

Министерство образования и науки РФ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Омский государственный технический университет»

Факультет (институт) Факультет информационных технологий и компьютерных систем

Кафедра Информатики и вычислительной техники

КУРСОВОЙ ПРОЕКТ (РАБОТА)

по дисциплине Операционные системы

на тему Многопоточная Linux программная модель танцевального вечера юношей и девушек.

Пояснительная записка

Шифр проекта (работы) КП-02068999-20-97

Студента (ки) Даниловой Александры Юрьевны
фамилия, имя, отчество полностью

Курс 2 Группа ПИН-181

Направление (специальность) 09.03.04
Программная инженерия
код, наименование

Руководитель доцент, К.Т.Н
ученая степень, звание
Флоренсов Александр Николаевич
фамилия, инициалы

Выполнил (а) _____
дата, подпись студента (ки)

К защите _____
дата, подпись руководителя

Выполнение и подготовка к защите КП (КР)	Защита КП (КР)	Итоговый рейтинг

Проект (работа) защищен (а) с оценкой _____

Омск 2020

Министерство образования и науки РФ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Омский государственный технический университет»

ОТЗЫВ
на курсовой проект (работу)

Факультет (институт) Факультет информационных технологий и компьютерных систем

Кафедра Информатики и вычислительной техники

Дисциплина Операционные системы

Тема Многопоточная Linux программная модель танцевального вечера юношей и девушек.

Студент (ка) Даниловой Александры Юрьевны
фамилия, имя, отчество полностью

Курс 2 ПИН-
Группа 181

Руководитель Флоренсов Александр Николаевич
ученая степень, звание, ФИО

Содержание отзыва

Была разработана программа в соответствии с требованиями задания,
демонстрирующая модель танцевального вечера юношей и девушек. В программе
реализованы методы программного представления мьютексов, принципы создания
многопоточных приложений и организовано их корректное выполнение.

Рейтинговые баллы за выполнение и подготовку к защите курсового проекта (работы)	
Заключение о допуске к защите	

Руководитель Флоренсов А.Н. Дата 29 мая 20 20 г.

Реферат

Пояснительная записка по курсовому проекту 26 с., 4 ч., 7 рис., 3 источ, 1 прил.

C, LINUX, XLIB, GCC, МНОГОПОТОЧНОЕ ПРИЛОЖЕНИЕ.

Объектом исследования является алгоритм взаимодействия нескольких потоков в операционной системе Linux при работе в графическом окне.

Цель работы – разработка многопоточной программной модели танцевального вечера юношей и девушек.

В ходе работы реализован алгоритм взаимодействия нескольких потоков при работе с графическим окном в операционной системе Linux.

В результате была получена программа, которая демонстрирует модель танцевального вечера юношей и девушек.

Содержание

Введение	5
1 Введение в проблематику разработки многопоточных приложений.....	4
2 Декомпозиция разрабатываемой программы снизу-вверх с формированием основных процедур ее функционирования и описанием их функционального назначения	5
3 Описание глобальных информационных объектов программы: глобальных переменных, средств синхронизации потоков и используемых структур данных в случае их применения.....	7
4 Детальное текстовое описание на основе сочетания естественного языка и программных конструкций алгоритмов всех процедур.....	8
Заключение.....	19
Список использованных источников	20
Приложение.....	21

Введение

Курсовой проект по дисциплине «Операционные системы», 2 курс. В проекте использовался язык программирования C.

Задача: Разработать для ОС Linux многопоточную модель “танцевального вечера” с участием фиксированного числа юношей и девушек. Число юношей должно быть 12, а число девушек – 15. Каждая девушка и каждый юноша представляются отдельной нитью и их поведение отображается одним из алфавитных символов, для юношей использовать прописные, для девушек – строчные буквы. Поведение участников танцев описывается циклом с 5-кратным повторением. В цикле поведения участников задается интервал ожидания танца 8 сек, этап приглашения партнера – 3 сек, сам танец – 20 секунд во времени модели, время раскланяться и новая итерация цикла. Каждый юноша случайным образом выбирает себе предпочтительную девушку и приглашает ее на танец. Реализация модели в Linux. Поведение модели должно отображаться в консольном окне, размер окна не менее 40 строк на 100 столбцов, занимая в ходе моделирующих движений все пространство окна. Начальные размещения символов отдельных юношей и девушек вдоль двух смежных сторон окна. Положение для совместного танца рядом – по середине окна. Сами движения партнеров в танце или друг к другу не отображать. Для правильного взаимодействия использовать семафоры и мьютексы.

Проект состоит из пяти разделов:

- Введение в проблематику разработки многопоточных приложений
- Декомпозиция разрабатываемой программы снизу-вверх с формированием основных процедур ее функционирования и описанием их функционального назначения
- Описание глобальных информационных объектов программы: глобальных переменных, средств синхронизации потоков и используемых структур данных в случае их применения
- Детальное текстовое описание на основе сочетания естественного языка и программных конструкций алгоритмов всех процедур

1 Введение в проблематику разработки многопоточных приложений

При разработке многопоточных приложений следует иметь в виду необходимость явного указания для системы разработки, что данное приложение будет использовать более одной нити. Для таких приложений в процессе построения исполняемого файла подключаются специальные библиотеки подпрограмм.

При разработке многопоточных программ для Linux следует указывать соответствующую библиотеку поддержки. Такое указание может задаваться в одной из двух основных форм. Первая из них явно задает библиотеку и имеет вид:

```
gcc prog.c /usr/lib/libpthread
```

а вторая задает эту же библиотеку неявно и записывается в виде:

```
gcc prog.c -lpthread
```

Естественно, что при этом могут быть использованы и другие опции вызова компилятора, в частности, явное именование результирующего исполняемого файла и добавление отладочной информации.

В современных операционных системах широко используются нити (thread), называемые несколько неточно в русском переводе также потоками. Обычно нить своей работой реализует действия одной из процедур программы. Теоретически любой нити процесса доступны все части программы процесса, в частности, все его процедуры, но реально работа организуется так, чтобы нить отвечала отдельная процедура. Учитывая, что процедуре для нормальной работы необходимы локальные переменные, становится понятным закрепление области этих переменных за нитью. Объект хранения локальных переменных (вместе со служебной информацией при вызове подпрограмм) называют стеком. Этот стек в действительности является частью оперативной памяти, он используется не только программно, но и аппаратно, в частности, при реализации прерываний. Стек процедуры является неотъемлемой частью ресурсов, принадлежащих процедуре.

