Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования «Алтайский государственный технический университет им. И.И. Ползунова»

Факультет Информационных технологий
Кафедра Прикладной математики
Специальность Программная инженерия
(направление, профиль)

Курсовой проект
защищен с оценкой _____
А.С. Ненайденко
(подпись руководителя проекта) (инициалы, фамилия)
“____” _____ 2022 г.

КУРСОВОЙ ПРОЕКТ

Моделирование поведения мультизадачной системы пакетного режима с фиксированным числом задач и дисциплиной обслуживания FCFS.
(тема курсового проекта)

Пояснительная записка

по дисциплине Курсовая работа по операционным системам

КП 09.03.04.1.000 ПЗ
(обозначение документа)

Студент группы ПИ-92 Шинтяпин И.И. 30.05.2022
(фамилия, имя, отчество) (подпись) (дата)
Руководитель
проекта к.т.н., доцент А.С. Ненайденко
(должность, ученое звание) (подпись) (инициалы, фамилия)

БАРНАУЛ 2022

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «Алтайский государственный технический университет

имени И.И. Ползунова»

Факультет информационных технологий

Кафедра «Прикладная математика»

З А Д А Н И Е

на курсовой проект по дисциплине «Курсовая работа по операционным системам»

студенту группы ПИ-92 Шинтяпину Илье Игоревичу

Тема курсового проекта: «Моделирование поведения мультизадачной системы пакетного режима с фиксированным числом задач и дисциплиной обслуживания FCFS».

Календарный план работы:

№ этапа	Содержание этапа	Недели семестра
1	Получение задания	1
2	Описание предметной области и постановка задачи	2
3	Проектирование программы	3-4
4	Реализация программы	5-13
5	Оформление пояснительной записки	14
6	Защита курсового проекта	15-16

Руководитель проекта доцент, к.т.н. _____ А.С. Ненайденко
подпись

Дата выдачи задания «11» февраля 2022 г.
число месяц год

Задание принял к исполнению _____Шинтяпин И.И.
подпись

Содержание

Введение

Мультизадачный режим операционной системы обеспечивает параллельную обработку нескольких процессов (то есть решать несколько задач одновременно), при этом задачи распределяют между собой общие ресурсы (resources sharing), а также выполняют планирование (scheduling) задачи в очереди исполнения.

Существует 2 типа многозадачности:

- **Процессная многозадачность**, основанная на процессах — одновременно выполняющихся программах. Здесь программа — наименьший элемент управляемого кода, которым может управлять планировщик операционной системы.
- **Поточная многозадачность**, основанная на потоках. Наименьший элемент управляемого кода — поток (одна программа может выполнять 2 и более задачи одновременно).

Самые простые многозадачные среды обеспечивают чистое «разделение ресурсов» — за каждой задачей закрепляют определенный участок памяти, и задача выполняется в определенные интервалы времени. Более развитые многозадачные системы обеспечивают распределение ресурсов динамически, когда задача стартует в памяти или покидает память в зависимости от ее приоритета и от стратегии системы. Некоторые особенности таких мультизадачных систем:

- Каждая задача имеет свой приоритет, в соответствии с которым получает процессорное время и память
- Система организует очереди задач так, чтобы все задачи получили ресурсы, в зависимости от приоритетов и стратегии системы
- Система организует обработку прерываний, по которым задачи могут активироваться, деактивироваться и удаляться
- По окончании положенного кванта времени ядро временно переводит задачу из состояния выполнения в состояние готовности, отдавая ресурсы другим задачам
- Система обеспечивает защиту адресного пространства задачи от несанкционированного вмешательства других задач
- Система распознаёт сбои и зависания отдельных задач и прекращает их
- Система решает конфликты доступа к ресурсам и устройствам, не допуская тупиковых ситуаций общего зависания от ожидания заблокированных ресурсов
- Система обеспечивает коммуникацию между процессами

Такой подход можно реализовать с помощью мультипрограммирования - способа организации выполнения нескольких программ на одном компьютере. Разделяют мультипрограммирование в пакетных системах, системах реального времени и в системах разделения времени.

1. Описание предметной области и постановка задачи

1.1. Описание предметной области

Режим классического мультипрограммирования, или пакетной обработки, применительно к однопроцессорным компьютерам является основой для построения всех других видов многопрограммной работы. Режим имеет целью обеспечить минимальное время обработки пакета заданий и максимально загрузить процессор. Задачи, планируемые к выполнению, называются пакетом. Переключение между задачами в пакетном режиме инициируется выполняющейся в данный момент задачей, поэтому промежутки времени выполнения той или иной задачи не определены. Пакетное задание фактически представляет собой список запускаемых программ с указанием параметров запуска и входных данных.
Пакет заданий упорядочивается в соответствии с приоритетами заданий, и обслуживание программ ведется в порядке очередности. Обычно процессор обслуживает наиболее приоритетную программу. Как только ее решение завершается, процессор переключается на следующую по приоритетности программу.

Главные задачи алгоритмов планирования пакетных систем:

- производительность — выполнение максимального количества заданий в час;
- оборотное время — минимизация времени между представлением задачи и ее завершением;
- использование центрального процессора — поддержка постоянной загруженности процессора.

Существует множество алгоритмов планирования в многозадачных системах: JSF - Job Shortest First, циклического планирования Round Robin, «справедливое» планирование итд. Одним из самых первых является алгоритм FCFS:

Алгоритм планирования FCFS (первый пришел – первый обслужен) – наиболее простой в плане понимания и программирования алгоритм диспетчеризации, суть которого заключается в обслуживании процессов в порядке их поступления в очередь, причем каждый процесс выполняется до своего завершения (за исключением случаев, когда процесс переходит в заблокированное состояние). Отслеживание готовых процессов осуществляется с помощью единого связанного списка. Выбор следующего выполняемого процесса сводится к извлечению одного процесса из начала очереди. Добавление нового задания или разблокированного процесса сводится к присоединению его к концу очереди. Фактически это алгоритм без переключения.

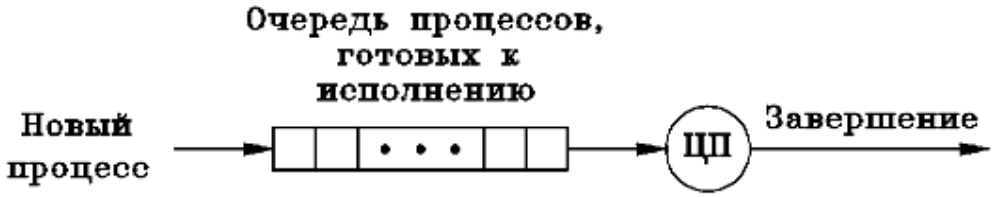

Рис. 1.1 – планирование процессов по принципу FCFS

В случае перехода какой-либо задачи в заблокированное состояние, она, с сохранением своего состояния, попадает в очередь заблокированных задач, при этом процессору, из общей очереди задач, предоставляется следующая задача на выполнение. После завершения времени блокировки, задача попадает в конец общей очереди задач, готовых к выполнению.

Главным недостатком алгоритма планирования FCFS является возможность монополизации времени одним из процессов, требующего больших затрат процессорного времени. Также, стоит заметить, что среднее время ожидания может неограниченно расти по мере того, как система приближается к пределу своей загруженности.

В современных системах алгоритм FCFS почти не используется самостоятельно в качестве основной дисциплины обслуживания, а чаще комбинируется с другими схемами. Например, во многих схемах реализуется диспетчеризация процессов в соответствии с их приоритетами, однако процессы с одинаковыми уровнями приоритета планируются по принципу FCFS.

