

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»

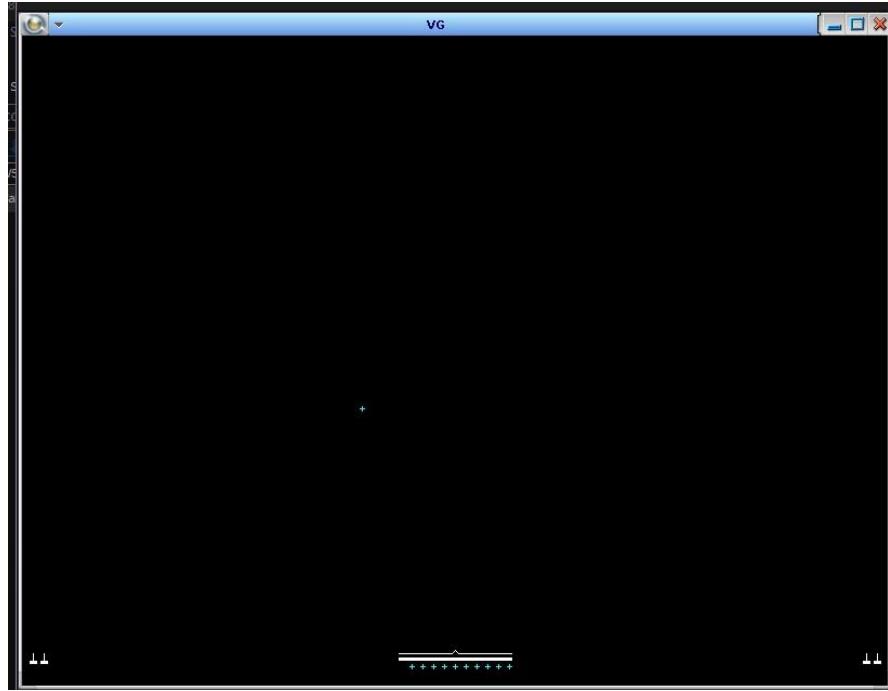
Выполнил:
студент группы ИП-216
Русецкий А.С.

Работу проверил:
Ассистент
Шевелькова В.Ю.

Новосибирск 2025 г.

Ступень 0

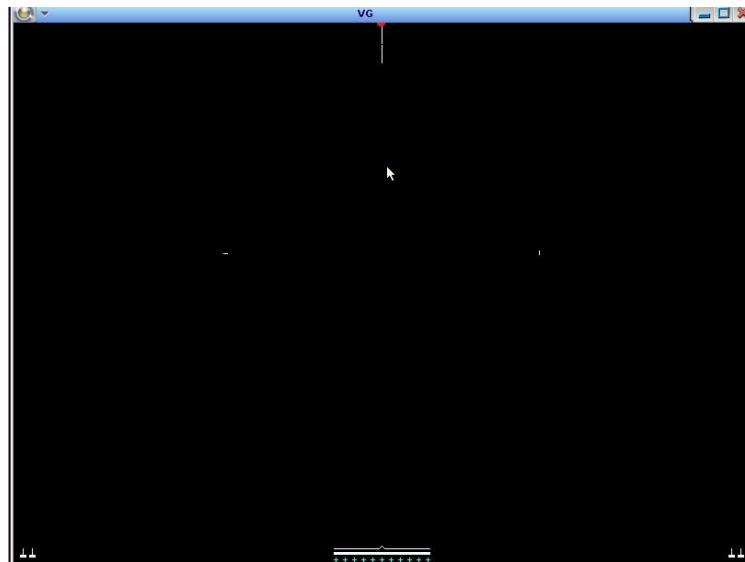
Написать программу, осуществляющую полёт управляемого снаряда по квадрату 200x200, а затем по прямоугольнику 500x200 точек. Тарелок нет.



```
1 #include <sys/neutrino.h>
2 #include <unistd.h>
3 #include <vgraph.h>
4 #include "/root/labs/plates.h"
5
6 int main()
7 {
8     StartGame(0);
9
10    putreg(RG_RCMN, 0);
11    putreg(RG_RCMC, RCMC_START);
12    usleep(100000);
13
14    // ПОЛЕТ ПО КВАДРАТУ 200x200
15    putreg(RG_RCMC, RCMC_LEFT);
16    usleep(400000);
17
18    putreg(RG_RCMC, RCMC_UP);
19    usleep(800000);
20
21    putreg(RG_RCMC, RCMC_RIGHT);
22    usleep(800000);
23
24    putreg(RG_RCMC, RCMC_DOWN);
25    usleep(800000);
26
27    putreg(RG_RCMC, RCMC_LEFT);
28    usleep(400000);
29
30    usleep(500000);
31
32    // ПОЛЕТ ПО ПРЯМОУГОЛЬНИКУ 500x200
33    putreg(RG_RCMN, 1);
34    putreg(RG_RCMC, RCMC_START);
35    usleep(100000);
36
37    putreg(RG_RCMC, RCMC_LEFT);
38    usleep(1000000);
39
40    putreg(RG_RCMC, RCMC_UP);
41    usleep(800000);
42
43    putreg(RG_RCMC, RCMC_RIGHT);
44    usleep(2000000);
45
46    putreg(RG_RCMC, RCMC_DOWN);
47    usleep(800000);
48
49    putreg(RG_RCMC, RCMC_LEFT);
50    usleep(1000000);
51
52    usleep(1000000);
53
54    EndGame();
55
56 }
```

