МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)**

Институт информационных технологий, математики и механики

Направление подготовки: «Программная инженерия»

ОТЧЕТ

по лабораторной работе №4

ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ СИСТЕМЫ ОБСЛУЖИНИЯ ПОТОКА ЗАДАНИЙ НА ЭВМ (ОЧЕРЕДИ)

Выполнил: студент группы
3822Б1ПР2

_____В. Е. Филатьев
Подпись

Нижний Новгород
2023

Содержание

1.Введение:

Цель лабораторной работы заключается в освоении динамической структуры данных очередь путем изучения различных способов хранения очереди и разработки методов и программ для решения задач с использованием очередей. Основной областью применения является эффективная организация выполнения задач на вычислительных системах. Очередь характеризуется порядком обработки значений, при котором новые элементы вставляются в конец очереди, а извлечение происходит из начала. Примером использования очереди может служить задача разработки системы имитации однопроцессорной ЭВМ. Эта схема имитации является одной из простых моделей обслуживания заданий в вычислительной системе.

Имитационное моделирование реально существующих объектов и явлений-физических, химических, биологических, социальных процессов, живых и неживых систем, инженерных конструкций, конструируемых объектов – представляет собой построение математической модели, которая описывает изучаемое явление с достаточной точностью, и последующую реализацию разработанной модели на ЭВМ для проведения вычислительных экспериментов с целью изучения свойств моделируемых явлений. Использование имитационного моделирования позволяет проводить изучение исследуемых объектов и явлений без проведения реальных экспериментов.

Очередь — это абстрактный тип данных, который представляет собой список элементов, где добавление новых элементов происходит в конец очереди, а удаление — из начала. Это означает, что элемент, добавленный первым, будет удален первым (First-In-First-Out, FIFO).

2. Постановка задачи:

Для вычислительной системы (ВС) с одним процессором и однопрограммным последовательным режимом выполнения поступающих заданий требуется разработать программную систему для имитации процесса обслуживания заданий в ВС. При построении модели функционирования ВС должны учитываться следующие основные моменты обслуживания заданий:

- генерация нового задания;
- постановка задания в очередь для ожидания момента освобождения процессора;
- выборка задания из очереди при освобождении процессора после обслуживания очередного задания.

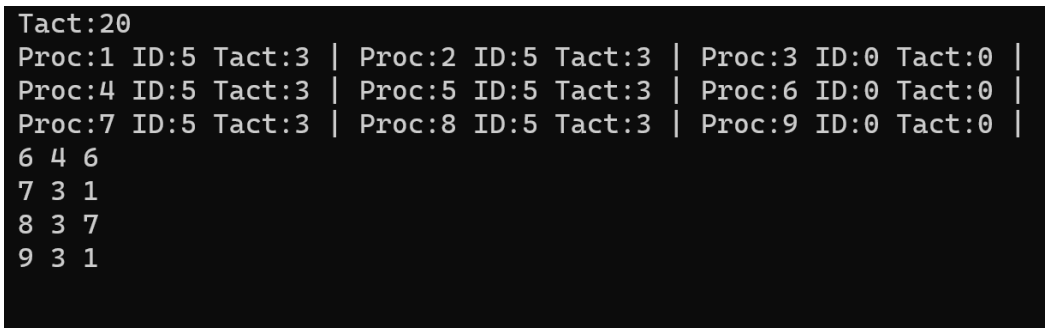
По результатам проводимых вычислительных экспериментов система имитации должна выводить информацию об условиях проведения эксперимента (интенсивность потока заданий, размер очереди заданий, производительность процессора, число тактов имитации) и полученные в результате имитации показатели функционирования вычислительной системы, в т.ч:

- количество поступивших в ВС заданий;
- количество отказов в обслуживании заданий из-за переполнения очереди;
- среднее количество тактов выполнения заданий;
- количество тактов простоя процессора из-за отсутствия в очереди заданий для обслуживания.

3. Руководство пользователя

3.1. Консольное приложение

При начале работы приложения, запускается работа процессора и на экране мы видим такт процессора и таблицу ядер (*рис 1*). У каждого ядра мы видим ID процесса и количество оставшихся тактов. Каждая из задач распределяется на необходимое количество ядер. После таблицы ядер, идет список задач. Список задач состоит из ID задачи количества ядер и количество тактов необходимых для выполнения задачи.



Рису 1

3.2. Визуальное приложение:

В визуальном приложении при начале работы, запускается работа процессора и на экране мы видим таблицу ядер (*рис 2*). У каждого ядра мы видим ID процесса и количество оставшихся тактов. Каждая из задач распределяется на необходимое количество ядер. Рядом с таблице ядер расположены таблица задач и таблица очереди. В таблице задач расположены задачи, которые в данный момент выполняются процессором, в таблице очереди расположены задачи, которые требуют выполнения. Задач состоит из ID задачи количества ядер и количество тактов необходимых для выполнения задачи.

