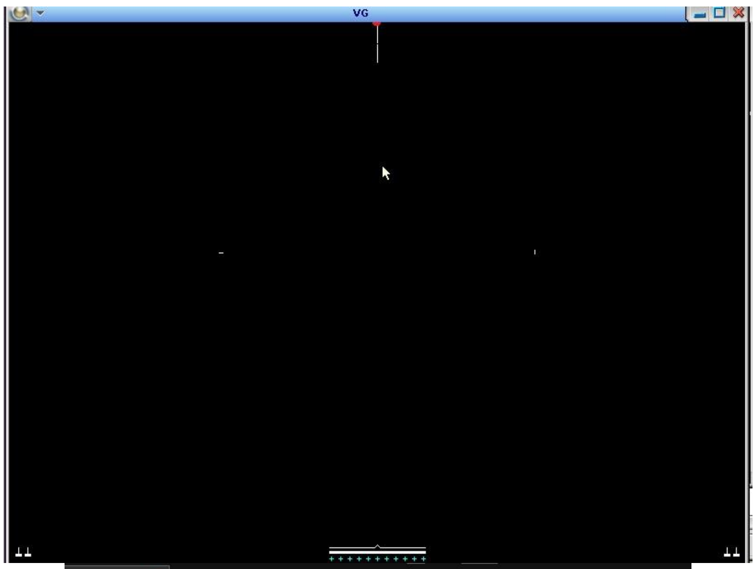


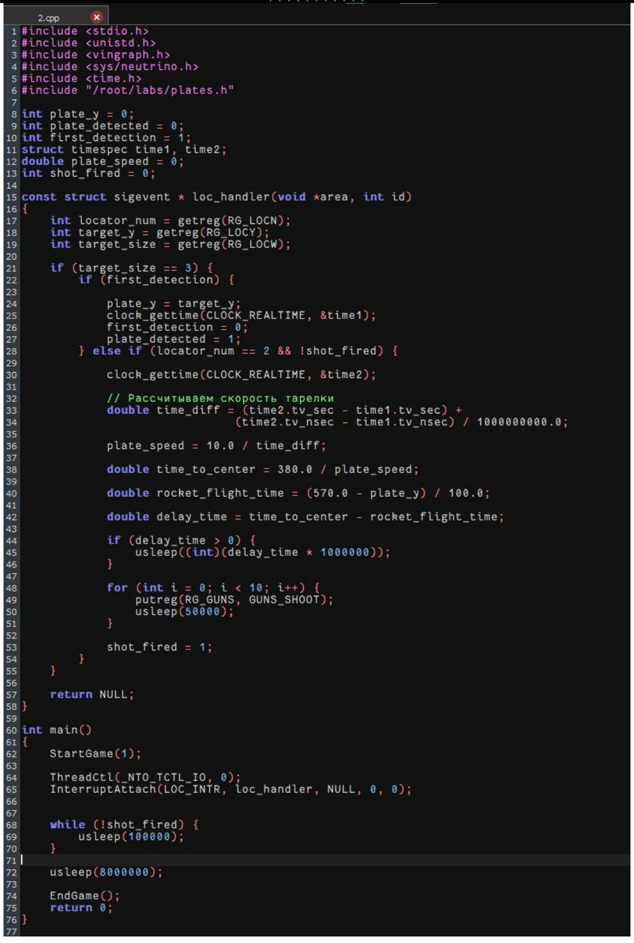


```
#include <sys/neutrino.h>
#include <unistd.h>
#include <vingraph.h>
#include "/root/labs/plates.h"

int main()
{
    StartGame(0);

    putreg(RG_RCMN, 0);
    putreg(RG_RCMC, RCMC_START);
    usleep(100000);

    // ПОЛЕТ ПО КВАДРАТУ 200x200
    putreg(RG_RCMC, RCMC_LEFT);
    usleep(400000);

    putreg(RG_RCMC, RCMC_UP);
    usleep(800000);

    putreg(RG_RCMC, RCMC_RIGHT);
    usleep(800000);

    putreg(RG_RCMC, RCMC_DOWN);
    usleep(800000);

    putreg(RG_RCMC, RCMC_LEFT);
    usleep(400000);

    usleep(500000);

    // ПОЛЕТ ПО ПРЯМОУГОЛЬНИКУ 500x200
    putreg(RG_RCMN, 1);
    putreg(RG_RCMC, RCMC_START);
    usleep(100000);

    putreg(RG_RCMC, RCMC_LEFT);
    usleep(1000000);

    putreg(RG_RCMC, RCMC_UP);
    usleep(800000);

    putreg(RG_RCMC, RCMC_RIGHT);
    usleep(2000000);

    putreg(RG_RCMC, RCMC_DOWN);
    usleep(800000);

    putreg(RG_RCMC, RCMC_LEFT);
    usleep(1000000);

    usleep(1000000);

    EndGame();
    return 0;
}
```

Ступень 1

Написать программу, сбивающую одну тарелку с помощью ракеты (тарелка движется слева направо)

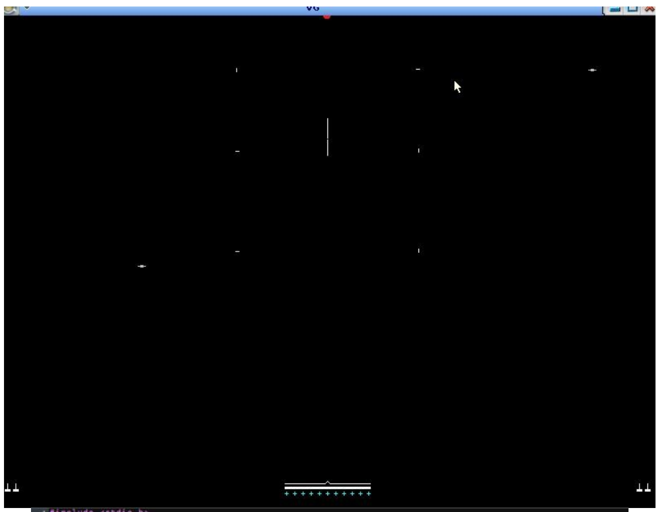


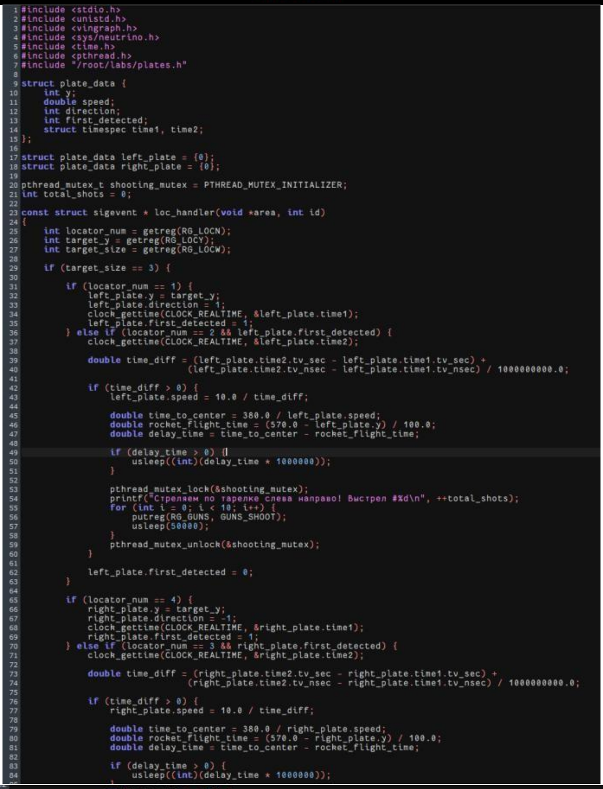


```cpp
#include <stdio.h>
#include <unistd.h>
#include <vingraph.h>
#include <sys/neutrino.h>
#include <time.h>
#include "/root/labs/plates.h"

int plate_y = 0;
int plate_detected = 0;
int first_detection = 1;
struct timespec time1, time2;
double plate_speed = 0;
int shot_fired = 0;

const struct sigevent * loc_handler(void *area, int id)
{
    int locator_num = getreg(RG_LOCN);
    int target_y = getreg(RG_LOCY);
    int target_size = getreg(RG_LOCW);

    if (target_size == 3) {
        if (first_detection) {

            plate_y = target_y;
            clock_gettime(CLOCK_REALTIME, &time1);
            first_detection = 0;
            plate_detected = 1;
        } else if (locator_num == 2 && !shot_fired) {

            clock_gettime(CLOCK_REALTIME, &time2);

            // Рассчитываем скорость тарелки
            double time_diff = (time2.tv_sec - time1.tv_sec) +
