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

Институт информатики и вычислительной техники
09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств вычислительной техники и автоматизированных систем"

Кафедра прикладной математики и кибернетики

# Операционные системы реального времени
# Лабораторная работа №2
# Процессы и асинхронное взаимодействие

Выполнил:

студент гр.ИП-213 Дмитриев Антон Александрович
ФИО студента

«__» _____ 2025 г.

Проверил:

Преподаватель

ФИО преподавателя

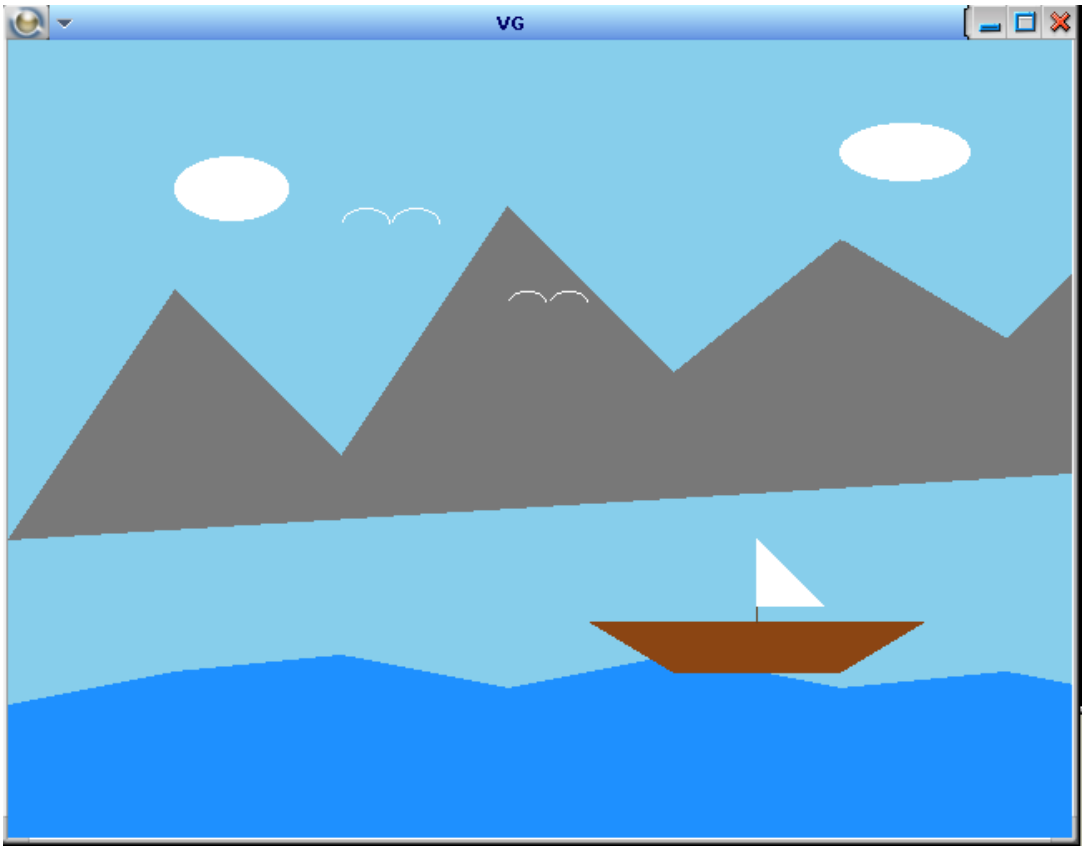
«__» _____ 2025 г. Оценка_____

Новосибирск 2025 г.

1. Используя функции библиотеки VinGraph, нарисовать абстрактную картину, которой представлены (почти) все доступные графические элементы.

```
#include <vingraph.h>
#include <unistd.h>

int main() {
    ConnectGraph();
    Clear(0);
    Fill(0, RGB(135, 206, 235));

    // Горы
    tPoint mountains[] = {{0, 300}, {100, 150}, {200, 250}, {300, 100},
                          {400, 200}, {500, 120}, {600, 180}, {700, 80}, {800,
250}};
    Polygon(mountains, 9, RGB(120, 120, 120));
    Fill(Polygon(mountains, 9, RGB(120, 120, 120), 0), RGB(120, 120, 120));

    // Озеро
    tPoint lake[] = {{0, 400}, {0, 600}, {800, 600}, {800, 450},
                    {700, 400}, {600, 380}, {500, 390}, {400, 370},
                    {300, 390}, {200, 370}, {100, 380}};
    Polygon(lake, 11, RGB(30, 144, 255));
    Fill(Polygon(lake, 11, RGB(30, 144, 255), 0), RGB(30, 144, 255));

    // Лодка
    tPoint boat[] = {{400, 380}, {500, 380}, {550, 350}, {350, 350}};
    Polygon(boat, 4, RGB(139, 69, 19));
    Fill(Polygon(boat, 4, RGB(139, 69, 19), 0), RGB(139, 69, 19));

    // Мачта лодки
    Line(450, 350, 450, 300, RGB(101, 67, 33));

    // Парус
    tPoint sail[] = {{450, 300}, {490, 340}, {450, 340}};
    Polygon(sail, 3, RGB(255, 255, 255));
    Fill(Polygon(sail, 3, RGB(255, 255, 255), 0), RGB(255, 255, 255));

    // Солнце
    int sun = Ellipse(650, 80, 60, 60, RGB(255, 215, 0));
    Fill(sun, RGB(255, 215, 0));

    // Птицы
    Arc(200, 100, 30, 20, 0, 1800, RGB(0, 0, 0));
    Arc(230, 100, 30, 20, 0, 1800, RGB(0, 0, 0));
    Arc(300, 150, 25, 15, 0, 1800, RGB(0, 0, 0));
    Arc(325, 150, 25, 15, 0, 1800, RGB(0, 0, 0));


    // Облака
    int cloud1 = Ellipse(100, 70, 70, 40, RGB(255, 255, 255));
    Fill(cloud1, RGB(255, 255, 255));
    int cloud2 = Ellipse(500, 50, 80, 35, RGB(255, 255, 255));
    Fill(cloud2, RGB(255, 255, 255));

    delay(30000);
```

```
    CloseGraph();
    return 0;
}
```