В операционной системе Unix многопоточное программирование появилось достаточно поздно. К настоящему времени эта возможность входит в стандарт POSIX для Unix и поддерживается во всех современных ОС. Использование нитей при этом требует подключения заголовочного файла pthread.h.

Важной задачей в многопоточном программировании является правильная реализация потоков. Если в программе требуется определённая последовательность работы потоков, то в таком случае следует использовать

семафоры или мьютексы, потому что последовательность работы потоков зависит от скорости выполнения других процессов, соответственно, если второй поток выполнится быстрее первого, то он и быстрее обработает данные, что собственно не должно произойти, поэтому не стоит забывать о правильной работе семафоров и мьютексов, если таковое имеется в программе. Потому что бывают такие ситуации, когда один поток заблокировал другой поток и после выполнения должен разблокировать, а этого не происходит, потому что программист просто не реализовал освобождение заблокированного потока.

2 Декомпозиция разрабатываемой программы снизу-вверх с формированием основных процедур ее функционирования и описанием их функционального назначения

Основным элементом функционирования являются графические примитивы, которые собственно и позволяют пользователю отслеживать действия объектов модели. Все происходящие действия объектов происходят в графическом окне 600x800 пикселей. Графические примитивы рисуются, путём получения координат объектов по x и y . Графический объект также имеет состояния, которые присутствуют в структуре данных этого объекта, например состояния танца или ожидания.

При помощи процедур изменения координат объекта, его состояний, а также очистки графического окна, рисуются графические примитивы. Сначала очищается окно, после же рисуются все графические примитивы и окно вновь очищается, что собственно представляет собой покадровое рисование объектов модели.

В процедуре `main` реализовано инициализация мест на танцполе, юношей и девушек. Также в `main` происходит создание графического окна и запуск потоков, а в конце `main` вызывается бесконечный цикл для отрисовки графических объектов.

Процедура `BoyActions` отвечает за логику действий юношей: выбор случайной девушки, ожидание ответа на предложение потанцевать, блокировка ее от других юношей в случае положительного ответа. Также осуществляется подбор свободного места и сам танец.

Процедура `GirlActions` отвечает за логику действий девушки: ожидание предложения, решение отказать или принять его.

Процедура `LogicActions` отвечает за смену танцев.

Процедура `InitBoys` отвечает за выделение памяти для массива юношей и их инициализацию.

Процедура `InitGirls` отвечает за выделение памяти для массива девушек и их инициализацию.

Процедура `InitPlaces` отвечает за выделение памяти для массива мест на танцполе и их инициализацию.

Процедура `InitStreams` отвечает за инициализацию и запуск потоковых функций.

Процедура `CreateWindow` отвечает за создание графического окна.

Процедура `Draw` отвечает за отрисовку объектов в графическом окне. В данной процедуре реализован бесконечный цикл, который собственно и содержит вызовы функции рисования.

3 Описание глобальных информационных объектов программы: глобальных переменных, средств синхронизации потоков и используемых структур данных в случае их применения

В программе используются следующие глобальные переменные:

1. `Display *dspl` – это внутренняя структура библиотеки `Xlib`, на которую ссылается программа пользователя при вызове функций, требующих обмена данными с X-сервером или функций, которым необходимы сведения о X-сервере. Поля этой структуры в документации не описываются и прямые обращения из программы к полям данных этой структуры запрещены.
2. `int screen` – номер экрана, используемый по умолчанию. Этот номер указывается при подключении к X-серверу.
3. `Window hwnd` – идентификатор окна.
4. `GC gc` – дескриптор контекста вывода графики. Дескриптор представляет из себя указатель на структуру данных, в которой хранятся специфичные для библиотеки `Xlib` данные и идентификатор графического контекста на X-сервере.
5. `Boy` – структура, которая содержит в себе все необходимые переменные для описания поведения юноши, где `id` – его идентификатор; `x` и `y` – текущие координаты; `oldX` и `oldY` – координаты, на которых он стоит слева; `idGirl` – идентификатор девушки, которую он выбирает; `idPlace` – идентификатор места, на которое он встаёт; `dancing` – состояние танца; `reverence` – поклон; `pthread_mutex_t dance` – mutex для блокировки и разблокировки при танце; `pthread_mutex_t wait` – mutex для взаимодействия с девушкой.
6. `Girl` – структура, которая содержит в себе все необходимые переменные для описания поведения девушки, где `id` – ее идентификатор; `x` и `y` – текущие координаты; `oldX` и `oldY` – координаты, на которых юноша стоит слева; `idBoy` – идентификатор юноши, с которым она танцует; `pthread_mutex_t lock` – mutex для блокировки от других парней; `pthread_mutex_t wait` – mutex для взаимодействия с юношей.

7. Place - структура данных места на танцполе, где id – идентификатор; x и y – текущие координаты; busy – состояние занято, либо свободно.
8. Stop - переменная, отвечающая за начало и остановку танца.
9. places – массив структур Place.
10. boys – массив структур Boy.
11. girls – массив структур Girl.
12. countBoys - количество юношей.
13. countGirls - количество девушек.
14. countDancing - количество танцев.

4 Детальное текстовое описание на основе сочетания естественного языка и программных конструкций алгоритмов всех процедур

Программа работает за счёт изменения координат x и y с заданной задержкой руководствуясь условиями, определяющими текущее состояние объекта. В результате этих действий координаты графических примитивов изменяются с заданной частотой. Изменение координат происходит за счёт простого инкремента или декремента переменных, отвечающих за соответствующие координаты y объекта графического примитива.

Графическое сопровождение программы реализовано при помощи графического окна и графических примитивов, которые программа рисует в данном графическом окне. В программе реализовано покадровое рисование графических примитивов при помощи бесконечного цикла `while`, который каждый последующий шаг рисует графические примитивы, а после, в конце цикла, очищает графический экран. В данном цикле обрабатываются данные юношей и девушек, которые в последствии используются для отображения их танца. Данный цикл расположен в функции `Draw`.

В главной функции `main` мы инициализируем места на танцполе, m и девушек, затем создаем графическое окно, инициализируем и запускаем потоки, запускаем отрисовку графических объектов. После чего, запущенные потоки начинают исполнять свои операции, которые определены в их потоковых функциях, которые собственно и отвечают за логику действий юношей и девушек.

За поведение юношей отвечает функция `BoyActions`. В данной функции описаны выбор случайной девушки, обращение к ней с предложением потанцевать, блокировка ее от других юношей на время ожидания ответа, и, в случае отказа, освобождение заблокированного потока. Далее происходит перебор мест на танцполе и сам танец, поклон, освобождение места на танцполе и возвращение на исходную позицию.