                              (time2.tv_nsec - time1.tv_nsec) / 1000000000.0;

            plate_speed = 10.0 / time_diff;

            double time_to_center = 380.0 / plate_speed;

            double rocket_flight_time = (570.0 - plate_y) / 100.0;

            double delay_time = time_to_center - rocket_flight_time;

            if (delay_time > 0) {
                usleep((int)(delay_time * 1000000));
            }

            for (int i = 0; i < 10; i++) {
                putreg(RG_GUNS, GUNS_SHOOT);
                usleep(50000);
            }

            shot_fired = 1;
        }
    }

    return NULL;
}

int main()
{
    StartGame(1);

    ThreadCtl(_NTO_TCTL_IO, 0);
    InterruptAttach(LOC_INTR, loc_handler, NULL, 0, 0);


    while (!shot_fired) {
        usleep(100000);
    }

    usleep(8000000);

    EndGame();
    return 0;
}
```

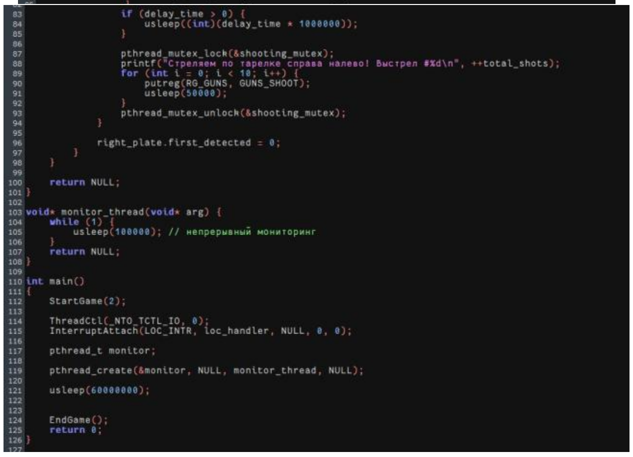
Ступень 2

Написать программу, сбивающую несколько тарелок с помощью ракет (тарелки движутся в разных направлениях).





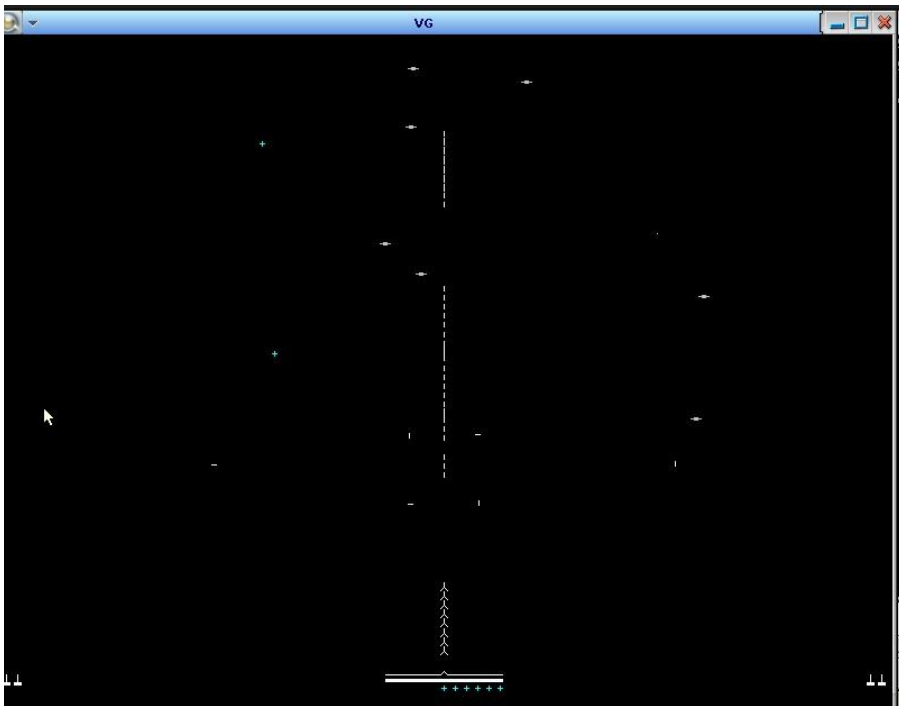
```c
#include <stdio.h>
#include <unistd.h>
#include <vingraph.h>
#include <sys/neutrino.h>
#include <time.h>
#include <pthread.h>
#include "/root/labs/plates.h"

struct plate_data {
    int y;
    double speed;
    int direction;
    int first_detected;
    struct timespec time1, time2;
};

struct plate_data left_plate = {0};
struct plate_data right_plate = {0};

pthread_mutex_t shooting_mutex = PTHREAD_MUTEX_INITIALIZER;
int total_shots = 0;

const struct sigevent * loc_handler(void *area, int id)
{
    int locator_num = getreg(RG_LOCN);
    int target_y = getreg(RG_LOCY);
    int target_size = getreg(RG_LOCW);

    if (target_size == 3) {

        if (locator_num == 1) {
            left_plate.y = target_y;
            left_plate.direction = 1;
            clock_gettime(CLOCK_REALTIME, &left_plate.time1);
            left_plate.first_detected = 1;
        } else if (locator_num == 2 && left_plate.first_detected) {
            clock_gettime(CLOCK_REALTIME, &left_plate.time2);

            double time_diff = (left_plate.time2.tv_sec - left_plate.time1.tv_sec) +