1.2. Постановка задачи

В рамках реализации данной модели будет создан главный класс «System», включающий в себя все необходимые классы и структуры, необходимые для моделирование поведения мультизадачной системы. Сами задачи (Tasks) должны храниться в одной очереди (Queue) процессов. Так как задач фиксированное кол-во, они будут инициализироваться при запуске системы, и их нельзя будет добавлять по ходу работы программы, причем кол-во этих задач и их сложность будет вводить пользователь при старте программы.
Также в данной модели будет различное кол-во процессоров (Processor), которые необходимы для обработки этих задач, от 1 до 12 и выбирать это кол-во будет также пользователь при старте программы. Сами процессоры будут храниться в динамическом массиве, и обращение к ним будет происходить по индексу (от 0 до 11).

В текущий момент времени один процессор может обрабатывать только одну задачу, после завершения которой, из общей очереди берется следующая задача и начинает выполняться данным процессором. Так как суть этой работы заключается в моделировании поведения мультизадачной системы, то обработка задач (процессов) будет эмулироваться в методе «execution()» класса «Task», путем последовательного отнимания от общей сложности задачи константного числа (один вызов функции - одно отнимание). По достижении нулевого значения (установки флага is_finish=true), задача будет считаться завершенной. Прогресс завершения задачи будет пересчитываться в проценты, и выводиться на экран пользователя.

Сложность задачи будет являться ключевым параметром времени выполнения данной задачи, то есть предполагается, чем выше сложность, тем задача более трудоемкая и требует большего времени выполнения. Чтобы наглядно видеть в течение некоторого времени сам процесс обработки задач, скорость их обработки будет искусственно замедляться. Это будет реализовано за счет добавления таймера в систему. Процесс обработки задач будет запускаться на каждый тик таймера с интервалом в одну секунду. Фактически сложность задачи – это время ее выполнения.

Кроме того, в системе будет реализована возможность ручной блокировки любой выполняющейся задачи на определенный срок, который будет определяться рандомом и не будет превышать общий срок выполнения этой задачи. Так, заблокированные задачи, с

сохранением своего состояния, будут добавляться в соответствующий массив (реализован за счет очереди Queue<Tasks> с перегрузкой оператора [ ]) – *block_tasks, на срок своей блокировки. После чего, снова будут добавлены в конец общей очереди задач, готовых к выполнению.

Очередь задач будет реализована двумя классами: внешним - «Queue» и вложенным классом «Node» - узел очереди. Каждый узел (элемент) очереди хранить в себе ссылку на следующий узел, то есть будет применяться концепция односвязного списка с методами: добавления в конец очереди, удаления из начала очереди, удаление по индексу и перегрузкой оператора [ ].

При запуске данной программы будет запущено стартовое окно, где пользователю будет предложено выбрать кол-во процессов (1 до 12), кол-во задач (от 1 до 1000) и порядок определения сложности задач: ручной ввод или автоматическая генерация. После ввода необходимых данных и старта инициализации системы, появиться главная форма программы с выводом всей необходимой информации в интерактивном динамическом режиме. Для каждого процессора будет показано:

- его номер
- %завершения текущей задачи
- ID текущей задачи
- Кол-во выполненных задач данным процессором
- Кнопка перевода текущей задачи в заблокированное состояние

Кроме того, снизу формы будет размещены дополнительные возможности системы, именно:

- кнопка «Очередь», выводит на экран форму с текущей очередью задач и их сложность
- кнопка «Блок процессы», выводит на экран форму с текущим массивом заблокированных процессов
- Кнопка «Выполненные процессы» выводит на экран форму с массивом выполненных задач
- кнопка «Пауза» приостанавливает работу всей системы
- кнопка «Выход»
- также вывод состояния и общего времени работы системы

Примерная схема главного окна приложения:

CPU №	% завершения задачи	ID задачи	Сложность	Количество выполненных задач	Блокировка задачи
CPU 1	45%	11	100	10	Блокировка
CPU 2	25%	15	50	8	Блокировка
CPU 3	10%	21	45	12	Блокировка
CPU 4	71%	18	120	4	Блокировка
CPU 5	90%	25	80	5	Блокировка
CPU 6	51%	14	95	7	Блокировка
CPU 7	3%	25	54	10	Блокировка

Состояние системы: работает
Время работы системы: 5 мин 21 сек

Количество задач: 7
Количество выполненных задач: 0

Очередь | Заблокированные процессы | Выполненные процессы | Пауза | Выход

Рисунок 1.2 – схема главного окна приложения

По завершении работы системы, в окне будет выведено соответствующее сообщение с указанием времени работы системы, общего числа выполненных задач и суммарной сложности процессов.

2. Проектирование

2.1. Описание структуры классов

1. Класс «**System**» - главный, связующий класс программы, содержащий и управляющий всеми компонентами системы. При остановке работы данного класса, останавливается вся системы в целом

   Поля private:

   - **bool is_pause** – флаг, остановлена система или нет
   - **bool is_finish -** финиш
   - **size_t count_cpu -** кол-во процессоров
   - **My_queue *general_queue_tasks -** указатель на главную очередь задач
   - **My_queue *block_tasks -** указатель на очередь заблокированных задач
   - **My_queue *finish_tasks -** указатель на очередь выполненных задач
   - **processor *cpu -** указатель на массив процессоров
   - **size_t count_tasks –** кол-во задач

   Методы public:

   - **My_system(My_queue *tasks, size_t count_cpu, processor *cpu)** - конструктор с параметрами
   - **~My_system()** - деструктор
   - **void setup()** - установка начальных параметров
   - **bool work()** - метод работы системы
   - **void pause()** - метод паузы
   - **void start()** - возобновление работы
   - **bool get_is_pause()** - геттер для паузы
   - **void block_task(int index)** - блокировка выбранной задачи

2. Класс «**My_queue**» - класс, реализующий очередь задач по концепции односвязного списка с хранением ссылки на первый (root) элемент очереди и размер очереди. Содержит вложенный (inner) класс узла (элемента) очереди «Node»

   Поля private:

   - **Node *head** – указатель на последний элемент очереди
   - **size_t size** – размер очереди

   Методы public:

   - **Queue()** – конструктор класса
   - **Operator [](int index)** – перегрузка оператора [ ]
   - **void push_back(task *t)** – добавление элемента в конец очереди

- **task *pop_front()** – удаление первого элемента
- **Size()** – получить текущий размер очереди
- **task *removeAt(int index)** - удаление элемента по индексу

3. Класс «**Node**» - вложенный класс, узел очереди, содержит ссылку на следующий элемент и сами задачи, хранящиеся в очереди.

   Поля public:

   - **Node *pNext** – указатель на следующий элемент в очереди
   - **task *data** – сами данные

   Методы public:

   - **Node()** – конструктор класса

4. Класс «**Processor**» - один процессор в системе, обеспечивающий обработку задач из общей очереди по принципу FCFS. В системе может быть от одного до двенадцати процессоров.

   Поля private:

   - **int number** – номер процессора
   - **task *current_task** - текущая задача
   - **int count_completed** - кол-во выполненных задач
   - **int current_procent** - процент выполнения текущей задачи
   - **bool is_finish** – флаг, окончания работы процессором

   Методы public:

   - **processor()** - конструткор
   - **processor(int number)** - конструткор с параметрами
   - **~processor()** - деструктор
   - **bool work()** - метод работы процессора
   - **void new_task(task *new_task)** - добавление новой задачи
   - **task* block_task()** - блокировка текущей задачи

5. Класс «**Task**» - класс самой задачи, содержащий все необходимые компоненты для моделирования обработки задачи.