2. Заставить нарисованные элементы двигаться независимо друг от друга с помощью параллельных процессов (можно изменять во времени положение, цвет, размеры, конфигурацию графических элементов). Предусмотреть завершение программы по нажатию на любую клавишу.

```
#include <vingraph.h>
#include <unistd.h>
#include <pthread.h>
#include <signal.h>
#include <stdlib.h>
#include <time.h>

// Глобальные переменные для управления потоками
volatile int running = 1;
pthread_t threads[6];

// Обработчик сигнала для корректного завершения
void signal_handler(int sig) {
    running = 0;
}

// Поток для движущегося прямоугольника
void* moving_rect_thread(void* arg) {
```

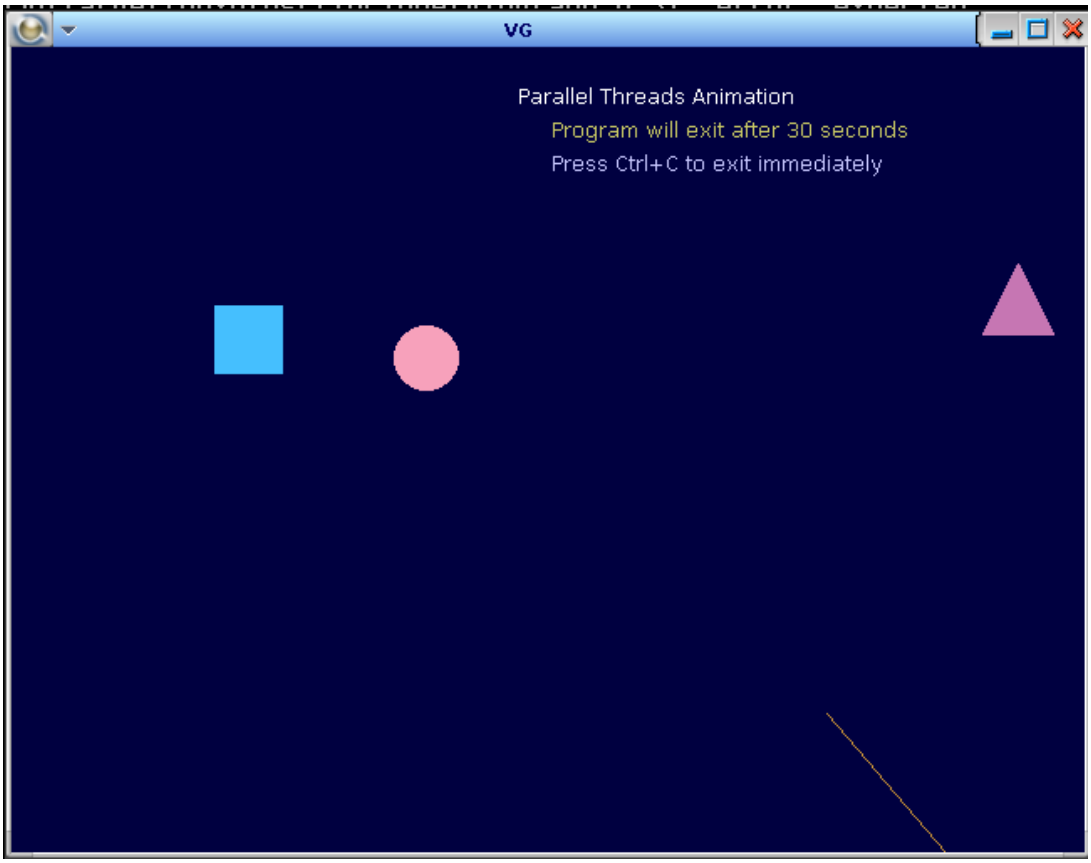
```
    int x = 100, y = 100;
    int dx = 3, dy = 2;
    int color = RGB(255, 0, 0);
    int size = 40;

    int rect = Rect(x, y, size, size, color);
    Fill(rect, color);

    while (running) {
        x += dx;
        y += dy;

        // Отскок от границ экрана
        if (x <= 0 || x >= 800 - size) {
            dx = -dx;
            color = RGB(rand() % 256, rand() % 256, rand() % 256);
            SetColor(rect, color);
            Fill(rect, color);
        }
        if (y <= 0 || y >= 600 - size) {
            dy = -dy;
            color = RGB(rand() % 256, rand() % 256, rand() % 256);
            SetColor(rect, color);
            Fill(rect, color);
        }

        MoveTo(x, y, rect);
        usleep(30000);
    }

    Delete(rect);
    return NULL;
}

// Поток для движущегося круга
void* moving_circle_thread(void* arg) {
    int x = 400, y = 300;
    int dx = -2, dy = 3;
    int color = RGB(0, 255, 0);
    int radius = 30;

    int circle = Ellipse(x, y, radius * 2, radius * 2, color);
    Fill(circle, color);

    while (running) {
        x += dx;
        y += dy;

        // Отскок от границ с изменением размера
        if (x <= radius || x >= 800 - radius) {
            dx = -dx;
            radius = 20 + rand() % 40;
            EnlargeTo(x - radius, y - radius, radius * 2, radius * 2, circle);
```