Ступень 1

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



```
2.cpp x
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/types.h>
4 #include <sys/neutrino.h>
5 #include <time.h>
6 #include "/root/labs/plates.h"
7
8 int plate_y = 0;
9 int plate_detected = 0;
10 int first_detection = 1;
11 struct timespec time1, time2;
12 double plate_speed = 0;
13 int shot_fired = 0;
14
15 const struct sigevent * loc_handler(void *area, int id)
16 {
17     int locator_num = getreg(RG_LOCN);
18     int target_y = getreg(RG_LCY);
19     int target_size = getreg(RG_LOCW);
20
21     if (target_size == 3) {
22         if (first_detection) {
23             plate_y = target_y;
24             clock_gettime(CLOCK_REALTIME, &time1);
25             first_detection = 0;
26             plate_detected = 1;
27         } else if (locator_num == 2 && !shot_fired) {
28             clock_gettime(CLOCK_REALTIME, &time2);
29
30             // Рассчитываем скорость тарелки
31             double time_diff = (time2.tv_sec - time1.tv_sec) +
32                                 (time2.tv_nsec - time1.tv_nsec) / 1000000000.0;
33
34             plate_speed = 10.0 / time_diff;
35
36             double time_to_center = 380.0 / plate_speed;
37
38             double rocket_flight_time = (570.0 - plate_y) / 100.0;
39
40             double delay_time = time_to_center - rocket_flight_time;
41
42             if (delay_time > 0) {
43                 usleep((int)(delay_time * 1000000));
44             }
45
46             for (int i = 0; i < 10; i++) {
47                 putreg(RG_GUNS, GUNS_SHOOT);
48                 usleep(50000);
49             }
50
51             shot_fired = 1;
52         }
53     }
54
55     return NULL;
56 }
57
58
59
60 int main()
61 {
62     StartGame(1);
63
64     ThreadCtl(NTO_TCTL_IO, 0);
65     InterruptAttach(LOC_INTR, loc_handler, NULL, 0, 0);
66
67
68     while (!shot_fired) {
69         usleep(100000);
70     }
71
72     usleep(800000);
73
74     EndGame();
75
76     return 0;
77 }
```