                               (left_plate.time2.tv_nsec - left_plate.time1.tv_nsec) / 1000000000.0;

            if (time_diff > 0) {
                left_plate.speed = 10.0 / time_diff;

                double time_to_center = 380.0 / left_plate.speed;
                double rocket_flight_time = (570.0 - left_plate.y) / 100.0;
                double delay_time = time_to_center - rocket_flight_time;

                if (delay_time > 0) {
                    usleep((int)(delay_time * 1000000));
                }

                pthread_mutex_lock(&shooting_mutex);
                printf("Стреляем по тарелке слева направо! Выстрел #%d\n", ++total_shots);
                for (int i = 0; i < 10; i++) {
                    putreg(RG_GUNS, GUNS_SHOOT);
                    usleep(50000);
                }
                pthread_mutex_unlock(&shooting_mutex);
            }

            left_plate.first_detected = 0;
        }

        if (locator_num == 4) {
            right_plate.y = target_y;
            right_plate.direction = -1;
            clock_gettime(CLOCK_REALTIME, &right_plate.time1);
            right_plate.first_detected = 1;
        } else if (locator_num == 3 && right_plate.first_detected) {
            clock_gettime(CLOCK_REALTIME, &right_plate.time2);

            double time_diff = (right_plate.time2.tv_sec - right_plate.time1.tv_sec) +
                               (right_plate.time2.tv_nsec - right_plate.time1.tv_nsec) / 1000000000.0;

            if (time_diff > 0) {
                right_plate.speed = 10.0 / time_diff;

                double time_to_center = 380.0 / right_plate.speed;
                double rocket_flight_time = (570.0 - right_plate.y) / 100.0;
                double delay_time = time_to_center - rocket_flight_time;

                if (delay_time > 0) {
                    usleep((int)(delay_time * 1000000));
```

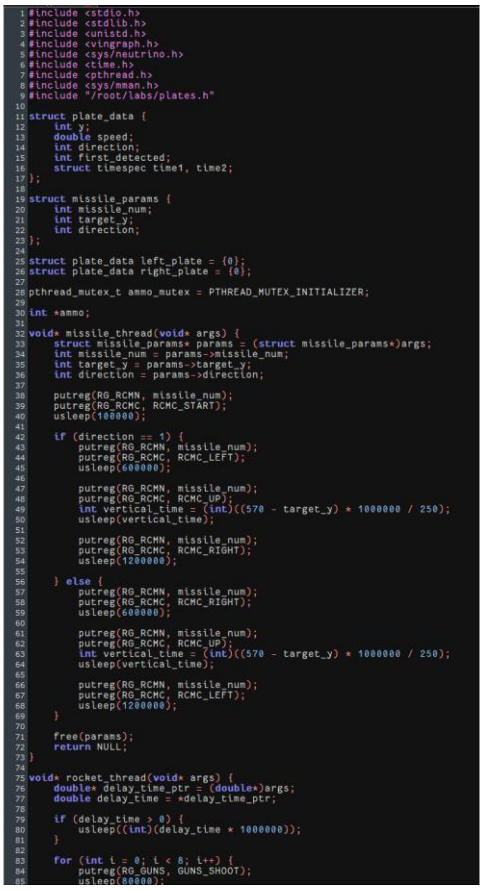


```c
                if (delay_time > 0) {
                    usleep((int)(delay_time * 1000000));
                }

                pthread_mutex_lock(&shooting_mutex);
                printf("Стреляем по тарелке справа налево! Выстрел #%d\n", ++total_shots);