За поведение девушек отвечает функция `GirlActions`. В данной функции описаны выбор ожидания предложения потанцевать, принятие решения об отказе или согласии, сам ответ и возвращение на исходную позицию.

За смену танцев отвечает функция `LogicActions`. В данной функции определяется время ожидания и самого танца, подсчет количества повторений.

За инициализацию парней отвечает функция `InitBoys`. В данной функции происходит выделение памяти под массив юношей, перебор всех юношей и, в цикле, установка нужных данных: идентификатор, позиция, идентификатор девушки, идентификатор места, состояние покоя, а также мьютексы.

За инициализацию девушек отвечает функция `InitGirls`. В данной функции происходит выделение памяти под массив девушек, перебор всех девушек и, в цикле, установка нужных данных: идентификатор, позиция и мьютекс.

За инициализацию мест на танцполе отвечает функция `InitPlaces`. В данной функции происходит выделение памяти под массив мест на танцполе, перебор всех мест и присвоение им идентификаторов и состояния занятости, а также распределение мест по координатам в графическом окне.

За инициализацию и запуск потоковых функций отвечает функция `InitStreams`. В данной функции происходит динамическое выделение памяти и создание потоков.

За создание графического окна отвечает функция `CreateWindow`. В данной функции происходит создание окна, размером 600 на 800 пикселей, с координатами левого верхнего угла: $x=10$, $y=50$, черным цветом рамки и белым фоном. Также в функции присутствует бесконечный цикл, который нужен для ожидания event об открытии графического окна.

За отрисовку графических объектов отвечает функция `Draw`. В данной функции находится бесконечный цикл, в котором происходит рисование танцпола, юношей и девушек, вывод количества оставшихся танцев. А также очищение окна от графических элементов.

Схема алгоритма главной процедуры представлена на рисунке 1.

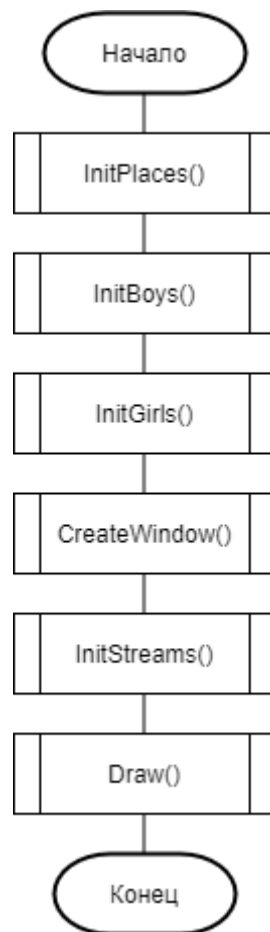


Рисунок 1 – схема алгоритма основной процедуры

Схема алгоритма процедуры BoyActions представлена на рисунке 2.

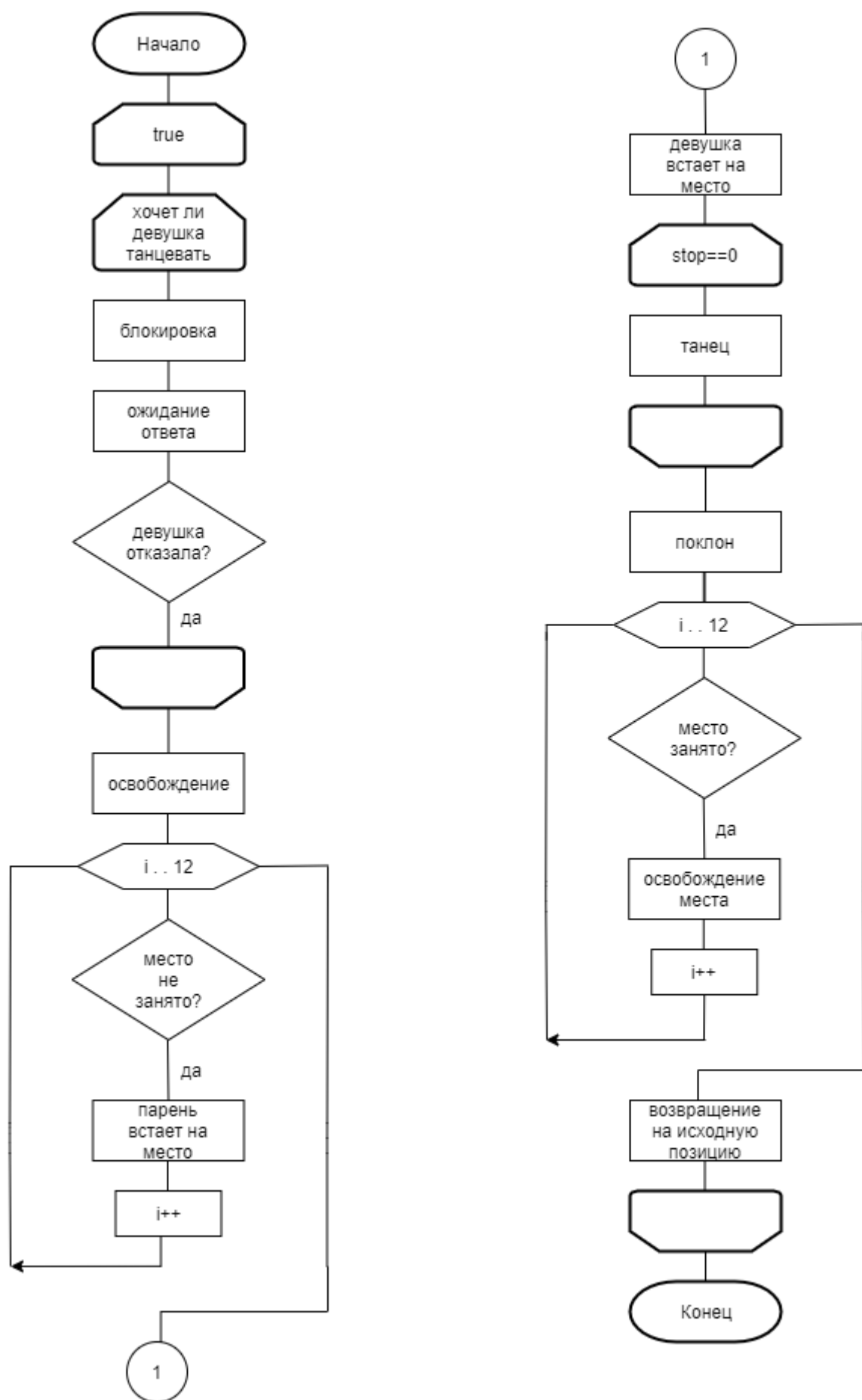
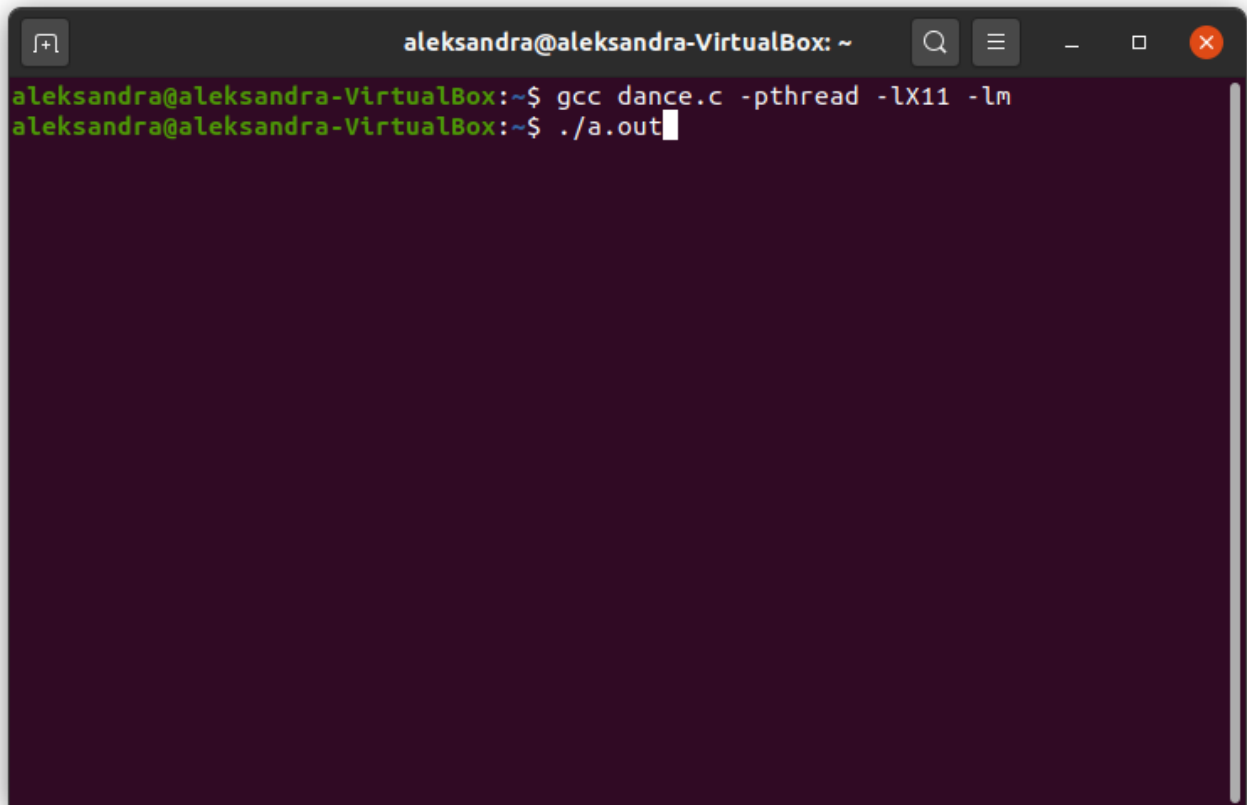