Ступень 2

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



```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/neutrino.h>
#include <clms.h>
#include <clmsd.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 *ares, int id)

{
    int locator_num = getreg(RG_LOCN);
    int target_y = getreg(RG_LOCY);
    int target_size = getreg(RG_LCW);

    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 = 0;
        } 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((1m)(delay_time * 1000000));
                }
                pthread_mutex_lock(&shooting_mutex);
                printf("Спрейка по ракете Chess напасо! Выстрелил #d\n", ++total_shots);
                for (int i = 0; i < 10; i++) {
                    putreg(80, GUNS, GUNS_SHOOT);
                    usleep(50000);
                }
                pthread_mutex_unlock(&shooting_mutex);
            }
            left_plate.first_detected = 0;
        }

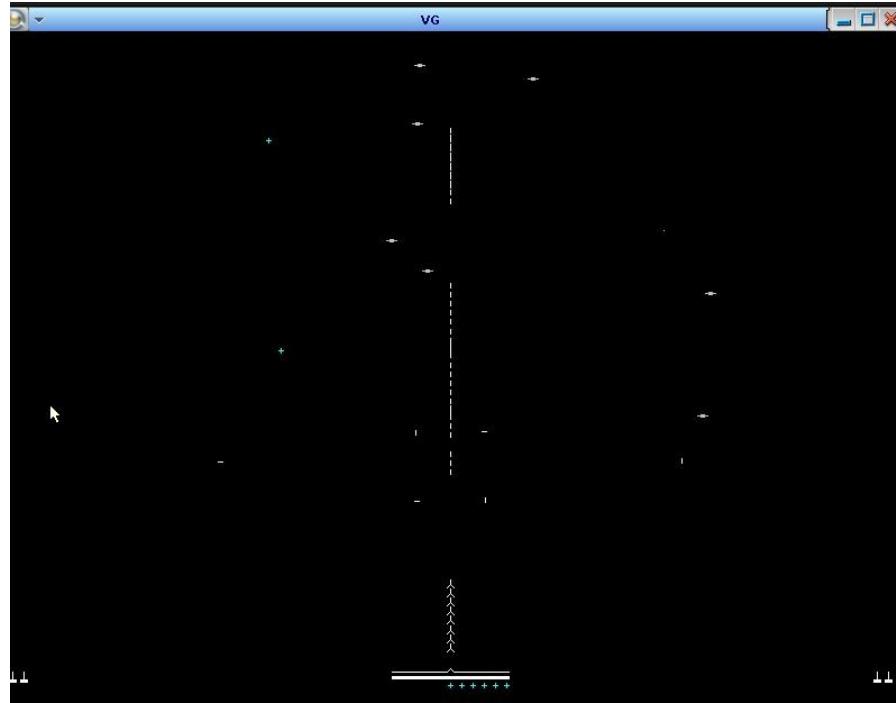
        if (locator_num == 4) {
            right_plate.y = target_y;
            right_plate.direction = -1;
            clock_gettime(CLOCK_REALTIME, &right_plate.time1);
            right_plate.first_detected = 0;
        } else if (locator_num == 5 && right_plate.first_detected) {
            clock_gettime(CLOCK_REALTIME, &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((1m)(delay_time * 1000000));
                }
                if (delay_time > 0) {
                    usleep((1m)(delay_time * 1000000));
                }
                pthread_mutex_lock(&shooting_mutex);
                printf("Спрейка по ракете Chess напасо! Выстрелил #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();
    ThreadCtl(_NTO_TCTL_IO, 0);
    InterruptAttach(LOC_INTR, loc_handler, NULL, 0, 0);
    pthread_t monitor;
    pthread_create(&monitor, NULL, monitor_thread, NULL);
    usleep(6000000);
    EndGame();
    return 0;
}
```