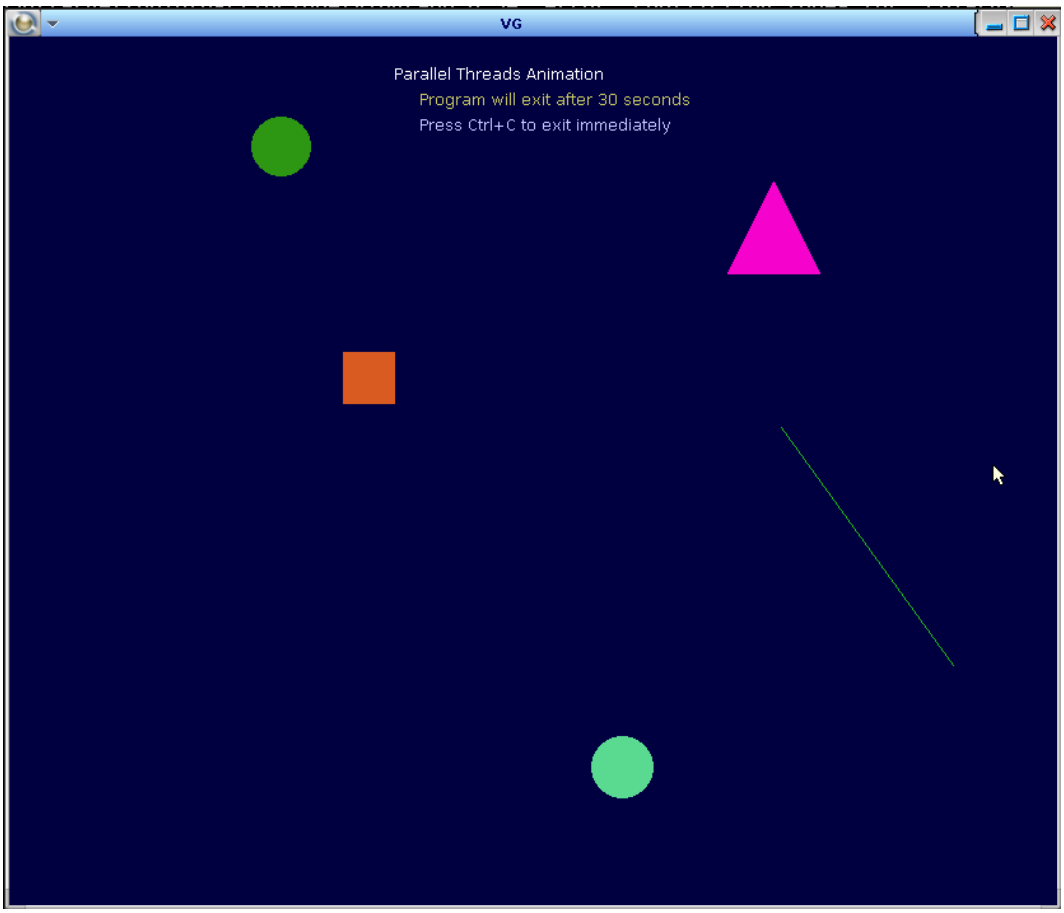
```
        }
        if (y <= radius || y >= 600 - radius) {
            dy = -dy;
            color = RGB(rand() % 256, rand() % 256, rand() % 256);
            SetColor(circle, color);
            Fill(circle, color);
        }

        MoveTo(x - radius, y - radius, circle);
        usleep(40000);
    }

    Delete(circle);
    return NULL;
}

// Поток для пульсирующего треугольника
void* pulsating_triangle_thread(void* arg) {
    int center_x = 600, center_y = 150;
    int size = 30;
    int growth = 1;
    int color = RGB(0, 0, 255);

    while (running) {
        tPoint triangle[] = {
            {center_x, center_y - size},
            {center_x - size, center_y + size},
            {center_x + size, center_y + size}
        };

        int poly = Polygon(triangle, 3, color);
        Fill(poly, color);

        // Пульсация размера
        size += growth;
        if (size > 50 || size < 10) {
            growth = -growth;
            color = RGB(rand() % 256, rand() % 256, rand() % 256);
        }

        usleep(100000);
        Delete(poly);
    }

    return NULL;
}

// Поток для летающей линии
void* flying_line_thread(void* arg) {
    int x1 = 50, y1 = 400;
    int x2 = 150, y2 = 450;
    int dx1 = 4, dy1 = -3;
    int dx2 = 3, dy2 = -4;
```

```
    int color = RGB(255, 255, 0);

    int line = Line(x1, y1, x2, y2, color);

    while (running) {
        x1 += dx1;
        y1 += dy1;
        x2 += dx2;
        y2 += dy2;

        // Отскок от границ
        if (x1 <= 0 || x1 >= 800) dx1 = -dx1;
        if (y1 <= 0 || y1 >= 600) dy1 = -dy1;
        if (x2 <= 0 || x2 >= 800) dx2 = -dx2;
        if (y2 <= 0 || y2 >= 600) dy2 = -dy2;

        // Изменение цвета при столкновении с границей
        if (x1 <= 0 || x1 >= 800 || y1 <= 0 || y1 >= 600 ||
            x2 <= 0 || x2 >= 800 || y2 <= 0 || y2 >= 600) {
            color = RGB(rand() % 256, rand() % 256, rand() % 256);
            SetColor(line, color);
        }

        // Перерисовываем линию с новыми координатами
        Delete(line);
        line = Line(x1, y1, x2, y2, color);

        usleep(50000);
    }

    Delete(line);
    return NULL;
}

// Поток для прыгающего шара
void* bouncing_ball_thread(void* arg) {
    int x = 300, y = 80;
    int dx = 2, dy = 4;