                for (int i = 0; i < 10; i++) {
                    putreg(RG_GUNS, GUNS_SHOOT);
                    usleep(50000);
                }
                pthread_mutex_unlock(&shooting_mutex);
            }

            right_plate.first_detected = 0;
        }
    }

    return NULL;
}

void* monitor_thread(void* arg) {
    while (1) {
        usleep(100000); // непрерывный мониторинг
    }
    return NULL;
}

int main()
{
    StartGame(2);

    ThreadCtl(_NTO_TCTL_IO, 0);
    InterruptAttach(LOC_INTR, loc_handler, NULL, 0, 0);

    pthread_t monitor;

    pthread_create(&monitor, NULL, monitor_thread, NULL);

    usleep(60000000);


    EndGame();
    return 0;
}
```

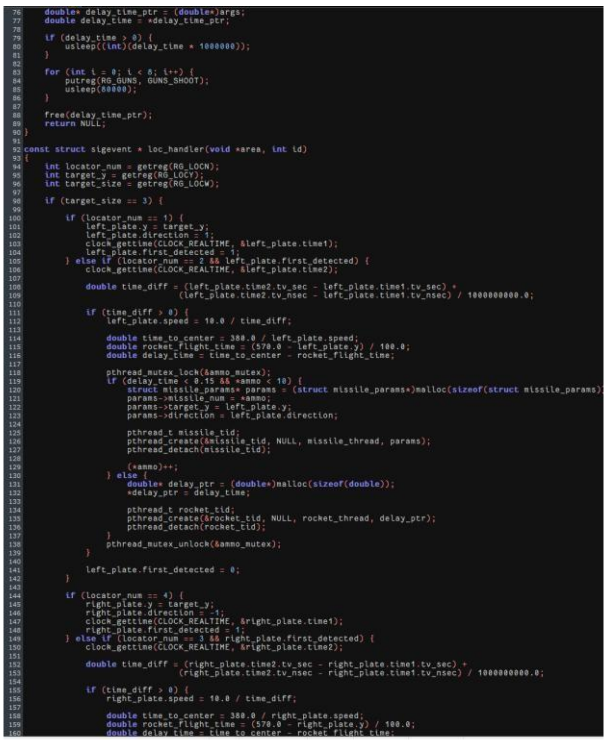
Ступень 3

Написать программу, сбивающую медленные тарелки ракетами, а быстрые – управляемыми тарелками

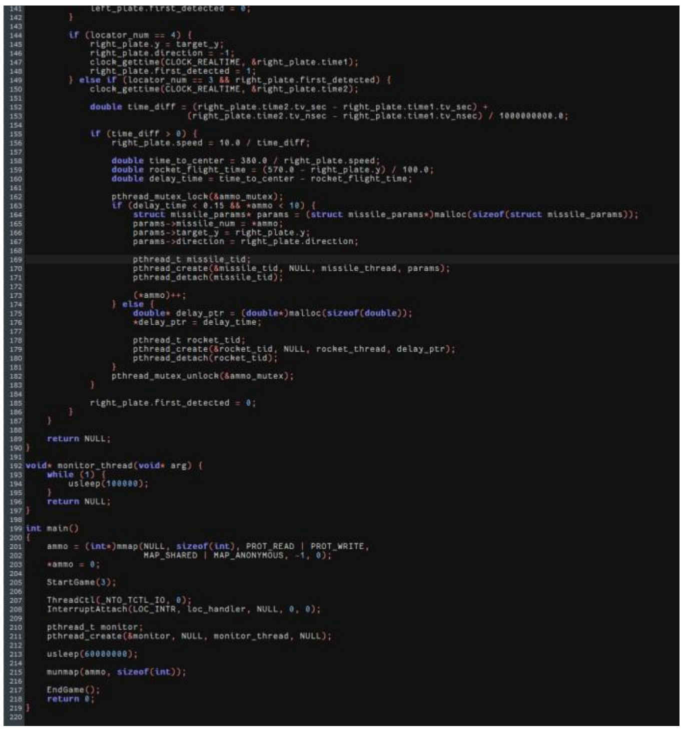




```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <vingraph.h>
#include <sys/neutrino.h>
#include <time.h>
#include <pthread.h>
#include <sys/mman.h>
#include "/root/labs/plates.h"

struct plate_data {
    int y;
    double speed;
    int direction;
    int first_detected;
    struct timespec time1, time2;
};

struct missile_params {
    int missile_num;
    int target_y;
    int direction;
};

struct plate_data left_plate = {0};
struct plate_data right_plate = {0};

pthread_mutex_t ammo_mutex = PTHREAD_MUTEX_INITIALIZER;

int *ammo;

void* missile_thread(void* args) {
    struct missile_params* params = (struct missile_params*)args;
    int missile_num = params->missile_num;
    int target_y = params->target_y;
    int direction = params->direction;

    putreg(RG_RCMN, missile_num);
    putreg(RG_RCMC, RCMC_START);
    usleep(100000);

    if (direction == 1) {
        putreg(RG_RCMN, missile_num);
        putreg(RG_RCMC, RCMC_LEFT);
        usleep(600000);

        putreg(RG_RCMN, missile_num);
        putreg(RG_RCMC, RCMC_UP);
        int vertical_time = (int)((570 - target_y) * 1000000 / 250);
        usleep(vertical_time);

        putreg(RG_RCMN, missile_num);
        putreg(RG_RCMC, RCMC_RIGHT);
        usleep(1200000);

    } else {
        putreg(RG_RCMN, missile_num);