Ступень 3

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



```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/types.h>
5 #include <sys/neutrino.h>
6 #include <time.h>
7 #include <pthread.h>
8 #include <sys/mman.h>
9 #include "/root/labs/plates.h"
10
11 struct plate_data {
12     int y;
13     double speed;
14     int direction;
15     int first_detected;
16     struct timespec time1, time2;
17 };
18
19 struct missile_params {
20     int missile_num;
21     int target_y;
22     int direction;
23 };
24
25 struct plate_data left_plate = {};
26 struct plate_data right_plate = {};
27
28 pthread_mutex_t ammo_mutex = PTHREAD_MUTEX_INITIALIZER;
29
30 int *ammo;
31
32 void* missile_thread(void* args) {
33     struct missile_params* params = (struct missile_params*)args;
34     int missile_num = params->missile_num;
35     int target_y = params->target_y;
36     int direction = params->direction;
37
38     putreg(RG_RCMN, missile_num);
39     putreg(RG_RCMC, RCMC_START);
40     usleep(100000);
41
42     if (direction == 1) {
43         putreg(RG_RCMN, missile_num);
44         putreg(RG_RCMC, RCMC_LEFT);
45         usleep(600000);
46
47         putreg(RG_RCMN, missile_num);
48         putreg(RG_RCMC, RCMC_UP);
49         int vertical_time = (int)((570 - target_y) * 1000000 / 250);
50         usleep(vertical_time);
51
52         putreg(RG_RCMN, missile_num);
53         putreg(RG_RCMC, RCMC_RIGHT);
54         usleep(1200000);
55     }
56     else {
57         putreg(RG_RCMN, missile_num);
58         putreg(RG_RCMC, RCMC_RIGHT);
59         usleep(600000);
60
61         putreg(RG_RCMN, missile_num);
62         putreg(RG_RCMC, RCMC_UP);
63         int vertical_time = (int)((570 - target_y) * 1000000 / 250);
64         usleep(vertical_time);
65
66         putreg(RG_RCMN, missile_num);
67         putreg(RG_RCMC, RCMC_LEFT);
68         usleep(1200000);
69     }
70     free(params);
71     return NULL;
72 }
73
74 void* rocket_thread(void* args) {
75     double* delay_time_ptr = (double*)args;
76     double delay_time = *delay_time_ptr;
77
78     if (delay_time > 0) {
79         usleep((int)(delay_time * 1000000));
80     }
81
82     for (int i = 0; i < 8; i++) {
83         putreg(RG_GUNS, GUNS_SHOOT);
84         usleep(100000);
85     }
}
```

```

76    double* delay_time_ptr = (double*)arg0;
77    double delay_time = *delay_time_ptr;
78
79    if (delay_time > 0) {
80        usleep((int)(delay_time * 1000000));
81    }
82
83    for (int i = 0; i < 8; i++) {
84        int locator_x = getreg(R5_LOCK);
85        int target_y = getreg(R6_LOCK);
86        int target_size = getreg(R6_LOCK);
87
88        if (target_size == 3) {
89            if (locator_num == i) {
90                left_plate.y = target_y;
91                left_plate.time = time;
92                clock_gettime(CLOCK_REALTIME, &left_plate.time);
93                left_plate.first_detected = 1;
94            } else {
95                if (left_plate.time <= left_plate.first_detected) {
96                    clock_gettime(CLOCK_REALTIME, &left_plate.time2);
97
98                    double time_diff = (left_plate.time2.tv_sec - left_plate.time.tv_sec) +
99                        (left_plate.time2.tv_nsec - left_plate.time.tv_nsec) / 1000000000.0;
100
101                    if (time_diff > 0) {
102                        left_plate.speed = 10.0 / time_diff;
103
104                        double time_to_center = 388.0 / left_plate.speed;
105                        double rocket_flight_time = (370.0 - right_plate.y) / 100.0;
106                        double delay_time = time_to_center - rocket_flight_time;
107
108                        pthread_mutex.lock(&ammo_mutex);
109                        if (delay_time < 0.15 && ammo < 10) {
110                            struct missile_params* params = (struct missile_params*)malloc(sizeof(struct missile_params));
111                            params->missile_num = i+10;
112                            params->target_y = left_plate.y;
113                            params->target_x = left_plate.x;
114                            params->direction = left_plate.direction;
115
116                            pthread_t missile_tid;
117                            pthread_create(&missile_tid, NULL, missile_thread, params);
118                            pthread_detach(missile_tid);
119
120                            (ammo)++;
121
122                            double* delay_ptr = (double*)malloc(sizeof(double));
