

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»

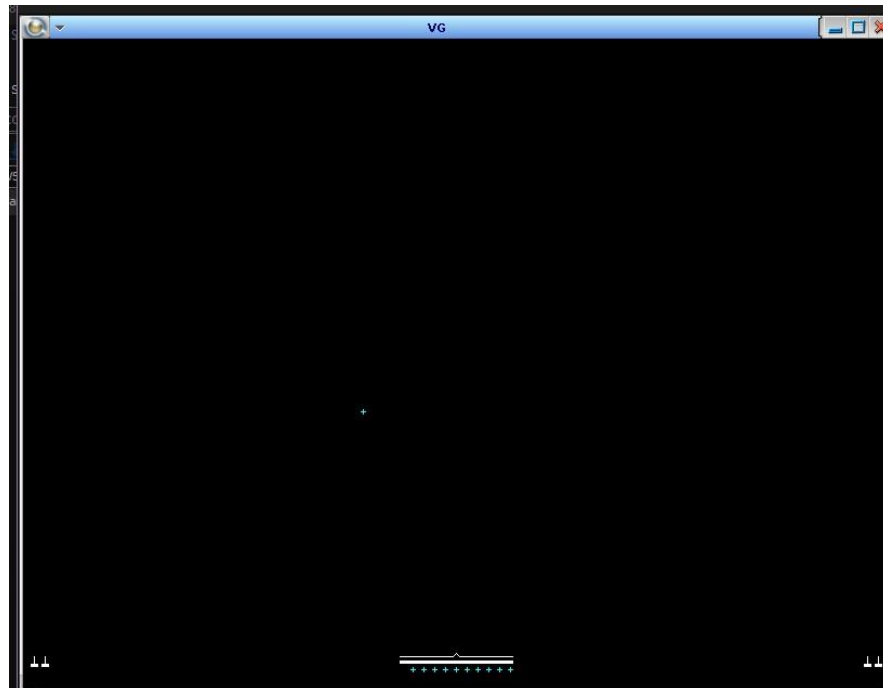
Выполнил:  
студент группы ИП-216  
Русецкий А.С.

Работу проверил:  
Ассистент  
Шевелькова В.Ю.

Новосибирск 2025 г.

## Ступень 0

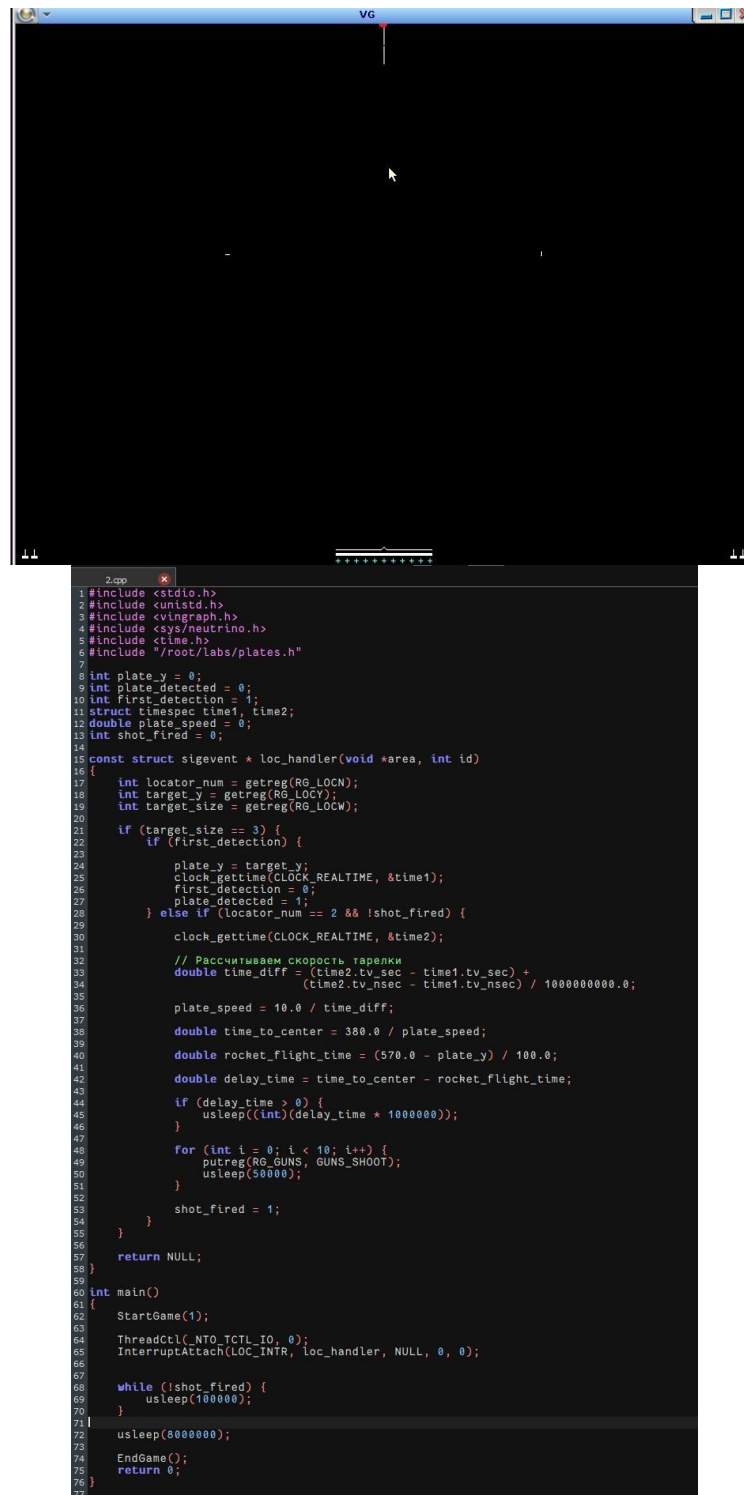
Написать программу, осуществляющую полёт управляемого снаряда по квадрату 200x200, а затем по прямоугольнику 500x200 точек. Тарелок нет.



```
1 #include <sys/neutrino.h>
2 #include <unistd.h>
3 #include <vingraph.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    return 0;
56 }
```