   Поля private:

   - **int ID** - ID задачи
   - **int difficult** - сложность (время выполнения)
   - **bool is_start** - старт
   - **bool is_finish** - финиш
   - **int current_difficult** - оставшееся время выполнения
   - **bool is_block** - заблокирована ли задача
   - **int counter_block** - счетчки блокировки

- **int const_number** - константное число, необходимое для отнимания от сложности в процессе обработки задачи
- **int current_procent** - текущий процент выполнения при блокировке
- **bool was_block** - была ли блокировка

Методы public

- **task(int ID, int difficult)** - конструктор с параметрами
- **~task()** - деструктор
- **bool execution()** - метод моделирования обработки задачи
- **void start()** - старт обработки
- **void block(int current_procent)** - блокировка задачи
- **bool time_block()** - генерация времени блокировки

2.2 Диаграмма классов

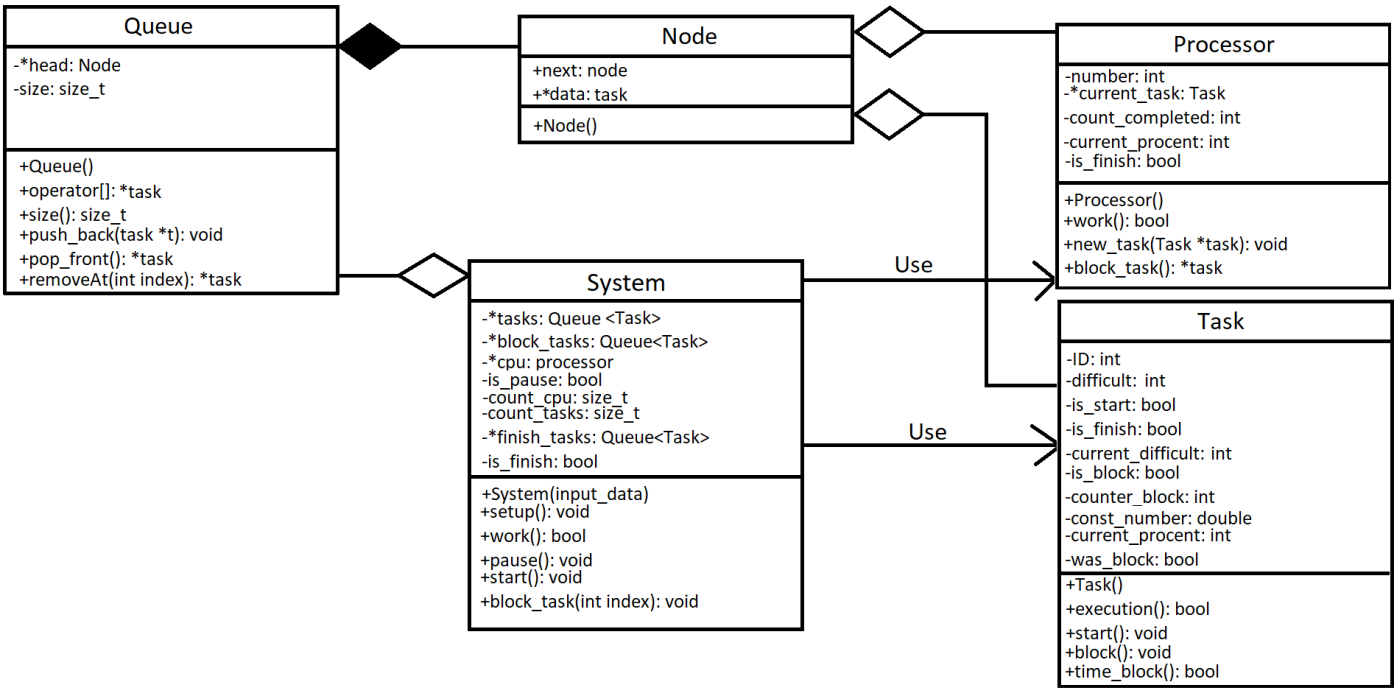

Рисунок 2.1 – диаграмма классов

3. Разработка программного обеспечения

3.1 Инструменты и технологии

Программа написана на одном из самых самых известных и популярных языков программирования C++ с помощью интерфейса программирования приложений Windows Forms.

C++ - компилируемый, статически типизированный язык программирования общего назначения, поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности.

C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также игр.

Windows Forms — интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Приложение Windows Forms представляет собой событийно-ориентированное приложение, поддерживаемое Microsoft .NET Framework. В отличие от пакетных программ, большая часть времени тратится на ожидание от пользователя каких-либо действий, как, например, ввод текста в текстовое поле или клика мышкой по кнопке

3.2 Реализация

При запуске приложения открывается стартовая форма, где пользователю предлагается выбрать количество процессоров в системе, их может быть от одного до двенадцати, ввести количество задач и определиться с выбором установки сложности (времени выполнения задачи) и ID задачи: автоматическая генерация или ручной ввод. В первом случае, будут отображены соответствующие поля для ввода ID и сложности для каждой задачи. Если будут введены некорректные данные, программа сообщит об этом и предложит повторные ввод.

При автоматической генерации, сразу после нажатия кнопки «Далее», открывается главная форма приложения, в центре которой отображена таблица со столбцами: номер процессора, % завершения задачи, ID текущей задачи, сложность данной задачи (время выполнения), количество выполненных задач этим процессором и кнопка блокировки текущей задачи. В случае превышения количества процессоров над количеством задач, в ячейках cpu, которым не досталась задача, будет выводиться значение «NULL».

Под таблицей выводится общая информация о системе: состояние, текущее время работы, общее количество задач в системе, количество выполненных задач.

Внизу формы расположены кнопки управления и вывода информации:

- «Очередь»
- «Блок. процессы»
- «Выполненные процессы»
- «Запуск/Пауза» - при нажатии переводит систему в состояние работы/паузы соответственно
- «Выход» - выход из главной формы с завершением работы системы

При нажатии кнопок «Очередь», «Блок. процессы» и «Выполненные процессы» открывается форма «Queue_form» с выводом очереди процессов готовых к выполнению, заблокированных процессов и завершенных процессов соответственно. По центру данной формы расположена таблица со столбцами: ID задачи, сложность задачи, %завершения задачи и, в случае общей очереди и очереди выполненных процессов, сведение о том была блокировка у данной задачи или нет, а в случае «Блок. процессы» в последнем столбце динамически выводится оставшееся время блокировки задачи в секундах. Чтобы скрыть данную форму необходимо нажать на соответствующую кнопку сверху формы.

Изначально система находится в состоянии паузы и чтобы она начала работать, необходимо нажать на кнопку «Запуск». После этого процессы начнут обработку задач, и данные в таблицах будут динамически обновляться каждый тик таймера, который равен одной секунде.

При нажатии кнопки «блокировка» в строке таблицы, соответствующая задача, обрабатываемая данным процессором, будет переведена в заблокированное состояние на срок, который определяется случайным числом в диапазоне от единицы до сложности данной задачи. После перехода задачи в заблокированное состояние, процессору дается на обработку следующая задача из общей очереди, в случае пустой очереди, процессор переходит в состояние паузы. Сама заблокированная задача, с сохранением своего состояния, на время блокировки, переводится в массив заблокированных задач, который можно увидеть, нажав на кнопку «Блок. процессы». После завершения времени блокировки, задача добавляется в очередь процессов, готовых к выполнению.

После выполнения всех процессов работа системы завершается, на экран, поверх главной формы, выводится форма с таблицей выполненных процессов, которую можно скрыть, нажав соответствующую кнопку вверху формы. Для выхода из приложения необходимо нажать кнопку «Выход» внизу главной формы.