    int color = RGB(255, 165, 0);
    int radius = 25;

    int ball = Ellipse(x, y, radius * 2, radius * 2, color);
    Fill(ball, color);

    while (running) {
        x += dx;
        y += dy;

        // Гравитация
        if (y < 550) {
            dy += 1;  // Ускорение падения
        }
```

```
        // Отскок от границ
        if (x <= radius || x >= 800 - radius) {
            dx = -dx;
            color = RGB(rand() % 256, rand() % 256, rand() % 256);
            SetColor(ball, color);
            Fill(ball, color);
        }
        if (y <= radius) {
            dy = -dy;
        }
        if (y >= 600 - radius) {
            dy = -dy * 0.8;  // Потеря энергии при ударе о землю
            y = 600 - radius;
            color = RGB(rand() % 256, rand() % 256, rand() % 256);
            SetColor(ball, color);
            Fill(ball, color);
        }

        MoveTo(x - radius, y - radius, ball);
        usleep(30000);
    }

    Delete(ball);
    return NULL;
}

// завершение через 30 секунд
void* timer_thread(void* arg) {
    sleep(30);
    running = 0;
    return NULL;
}

int main() {
    // Устанавливаем обработчик сигналов
    signal(SIGINT, signal_handler);

    // Подключаемся к графической системе
    ConnectGraph();
    Clear(0);
    Fill(0, RGB(0, 0, 64));  // Темно-синий фон

    // Инициализация случайных чисел
    srand(time(NULL));

    Text(300, 20, "Parallel Threads Animation", RGB(255, 255, 255));
    Text(320, 40, "Program will exit after 30 seconds", RGB(200, 200, 100));
    Text(320, 60, "Press Ctrl+C to exit immediately", RGB(200, 200, 255));

    // Создаем потоки для каждого движущегося элемента
    pthread_create(&threads[0], NULL, moving_rect_thread, NULL);
    pthread_create(&threads[1], NULL, moving_circle_thread, NULL);
    pthread_create(&threads[2], NULL, pulsating_triangle_thread, NULL);
```

```
    pthread_create(&threads[3], NULL, flying_line_thread, NULL);
    pthread_create(&threads[4], NULL, bouncing_ball_thread, NULL);
    pthread_create(&threads[5], NULL, timer_thread, NULL);

    // Главный поток ждет завершения всех потоков
    for (int i = 0; i < 6; i++) {
        pthread_join(threads[i], NULL);
    }

    Clear(0);
    Text(350, 300, "Animation finished", RGB(255, 255, 255));
    sleep(1);

    CloseGraph();
    return 0;
}
```