## Ступень 1

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

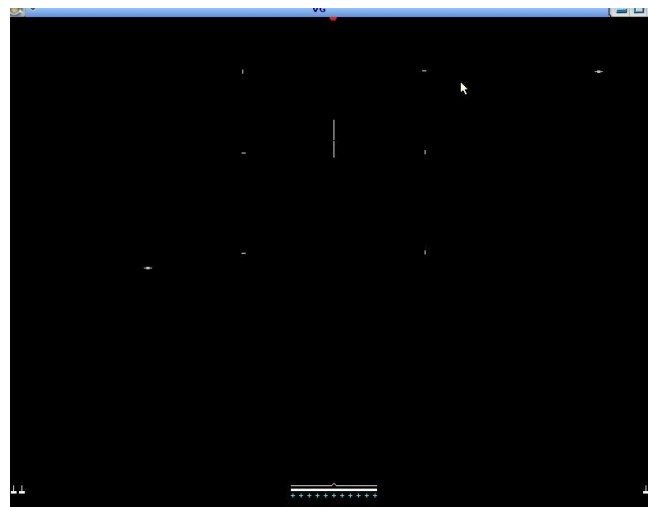


The image shows a terminal window with a game interface at the top and its C source code below. The game interface, titled 'vg', displays a dark screen with a small red dot at the top center, representing a target. A mouse cursor is visible near the center. The code, titled '2.cpp', is a C program that implements a game logic. It includes headers for standard I/O, time, and a custom header 'plates.h'. It defines variables for plate position, detection, and speed. A signal handler 'loc\_handler' is registered to handle events, which includes logic for detecting the target, calculating the time to center, and firing a rocket. The main function initializes the game, starts the signal handler, and enters a loop that waits for the shot to be fired before ending the game.