Заключение

Разработанное приложение моделирует поведение мультизадачной системы пакетного режима с фиксированным числом задач и дисциплиной обслуживания FCFS. Основные возможности программы:

- Моделирование обработки задач с динамическим отслеживанием прогресса
- Различное число процессоров, от 1 до 12
- Количество задач ограничивается объемом памяти устройства
- Задачи, готовые к выполнению находятся в очереди и обрабатываются последовательно, в порядке очереди
- Блокировка задач на определенный срок, их отображение в отдельном окне
- Подсчет общего времени работы системы и количества выполненных задач, каждым процессором
- Пауза/Запуск системы по нажатию на соответствующую кнопку
- Вывод выполненных задач в отдельном окне

В дальнейшем возможно расширение функционала системы в виде:

- Добавления звукового сопровождения в процессе функционирования системы
- Реализация графического отображения процесса обработки задач
- Возможность автоматической блокировки задач
- Усложнение процесса обработки задач, путем добавления реальных ресурсо-затратных операций для каждой задачи

Список использованных источников

1. Habr [Электронный ресурс]. – Режим доступа: https://habr.com, свободный.

2. StackOverflow [Электронный ресурс]. – Режим доступа: https://stackoverflow.com, свободный.

3. “MSDN” – информационный сервис для разработчиков [Электронный ресурс] – Режим доступа http://msdn.microsoft.com/ru-ru/ms348103/library, свободный.

4. Буч, Г. Язык UML. Руководство пользователя : руководство / Г. Буч, Д. Рамбо, И. Якобсон. — Москва : ДМК Пресс, 2008. — 496 с. — ISBN 5-94074-334-X.