Рисунок 2 – схема алгоритма процедуры BoyActions

Скриншоты работы программы представлены на рисунках 2-6.

A terminal window titled 'aleksandra@aleksandra-VirtualBox: ~' with standard window controls. The terminal shows two commands: 'gcc dance.c -pthread -lX11 -lm' and './a.out', both preceded by the prompt 'aleksandra@aleksandra-VirtualBox:~\$'. The background is dark purple.

```
aleksandra@aleksandra-VirtualBox:~$ gcc dance.c -pthread -lX11 -lm
aleksandra@aleksandra-VirtualBox:~$ ./a.out
```

Рисунок 3 – компиляция и запуск программы

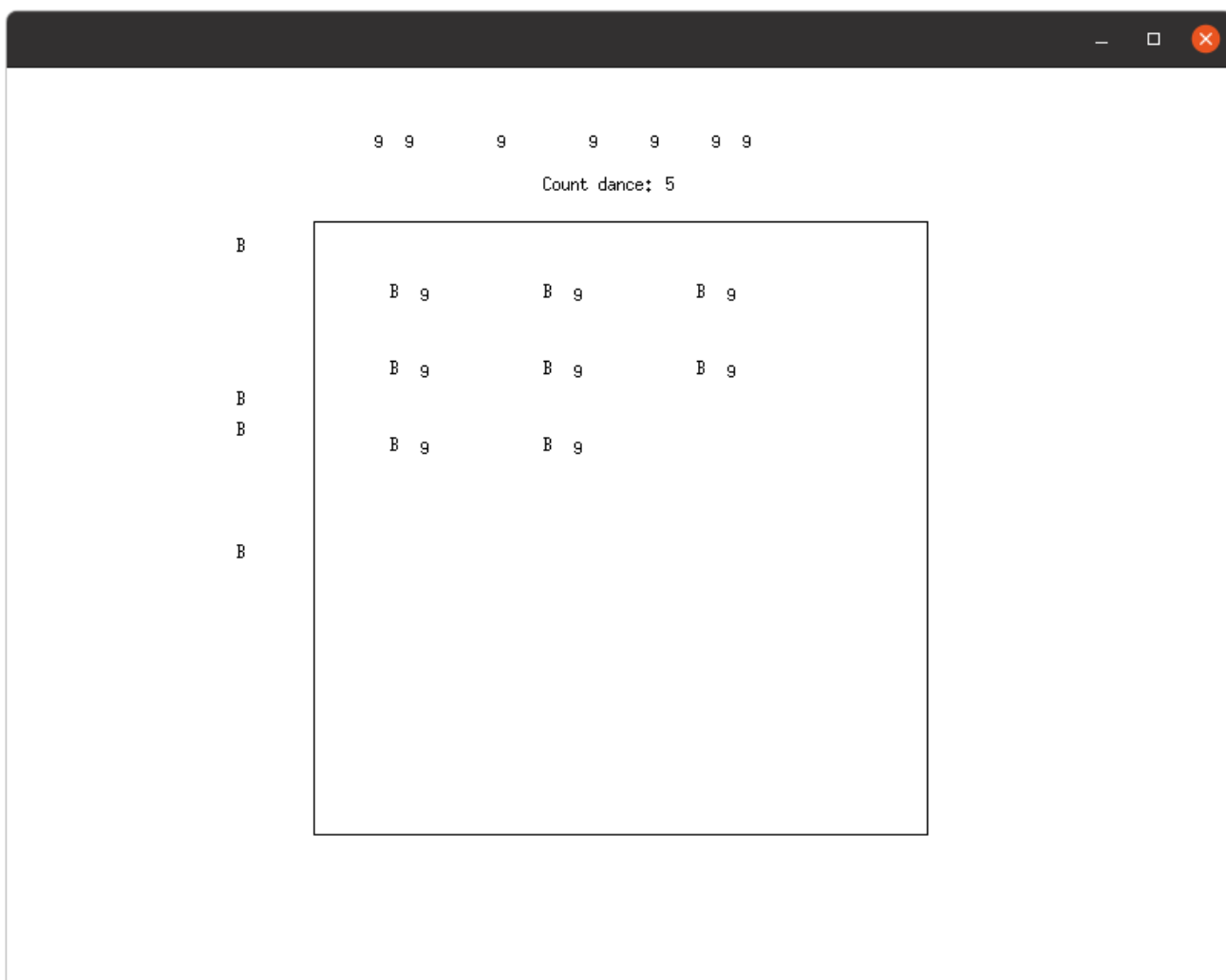


Рисунок 4 – ожидание танца

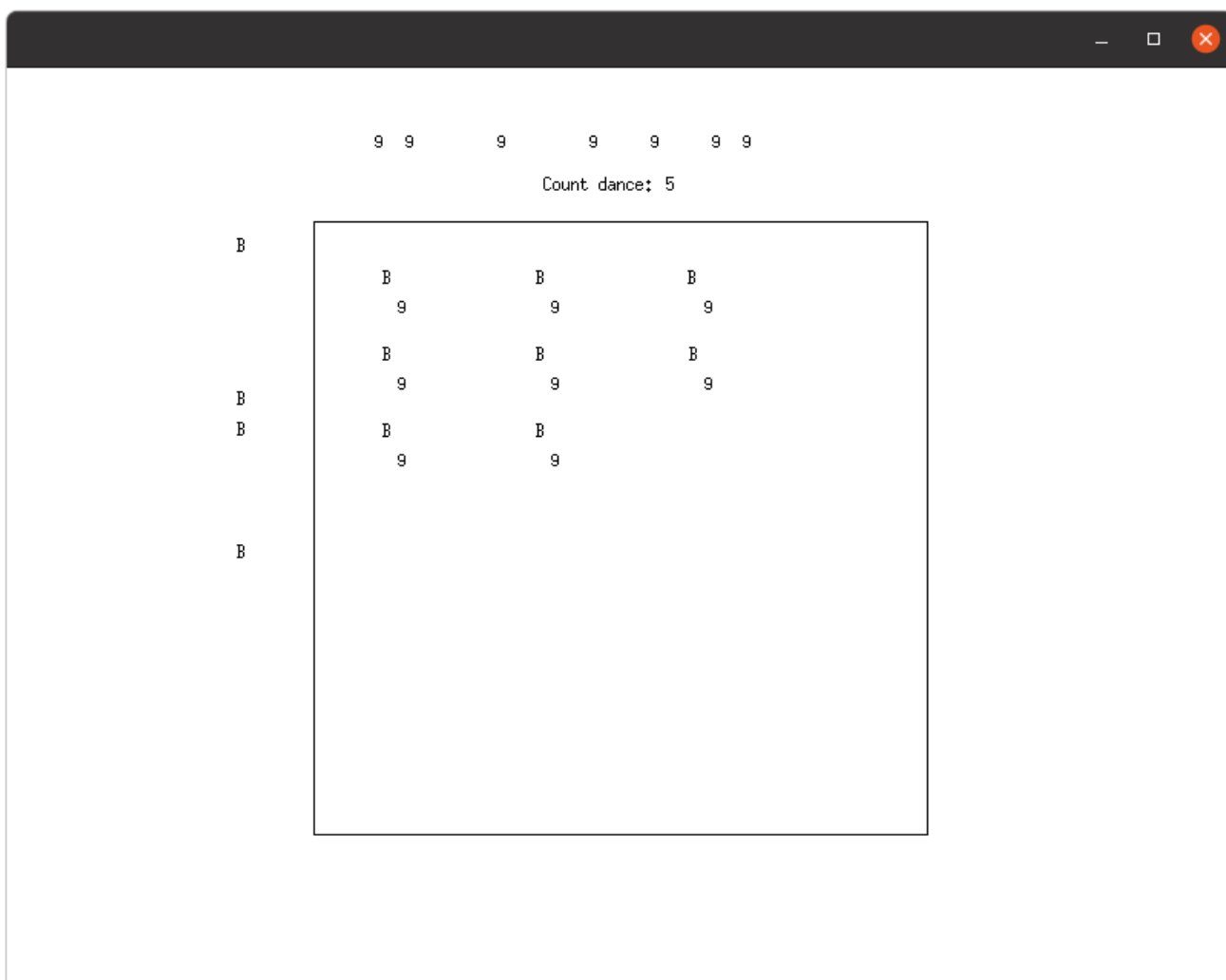


Рисунок 5 – танец

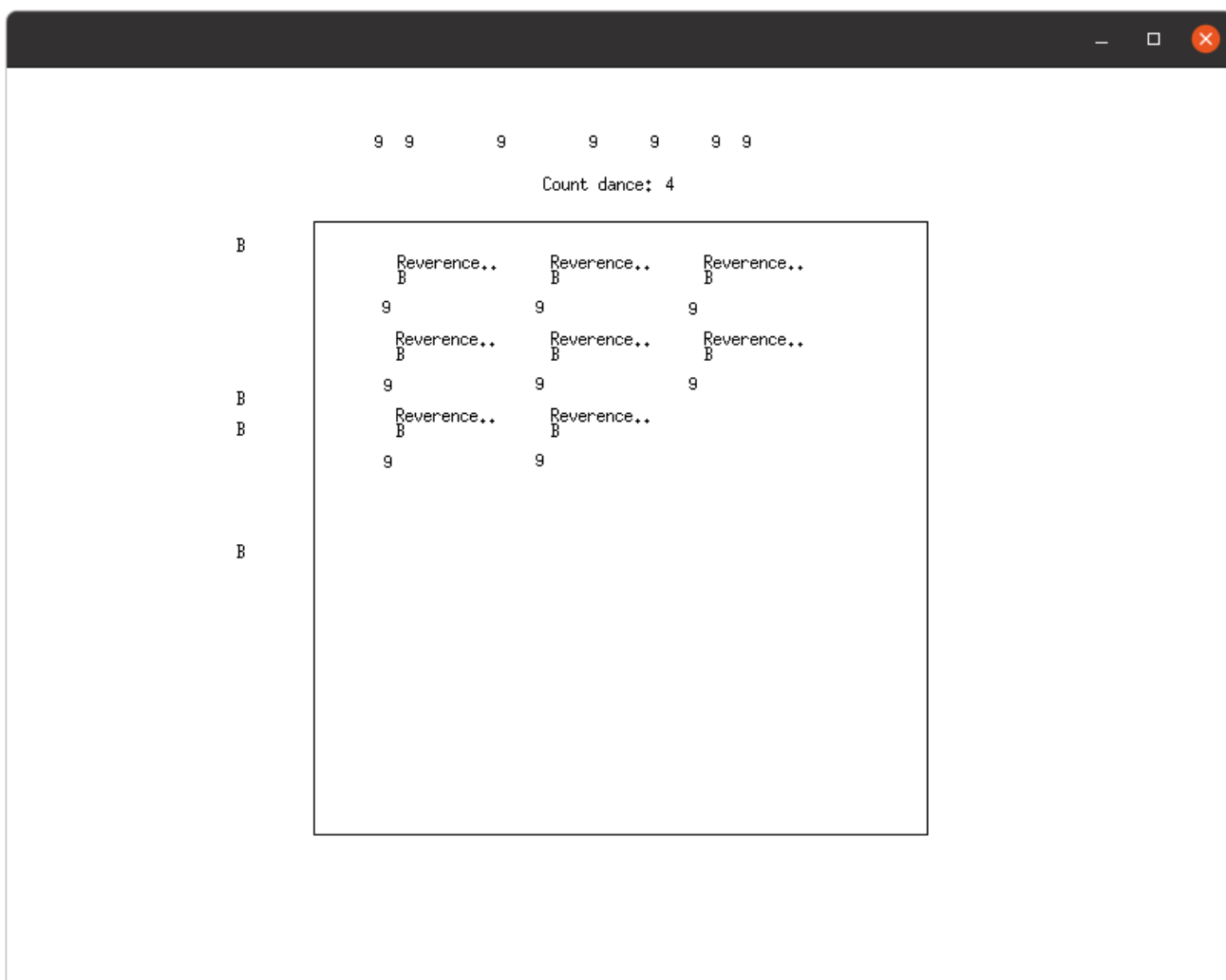


Рисунок 6 – поклон

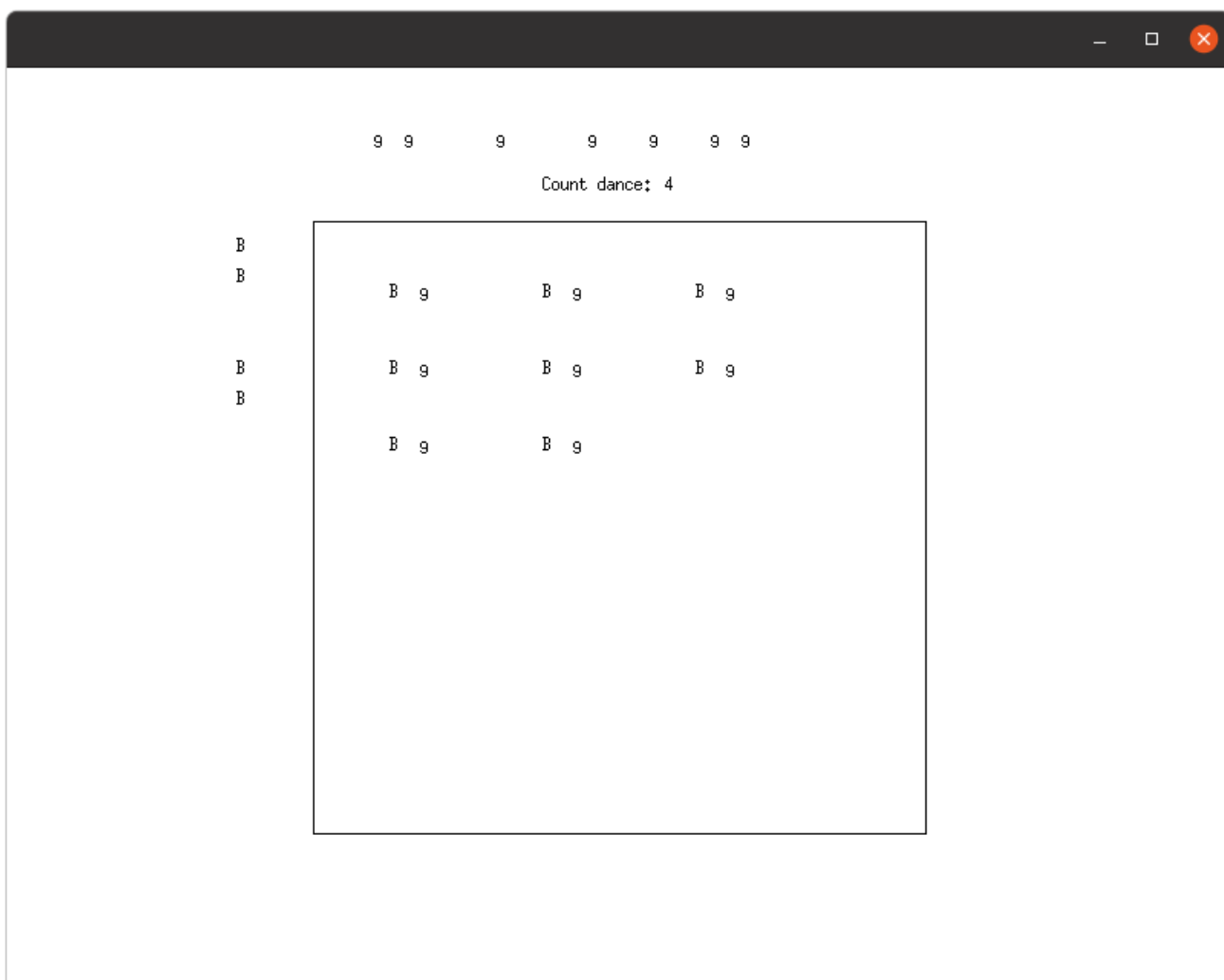


Рисунок 7 – смена партнера

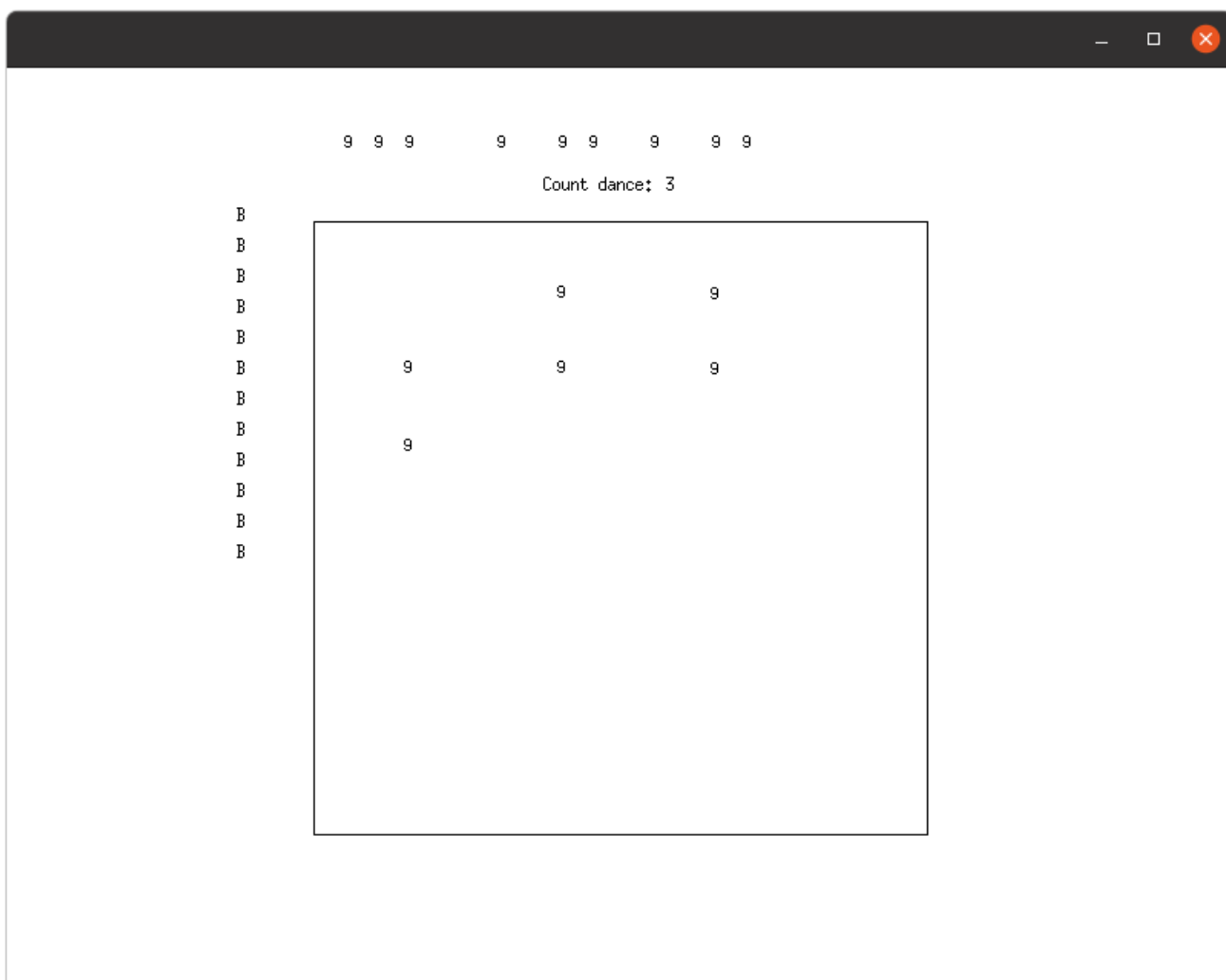


Рисунок 6 – возвращение юношей на исходное место

Заключение

