Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики» (СибГУТИ)

Лабораторная работа №2

«Процессы и асинхронное взаимодействие»

Выполнили: студенты 4 курса

ИВТ, гр. ИП-111

Маландий И.И.

Толкач И.А.

Проверил:

профессор кафедры ПМиК

Фионов Андрей Николаевич

Новосибирск, 2024 г.

Оглавление

Задание на лабораторную работу

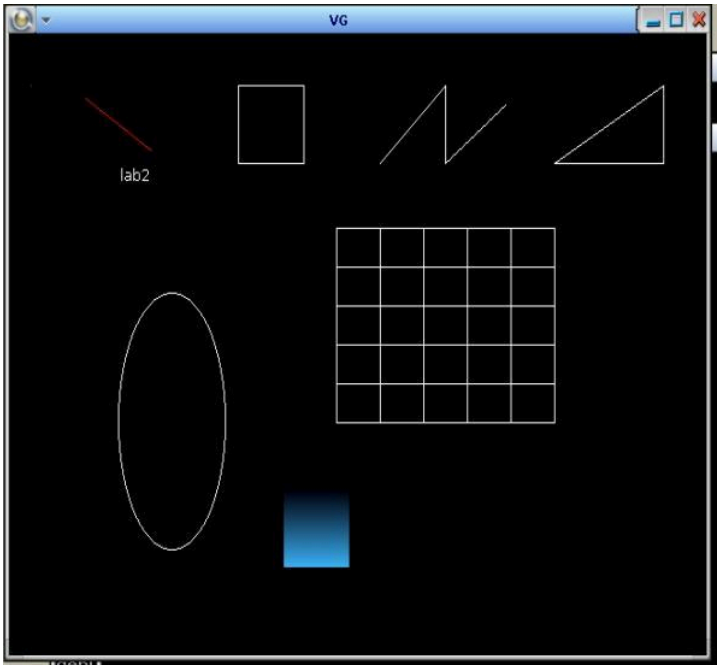
1. Тщательно изучить библиотеку VinGraph.
2. Используя функции библиотеки VinGraph, нарисовать абстрактную картину, которой представлены (почти) все доступные графические элементы.
3. Заставить нарисованные элементы двигаться независимо друг от друга с помощью параллельных процессов (можно изменять во времени положение, цвет, размеры, конфигурацию графических элементов). Предусмотреть завершение программы по нажатию на любую клавишу.
4. 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.
5. Затем последнюю программу сделать с помощью нитей в одном процессе.

Выполнение

1) Изучена библиотека VinGraph на основе справочного материала с эиоса. Ознакомлены с основными фигурами в VinGraph и функциями.

2) Реализована программа, выводящая в терминал VinGraph почти все доступные графические элементы.

Выполнение программы:



Код программы:

```
#include <stdlib.h>
#include <unistd.h>
#include <vingraph.h>
#include <iostream>
#include <math.h>

int main(){
  ConnectGraph();
  int c = RGB(255,0,0);
  Text(100,100, "lab2");
  Pixel(20,40,c);
  Line(70,50,130,90,c);
  Rect(210,40,60,60);
  tPoint p1 [] = { {340,100}, {400,40} , {400, 100}, {455,55} };
  Polyline(p1,4);
  tPoint p2 [] = { {500,100}, {600,40} , {600, 100}};
  Polygon (p2,3);
  Ellipse(100,200,100,200);
  //Arc(300,250,300,250,0,1800);
  Grid(300,150,200,150,5,5);
  int *im_buf = (int*) malloc (60*60*4);
  for (int i = 0, c = 10; i < 60;i++, c += 0x010304)
    for(int j = 0;j < 60;j++) im_buf [60*i + j] = c;
  Image32(250,350,60,60,im_buf);
  CloseGraph();
  return 0;
}
```

Объяснение кода программы:

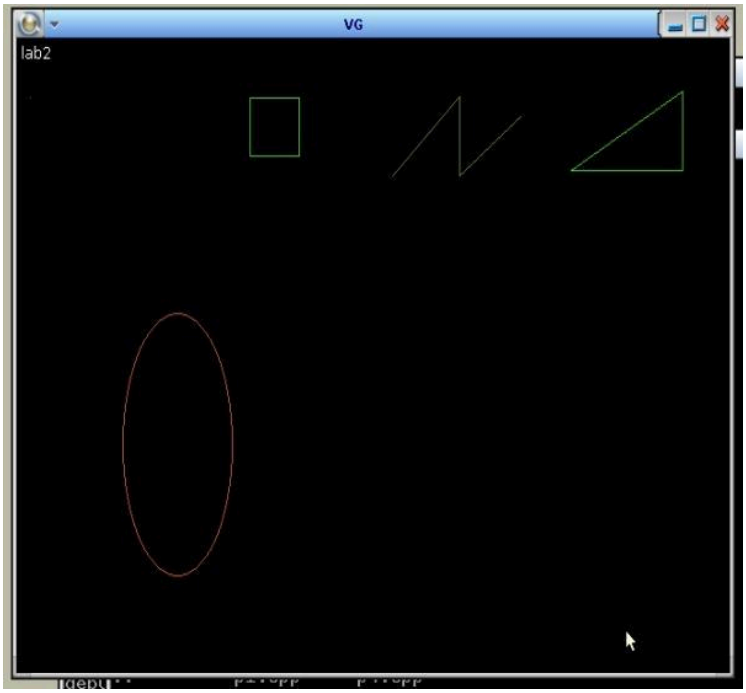
В начале происходит инициализация графического режима с помощью ConnectGraph(), который включает графическую среду. Затем устанавливается красный цвет через функцию RGB, и он используется в нескольких элементах рисунка, включая линию (Line) и пиксель (Pixel), что располагаются в заданных координатах. Далее выводится текст в позицию (100, 100) для метки задания.

Создается несколько различных геометрических фигур через функции библиотеки. Для отрисовки прямоугольника вызвана функция Rect с заданием определенной позиции, ширины и высоты, а сложные фигуры создаются с использованием массивов точек. Полилиния и многоугольник построены на основе массивов координат p1 и p2. Элипс создается через отрисовку закругленных фигур с параметрами радиуса. Команда Grid выводит сетку в указанных координатах и размерах ячеек.

Последний графический элемент – изображение, созданное через буфер im_buf, который выделяется с помощью malloc и заполняется значениями цвета. Вложенный цикл итерирует по строкам и столбцам, тем самым изменяя каждый пиксель с помощью инкрементации, создавая градиент с голубым оттенком. Стоит отметить, что код построен линейно, функции VinGraph выводятся последовательно.

3) 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.

Результат работы программы:



Код программы:

```
int main() {
  ConnectGraph();
  int c = RGB(255, 0, 0);

  // ########## ##########
  Text(2, 2, "lab2");
  int pixel = Pixel(20, 40, c);
  int rect = Rect(210, 40, 50, 50);
  tPoint p1[] = {{340, 100}, {400, 40}, {400, 100}, {455, 55}};
  int polyline = Polyline(p1, 4);
  tPoint p2[] = {{500, 100}, {600, 40}, {600, 100}};
  int polygon = Polygon(p2, 3);
  int ellipse = Ellipse(100, 200, 100, 200);

  pid_t pids[5];
  pids[0] = fork();

  if (pids[0] == 0) {
    srand(time(0));
    while (true) {
      int c = RGB(rand() % 255, rand() % 255, rand() % 255);
      int x = (rand() % 3) - 1;
      int y = (rand() % 3) - 1;
      Move(pixel, x, y);
      SetColor(pixel, c);
      usleep(100000);
    }
    exit(0);
  } else if (pids[0] > 0) {
    pids[1] = fork();
    if (pids[1] == 0) {
      srand(time(0));
      while (true) {
        int c = RGB(rand() % 255, rand() % 255, rand() % 255);
        int x = (rand() % 3) - 1;
        int y = (rand() % 3) - 1;
        Move(ellipse, x, y);
        SetColor(ellipse, c);
        usleep(200000);
      }
      exit(0);
    } else if (pids[1] > 0) {
      pids[2] = fork();
      if (pids[2] == 0) {
        srand(time(0));
        while (true) {
          int c = RGB(rand() % 255, rand() % 255, rand() % 255);
          int lr = (rand() % 20) - 10;
          int x = (rand() % 3) - 1;
          int y = (rand() % 3) - 1;
          Move(rect, x, y);
          SetColor(rect, c);
          Enlarge(rect, lr, lr);
          x = (rand() % 3) - 1;
          y = (rand() % 3) - 1;
          Move(polygon, x, y);
          Move(polygon, x, y);
          SetColor(polygon, c);
          usleep(300000);
        }
        exit(0);
      } else if (pids[2] > 0) {
        pids[3] = fork();
        if (pids[3] == 0) {
          srand(time(0));
          while (true) {
            int x = (rand() % 3) - 1;
            int y = (rand() % 3) - 1;
            x = (rand() % 3) - 1;
            y = (rand() % 3) - 1;
            Move(polygon, x, y);
            usleep(400000);
          }
          exit(0);
        } else if (pids[3] > 0) {
          srand(time(0));
          while (true) {
            int c = RGB(rand() % 255, rand() % 255, rand() % 255);
            int x = (rand() % 3) - 1;
            int y = (rand() % 3) - 1;
            Move(polyline, x, y);
            SetColor(polyline, c);
            usleep(500000);
          }
          exit(0);
        }
      }
    }
  }

  for (int i = 0; i < 5; i++) {
    wait(NULL);
  }

  CloseGraph();
  return 0;
}
```