        putreg(RG_RCMC, RCMC_RIGHT);
        usleep(600000);

        putreg(RG_RCMN, missile_num);
        putreg(RG_RCMC, RCMC_UP);
        int vertical_time = (int)((570 - target_y) * 1000000 / 250);
        usleep(vertical_time);

        putreg(RG_RCMN, missile_num);
        putreg(RG_RCMC, RCMC_LEFT);
        usleep(1200000);
    }

    free(params);
    return NULL;
}

void* rocket_thread(void* args) {
    double* delay_time_ptr = (double*)args;
    double delay_time = *delay_time_ptr;

    if (delay_time > 0) {
        usleep((int)(delay_time * 1000000));
    }

    for (int i = 0; i < 8; i++) {
        putreg(RG_GUNS, GUNS_SHOOT);
        usleep(80000);
```



```c
    double* delay_time_ptr = (double*)args;
    double delay_time = *delay_time_ptr;

    if (delay_time > 0) {
        usleep((int)(delay_time * 1000000));
    }

    for (int i = 0; i < 8; i++) {
        putreg(RG_GUNS, GUNS_SHOOT);
        usleep(80000);
    }

    free(delay_time_ptr);
    return NULL;
}

const struct sigevent * loc_handler(void *area, int id)
{
    int locator_num = getreg(RG_LOCN);
    int target_y = getreg(RG_LOCY);
    int target_size = getreg(RG_LOCW);

    if (target_size == 3) {

        if (locator_num == 1) {
            left_plate.y = target_y;
            left_plate.direction = 1;
            clock_gettime(CLOCK_REALTIME, &left_plate.time1);
            left_plate.first_detected = 1;
        } else if (locator_num == 2 && left_plate.first_detected) {
            clock_gettime(CLOCK_REALTIME, &left_plate.time2);

            double time_diff = (left_plate.time2.tv_sec - left_plate.time1.tv_sec) +
                               (left_plate.time2.tv_nsec - left_plate.time1.tv_nsec) / 1000000000.0;

            if (time_diff > 0) {
                left_plate.speed = 10.0 / time_diff;

                double time_to_center = 380.0 / left_plate.speed;
                double rocket_flight_time = (570.0 - left_plate.y) / 100.0;
                double delay_time = time_to_center - rocket_flight_time;

                pthread_mutex_lock(&ammo_mutex);
                if (delay_time < 0.15 && *ammo < 10) {