3. Нарисовать нечто, движущееся по замкнутой кривой. Организовать изменение траектории движения по нажатию на клавиши (организуя взаимодействие процессов через общую область памяти (shared memory)). В качестве фона можно использовать (оживленную) картину, созданную на предыдущих этапах работы.

```
#include <vingraph.h>
#include <unistd.h>
#include <sys/mman.h>
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <sys/wait.h>

// Структура для разделяемой памяти
typedef struct {
    double a;
    double b;
    double scale;
    int curve_type;  // 0: Улитка Паскаля, 1: Эллипс, 2: Лемниската, 3: Кардиоида
    int running;
    int key_pressed;
} shared_data_t;

shared_data_t *shared;
```

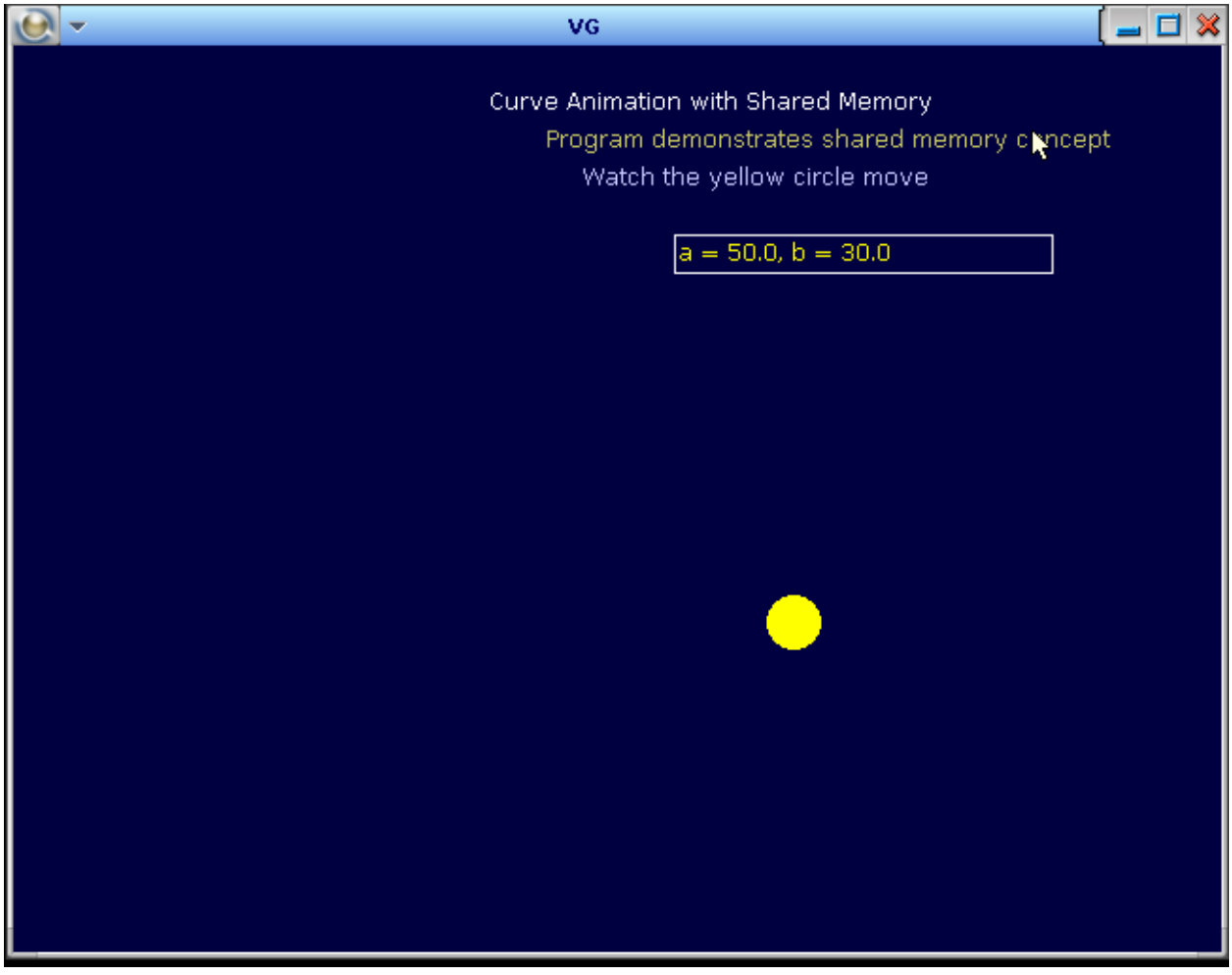
```
// Обработчик сигнала для корректного завершения
void signal_handler(int sig) {
    shared->running = 0;
}

// Функция для получения координат в зависимости от типа кривой
void get_curve_point(int curve_type, double phi, double a, double b, double scale,
                     double *x, double *y) {
    double rho;

    switch (curve_type) {
        case 0: // Улитка Паскаля
            rho = a * cos(phi) + b;
            *x = rho * cos(phi) * scale + 400;
            *y = rho * sin(phi) * scale + 300;
            break;

        case 1: // Эллипс
            *x = a * scale * cos(phi) + 400;
            *y = b * scale * sin(phi) + 300;
            break;

        case 2: // Лемниската Бернулли
            rho = a * sqrt(cos(2 * phi));
            *x = rho * cos(phi) * scale + 400;
            *y = rho * sin(phi) * scale + 300;
            break;

        case 3: // Кардиоида
            rho = a * (1 + cos(phi));
            *x = rho * cos(phi) * scale + 400;
            *y = rho * sin(phi) * scale + 300;
            break;

        default:
            *x = 400;
            *y = 300;
    }
}

// Процесс-обработчик клавиш
void key_handler_process() {
    printf("Key controls:\n");
    printf("1 - Улитка Паскаля\n");
    printf("2 - Эллипс\n");
    printf("3 - Лемниската\n");
    printf("4 - Кардиоида\n");
    printf("+ - Увеличить масштаб\n");
    printf("- - Уменьшить масштаб\n");
    printf("a/A - Изменить параметр a\n");
    printf("b/B - Изменить параметр b\n");
    printf("q - Выход\n");
```

```
while (shared->running) {
    if (kbhit()) {
        char key = getch();
        shared->key_pressed = 1;

        switch (key) {
            case '1':
                shared->curve_type = 0;
                printf("Траектория: Улитка Паскаля\n");
                break;
            case '2':
                shared->curve_type = 1;
                printf("Траектория: Эллипс\n");
                break;
            case '3':
                shared->curve_type = 2;
                printf("Траектория: Лемниската Бернулли\n");
                break;
            case '4':
                shared->curve_type = 3;
                printf("Траектория: Кардиоида\n");
                break;
            case '+':
                shared->scale *= 1.1;
                printf("Масштаб: %.1f\n", shared->scale);
                break;
            case '-':
                shared->scale /= 1.1;
                printf("Масштаб: %.1f\n", shared->scale);
                break;
            case 'a':
                shared->a += 5;
                printf("Параметр a: %.1f\n", shared->a);
                break;
            case 'A':
                shared->a -= 5;
                if (shared->a < 1) shared->a = 1;
                printf("Параметр a: %.1f\n", shared->a);
                break;
            case 'b':
                shared->b += 5;
                printf("Параметр b: %.1f\n", shared->b);
                break;
            case 'B':
                shared->b -= 5;