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <vring.h>
4 #include <sys/neutrino.h>
5 #include <time.h>
6 #include "/root/tabs/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_LCW);
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 int main()
59 {
60     StartGame(1);
61     ThreadCtl(_NTO_TCTL_IO, 0);
62     InterruptAttach(LOC_INTR, loc_handler, NULL, 0, 0);
63
64     while (!shot_fired) {
65         usleep(100000);
66     }
67
68     usleep(8000000);
69     EndGame();
70     return 0;
71 }
```

## Ступень 2

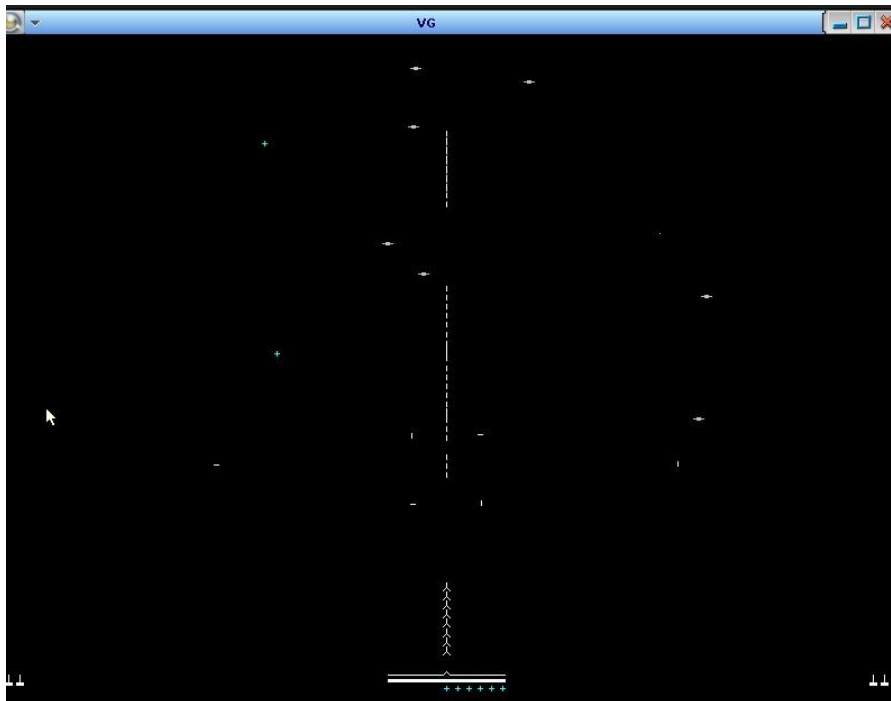
Написать программу, сбивающую несколько тарелок с помощью ракет (тарелки движутся в разных направлениях).