Приложение А. Тестирование программного обеспечения

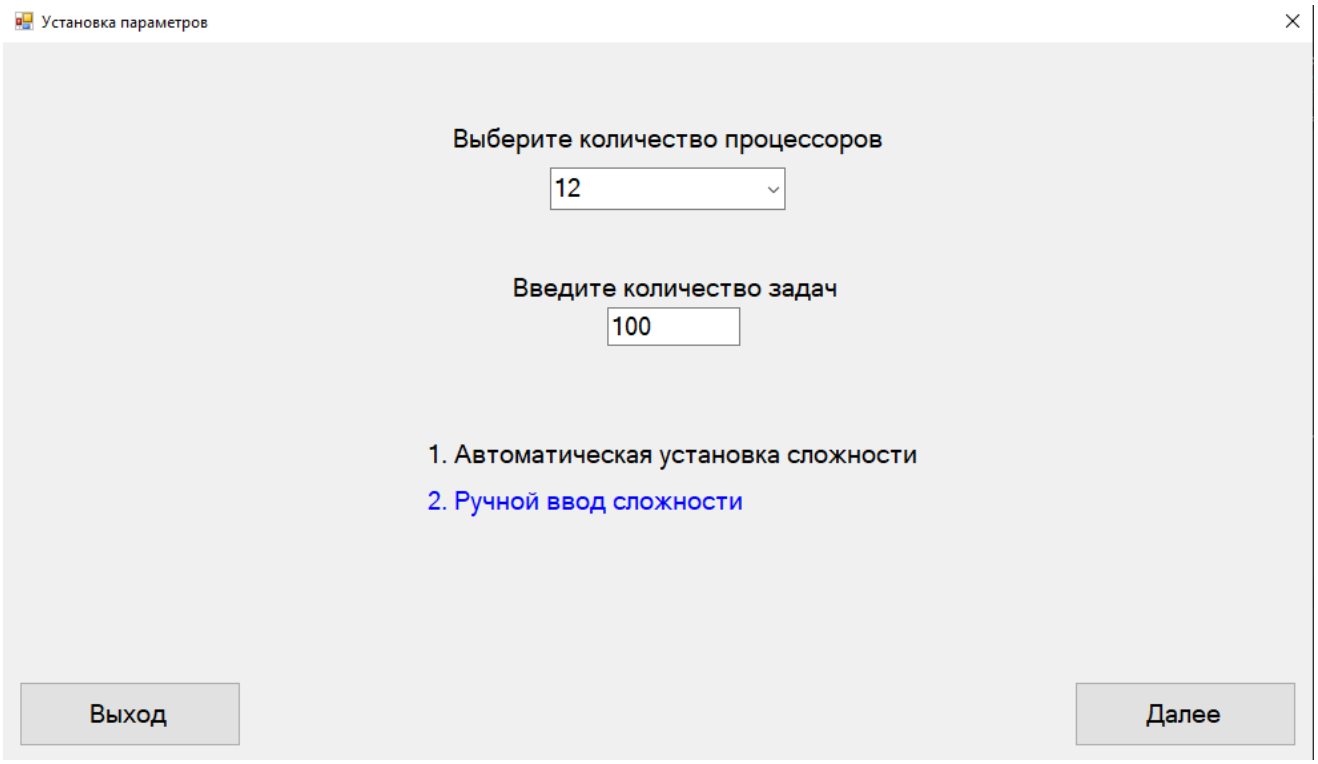

Рисунок А.1 – Стартовая форма программы

Рисунок А.2 – Ручной ввод сложности и ID

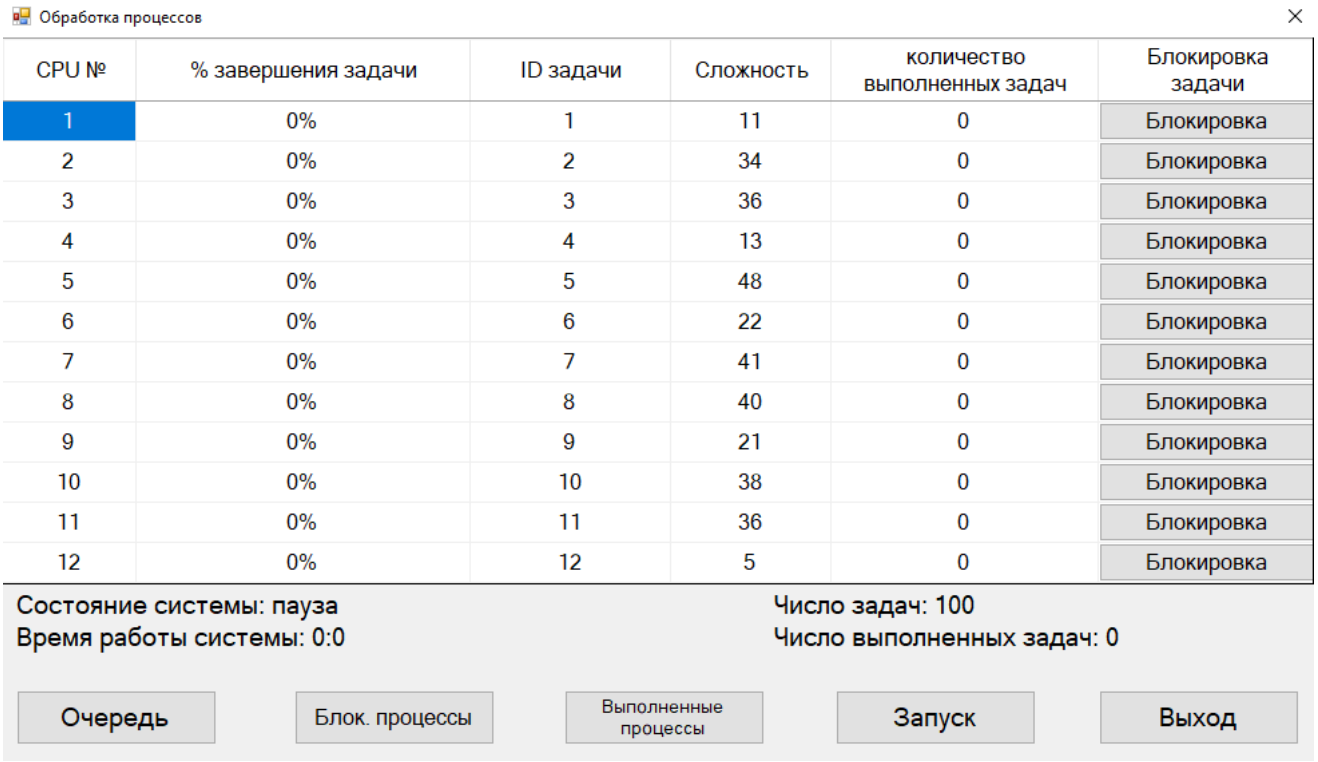

Рисунок А.3 – Главная форма программы, после инициализации параметров

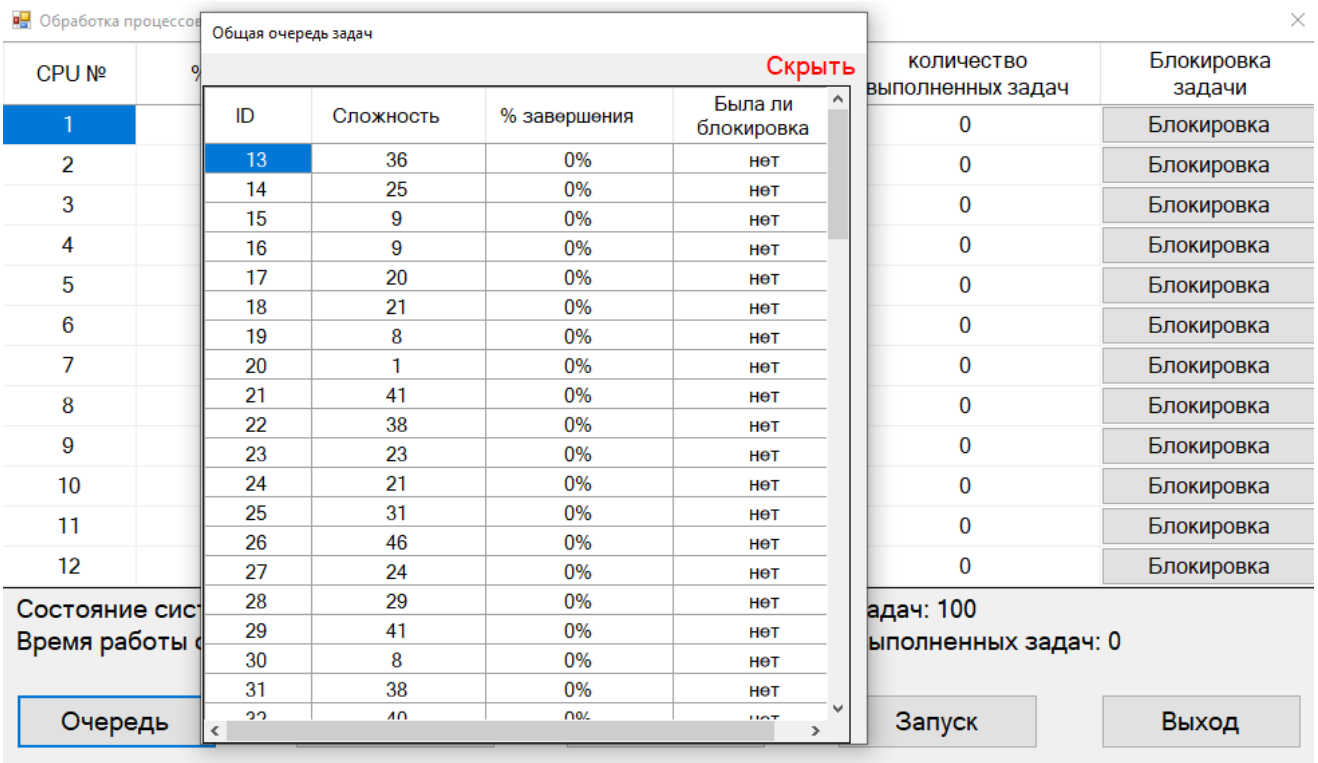

Рисунок А.4 – Очередь задач, готовых к выполнению

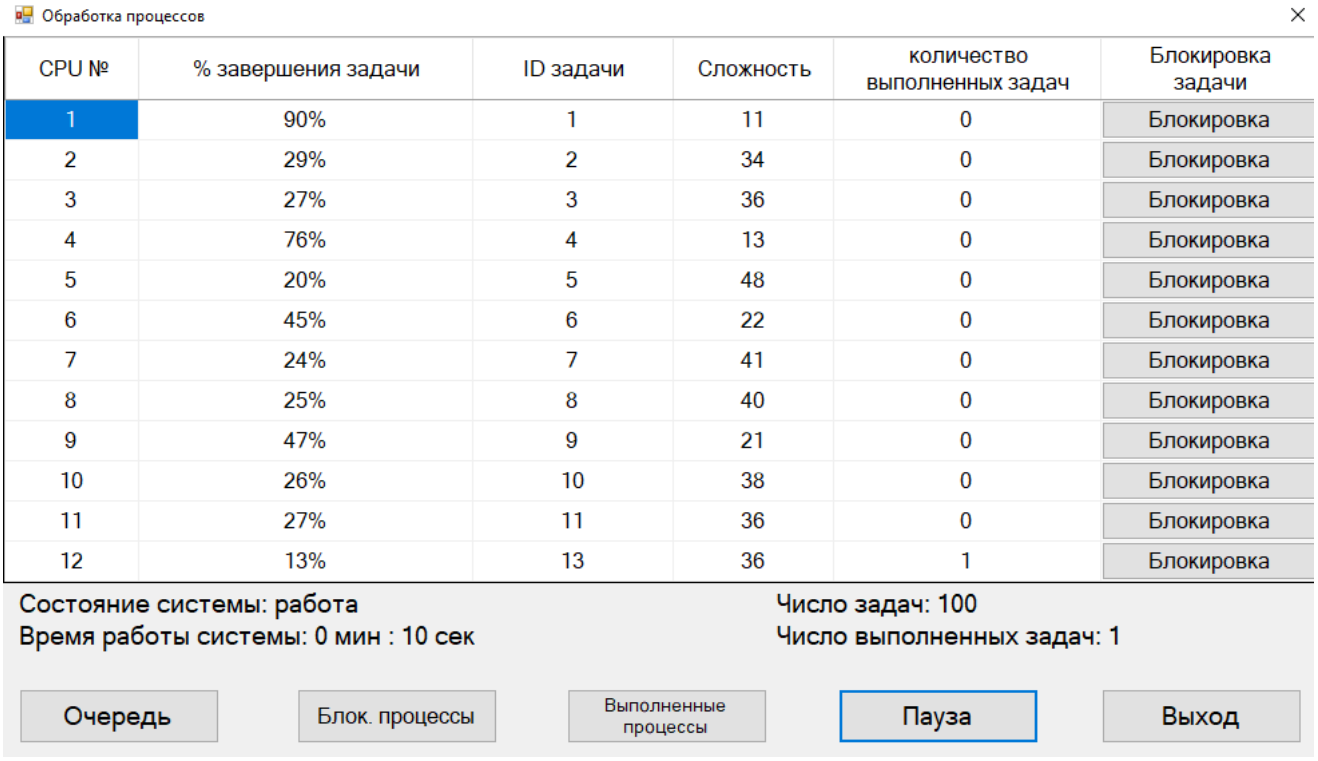

Рисунок А.5 – Процесс обработки задач

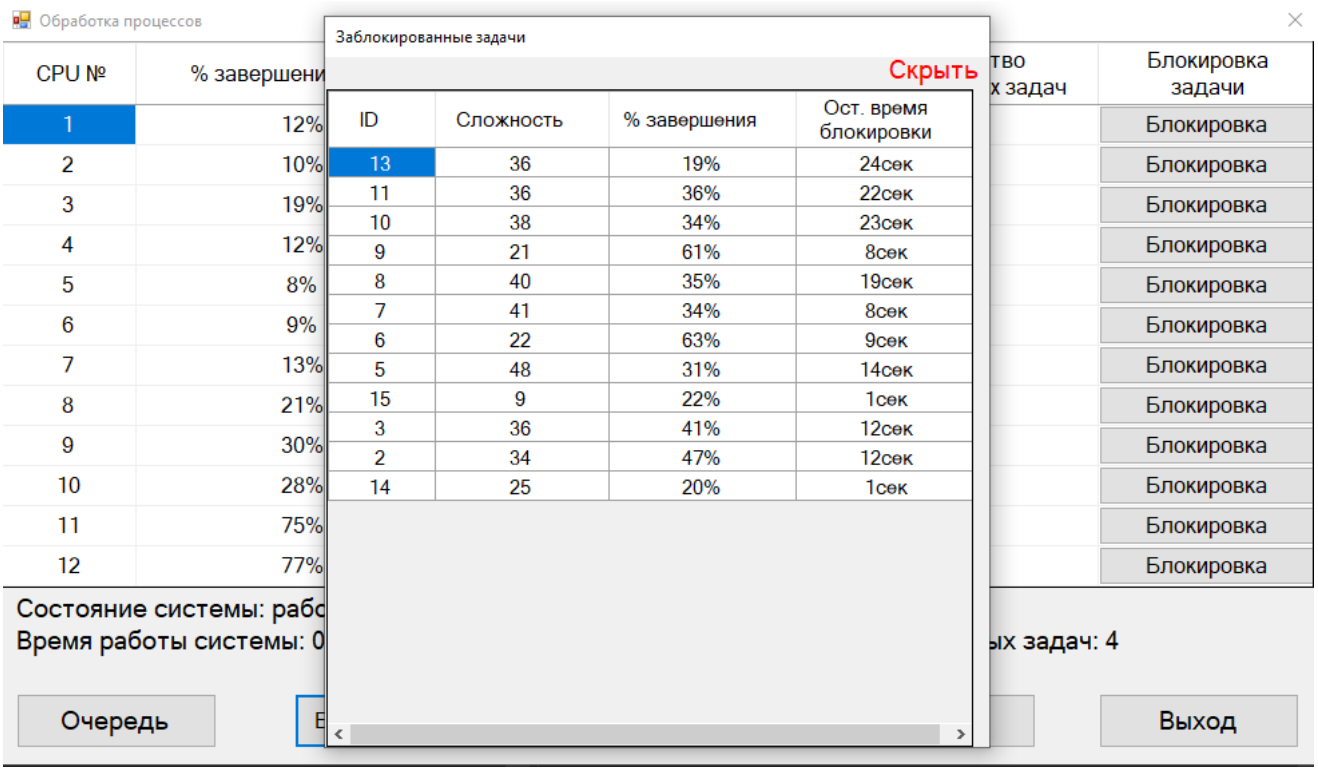

Рисунок А.6 – массив заблокированных задач

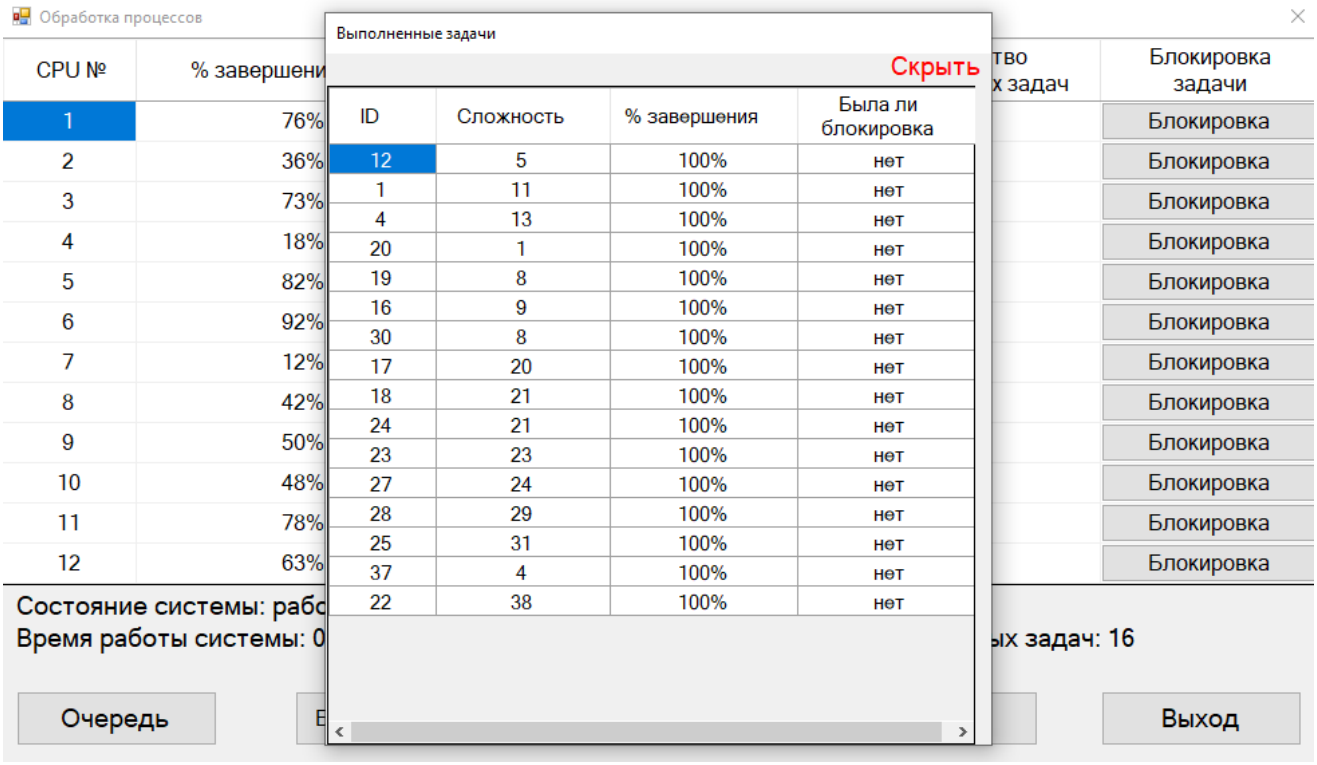

Рисунок А.7 – Таблица выполненных процессов

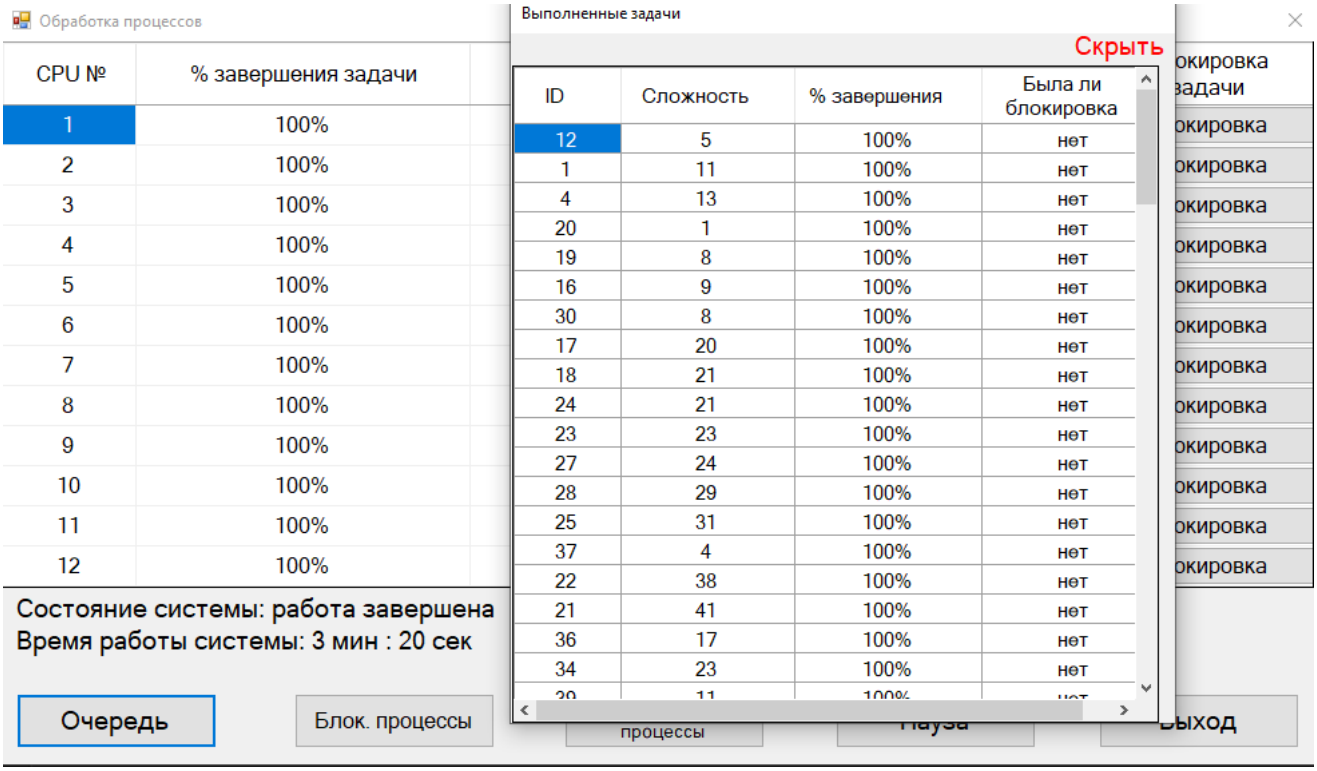

Рисунок А.8 – Работа завершена

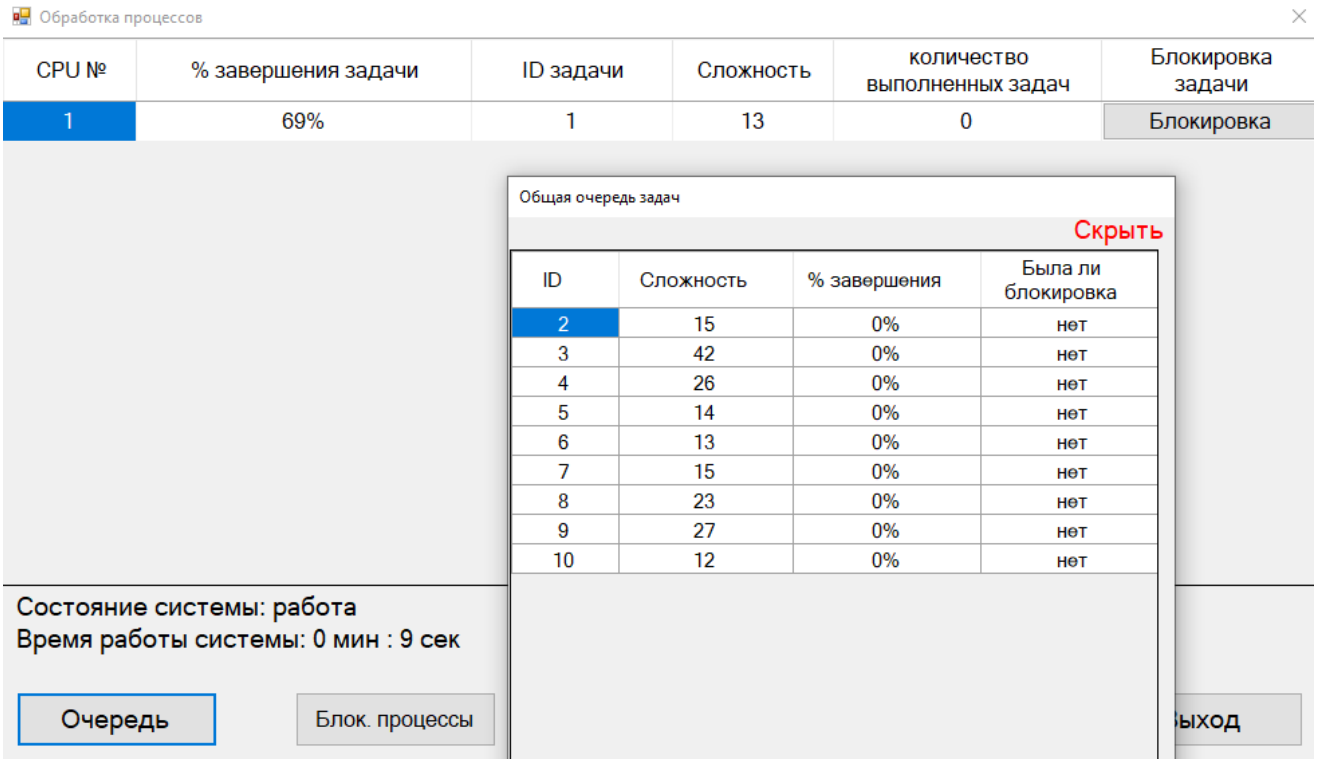

Рисунок А.9 – Случай, когда процессор один, задач много

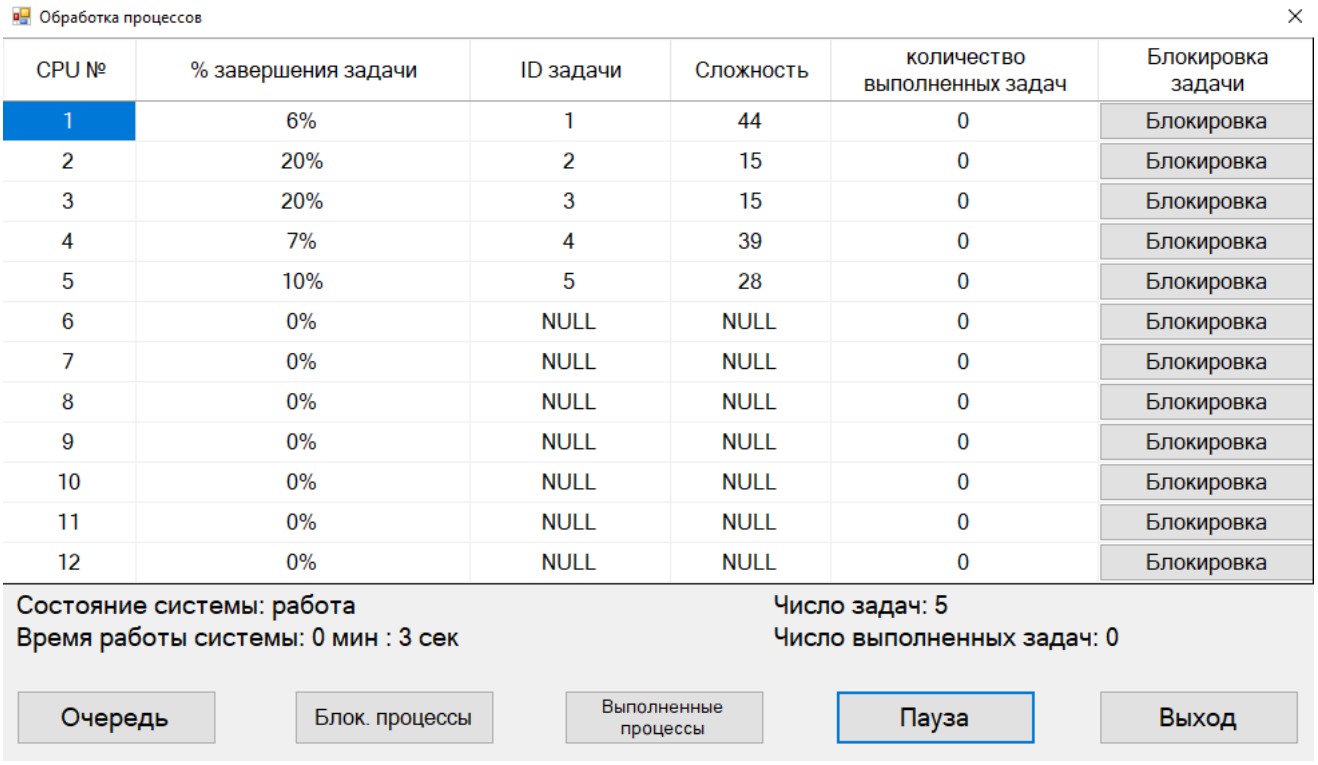

Рисунок А.10 – Случай когда процессоров больше, чем задач

Приложение Б. Исходный код программы

Файл **System.h**

```
#pragma once
#include<iostream>
#include"My_queue.h"
#include"Processor.h"
#include"Task.h"

using namespace std;
// класс система
class My_system
{
private:
	bool is_pause;// пауза
	bool is_finish;// финиш
	size_t count_cpu;// кол-во процессоров
	My_queue *general_queue_tasks;// указатель на главную очередь задач
	My_queue *block_tasks;// указатель на очередь заблокированных задач
	My_queue *finish_tasks;// // указатель на очередь выполненных задач
	processor *cpu;// указатель на массив процессоров
	size_t count_tasks;// кол-во задач
public:
	My_system(My_queue *tasks, size_t count_cpu, processor *cpu);// конструктор с
параметрами
	~My_system();// деструктор
	void setup();// установка начальных параметров
	bool work();// метод работы системы
	void pause();// метод паузы