                    struct missile_params* params = (struct missile_params*)malloc(sizeof(struct missile_params)
                    params->missile_num = *ammo;
                    params->target_y = left_plate.y;
                    params->direction = left_plate.direction;

                    pthread_t missile_tid;
                    pthread_create(&missile_tid, NULL, missile_thread, params);
                    pthread_detach(missile_tid);

                    (*ammo)++;
                } else {
                    double* delay_ptr = (double*)malloc(sizeof(double));
                    *delay_ptr = delay_time;

                    pthread_t rocket_tid;
                    pthread_create(&rocket_tid, NULL, rocket_thread, delay_ptr);
                    pthread_detach(rocket_tid);
                }
                pthread_mutex_unlock(&ammo_mutex);
            }

            left_plate.first_detected = 0;
        }

        if (locator_num == 4) {
            right_plate.y = target_y;
            right_plate.direction = -1;
            clock_gettime(CLOCK_REALTIME, &right_plate.time1);
            right_plate.first_detected = 1;
        } else if (locator_num == 3 && right_plate.first_detected) {
            clock_gettime(CLOCK_REALTIME, &right_plate.time2);

            double time_diff = (right_plate.time2.tv_sec - right_plate.time1.tv_sec) +
                               (right_plate.time2.tv_nsec - right_plate.time1.tv_nsec) / 1000000000.0;

            if (time_diff > 0) {
                right_plate.speed = 10.0 / time_diff;

                double time_to_center = 380.0 / right_plate.speed;
                double rocket_flight_time = (570.0 - right_plate.y) / 100.0;
                double delay_time = time_to_center - rocket_flight_time;
```



```c
            left_plate.first_detected = 0;
        }

        if (locator_num == 4) {
            right_plate.y = target_y;
            right_plate.direction = -1;
            clock_gettime(CLOCK_REALTIME, &right_plate.time1);
            right_plate.first_detected = 1;
        } else if (locator_num == 3 && right_plate.first_detected) {