123                            *delay_ptr = delay_time;
124
125                            pthread_t rocket_tid;
126                            pthread_create(&rocket_tid, NULL, rocket_thread, delay_ptr);
127                            pthread_detach(rocket_tid);
128
129                            pthread_mutex.unlock(&ammo_mutex);
130
131                        } else {
132                            double* delay_ptr = (double*)malloc(sizeof(double));
133                            *delay_ptr = delay_time;
134
135                            pthread_t rocket_tid;
136                            pthread_create(&rocket_tid, NULL, rocket_thread, delay_ptr);
137                            pthread_detach(rocket_tid);
138
139                            pthread_mutex.unlock(&ammo_mutex);
140
141                            left_plate.first_detected = 0;
142
143                        }
144
145                        if (locator_num == 4) {
146                            right_plate.y = target_y;
147                            right_plate.time = time;
148                            clock_gettime(CLOCK_REALTIME, &right_plate.time);
149                            right_plate.first_detected = 1;
150                        } else {
151                            if (right_plate.time <= right_plate.first_detected) {
152                                clock_gettime(CLOCK_REALTIME, &right_plate.time2);
153
154                                double time_diff = (right_plate.time2.tv_sec - right_plate.time.tv_sec) +
155                                    (right_plate.time2.tv_nsec - right_plate.time.tv_nsec) / 1000000000.0;
156
157                                if (time_diff > 0) {
158                                    right_plate.speed = 10.0 / time_diff;
159
160                                    double time_to_center = 388.0 / right_plate.speed;
161                                    double rocket_flight_time = (370.0 - right_plate.y) / 100.0;
162                                    double delay_time = time_to_center - rocket_flight_time;
163
164                                    pthread_mutex.lock(&ammo_mutex);
165                                    if (delay_time < 0.15 && ammo < 10) {
166                                        struct missile_params* params = (struct missile_params*)malloc(sizeof(struct missile_params));
167                                        params->missile_num = i+10;
168                                        params->target_y = right_plate.y;
169                                        params->target_x = right_plate.x;
170                                        params->direction = right_plate.direction;
171
172                                        pthead_t missile_tid;
173                                        pthread_create(&missile_tid, NULL, missile_thread, params);
174                                        pthread_detach(missile_tid);
175
176                                        (ammo)++;
177
178                                        double* delay_ptr = (double*)malloc(sizeof(double));
179                                        *delay_ptr = delay_time;
180
181                                        pthread_t rocket_tid;
182                                        pthread_create(&rocket_tid, NULL, rocket_thread, delay_ptr);
183                                        pthread_detach(rocket_tid);
184
185                                        pthread_mutex.unlock(&ammo_mutex);
186
187                                    } else {
188                                        right_plate.first_detected = 0;
189
190                                    }
191
192                                }
193
194                            void* monitor_thread(void* arg) {
195                                while (1) {
196                                    usleep(100000);
197                                }
198                            return NULL;
199
200
201    int main()
202    {
203        ammo = (int*)mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE,
204                           MAP_SHARED | MAP_ANONYMOUS, -1, 0);
205        ammo = 0;
206
207        StartGame();
208
209        ThreadCtl(NTO_TIDL_IO_0);
210        InterruptAttach(LOC_INTR, loc_handler, NULL, 0, 0);
211
212        pthread_t monitor;
213        pthread_create(&monitor, NULL, monitor_thread, NULL);
214
215        usleep(60000000);
216
217        munmap(ammo, sizeof(int));
218
219        EndGame();
220
221        return 0;
222
223    }

```