	void start();// метод начала работы системы
	bool get_is_pause();// геттер для паузы
	void block_task(int index);// блокировка выбранной задачи
	// геттеры
	task* get_task(size_t index);
	task* get_block_task(size_t index);
	processor* get_cpu(size_t index);
	size_t get_count_cpu();
	size_t get_count_general_queue_tasks();
	size_t get_count_block_tasks();
	size_t get_count_finish_tasks();
	size_t get_count_tasks();
	task* get_finish_task(size_t index);
};
```

Файл **System.cpp**

```
#include "System.h"
// конструктор
My_system::My_system(My_queue *tasks, size_t count_cpu, processor *cpu)
{
	this->general_queue_tasks = tasks;
	this->count_cpu = count_cpu;
	this->cpu = cpu;
	is_pause = true;
	is_finish = false;
	block_tasks = new My_queue();
	finish_tasks = new My_queue();
	count_tasks = tasks->Size();
}
//деструктор
My_system::~My_system()
```

```
{
	delete[] cpu;
	delete block_tasks;
	delete finish_tasks;
	delete general_queue_tasks;
}
//установка начальных параметров системы
void My_system::setup()
{
	task *current_task;
	for (int i = 0; i < count_cpu; i++)
	{
		if (general_queue_tasks->Size() == 0)
		{
			break;
		}
		current_task = general_queue_tasks->pop_front();
		cpu[i].new_task(current_task);
	}
}
// метод работы системы
bool My_system::work()
{
	if (is_finish)
		return is_finish;
	task *current_task;
	for (int i = 0; i < block_tasks->Size(); i++)
	{
		current_task = block_tasks->operator[](i);
		if (!current_task->time_block())
		{
			block_tasks->removeAt(i);
			general_queue_tasks->push_back(current_task);
		}
	}
	for (int i = 0; i < count_cpu; i++)
	{
		if (cpu[i].is_finish_task()&&general_queue_tasks->Size() == 0)// пропуск
процессора при выполненной задачи и пустой очереди задач