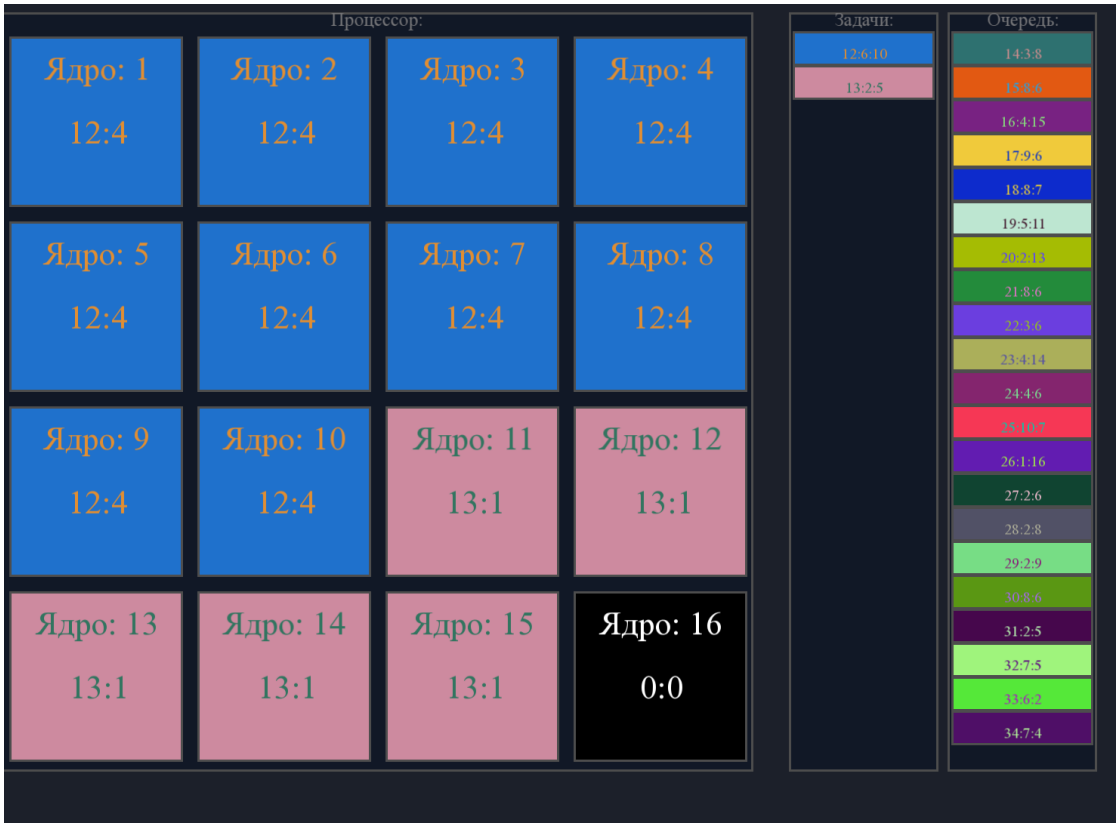
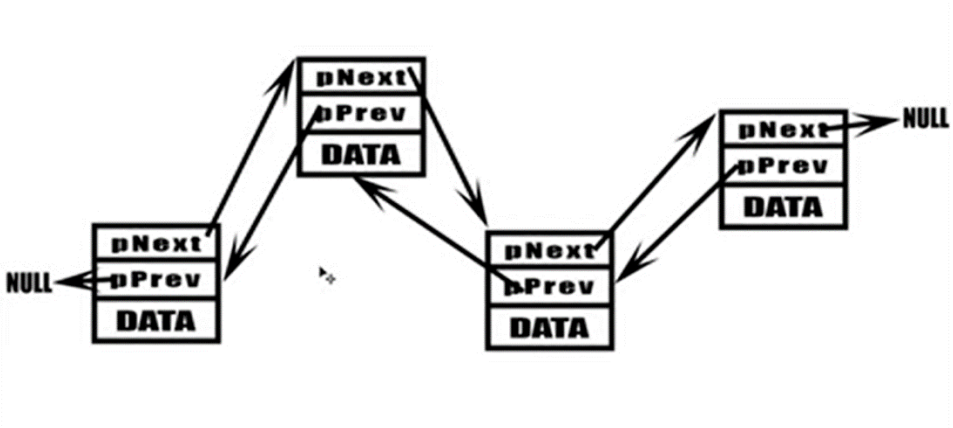


Рис 2

3. Руководство программиста:

3.1 Описание структур:

Очередь, построенная на двухсвязном списке в языке программирования C++, представляет собой структуру данных, которая поддерживает операции добавления элемента в конец очереди (push) и удаления элемента из начала очереди (pop). Двухсвязной список состоит из узлов, каждый из которых содержит указатели на предыдущий и следующий элементы. Это позволяет эффективно добавлять и удалять элементы как в начале, так и в конце списка.