В ходе курсового проектирования была разработана программа в соответствии с требованиями задания, демонстрирующая модель танцевального вечера юношей и девушек. Были изучены методы и средства программирования в операционной системе Linux при помощи функций языка Си. В программе реализованы методы программного представления мьютексов, принципы создания многопоточных приложений и организовано их корректное выполнение.

Список использованных источников

1. Флоренсов, А.Н. Операционные системы для программиста. Омск. ОмГТУ, 2005.
2. Гордеев, А.В. Операционные системы / А.В. Гордеев. – 2-е изд. – СПб.: Питер, 2007
3. ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание. Общие требования и правила составления.

Приложение

Исходный код программы

```
#include <stdio.h>
#include <malloc.h>
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>

#include <X11/Xlib.h>
#include <X11/XKBlib.h>

Display *dspl;
int screen;
Window hwnd;
XEvent event;
GC gc;

typedef struct {
    int id;
    float x, y;
    float oldX, oldY;
    int idGirl;
    int idPlace;
    int dancing;
    int reverence;
    pthread_mutex_t dance;
    pthread_mutex_t wait;
} Boy;

typedef struct {
    int id;
    float x, y;
    float oldX, oldY;
    int idBoy;
    pthread_mutex_t lock;
    pthread_mutex_t wait;
} Girl;

typedef struct {
    int id;
    float x, y;
    int busy;
} Place;

static int stop;

static Place* places;
static Boy* boys;
static Girl* girls;

static int countBoys;
```

```

static int countGirls;
static int countDancing;

void* BoyActions (void* thread_data){

    Boy *boy = (Boy*) thread_data;

    int idGirl;
    idGirl = rand()%countGirls;
    float timeW1 = 1;
    float timeW2 = 180;

    while(1){

        while(girls[idGirl].idBoy != boy -> id){
            pthread_mutex_lock(&girls[idGirl].lock);
            girls[idGirl].idBoy = boy -> id;
            pthread_mutex_unlock(&girls[idGirl].wait);
            pthread_mutex_lock(&boy -> wait);
            if(girls[idGirl].idBoy != boy -> id)
pthread_mutex_unlock(&girls[idGirl].lock);
        }

        boy -> dancing = 1;

        for(int i=0; i<12; i++){

            if(places[i].busy==0){
                boy->idPlace = places[i].id;
                boy->x = places[i].x;
                boy->y = places[i].y;
                places[i].busy = 1;
                break;
            }
        }

        girls[idGirl].x = boy->x+20;
        girls[idGirl].y = boy->y;
        pthread_mutex_lock(&boy->dance);

        while(stop==0){

            boy -> x = places[boy -> idPlace].x + 10 * cos (timeW1);
            boy -> y = places[boy -> idPlace].y + 10 * sin (timeW1);

            girls[idGirl].x = places[boy -> idPlace].x + 10 * cos (timeW2);
            girls[idGirl].y = places[boy -> idPlace].y + 10 * sin (timeW2);

            if(timeW1 == 360) timeW1 = 1;
            if(timeW2 == 360) timeW2 = 1;

            timeW1 += 0.01;
            timeW2 += 0.01;
            usleep(5000);
        }
    }
}