Объяснение кода программы:

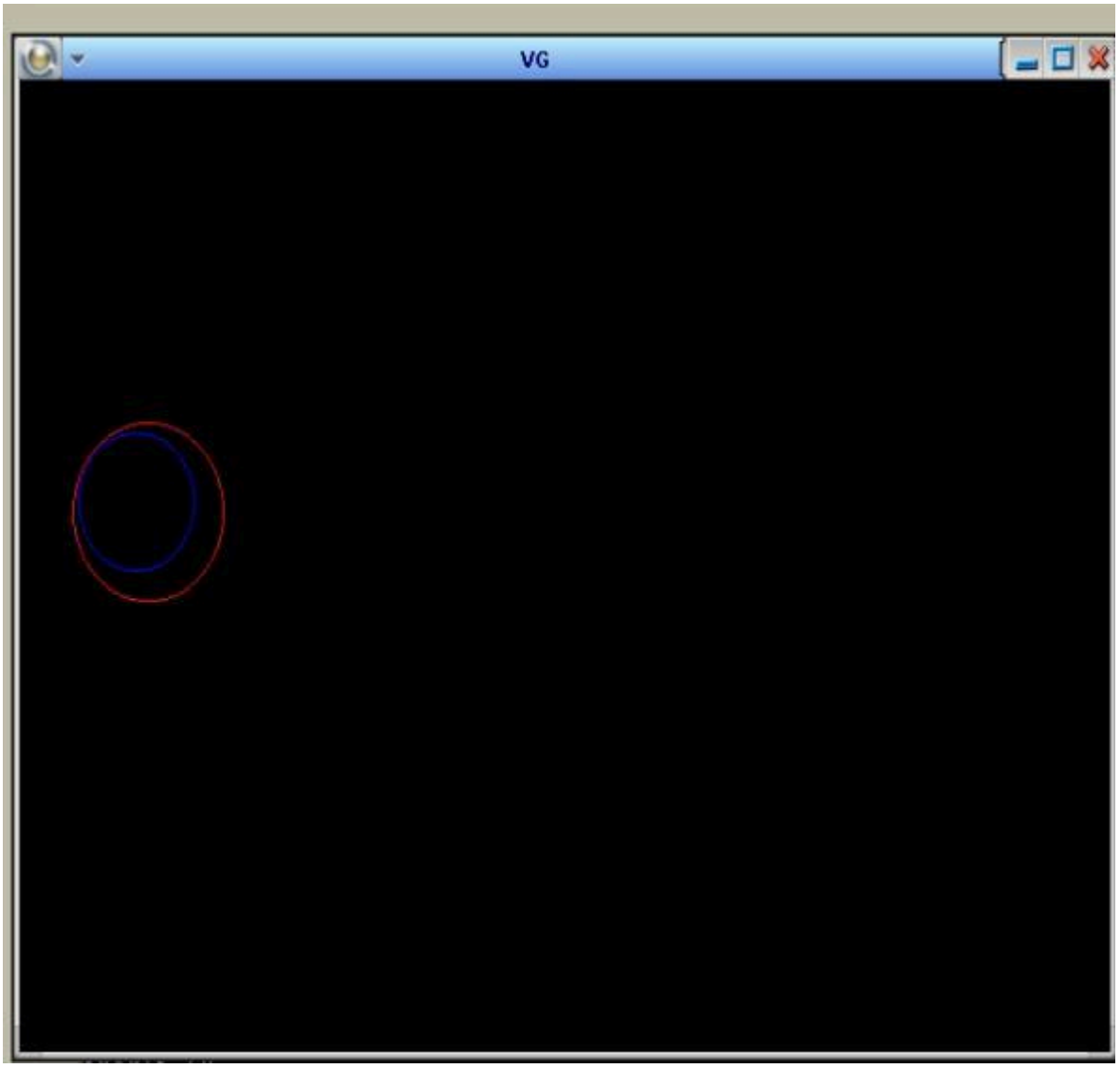
Здесь реализовано независимое движение графических элементов с помощью порождения параллельных процессов с использованием системных вызовов fork. В начале программа подключается к графической системе, задает цвет, а затем создает несколько графических элементов. Каждому из этих элементов назначены координаты, и они будут изменять положение, цвет и размер в ходе выполнения программы. Далее создаются пять процессов для управления анимацией.