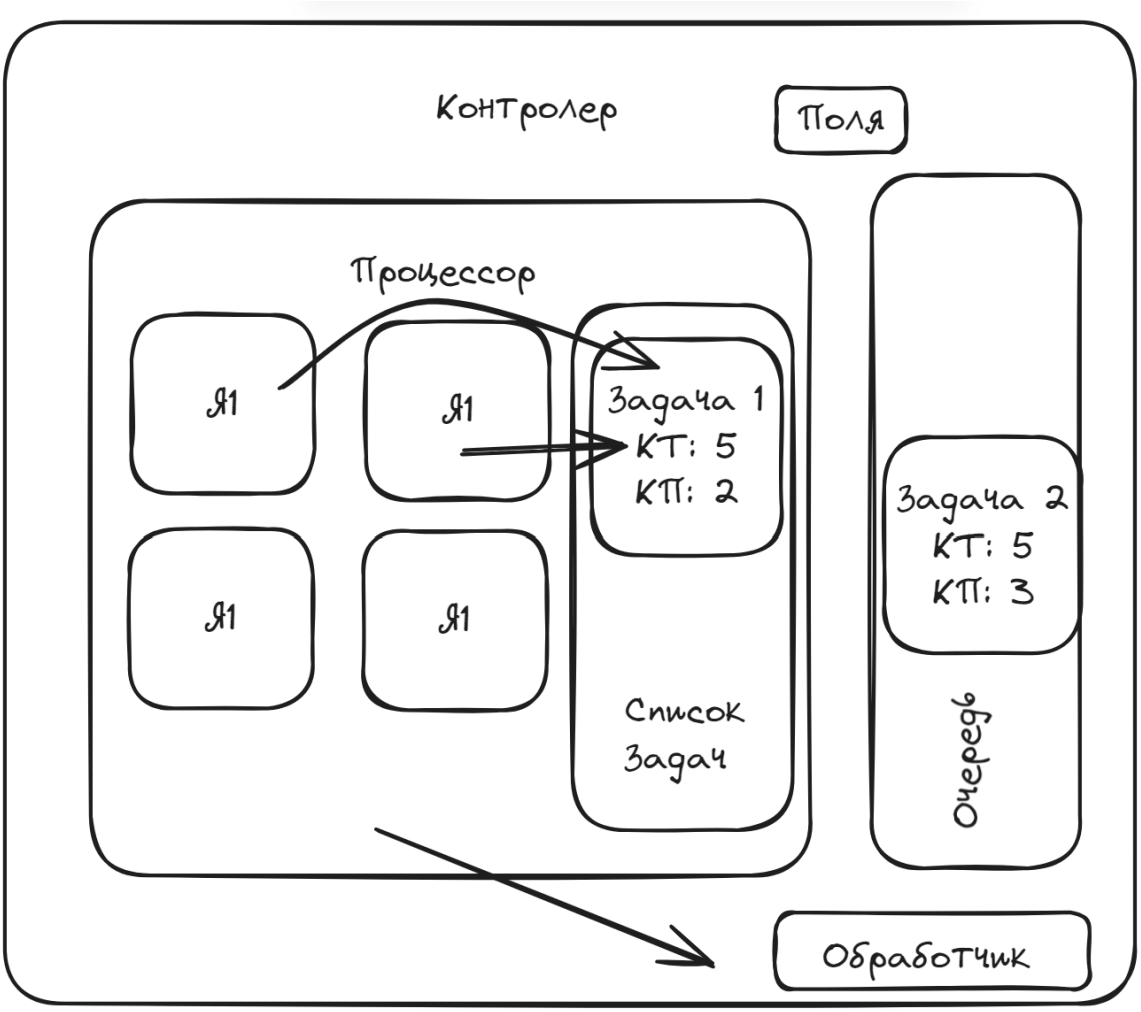


Для реализации очереди на двухсвязном списке в C++ необходимо создать класс, который будет содержать методы для добавления и удаления элементов, а также указатели на начало и конец списка. Метод push будет добавлять новый элемент в конец списка, а метод pop будет удалять элемент из начала списка. Преимуществом использования двухсвязного списка для реализации очереди является то, что операции добавления и удаления элементов выполняются за константное время O(1). Однако, необходимо учитывать, что использование двухсвязного списка требует больше памяти для хранения указателей на предыдущий и следующий элементы. Таким образом, очередь, построенная на двухсвязном списке в C++, является эффективной структурой данных для реализации очереди с быстрым доступом к началу и концу списка.

Схема имитации процесса поступления и обслуживания заданий в вычислительной системе состоит в следующем.

1) Контролера – он хранит в себе процессор и очередь задач и на каждой итерации пытается передать задачу из очереди в процессор также обрабатывает отображение процессора и очереди на экране

2) Процессор состоит из списка ядер и списка задач выполняющихся на процессоре. Процессор выполняет действия по добавления задачи в ядра, обрабатывает такты, а также освобождает ядра при завершение задачи.


Контролер
Поля
Процессор
Я1
Я1
Я1
Я1
Задача 1
КТ: 5
КП: 2
Список
Задач
Задача 2
КТ: 5
КП: 3
Очередь
Обработчик

3.2.Описание программы:

Программа состоит из нескольких основных классов:

Класс очередь:

Класс очередь – шаблонный класс с изменяемым типом хранения содержащий:

Внутренний класс:

- class Note; - Данный класс хранит значение ячейки (T value;) и указатель на следующую и предыдушую ячейку (Note* next, prev). В классе перегружен конструктор инициализации.

Поля:

- Note<T>* head; - указатель на первый элемент очереди
- Note<T>* tail; - указывает на последний элемент очереди
- size_t sz; - размер очереди

Методы:

- MyQueue() noexcept; - конструктор по умолчанию
- bool empty() noexcept; - метод проверки на пустоту стак
- void push(T value); - добавление элемента в начала стека
- void pop(); - удаление элемента из начала стека
- void clear(); - очистка стека
- size_t size() noexcept – метод возвращает размер стека
- T& top(); - метод возвращает верхний элемент стека
- ~ MyQueue(); - деструктор
- Note<T>* begin() – возвращает указатель на первый элемент
- Note<T>* end() – возвращает указатель на следующий элемент конец очереди

Класс задача:

Поля:

- static int kolTask; - Общее поле для все задач определяет уникальный номер для задачи

- int id; - уникальный номер задачи
- int kolProc; - количество процессоров необходимых для выполнения задачи
- int kolTact; - количество оставшихся тактов для выполнения задачи
- int allTact; - количество тактов необходимое для выполнения задачи

Методы:

- Task(); - конструктор по умолчанию
- Task(int kolProc, int kolTact); - конструктор инициализации
- bool Tact(); - метод выполняющий так для задачи
- int ID() – метод возвращающий ID задачи
- int GetKolProc() – метод возвращающий количество ядер необходимы для выполнения задачи
- int GetKolTract() - метод возвращающий количество оставшихся тактов для выполнения данной задачи
- int GetAllTact() – метод возвращающий количество тактов необходимых для выполнения задачи

Класс процессор:

Внутренний класс:

Класс ядро: - Данный класс хранит данные о всех ядрах данного процессора.

Поля:

- vector<Core> Cores; - Вектор ядер
- list<Task> Tasks; Список задач выполняющихся на процессоре
- int kolProz; - Общее число ядер
- int kolFree; - Количество свободных ядер
- vector<int> id; ID освободившихся процессоров

Методы:

- Proc(int kol = 0); - Конструктор инициализации
- void setKolProc(int kol); - Метод возвращающий количество процессоров
- bool SetTask(Task task); - Метод распределения задачи между ядрами

- void Tact(); - Метод обрабатывающий такт процессора
- vector<Core>& GetCore() ; - Метод возвращающий список ядер

Класс контроллер:

Поля:

- Proc proc; - Процессор контроллера
- MyQueue<Task> tasks; - Очередь задач
- int Width, Height; - Высота и ширина таблицы ядер
- vector<vector<CoreInfo>> TablCores; - Таблица ядер
- int kolProc; - Количество ядер
- int Shans; - Шансы поступления задачи на процессор
- int minKolProc, maxKolProc; - Максимальное и минимальное количество ядер
- int minKolTact, maxKolTact; Максимальное и минимальное количество тактов