```

```

        boy -> reverence = 1;
        sleep(4);
        boy -> reverence = 0;

        for(int i=0; i<12; i++)
            if(boy->idPlace == places[i].id){
                places[i].busy = 0;
                break;
            }

        pthread_mutex_unlock(&girls[idGirl].wait);
        pthread_mutex_unlock(&girls[idGirl].lock);

        boy -> dancing = 0;
        boy -> x = boy -> oldX;
        boy -> y = boy -> oldY;
        sleep(1);
    }
}

void* GirlActions (void* thread_data){
    Girl *girl = (Girl*) thread_data;

    int idBoy;

    while(1){
        pthread_mutex_lock(&girl->wait);
        idBoy = girl->idBoy - 1;
        if (1 == rand()%8) girl->idBoy = -1;
        pthread_mutex_unlock(&boys[idBoy].wait);
        pthread_mutex_lock(&girl->wait);

        girl->x = girl->oldX;
        girl->y = girl->oldY;
        sleep(1);
    }
}

void *LogicActions (void *unused){
    while(countDancing>1){

        sleep(8);
        stop = 0;

        for(int i = 0; i < countBoys; i++)

            if(boys[i].dancing == 1)
                pthread_mutex_unlock(&boys[i].dance);
            sleep(20);
            stop=1;
            countDancing--;
    }
}

```



```

void InitBoys(){

    boys = (Boy*) malloc(countBoys * sizeof(Boy));
    for(int i = 0; i < countBoys; i++){
        boys[i].id = i+1;
        boys[i].oldX = 150;
        boys[i].oldY = 100 + (i*20);
        boys[i].x = boys[i].oldX;
        boys[i].y = boys[i].oldY;
        boys[i].idGirl = -1;
        boys[i].idPlace = -1;
        boys[i].dancing = 0;
        pthread_mutex_lock(&boys[i].dance);
        pthread_mutex_lock(&boys[i].wait);
    }
}

void InitGirls(){

    girls = (Girl*) malloc(countGirls * sizeof(Girl));

    for(int i=0; i<countGirls; i++){
        girls[i].id = i+1;
        girls[i].oldX = 200 + (i*20) ;
        girls[i].oldY = 50;
        girls[i].x = girls[i].oldX;
        girls[i].y = girls[i].oldY;
        girls[i].idBoy = -1;
        pthread_mutex_lock(&girls[i].wait);
    }
}

void InitPlaces(){

    places = (Place*) malloc(12 * sizeof(Place));

    for(int i=0; i<12; i++){
        places[i].id=i;
        places[i].busy=0;
    }

    places[0].x = 250; places[0].y = 150;
    places[1].x = 350; places[1].y = 150;
    places[2].x = 450; places[2].y = 150;
    places[3].x = 250; places[3].y = 200;
    places[4].x = 350; places[4].y = 200;
    places[5].x = 450; places[5].y = 200;
    places[6].x = 250; places[6].y = 250;
    places[7].x = 350; places[7].y = 250;
    places[8].x = 450; places[8].y = 250;
    places[9].x = 250; places[9].y = 300;
    places[10].x = 350; places[10].y = 300;
    places[11].x = 450; places[11].y = 300;
}

```

```

void InitStreams(){

    pthread_t* threadsBoys = (pthread_t*) malloc(countBoys * sizeof(pthread_t));
    pthread_t* threadsGirls = (pthread_t*) malloc(countGirls *
sizeof(pthread_t));
    pthread_t threadsLogic;

    for(int i=0; i<countBoys; i++)    pthread_create(&(threadsBoys[i]), NULL,
BoyActions, &boys[i]);
    for(int i=0; i<countGirls; i++)    pthread_create(&(threadsGirls[i]), NULL,
GirlActions, &girls[i]);

    pthread_create(&threadsLogic, NULL, LogicActions, NULL);
}

void CreateWindow(){

    dspl = XOpenDisplay(NULL);
    if (dspl == NULL) {
        printf("Error XOpenDisplay\n");
        exit(1);
    }
    screen = XDefaultScreen(dspl);

    hwnd = XCreateSimpleWindow(dspl, RootWindow(dspl, screen), 100, 50,
800, 600, 3, BlackPixel(dspl, screen),
WhitePixel(dspl, screen));

    if (hwnd == 0) {
        printf("Error XCreateSimpleWindow\n");
        exit(1);
    }

    XSelectInput(dspl, hwnd, ExposureMask | StructureNotifyMask);
    XMapWindow(dspl, hwnd);
    gc = XDefaultGC(dspl, screen);

    while(1) {
        XEvent event;
        XNextEvent(dspl, &event);
        if (MapNotify == event.type) break;
    }
}

void Draw(){

    char bufStr[3];

    while (1){

        XDrawRectangle(dspl, hwnd, gc, 200, 100, 400, 400);
        XDrawString(dspl, hwnd, gc, 350, 80, "Count dance:", 12);

        sprintf(bufStr, "%d", countDancing);
    }
}

```

```

        if(countDancing < 10) XDrawString(dspl, hwnd, gc, 430, 80,
bufStr, 1);
        else if(countDancing > 9) XDrawString(dspl, hwnd, gc, 430, 80,
bufStr, 2);

        for(int i=0; i<countBoys; i++){
            XDrawString(dspl, hwnd, gc, boys[i].x, boys[i].y, "B", 1);
            if(boys[i].reverence == 1) XDrawString(dspl, hwnd, gc,
boys[i].x, boys[i].y-10, "Reverence..", 11);
        }

        for(int i=0; i<countGirls; i++)
            XDrawString(dspl, hwnd, gc, girls[i].x, girls[i].y, "g", 1);

        XFlush(dspl);
        XClearWindow(dspl, hwnd);
        usleep(50000);
    }
}

int main() {

    countBoys = 12;
    countGirls = 15;
    countDancing = 5;

    InitPlaces();
    InitBoys();
    InitGirls();
    CreateWindow();
    InitStreams();
    Draw();
    return 0;
}

```