                if (shared->b < 1) shared->b = 1;
                printf("Параметр b: %.1f\n", shared->b);
                break;
            case 'q':
                shared->running = 0;
                break;
        }
```

```
            usleep(100000); // Задержка для избежания множественных срабатываний
        }
        usleep(50000); // Небольшая задержка
    }
}


int main() {
    // Создание разделяемой памяти
    shared = (shared_data_t*)mmap(NULL, sizeof(shared_data_t),
                                  PROT_READ | PROT_WRITE,
                                  MAP_SHARED | MAP_ANON, -1, 0);

    if (shared == MAP_FAILED) {
        printf("Ошибка создания разделяемой памяти\n");
        exit(1);
    }

    // Инициализация раздельной памяти
    shared->a = 50.0;
    shared->b = 30.0;
    shared->scale = 2.0;
    shared->curve_type = 0;
    shared->running = 1;
    shared->key_pressed = 0;

    // обработчик сигналов
    signal(SIGINT, signal_handler);

    // дочерний процесс для обработки клавиш
    pid_t pid = fork();

    if (pid == -1) {
        printf("Ошибка создания процесса\n");
        munmap(shared, sizeof(shared_data_t));
        exit(1);
    }

    if (pid == 0) {
        // Дочерний процесс - обработчик клавиш
        key_handler_process();
        munmap(shared, sizeof(shared_data_t));
        exit(0);
    }

    // Родительский процесс - графика
    ConnectGraph();
    Clear(0);
    Fill(0, RGB(0, 0, 64));

    Text(200, 20, "Curve Animation with Shared Memory and Process Communication",
RGB(255, 255, 255));
    Text(280, 40, "Use keys 1-4 to change curve type, +/- for scale, a/A b/B for
parameters", RGB(200, 200, 100));
```

```
    Text(320, 60, "Press 'q' in terminal to exit", RGB(200, 200, 255));

    // объект для движения (желтый круг)
    int obj = Ellipse(0, 0, 20, 20, RGB(255, 255, 0));
    Fill(obj, RGB(255, 255, 0));

    // траекторию
    int prev_x = 0, prev_y = 0;
    int first_point = 1;

    double phi = 0.0;
    time_t start_time = time(NULL);
    char info[100];
    char curve_names[4][20] = {
        "Pascal's Snail", "Ellipse", "Lemniscate", "Cardioid"
    };

    // Главный цикл анимации
    while (shared->running) {
        // координаты в зависимости от типа кривой
        double x, y;
        get_curve_point(shared->curve_type, phi, shared->a, shared->b, shared-
>scale, &x, &y);


        if (!first_point) {
            Line(prev_x, prev_y, (int)x, (int)y, RGB(100, 100, 100));
        } else {
            first_point = 0;
        }
        prev_x = (int)x;
        prev_y = (int)y;

        // Перемещаем объект
        MoveTo((int)x - 10, (int)y - 10, obj);

        // Увеличиваем угол
        phi += 0.02;
        if (phi > 4 * M_PI) phi -= 4 * M_PI;

        // отображение параметров
        sprintf(info, "Curve: %s | a=%.1f, b=%.1f, scale=%.1f",
                curve_names[shared->curve_type], shared->a, shared->b, shared-
>scale);

        SetColor(0, RGB(0, 0, 64));
        Fill(Rect(250, 80, 500, 20, 0), RGB(0, 0, 64));
        Text(250, 80, info, RGB(255, 255, 0));

        // Отображаем время работы
        int elapsed = (int)(time(NULL) - start_time);
        sprintf(info, "Time: %d sec", elapsed);
        SetColor(0, RGB(0, 0, 64));
```

```
        Fill(Rect(700, 560, 100, 20, 0), RGB(0, 0, 64));
        Text(700, 560, info, RGB(200, 200, 255));

        delay(30);
    }

    // завершение программу
    shared->running = 0;

    waitpid(pid, NULL, 0);

    Delete(obj);
    Clear(0);
    Text(350, 300, "Animation finished", RGB(255, 255, 255));
    delay(1000);

    // Освобождение разделяемой памяти
    munmap(shared, sizeof(shared_data_t));

    CloseGraph();
    return 0;
}
```