Методы:

- Controller(int kolProc, int Shans, int minKolProc, int maxKolProc, int minKolTact, int maxKolTact); - конструктор иницилизации
- void Tact(); - Метод обрабатывающий так процессора
- Proc& GetProc() – Метод возвращающий количество ядер
- MyQueue<Task> GetTask() – Метод возвращающий очередь задач
- MyQueue<Task> GetTaskProc(); - Метод возвращающий очередь ядер
- vector<vector<CoreInfo>> GetTablCores() – Метод возвращающий таблицу ядер

Класс MainForm:

Класс основного окна, выполняет бесконечный цикл проверяет события и отображает объекты на окне

Поля:

- RenderWindow* window; - окно приложения
- button* exit; - кнопка выхода из приложения
- Text name; - текстовое поле для названия приложения
- Font font; - поля для хранения шрифта

- PanelProc* panelProc; - Панель для отображения ядер процессора
- PanelQueue* panelQueue; - Панель для отображения очереди задач
- PanelQueue* panelTask; - Панель для отображения панели задач процессора
- Controller* controller; - Контроллер

Методы:

- void Update(); - метод обрабатывает события
- void Draw(); - метод отрисовывает объекты на форме
- void Init(); - метод инициализирует объекты
- MainForm(int Windht = 1000, int Height = 1300); - конструктор инициализации
- ~MainForm(); - деструктор
- void Start(); - запускает окно и выполняет бесконечный цикл работы приложения
- void zap_knopki(); - заполняет имена кнопок
- void MousKlic(wstring s); - метод обрабатывающий нажатие на кнопки

Класс button:

Поля:

- RectangleShape rectagle; - прямоугольник ограничивавший область кнопки
- Font font; - поле содержавшие шрифт текста
- Text text; - поле текста кнопки
- Color Hover; - цвет кнопки при наведения на него мышки
- Color Fill; - цвет кнопки

Методы:

- button(int Windht = 100, int Height = 100); - конструктор
- Color GetColor() – выводит цвет кнопки
- Vector2f GetSize() - размеры кнопки
- Vector2f GetPosition() – положение кнопки
- void SetColor(Color color); - метод задает цвет кнопке
- void SetHoverColor(Color color) – метод задает цвет кнопки при наведение
- Text& Text() – получить текст кнопки

- void SetPosition(Vector2f vec); - задает положение кнопки
- void draw(RenderWindow* window); - отрисовка кнопки на форме
- bool MousHover(RenderWindow* window); - проверяет наведена ли курсор на кнопку

4.Эксперементы:

Количество тактов	Количество ядер	Среднее число необходимых процессоров	Среднее число тактов на задачу	Шанс получения задачи	Количество выполненных задач	Среднее число занятых ядер
100	9	5	5	50%	32	6
100	9	5	8	50%	20	5
100	9	8	5	50%	18	8
100	9	5	5	90%	22	6
1000	9	5	5	50%	210	5

5.Заключение

В ходе выполнения этой лабораторной работы, я успешно справился со всеми поставленными задачами. После изучения темы очередь был реализован класс очередь на основе двухсвязного списка, после в него были добавлены различные улучшения такие как, интегрируемость по очереди. Реализовали тесты для проверки основных функций класса. После был реализованы классы процессор и контроллер для выполнения имитационной модели процессора. Классы были оптимизированы для эффективной работы. Для проверки работоспособности системы были реализованы консольный и визуальный интерфейс. В интерфейсах проверили работоспособность приложений и верное добавления задач в процессор. Для работы с приложением были созданы кнопки управления приложением, а так же окно отображения процессора и очереди.

6.Литература:

1. Википедия/Очередь - https://ru.wikipedia.org/wiki/Очередь
2. Википедия/Список - https://ru.wikipedia.org/wiki/Связный_список
3. Учебно-методическое пособие - https://drive.google.com/drive
4. Библиотека SFML - https://www.sfml-dev.org/

7.Приложение:

Очередь:

```
#include <iostream>
using namespace std;
template<class T>

class MyQueue {
public:
	MyQueue() noexcept;
	MyQueue(MyQueue<T>& other);
	size_t size() noexcept { return sz; }
	void clear();
	void push(T value);
	T& top();
	bool empty() noexcept;
	void pop();
	~MyQueue();
private:
	template<class T>
	class Note {
	public:
		T value;
		Note* next;
		Note* prev;
		Note(T value = T(), Note* next = nullptr,Note* prev = nullptr) {
			this->value = value;
			this->next = next;
			this->prev = prev;
		}
	};

	Note<T>* tail;
	Note<T>* head;
	size_t sz;
public:
	class iterator {
		Note<T>* t;
	public:
		iterator() { t = nullptr; }
		void operator=(Note<T>* note) {
			t = note;
		}
		T operator*() {
			return t->value;
		}
		void operator++(int k) {
			t = t->prev;
		}
		bool operator!=(const Note<T>* note) {
			return t != note;
		}
	};

	Note<T>* begin() { return head; }
	Note<T>* end() { return (tail==nullptr)?tail:tail->prev; }
};
template<class T>
MyQueue<T>::MyQueue() noexcept {
```

```
	sz = 0;
	head = tail = nullptr;
}
template<class T>
MyQueue<T>::MyQueue(MyQueue<T>& other) {
	this->sz = 0;
	this->head = this->tail = nullptr;
	auto it = MyQueue<T>::iterator();
	it = other.begin();
	while (it != other.end()) {
		this->push((*it));
		it++;
	}
}
template<class T>
void MyQueue<T>::clear() {
	while (!this->empty())
		this->pop();
}
template<class T>
void MyQueue<T>::push(T value) {
	if (sz == 0) {
		head = new Note<T>(value);
		tail = head;
	}
	else {
		Note<T>* temp = new Note<T>(value, tail);
		tail->prev = temp;
		tail = temp;
	}
	sz++;
}
template<class T>
T& MyQueue<T>::top() {
	if (sz == 0)
		throw exception("Очередь пуста");
	return head->value;
}
template<class T>
bool MyQueue<T>::empty() noexcept {
	if (sz == 0)
		return true;
	return false;
}
template<class T>
void MyQueue<T>::pop() {
	if (sz == 0)
		throw exception("Очередь пуста");
	if (head->prev != nullptr) {
		Note<T>* temp = head->prev;
		temp->next = nullptr;
		delete head;
		head = temp;
	}
	else {
		delete head;
		head = nullptr;
		tail = nullptr;
	}
	sz--;
}
template<class T>
```

```
MyQueue<T>::~MyQueue() {
	this->clear();
	}
```