```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <wingraph.h>
4 #include <sys/mman.h>
5 #include <time.h>
6 #include <pthread.h>
7 #include <root/labs/plates.h>
8
9 struct plate_data {
10     int y;
11     double speed;
12     int direction;
13     int first_detected;
14     struct timespec time1, time2;
15 };
16
17 struct plate_data left_plate = {0};
18 struct plate_data right_plate = {0};
19
20 pthread_mutex_t shooting_mutex = PTHREAD_MUTEX_INITIALIZER;
21 int total_shots = 0;
22
23 const struct sigevent * loc_handler(void *area, int id)
24 {
25     int locator_num = getreg(RG_LOCK);
26     int target_y = getreg(RG_LOCKY);
27     int target_size = getreg(RG_LOCKW);
28
29     if (target_size == 3) {
30         if (locator_num == 1) {
31             left_plate.y = target_y;
32             left_plate.direction = 1;
33             clock_gettime(CLOCK_REALTIME, &left_plate.time1);
34             left_plate.first_detected = 1;
35         } else if (locator_num == 2 && left_plate.first_detected) {
36             clock_gettime(CLOCK_REALTIME, &left_plate.time2);
37
38             double time_diff = (left_plate.time2.tv_sec - left_plate.time1.tv_sec) +
39                               (left_plate.time2.tv_nsec - left_plate.time1.tv_nsec) / 1000000000.0;
40
41             if (time_diff > 0) {
42                 left_plate.speed = 10.0 / time_diff;
43
44                 double time_to_center = 300.0 / left_plate.speed;
45                 double rocket_flight_time = (570.0 - left_plate.y) / 100.0;
46                 double delay_time = time_to_center - rocket_flight_time;
47
48                 if (delay_time > 0) {
49                     usleep((int)(delay_time * 1000000));
50                 }
51
52                 pthread_mutex_lock(&shooting_mutex);
53                 printf("Сбиваем тарелку слева направо! Скорость: %d\n", ++total_shots);
54                 for (int i = 0; i < 10; i++) {
55                     putreg(RG_GUNS, GUNS_SHOOT);
56                     usleep(50000);
57                 }
58                 pthread_mutex_unlock(&shooting_mutex);
59             }
60             left_plate.first_detected = 0;
61         }
62
63         if (locator_num == 4) {
64             right_plate.y = target_y;
65             right_plate.direction = -1;
66             clock_gettime(CLOCK_REALTIME, &right_plate.time1);
67             right_plate.first_detected = 1;
68         } else if (locator_num == 3 && right_plate.first_detected) {
69             clock_gettime(CLOCK_REALTIME, &right_plate.time2);
70
71             double time_diff = (right_plate.time2.tv_sec - right_plate.time1.tv_sec) +
72                               (right_plate.time2.tv_nsec - right_plate.time1.tv_nsec) / 1000000000.0;
73
74             if (time_diff > 0) {
75                 right_plate.speed = 10.0 / time_diff;
76
77                 double time_to_center = 300.0 / right_plate.speed;
78                 double rocket_flight_time = (570.0 - right_plate.y) / 100.0;
79                 double delay_time = time_to_center - rocket_flight_time;
80
81                 if (delay_time > 0) {
82                     usleep((int)(delay_time * 1000000));
83                 }
84                 if (delay_time > 0) {
85                     usleep((int)(delay_time * 1000000));
86                 }
87
88                 pthread_mutex_lock(&shooting_mutex);
89                 printf("Сбиваем тарелку справа налево! Скорость: %d\n", ++total_shots);
90                 for (int i = 0; i < 10; i++) {
91                     putreg(RG_GUNS, GUNS_SHOOT);
92                     usleep(50000);
93                 }
94                 pthread_mutex_unlock(&shooting_mutex);
95             }
96             right_plate.first_detected = 0;
97         }
98     }
99     return NULL;
100 }
101
102 void* monitor_thread(void* arg) {
103     while (1) {
104         usleep(1000000); // непрерывный мониторинг
105     }
106     return NULL;
107 }
108
109 int main()
110 {
111     StartGame(2);
112     ThreadCtl(NTD_TCTL_IO, 0);
113     InterruptAttech(LOC_INTA, loc_handler, NULL, 0, 0);
114     pthread_t monitor;
115     pthread_create(&monitor, NULL, monitor_thread, NULL);
116     usleep(600000000);
117     EndGame();
118     return 0;
119 }
```

### 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 <vringgraph.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
11struct plate_data {
12    int y;
13    double speed;
14    int direction;
15    int first_detected;
16    struct timespec time1, time2;
17};
18
19struct missile_params {
20    int missile_num;
21    int target_y;
22    int direction;
23};
24
25struct plate_data left_plate = {0};
26struct plate_data right_plate = {0};
27
28pthread_mutex_t ammo_mutex = PTHREAD_MUTEX_INITIALIZER;
29
30int *ammo;
31
32void* 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(500000);
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(500000);
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
71    free(params);
72    return NULL;
73}
74
75void* rocket_thread(void* args) {
76    double* delay_time_ptr = (double*)args;
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        putreg(RG_GUNS, GUNS_SHOOT);
85        usleep(500000);
86    }
87}
```

```

76 double delay_time_ptr = (double*)&args;
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     pthread_t t;
85     pthread_create(&t, NULL, BOMB_SHOOT);
86     usleep(100000);
87 }
88 free(delay_time_ptr);
89 return NULL;
90 }
91
92 const struct sigevent = Loc_handler(void *area, int id)
93 {
94     int locator_num = getreg(RG_LOCN);
95     int target_y = getreg(RG_LOCY);
96     int target_size = getreg(RG_LOCW);
97     if (target_size == 3) {
98         if (locator_num == 1) {
99             left_plate.y = target_y;
100             left_plate.direction = 1;
101             clock_gettime(CLOCK_REALTIME, &left_plate.time1);
102             left_plate.first_detected = 1;
103         } else if (locator_num == 2 || left_plate.first_detected) {
104             clock_gettime(CLOCK_REALTIME, &left_plate.time2);
105
106             double time_diff = (left_plate.time2.tv_sec - left_plate.time1.tv_sec) +
107                 (left_plate.time2.tv_nsec - left_plate.time1.tv_nsec) / 1000000000.0;
108
109             if (time_diff > 0) {
110                 left_plate.speed = 10.0 / time_diff;
111
112                 double time_to_center = 380.0 / left_plate.speed;
113                 double rocket_flight_time = (570.0 - left_plate.y) / 100.0;
114                 double delay_time = time_to_center - rocket_flight_time;
115
116                 pthread_mutex_lock(&ammo_mutex);
117                 if (delay_time < 0.15 || &ammo < 10) {
118                     struct missile_params* params = (struct missile_params*)malloc(sizeof(struct missile_params));
119                     params->missile_num = &ammo;
120                     params->target_y = left_plate.y;
121                     params->direction = left_plate.direction;
122
123                     pthread_t missile_tid;
124                     pthread_create(&missile_tid, NULL, missile_thread, params);
125                     pthread_detach(missile_tid);
126                 } else {
127                     &ammo++;
128                 }
129                 double* delay_ptr = (double*)malloc(sizeof(double));
130                 *delay_ptr = delay_time;
131
132                 pthread_t rocket_tid;
133                 pthread_create(&rocket_tid, NULL, rocket_thread, delay_ptr);
134                 pthread_detach(rocket_tid);
135             }
136             pthread_mutex_unlock(&ammo_mutex);
137         }
138         left_plate.first_detected = 0;
139     }
140     if (locator_num == 4) {
141         right_plate.y = target_y;
142         right_plate.direction = -1;
143         clock_gettime(CLOCK_REALTIME, &right_plate.time1);
144         right_plate.first_detected = 1;
145     } else if (locator_num == 3 || right_plate.first_detected) {
146         clock_gettime(CLOCK_REALTIME, &right_plate.time2);
147
148         double time_diff = (right_plate.time2.tv_sec - right_plate.time1.tv_sec) +
149             (right_plate.time2.tv_nsec - right_plate.time1.tv_nsec) / 1000000000.0;
150
151         if (time_diff > 0) {
152             right_plate.speed = 10.0 / time_diff;
153
154             double time_to_center = 380.0 / right_plate.speed;
155             double rocket_flight_time = (570.0 - right_plate.y) / 100.0;
156             double delay_time = time_to_center - rocket_flight_time;