4. Затем последнюю программу сделать с помощью нитей в одном процессе.

```
#include <vingraph.h>
#include <unistd.h>
#include <pthread.h>
#include <signal.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

// Структура для общих данных между потоками
typedef struct {
    double a;
    double b;
    double scale;
    int curve_type;  // 0: Улитка Паскаля, 1: Эллипс, 2: Лемниската, 3: Кардиоида
    int running;
    int key_pressed;
    pthread_mutex_t mutex;
} shared_data_t;

shared_data_t shared;

// Обработчик сигнала для корректного завершения
void signal_handler(int sig) {
    pthread_mutex_lock(&shared.mutex);
    shared.running = 0;
    pthread_mutex_unlock(&shared.mutex);
}

// Функция для получения координат в зависимости от типа кривой
void get_curve_point(int curve_type, double phi, double a, double b, double scale,
                     double *x, double *y) {
    double rho;

    switch (curve_type) {
        case 0: // Улитка Паскаля
            rho = a * cos(phi) + b;
            *x = rho * cos(phi) * scale + 400;
            *y = rho * sin(phi) * scale + 300;
            break;

        case 1: // Эллипс
            *x = a * scale * cos(phi) + 400;
            *y = b * scale * sin(phi) + 300;
            break;

        case 2: // Лемниската Бернулли
            rho = a * sqrt(fabs(cos(2 * phi))); // fabs для избежания NaN
            *x = rho * cos(phi) * scale + 400;
            *y = rho * sin(phi) * scale + 300;
            break;
```

```
        case 3: // Кардиоида
            rho = a * (1 + cos(phi));
            *x = rho * cos(phi) * scale + 400;
            *y = rho * sin(phi) * scale + 300;
            break;

        default:
            *x = 400;
            *y = 300;
    }
}

// Функция для обработки клавиш в отдельном потоке
void* key_handler_thread(void* arg) {
    printf("Key controls (in terminal):\n");
    printf("1 - Pascal's Snail\n");
    printf("2 - Ellipse\n");
    printf("3 - Lemniscate\n");
    printf("4 - Cardioid\n");
    printf("+ - Increase scale\n");
    printf("- - Decrease scale\n");
    printf("a - Increase parameter a\n");
    printf("A - Decrease parameter a\n");
    printf("b - Increase parameter b\n");
    printf("B - Decrease parameter b\n");
    printf("q - Quit\n");

    int running = 1;
    while (running) {
        char key = getchar();

        pthread_mutex_lock(&shared.mutex);

        switch (key) {
            case '1':
                shared.curve_type = 0;
                shared.key_pressed = 1;
                printf("Trajectory: Pascal's Snail\n");
                break;
            case '2':
                shared.curve_type = 1;
                shared.key_pressed = 1;
                printf("Trajectory: Ellipse\n");
                break;
            case '3':
                shared.curve_type = 2;
                shared.key_pressed = 1;
                printf("Trajectory: Lemniscate\n");
                break;
            case '4':
                shared.curve_type = 3;
                shared.key_pressed = 1;
```

```
            printf("Trajectory: Cardioid\n");
            break;
        case '+':
            shared.scale *= 1.1;
            shared.key_pressed = 1;
            printf("Scale: %.1f\n", shared.scale);
            break;
        case '-':
            shared.scale /= 1.1;
            if (shared.scale < 0.5) shared.scale = 0.5;
            shared.key_pressed = 1;
            printf("Scale: %.1f\n", shared.scale);
            break;
        case 'a':
            shared.a += 5;
            shared.key_pressed = 1;
            printf("Parameter a: %.1f\n", shared.a);
            break;
        case 'A':
            shared.a -= 5;
            if (shared.a < 1) shared.a = 1;
            shared.key_pressed = 1;
            printf("Parameter a: %.1f\n", shared.a);
            break;
        case 'b':
            shared.b += 5;
            shared.key_pressed = 1;
            printf("Parameter b: %.1f\n", shared.b);