CP.h:

```
#pragma once
#include <vector>
#include <list>
#include <algorithm>
#include <chrono>
#include <thread>
#include "../queue/lib/queue_lib.h"
using namespace std;

class Task {
private:
	static int kolTask;
	int id;
	int kolProc;
	int kolTact;
	int allTact;
public:
	int color[3];
	Task();
	Task(int kolProc, int kolTact);
	bool Tact();

	int ID() { return id; }
	int GetKolProc() { return kolProc; }
	int GetKolTract() { return kolTact; }
	int GetAllTact() { return allTact; }
};


class Proc {
public:
	Proc(int kol = 0);
	void setKolProc(int kol);
	bool SetTask(Task task);
	void Tact();

	int kol = 0;

protected:
	class Core {
	private:
		Task* task;
		bool flag;
	public:
		Core();
		Core(const Core& other);
		Core& operator=(const Core& other) {
			this->flag = other.flag;
			this->task = other.task;
			return *this;
		}
		bool SetTask(Task* task);
		bool free() { return flag; }
		void StopTask();
		int IdTask() { return (task == nullptr) ? 0 : task->ID();}
		int GetTact() { return (task == nullptr) ? 0 : task->GetKolTract();}
		int* GetColor() { return (task == nullptr) ? nullptr : task->color; }
		Task& GetTask() { return *task; }

	};
```

```
public:
	vector<Core>& GetCore() { return Cores; }
	vector<Core> Cores;

	list<Task> Tasks;
private:
	int kolProz, kolFree;
	vector<int> id;
};


class Controller : public Proc {
public:
	struct CoreInfo {
		int Id;
		int KolTact;
		int color[3]{};
		CoreInfo(int id = 0, int tact = 0, int* color = nullptr) {
			this->Id = id;
			this->KolTact = tact;
			if (color != nullptr) {
				for (int i = 0; i < 3; i++)
					this->color[i] = color[i];
			}
		}
	};

	Controller(int kolProc, int Shans = 50, int minKolProc = 1, int maxKolProc = 16, int minKolTact = 1, int  maxKolTact =
20);
	void Tact();
	Proc& GetProc() { return proc; }
	MyQueue<Task> GetTask() { return tasks; }
	MyQueue<Task> GetTaskProc() {
		MyQueue<Task> temp;
		for (auto a : proc.Tasks) {
			temp.push(a);
		}
		return temp;
	}


	int height() { return Height; }
	int width() { return Width; }
	int GetKolProc() { return kolProc; }

private:
	Proc proc;
	MyQueue<Task> tasks;


	int Width, Height;
	vector<vector<CoreInfo>> TablCores;



	int kolProc;
	int Shans;
	int minKolProc, maxKolProc;
	int minKolTact, maxKolTact;
```

```
public:
	vector<vector<CoreInfo>> GetTablCores() {
		for (int i = 0; i < kolProc; i++) {
			int x = (int)(i / Height);
			int y = (int)(i % Height);
			TablCores.at(y).at(x) = CoreInfo(proc.Cores[i].IdTask(),
proc.Cores[i].GetTact(),proc.Cores[i].GetColor());
		}
		return TablCores;
	}
	int GetKol() { return proc.kol; }
	};
```

Cp.cpp:

```
#include"CP.h"

//Task
Task::Task() {
	id = -1;
	kolProc = kolTact = allTact = 0;
}

Task::Task(int kolProc, int kolTact) {
	id = (++kolTask);
	this->kolProc = kolProc;
	this->kolTact = this->allTact = kolTact;
	for (auto& col : color) {
		col = rand() % 256;
	}
}
bool Task::Tact() {
	kolTact--;
	return kolTact == 0;
}

int Task::kolTask = 0;

//Proc
Proc::Proc(int kol) {
	setKolProc(kol);
}
void Proc::setKolProc(int kol) {
	Cores.clear();
	Tasks.clear();
	this->kolFree = this->kolProz = kol;
	for (int i = 0; i < kol; i++) {
		Cores.push_back(Core());
	}
}
bool Proc::SetTask(Task task) {
	if (kolFree < task.GetKolProc())
		return false;
	Tasks.push_back(task);
	int KolNProc = task.GetKolProc();
	for (auto& a : Cores) {
		if (a.SetTask(&Tasks.back()))
			KolNProc--;
		if (KolNProc == 0)
			break;
	}
	kolFree -= task.GetKolProc();
	return true;
}
void Proc::Tact() {
	if (!id.empty()) {
		for (auto& a : Cores) {
			if (!a.free())
				kol++;
			if (!a.free() && find(id.begin(), id.end(), a.IdTask()) != id.end()) {
				a.StopTask();
				kolFree++;
			}
		}
		auto it = Tasks.begin();
```

```
		while (it != Tasks.end()) {
			if (find(id.begin(), id.end(), (*it).ID()) != id.end())
				it = Tasks.erase(it);
			else
				it++;
		}
	}
	id.clear();
	for (auto& a : Tasks) {
		if (a.Tact())
			id.push_back(a.ID());
	}
}

//Proc::Core
class Core;
Proc::Core::Core()
{
	task = nullptr;
	flag = true;
}

Proc::Core::Core(const Core& other)
{
	this->flag = other.flag;
	this->task = other.task;
}


bool Proc::Core::SetTask(Task* task)
{
	if (!flag)
		return false;
	this->task = task;
	flag = false;
	return true;
}

void Proc::Core::StopTask()
{
	task = nullptr;
	flag = true;
}

//Controller
Controller::Controller(int kolProc, int Shans, int minKolProc, int maxKolProc, int minKolTact, int maxKolTact)
{
	srand((int)std::time(NULL));
	this->kolProc = kolProc;
	this->Shans = Shans;
	this->minKolProc = minKolProc;
	this->maxKolProc = maxKolProc;
	this->minKolTact = minKolTact;
	this->maxKolTact = maxKolTact;


	Height = sqrt(kolProc);
	Width = (kolProc % Height) ? kolProc / Height + 1 : kolProc / Height;


	proc = Proc(this->kolProc);
```

```
	TablCores.resize(Height);
	for (int i = 0; i < Height; i++) {
		TablCores[i].resize(Width);
	}
}

void Controller::Tact()
{
	if (rand() % 101 < Shans) {
		int KP = rand() % (maxKolProc - minKolProc + 1) + minKolProc;
		int KT = rand() % (maxKolTact - minKolTact + 1) + minKolTact;
		tasks.push(Task(KP, KT));
	}
	proc.Tact();
	if (!tasks.empty() && proc.SetTask(tasks.top())) {
		tasks.pop();
	}
	}
```