Каждый дочерний процесс управляет одним из графических элементов. Внутри цикла while каждый процесс случайным образом изменяет параметры своего элемента: цвет через SetColor, позицию через Move, а для прямоугольника также применяется изменение размера через Enlarge. Каждый из процессов работает с уникальной задержкой (usleep) для создания асинхронного движения. Использование srand(time(0)) позволяет генерировать псевдослучайные значения для обеспечения изменения положения и цвета графических объектов.

В данном случае код написан таким образом, чтобы каждый элемент изменял свои параметры независимо, тем самым создавая динамическую картину с разными визуальными эффектами.

4) 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.

Результат работы программы:


VG

Код программы:

```
int main() {
  ConnectGraph();
  pid_t proc;
  int c = RGB(255, 0, 0);
  int cc = RGB(0, 0, 255);
  int pic = Picture(100, 100);
  int ellipse1 = Ellipse(10, 50, 70, 70,cc, pic);
  int ellipse2 = Ellipse(7, 45, 90, 90, c, pic);
  int x, y, key=0;
  float phi = 0, rho = 0;
  float *a = static_cast<float*>(mmap(0, sizeof* a, PROT_READ + PROT_WRITE, MAP_SHARED + MAP_ANON, -1, 0));
  float *b = static_cast<float*>(mmap(0, sizeof* a, PROT_READ + PROT_WRITE, MAP_SHARED + MAP_ANON, -1, 0));
  *a = 100;
  *b = 100;
  if (proc = fork()) {
    while (key != 13) {
        key = InputChar();
        printf("%d\n", key);
        if (key == 'w')
            *a = *a + 10;
        if (key == 'a')
            *b = *b + 10;
        if (key == 's')
            *a = *a - 10;
        if (key == 'd')
            *b = *b - 10;
    }
    CloseGraph();
  }else{
  while(true){
    rho = *a * cos(phi) + *b;
    x = rho * cos(phi) + 20;
    y = rho * sin(phi) + 20;
    phi += 0.01;
    MoveTo(x,y,pic);
    delay(15);
    }
  }
  CloseGraph();
  munmap(a, sizeof(float));
  munmap(b, sizeof(float));
  return 0;
}
```

Объяснение кода программы:

Здесь реализуется движение графического элемента по замкнутой кривой с возможностью изменения траектории с помощью клавиш. В начале запускается графический режим и создается изображение, включающее два эллипса с разными размерами и цветами. Координаты `x` и `y` зависят от значений переменных `a` и `b`, которые отвечают за изменение траектории и реализованы в общей памяти с помощью `mmap`, что позволяет обменивать данные между процессами.