		{
			continue;
		}
		else if (cpu[i].is_finish_task()&&general_queue_tasks->Size() != 0)// передача
процессору новой задачи при завершении предыдущей
		{
			current_task = general_queue_tasks->pop_front();
			cpu[i].new_task(current_task);
		}
		if (cpu[i].work())
		{
			if (cpu[i].get_task() != nullptr)
			{
				finish_tasks->push_back(cpu[i].get_task());
			}
			if (general_queue_tasks->Size() != 0)
			{
				current_task = general_queue_tasks->pop_front();
				cpu[i].new_task(current_task);
			}
		}
	}
	if (count_tasks == finish_tasks->Size())
	{
		is_finish = true;
	}
```

```
	return is_finish;
}
// метод паузы
void My_system::pause()
{
	is_pause = true;
}
// метод начала работы системы
void My_system::start()
{
	is_pause = false;
}
// геттер для паузы
bool My_system::get_is_pause()
{
	return is_pause;
}
// геттеры
task* My_system::get_task(size_t index)
{
	return general_queue_tasks->operator[](index);
}

task* My_system::get_block_task(size_t index)
{
	return block_tasks->operator[](index);
}

task* My_system::get_finish_task(size_t index)
{
	return finish_tasks->operator[](index);
}

processor* My_system::get_cpu(size_t index)
{
	if (index<0 || index>count_cpu)
	{
		return nullptr;
	}
	return &cpu[index];
}
size_t My_system::get_count_cpu()
{
	return count_cpu;
}
size_t My_system::get_count_general_queue_tasks()
{
	return general_queue_tasks->Size();
}
size_t My_system::get_count_block_tasks()
{
	return  block_tasks->Size();
}
// блокировка выбранной задачи
void My_system::block_task(int index)
{
	task *temp = nullptr;
	if (!cpu[index].is_finish_task())
	{
		temp = cpu[index].block_task();
		if (temp)
		{
			block_tasks->push_back(temp);
		}
	}
```

```
	}
	size_t My_system::get_count_finish_tasks()
	{