Button:

```
#pragma once
#include <SFML\Graphics.hpp>
using namespace sf;
using namespace std;

class button {
	RectangleShape rectagle;
	Font font;
	Text text;

	Color Hover;
	Color Fill;


public:
	button(int Windht = 100, int Height = 100);
	Color GetColor() { return rectagle.getFillColor(); }
	Vector2f GetSize() { return rectagle.getSize(); }
	Vector2f GetPosition() { return rectagle.getPosition(); }
	void SetColor(Color color);
	void SetHoverColor(Color color) { Hover = color; }
	Text& Text() { return text; }
	void SetPosition(Vector2f vec);
	void draw(RenderWindow* window);
	bool MousHover(RenderWindow* window);

	};




button::button(int Windht, int Height)
{
	rectagle.setPosition(0, 0);
	rectagle.setSize(Vector2f(Windht, Height));
	Fill = Hover = Color::Green;
	rectagle.setFillColor(Fill);

	if (!font.loadFromFile("Times.ttf"))
		throw exception("NON font");
	text.setFont(font);
	text.setCharacterSize(Height - 12);
	text.setFillColor(Color::White);

}


void button::SetColor(Color color)
{
	rectagle.setFillColor(color);
	Fill = color;
	if (Hover != Color::Green)
		Hover = color;

}

void button::SetPosition(Vector2f vec)
{
	rectagle.setPosition(vec);
```

```
	text.setPosition(Vector2f(vec.x + rectagle.getSize().x / 2 - 15, vec.y + rectagle.getSize().y / 2 - 30));
}

void button::draw(RenderWindow* window)
{
	window->draw(rectagle);
	window->draw(text);
}

bool button::MousHover(RenderWindow* window)
{
	int x = Mouse::getPosition(*window).x;
	int y = Mouse::getPosition(*window).y;
	bool flag = true;
	if (x < rectagle.getPosition().x || x > rectagle.getPosition().x + rectagle.getSize().x)
		flag = false;
	if (y < rectagle.getPosition().y || y > rectagle.getPosition().y + rectagle.getSize().y)
		flag = false;
	flag ? rectagle.setFillColor(Hover) : rectagle.setFillColor(Fill);
	return flag;
	}
```