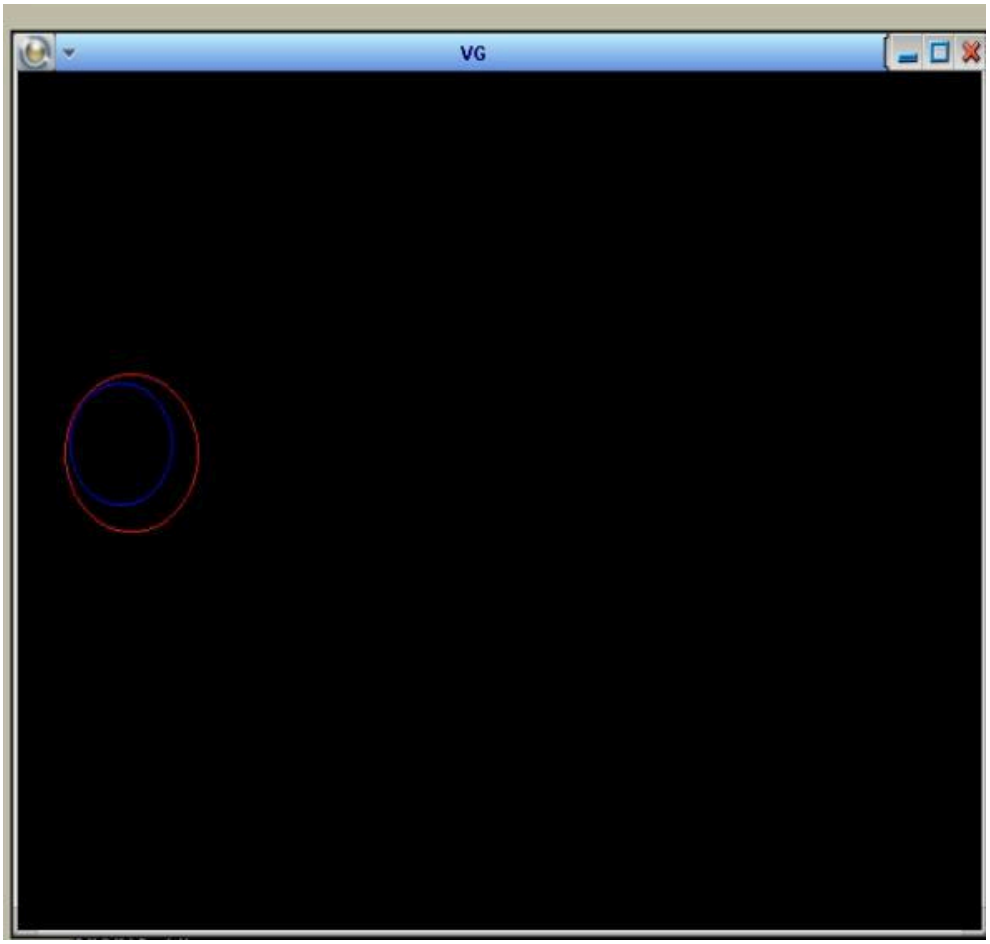
Программа создает два параллельных процесса с помощью `fork`. Родительский процесс принимает ввод от пользователя через `InputChar()` и изменяет значения переменных `a` и `b` в зависимости от нажатых клавиш: 'w', 'a', 's', 'd', которые отвечают за изменение положения по вертикали и горизонтали. Эти изменения передаются в дочерний процесс через общую память, благодаря чему удается добиться управления траекторией в режиме

реального времени. Когда пользователь нажимает Enter (код 13), родительский процесс завершает работу и графическую сессию.

Дочерний процесс вычисляет новые координаты `x` и `y` на основе значений `a`, `b` и текущего угла `phi` для реализации движения по кривой. Эти координаты используются функцией `MoveTo` для перемещения изображения по экрану в замкнутой петле. Задержка `delay(15)` контролирует скорость анимации для стабильного и плавного движения.

5) Затем последнюю программу сделать с помощью нитей в одном процессе.