            break;
        case 'B':
            shared.b -= 5;
            if (shared.b < 1) shared.b = 1;
            shared.key_pressed = 1;
            printf("Parameter b: %.1f\n", shared.b);
            break;
        case 'q':
            shared.running = 0;
            running = 0;
            printf("Quitting...\n");
            break;
        case '\n':
            break;
        default:
            printf("Unknown key: %c\n", key);
    }

    running = shared.running;
    pthread_mutex_unlock(&shared.mutex);

    usleep(10000);
}

return NULL;
```

```
}

// Функция для отрисовки информации
void draw_info(int elapsed, const char* curve_name) {
    char info[100];


    sprintf(info, "Curve: %s | a=%.1f, b=%.1f, scale=%.1f",
            curve_name, shared.a, shared.b, shared.scale);

    SetColor(0, RGB(0, 0, 64));
    Fill(Rect(200, 80, 500, 20, 0), RGB(0, 0, 64));
    Text(200, 80, info, RGB(255, 255, 0));


    // Инструкции на экране
    SetColor(0, RGB(0, 0, 64));
    Fill(Rect(250, 550, 400, 40, 0), RGB(0, 0, 64));
    Text(250, 550, "Switch to terminal and press keys: 1-4, +/-, a/A, b/B, q",
RGB(200, 200, 100));
}

int main() {
    // Инициализируем общие данные
    shared.a = 50.0;
    shared.b = 30.0;
    shared.scale = 2.0;
    shared.curve_type = 0;
    shared.running = 1;
    shared.key_pressed = 0;

    // Инициализируем мьютекс
    if (pthread_mutex_init(&shared.mutex, NULL) != 0) {
        printf("Mutex initialization failed\n");
        return 1;
    }

    // Устанавливаем обработчик сигналов
    signal(SIGINT, signal_handler);

    // Подключаемся к графической системе
    ConnectGraph();
    Clear(0);
    Fill(0, RGB(0, 0, 64));  // Темно-синий фон

    // Заголовок
    Text(250, 20, "Curve Animation with Threads (Pthreads)", RGB(255, 255, 255));
    Text(300, 40, "Multi-threaded application", RGB(200, 200, 100));

    // Создаем поток для обработки клавиш
    pthread_t key_thread;
    if (pthread_create(&key_thread, NULL, key_handler_thread, NULL) != 0) {
        printf("Thread creation failed\n");
```

```
    CloseGraph();
    return 1;
}

// Создаем объект для движения (желтый круг)
int obj = Ellipse(0, 0, 20, 20, RGB(255, 255, 0));
Fill(obj, RGB(255, 255, 0));

// Переменные для анимации
double phi = 0.0;
time_t start_time = time(NULL);
int prev_x = 0, prev_y = 0;
int first_point = 1;

// Массив имен кривых
char curve_names[4][20] = {
    "Pascal's Snail", "Ellipse", "Lemniscate", "Cardioid"
};

// Главный цикл анимации (графический поток)
while (1) {
    pthread_mutex_lock(&shared.mutex);
    if (!shared.running) {
        pthread_mutex_unlock(&shared.mutex);
        break;
    }

    // Получаем текущие параметры
    double a = shared.a;
    double b = shared.b;
    double scale = shared.scale;
    int curve_type = shared.curve_type;
    pthread_mutex_unlock(&shared.mutex);

    // Вычисляем координаты в зависимости от типа кривой
    double x, y;
    get_curve_point(curve_type, phi, a, b, scale, &x, &y);

    // Рисуем траекторию (тонкая серая линия)
    if (!first_point) {
        Line(prev_x, prev_y, (int)x, (int)y, RGB(100, 100, 100));
    } else {
        first_point = 0;
    }
    prev_x = (int)x;
    prev_y = (int)y;

    // Перемещаем объект
    MoveTo((int)x - 10, (int)y - 10, obj);

    // Увеличиваем угол
    phi += 0.02;
    if (phi > 4 * M_PI) phi -= 4 * M_PI;
```

```
        // Обновляем информацию на экране
        int elapsed = (int)(time(NULL) - start_time);
        draw_info(elapsed, curve_names[curve_type]);

        delay(30);
    }

    // Завершаем программу
    printf("Waiting for thread to finish...\n");

    // Ждем завершения потока обработки клавиш
    pthread_join(key_thread, NULL);

    // Уничтожаем мьютекс
    pthread_mutex_destroy(&shared.mutex);

    // Очищаем экран
    Delete(obj);
    Clear(0);
    Text(350, 300, "Animation finished", RGB(255, 255, 255));
    delay(1000);

    CloseGraph();
    return 0;
}
```