PanelProc:

```
#pragma once
#include <SFML\Graphics.hpp>
#include "..\Elements\button.h"
#include "..\Elements\CP.h"
using namespace sf;
using namespace std;

class ProcV {
	RectangleShape rectagle;
	Font font;
	Text idProc;
	Text info;
public:
	ProcV(int Windht = 100, int Height = 100, int nomer = 0);
	void draw(RenderWindow* window);
	void SetIdProc(wstring s);
	void SetInfo(wstring s);
	Text& IdProc() { return idProc; }
	Text& Info() { return info; }
	void SetPosition(Vector2f vector);
	Vector2f getPosition() { return rectagle.getPosition(); }
	void setColor(Color color) {
		rectagle.setFillColor(color);
		info.setFillColor(Color(255 - color.r, 255 - color.g, 255 - color.b));
		idProc.setFillColor(Color(255 - color.r, 255 - color.g, 255 - color.b));
	}
};


class PanelProc {
	Font font;
	Text text;
	RectangleShape rectagle;
public:
	PanelProc(int Windht, int Height, Controller* controller);
	void draw(RenderWindow* window);
	void SetText(wstring s);
	Text& Text() { return text; }
	void SetPosition(Vector2f vector);
	Vector2f getSize() { return rectagle.getSize(); }
	Vector2f getPosition() { return rectagle.getPosition(); }


	vector<ProcV*> proc;
	void SetProc(Controller* controller);
	};







PanelProc::PanelProc(int Windht, int Height,Controller* controller)
{

	rectagle.setPosition(0, 0);
	rectagle.setSize(Vector2f(Windht - 8, Height));
```

```
	rectagle.setPosition(Vector2f(4, 80));
	rectagle.setFillColor(Color(17, 24, 38));
	rectagle.setOutlineThickness(3);
	rectagle.setOutlineColor(Color(77,77,77));


	if (!font.loadFromFile("Times.ttf"))
		throw exception("NON font");

	text.setFont(font);
	text.setCharacterSize(25);
	text.setString(L"Процессор:");
	text.setPosition(rectagle.getPosition() + Vector2f((rectagle.getSize().x - text.getLocalBounds().width) / 2,-10));
	text.setFillColor(Color(117,117,120));


	float x = (Windht - 2) / controller->width();
	float y = (Height - 20) / controller->height();
	int h = controller->width();
	proc.resize(controller->GetKolProc());
	for (int i = 0; i < controller->GetKolProc();i++) {
		proc[i] = new ProcV(x * 0.9, y * 0.9, i + 1);
		Vector2f temp = Vector2f((i % h) * x + x * 0.05 + 0.5, (i / h) * y + y * 0.05 + 20);
		proc[i]->SetPosition(rectagle.getPosition() + temp);
	}
}

void PanelProc::draw(RenderWindow* window)
{
	window->draw(rectagle);
	window->draw(text);

	for (auto& a : proc) {
		a->draw(window);
	}
}


void PanelProc::SetText(wstring s)
{
	text.setString(s);
	text.setPosition(rectagle.getSize().x - text.getLocalBounds().width - 10, rectagle.getPosition().y + rectagle.getSize().y
/ 2 - 50);
}

void PanelProc::SetPosition(Vector2f vector)
{
	rectagle.setPosition(rectagle.getPosition() + vector);
	text.setPosition(text.getPosition() + vector);

	for (auto& a : proc) {
		a->SetPosition(vector);
	}
}

void PanelProc::SetProc(Controller* controller)
{
	vector<vector<Controller::CoreInfo>> temp = controller->GetTablCores();
	int h = controller->height();
	for (int i = 0; i < controller->GetKolProc(); i++) {
		string s = to_string(temp[i % h][i / h].Id) + ":" + to_string(temp[i % h][i / h].KolTact);
		wstring ws;
```

```
                for (auto a : s) {
                        ws += a;
                }
                proc[i]->SetInfo(ws);
                proc[i]->setColor(Color(temp[i % h][i / h].color[0], temp[i % h][i / h].color[1], temp[i % h][i / h].color[2]));

        }
}

ProcV::ProcV(int Windht, int Height, int Nomer)
{
        rectagle.setPosition(0, 0);
        rectagle.setSize(Vector2f(Windht, Height));
        rectagle.setFillColor(Color(180,180,180));
        rectagle.setOutlineThickness(3);
        rectagle.setOutlineColor(Color(77, 77, 77));


        if (!font.loadFromFile("Times.ttf"))
                throw exception("NON font");

        int mini = min(Height, Windht);

        idProc.setFont(font);
        idProc.setCharacterSize(mini* 0.2);
        idProc.setString(L"Ядро: " + to_string(Nomer));
        idProc.setPosition(rectagle.getPosition() + Vector2f((rectagle.getSize().x - idProc.getLocalBounds().width) / 2,10));
        idProc.setFillColor(Color(117, 117, 120));
        info.setFont(font);
        info.setCharacterSize(mini * 0.2);
        info.setString(L"0:0");
        info.setPosition(rectagle.getPosition() + Vector2f((rectagle.getSize().x - info.getLocalBounds().width) / 2, (rec-
tagle.getSize().y - info.getLocalBounds().height) / 2));
        info.setFillColor(Color(117, 117, 120));
}
void ProcV::draw(RenderWindow* window)
{
        window->draw(rectagle);
        window->draw(idProc);
        window->draw(info);
}

void ProcV::SetIdProc(wstring s)
{
        idProc.setString(s);
        idProc.setPosition(rectagle.getPosition() + Vector2f((rectagle.getSize().x - idProc.getLocalBounds().width) / 2, 10));
}

void ProcV::SetInfo(wstring s)
{
        info.setString(s);
        info.setPosition(rectagle.getPosition() + Vector2f((rectagle.getSize().x - info.getLocalBounds().width) / 2, (rec-
tagle.getSize().y - info.getLocalBounds().height) / 2));
}

void ProcV::SetPosition(Vector2f vector)
{
        rectagle.setPosition(rectagle.getPosition() + vector);
        idProc.setPosition(idProc.getPosition() + vector);
        info.setPosition(info.getPosition() + vector);
        }
```