Результат работы программы:



Код программы:

```
using namespace std;
float *a = static_cast<float*>(mmap(0, sizeof* a, PROT_READ + PROT_WRITE, MAP_SHARED + MAP_ANON, -1, 0));
float *b = static_cast<float*>(mmap(0, sizeof* a, PROT_READ + PROT_WRITE, MAP_SHARED + MAP_ANON, -1, 0));
int x = 0, y = 0 , pic;
float phi = 0, rho = 0;

Codeium: Refactor | Explain | Generate Function Comment | ×
void *pic_move_upr(void *args){
  int key = 0;
  while (key != 13) {
        key = InputChar();
        printf("%d\n", key);
        if (key == 'w')
            *a = *a + 10;
        if (key == 'a')
            *b = *b + 10;
        if (key == 's')
            *a = *a - 10;
        if (key == 'd')
            *b = *b - 10;
    }
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void *pic_move_bot(void *args){
    while(true){
     rho = *a * cos(phi) + *b;
     x = rho * cos(phi) + 20;
     y = rho * sin(phi) + 20;
     phi += 0.01;
     MoveTo(x,y,pic);
     delay(15);
     }
}

Codeium: Refactor | Explain | Generate Function Comment | ×
int main() {
  ConnectGraph();
  pid_t proc;
  int c = RGB(255, 0, 0);
  int cc = RGB(0, 0, 255);
  pic = Picture(100, 100);
  int ellipse1 = Ellipse(10, 50, 70, 70,cc, pic);
  int ellipse2 = Ellipse(7, 45, 90, 90, c, pic);
  *a = 100;
  *b = 100;
  pthread_t upr, bot;
  pthread_create(&upr,0,pic_move_upr,NULL);
  pthread_create(&bot,0,pic_move_bot,NULL);
  if(!pthread_join(upr,0))
    pthread_cancel(bot);
  CloseGraph();
  munmap(a, sizeof(float));
  munmap(b, sizeof(float));
  return 0;
}
```

Объяснение кода программы:

В данном случае вместо процессов используются нити (`pthread`), чтобы управлять движением графического элемента в одном процессе. В начале инициализируется графический режим и создается изображение, содержащее два эллипса с разными размерами и цветами, объединенные в переменную `pic`. Переменные `a` и `b` выделены в общей памяти, чтобы обе нити могли взаимодействовать, изменяя параметры движения изображения.

Эти переменные задают параметры траектории, и их значения могут быть изменены при вводе пользователем.

Две функции для нитей — `pic_move_upr` и `pic_move_bot` — управляют изменением траектории и движением элемента. Первая нить считывает ввод с клавиатуры, изменяя значения `a` и `b` на основе нажатий клавиш. Вторая нить, соответственно, вычисляет новые координаты `x` и `y` на основе параметров `a`, `b`, и угла `phi`. Функция `MoveTo` перемещает `pic` по этим координатам с заданной через `delay(15)` скоростью движения.

Основной поток программы создает две нити с помощью `pthread_create`. Когда `pic_move_upr` завершает выполнение после нажатия Enter, основная программа прерывает нить `pic_move_bot`, после чего завершает сессию. В данном случае получается одновременное управление движением и взаимодействие с пользователем, при этом обе нити могут совершенно безопасно обмениваться данными через общую память.

Вывод

Через выполнение этих задач были продемонстрированы основные возможности работы с графикой и многозадачностью с использованием библиотеки VinGraph и базовых функций управления процессами и потоками. В ходе работы были применены разные подходы к управлению графическими элементами: от статической отрисовки фигур до динамических изменений их положения и характеристик. Это позволило создать интерактивные и оживленные графические композиции. Постепенно расширяя функциональность, каждая следующая задача добавляла новые аспекты управления: параллельное исполнение через процессы и взаимодействие между ними с помощью общей памяти.

В поздних задачах наблюдается переход от статичного изображения к динамическому, при этом достигается независимое движение элементов с помощью процессов и общих данных. Через подобный подход удалось убедиться, как многозадачность позволяет добиться эффекта взаимодействия элементов в реальном времени.

Заключительная задача с использованием потоков вместо процессов сделала программу более легковесной и оптимизировала управление ресурсами. Переход на потоки упростил доступ к общей памяти и управление синхронностью.