            clock_gettime(CLOCK_REALTIME, &right_plate.time2);

            double time_diff = (right_plate.time2.tv_sec - right_plate.time1.tv_sec) +
                               (right_plate.time2.tv_nsec - right_plate.time1.tv_nsec) / 1000000000.0;

            if (time_diff > 0) {
                right_plate.speed = 10.0 / time_diff;

                double time_to_center = 380.0 / right_plate.speed;
                double rocket_flight_time = (570.0 - right_plate.y) / 100.0;
                double delay_time = time_to_center - rocket_flight_time;

                pthread_mutex_lock(&ammo_mutex);
                if (delay_time < 0.15 && *ammo < 10) {
                    struct missile_params* params = (struct missile_params*)malloc(sizeof(struct missile_params));
                    params->missile_num = *ammo;
                    params->target_y = right_plate.y;
                    params->direction = right_plate.direction;

                    pthread_t missile_tid;
                    pthread_create(&missile_tid, NULL, missile_thread, params);
                    pthread_detach(missile_tid);

                    (*ammo)++;
                } else {
                    double* delay_ptr = (double*)malloc(sizeof(double));
                    *delay_ptr = delay_time;

                    pthread_t rocket_tid;
                    pthread_create(&rocket_tid, NULL, rocket_thread, delay_ptr);
                    pthread_detach(rocket_tid);
                }
                pthread_mutex_unlock(&ammo_mutex);
            }

            right_plate.first_detected = 0;
        }
    }

    return NULL;
}

void* monitor_thread(void* arg) {
    while (1) {
        usleep(100000);
    }
    return NULL;
}

int main()
{
    ammo = (int*)mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE,
                      MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    *ammo = 0;

    StartGame(3);

    ThreadCtl(_NTO_TCTL_IO, 0);
    InterruptAttach(LOC_INTR, loc_handler, NULL, 0, 0);

    pthread_t monitor;
    pthread_create(&monitor, NULL, monitor_thread, NULL);

    usleep(60000000);

    munmap(ammo, sizeof(int));

    EndGame();
    return 0;
}
```