```

```

157         }
158         right_plate.first_detected = 0;
159     }
160     if (locator_num == 4) {
161         right_plate.y = target_y;
162         right_plate.direction = -1;
163         clock_gettime(CLOCK_REALTIME, &right_plate.time1);
164         right_plate.first_detected = 1;
165     } else if (locator_num == 3 || right_plate.first_detected) {
166         clock_gettime(CLOCK_REALTIME, &right_plate.time2);
167
168         double time_diff = (right_plate.time2.tv_sec - right_plate.time1.tv_sec) +
169             (right_plate.time2.tv_nsec - right_plate.time1.tv_nsec) / 1000000000.0;
170
171         if (time_diff > 0) {
172             right_plate.speed = 10.0 / time_diff;
173
174             double time_to_center = 380.0 / right_plate.speed;
175             double rocket_flight_time = (570.0 - right_plate.y) / 100.0;
176             double delay_time = time_to_center - rocket_flight_time;
177
178             pthread_mutex_lock(&ammo_mutex);
179             if (delay_time < 0.15 || &ammo < 10) {
180                 struct missile_params* params = (struct missile_params*)malloc(sizeof(struct missile_params));
181                 params->missile_num = &ammo;
182                 params->target_y = right_plate.y;
183                 params->direction = right_plate.direction;
184
185                 pthread_t missile_tid;
186                 pthread_create(&missile_tid, NULL, missile_thread, params);
187                 pthread_detach(missile_tid);
188             } else {
189                 &ammo++;
190             }
191             double* delay_ptr = (double*)malloc(sizeof(double));
192             *delay_ptr = delay_time;
193
194             pthread_t rocket_tid;
195             pthread_create(&rocket_tid, NULL, rocket_thread, delay_ptr);
196             pthread_detach(rocket_tid);
197         }
198         pthread_mutex_unlock(&ammo_mutex);
199     }
200     right_plate.first_detected = 0;
201 }
202 return NULL;
203 }
204
205 void* monitor_thread(void* arg) {
206     while (1) {
207         usleep(1000000);
208     }
209     return NULL;
210 }
211
212 int main()
213 {
214     ammo = (int*)mmap(NULL, sizeof(int), PROT_READ | PROT_WRITE,
215         MAP_SHARED | MAP_ANONYMOUS, -1, 0);
216     &ammo = 0;
217     StartGame(3);
218     ThreadCtl(NTD_TCTL_IO, 0);
219     InterruptAttach(LOC_INTR, loc_handler, NULL, 0, 0);
220     pthread_t monitor;
221     pthread_create(&monitor, NULL, monitor_thread, NULL);
222     usleep(60000000);
223     munmap(ammo, sizeof(int));
224     EndGame();
225     return 0;
226 }

```