PanelQoueu:

```
#pragma once
#include "..\Elements\button.h"
#include "..\Elements\CP.h"
using namespace sf;
using namespace std;


class TaskV {
	RectangleShape rectagle;
	Font font;
	Text info;
public:
	TaskV(int Windht = 100, int Height = 100);
	void draw(RenderWindow* window);
	void SetInfo(wstring s);
	Text& Info() { return info; }
	void SetPosition(Vector2f vector);
	Vector2f getPosition() { return rectagle.getPosition(); }
	void setColor(Color color) {
		rectagle.setFillColor(color);
		info.setFillColor(Color(255 - color.r, 255 - color.g, 255 - color.b));
	}
};


class PanelQueue {
	Font font;
	Text text;
	RectangleShape rectagle;
public:
	PanelQueue(int Windht, int Height, wstring s);
	void draw(RenderWindow* window);
	void SetText(wstring s);
	Text& Text() { return text; }
	void SetPosition(Vector2f vector);
	Vector2f getSize() { return rectagle.getSize(); }
	Vector2f getPosition() { return rectagle.getPosition(); }
	void getTasks(MyQueue<Task> tasks);


	vector<TaskV*> task;
	};







PanelQueue::PanelQueue(int Windht, int Height, wstring s)
{

	rectagle.setPosition(0, 0);
	rectagle.setSize(Vector2f(Windht - 8, Height));
	rectagle.setPosition(Vector2f(4, 80));
	rectagle.setFillColor(Color(17, 24, 38));
	rectagle.setOutlineThickness(3);
	rectagle.setOutlineColor(Color(77, 77, 77));


	if (!font.loadFromFile("Times.ttf"))
```

```
		throw exception("NON font");

	text.setFont(font);
	text.setCharacterSize(25);
	text.setString(s);
	text.setPosition(rectagle.getPosition() + Vector2f((rectagle.getSize().x - text.getLocalBounds().width) / 2, -10));
	text.setFillColor(Color(117, 117, 120));


}

void PanelQueue::draw(RenderWindow* window)
{
	window->draw(rectagle);
	window->draw(text);

	for (auto& a : task) {
		a->draw(window);
	}
}


void PanelQueue::SetText(wstring s)
{
	text.setString(s);
	text.setPosition(rectagle.getSize().x - text.getLocalBounds().width - 10, rectagle.getPosition().y + rectagle.getSize().y
/ 2 - 50);
}

void PanelQueue::SetPosition(Vector2f vector)
{
	rectagle.setPosition(rectagle.getPosition() + vector);
	text.setPosition(text.getPosition() + vector);
}

void PanelQueue::getTasks(MyQueue<Task> tasks)
{
	for (auto& a : task) {
		delete a;
	}
	task.clear();
	auto it = MyQueue<Task>::iterator();
	it = tasks.begin();
	int i = 0;
	while (it != tasks.end()) {
		task.push_back( new TaskV(rectagle.getSize().x - 10,40 ));
		string s = to_string((*it).ID())+ ":" + to_string((*it).GetAllTact()) + ":" + to_string((*it).GetKolProc());
		wstring ws;
		for (auto a : s) {
			ws += a;
		}
		task.back()->SetInfo(ws);
		task.back()->setColor(Color((*it).color[0], (*it).color[1], (*it).color[2]));
		task.back()->SetPosition(rectagle.getPosition() + Vector2f(5,25 + i * 45));
		it++;
		i++;
		if (45 * (i + 1) + 25 > rectagle.getSize().y)
			break;
	}
}
```

```
TaskV::TaskV(int Windht, int Height)
{
	rectagle.setPosition(0, 0);
	rectagle.setSize(Vector2f(Windht, Height));
	rectagle.setFillColor(Color(180, 180, 180));
	rectagle.setOutlineThickness(3);
	rectagle.setOutlineColor(Color(77, 77, 77));


	if (!font.loadFromFile("Times.ttf"))
		throw exception("NON font");


	info.setFont(font);
	info.setCharacterSize(Height * 0.5);
	info.setString(L"0:0");
	Vector2f temp = rectagle.getPosition();
	info.setPosition(Vector2f((rectagle.getSize().x - info.getLocalBounds().width) / 2, (rectagle.getSize().y - info.get-
LocalBounds().height) / 2 ) + temp);
	info.setFillColor(Color(117, 117, 120));
}

void TaskV::draw(RenderWindow* window)
{
	window->draw(rectagle);
	window->draw(info);
}

void TaskV::SetInfo(wstring s)
{
	info.setString(s);
	info.setPosition(rectagle.getPosition() + Vector2f((rectagle.getSize().x - info.getLocalBounds().width) / 2, (rec-
tagle.getSize().y - info.getLocalBounds().height) / 2));
}

void TaskV::SetPosition(Vector2f vector)
{
	rectagle.setPosition(rectagle.getPosition() + vector);
	info.setPosition(info.getPosition() + vector);
	}
```

MainForm:

```
#pragma once
#include <iostream>
#include <SFML\Graphics.hpp>
#include "Elements\button.h"
#include "Blocks\PanelProc.h"
#include "Blocks\PanelQueue.h"
using namespace sf;
using namespace std;

class MainForm
{
	RenderWindow* window;

	void Update();
	void Draw();
	void Init();
public:
	MainForm(int Windht = 2000, int Height = 1300);
	~MainForm();
	void Start();

private:
//Объекты
	button* exit;
	RectangleShape PanelMenu;
	Text name;
	Font font;


	PanelProc* panelProc;
	PanelQueue* panelQueue;
	PanelQueue* panelTask;

	Controller* controller;

};
```

```
#include "MainForm.h"

void MainForm::Update()
{
	sf::Event event;
	while (window->pollEvent(event))
	{
		if (event.type == sf::Event::Closed)
			window->close();
		if (event.type == Event::MouseButtonReleased) {
			if (event.mouseButton.button == Mouse::Left) {
				if (exit->MousHover(window)) {
					window->close();
				}
			}
		}
	}

	exit->MousHover(window);
```

```
}

void MainForm::Draw()
{
	window->clear(Color(28,32,42));

	window->draw(PanelMenu);

	exit->draw(window);

	panelProc->draw(window);
	panelQueue->draw(window);
	panelTask->draw(window);


	window->draw(name);

	window->display();
}

void MainForm::Init()
{
	exit = new button(40, 50);
	exit->SetColor(Color(28, 32, 42));
	exit->Text().setString("X");
	exit->SetPosition(Vector2f(Vector2f(window->getSize().x - exit->GetSize().x, 0)));
	exit->SetHoverColor(Color::Red);

	if (!font.loadFromFile("Times.ttf"))
		throw exception("NON font");

	name.setFont(font);
	name.setCharacterSize(30);
	name.setPosition(Vector2f(30, 10));
	name.setFillColor(Color::White);
	name.setString(L"Процессор");


	PanelMenu.setSize(Vector2f(window->getSize().x, 60));
	PanelMenu.setFillColor(Color(17, 24, 38));



	int kolProc = 16;
	controller = new Controller(kolProc, 50, 1, kolProc, 1, 10);


	panelProc = new PanelProc(1000, 1000, controller);
	panelProc->SetPosition(Vector2f(500, 0));

	panelQueue = new PanelQueue(200, 1000,L"Очередь:");
	panelQueue->SetPosition(Vector2f(1755, 0));


	panelTask = new PanelQueue(200, 1000,L"Задачи:");
	panelTask->SetPosition(Vector2f(1545, 0));

}

MainForm::MainForm(int Windht, int Height)
```

```
{
	window = new RenderWindow(VideoMode(Windht, Height), L"Процессор", Style::None);
	window->setPosition(Vector2i(100, 100));
	this->Init();
}

MainForm::~MainForm()
{
	delete window;
	delete exit;

	delete panelProc;
	delete panelQueue;
	delete panelTask;
}

void MainForm::Start()
{
	//window->setFramerateLimit(60);
	while (window->isOpen()) {
		Update();

		controller->Tact();
		panelProc->SetProc(controller);
		panelQueue->getTasks(controller->GetTask());
		panelTask->getTasks(controller->GetTaskProc());

		Draw();

		chrono::milliseconds timespan(400);
		this_thread::sleep_for(timespan);
	}

}
```