		return finish_tasks->Size();
	}
	size_t My_system::get_count_tasks()
	{
		return count_tasks;
}
```

Файл **Processor.h**

```
	#pragma once
	#include<iostream>
	#include"Task.h"
	using namespace std;

	// класс процессора
	class processor
	{
	private:
		int number;// номер
		task *current_task;// текущая задача
		int count_completed;// кол-во выполненных задач
		int current_procent;// процент выполнения текущей задачи
		bool is_finish;// финиш
	public:
		processor();// конструткор
		processor(int number);//конструткор с параметрами
		~processor();// деструктор
		bool work();//метод работы процессора
		void new_task(task *new_task);// добавление новой задачи
		task* block_task();// блокировка текущей задачи
		//геттеры
		int get_count_completed();
		int get_current_procent();
		int get_number();
		task* get_task();
		bool is_finish_task();
		void set_number(int number);
		void set_is_finish();
		bool get_is_finish();
};
```

Файл **Processor.cpp**

```
	#include "Processor.h"

	processor::processor()// конструткор
	{
		this->number = 0;
		current_task = nullptr;
		count_completed = 0;
		current_procent = 0;
		is_finish = false;
	}
	//конструткор с параметрами
	processor::processor(int number)
	{
		this->number = number;
		current_task = nullptr;
		count_completed = 0;
		current_procent = 0;
		is_finish = false;
```

```
}
// деструктор
processor::~processor()
{
	if (current_task)
	{
		delete current_task;
	}
}
//метод работы процессора
bool processor::work()
{
	if (!current_task)
		return true;
	if (current_task->execution())
		count_completed++;
	current_procent = ((current_task->get_difficult() -
current_task->get_current_difficult()) * 100) / current_task->get_difficult();
	current_task->set_current_procent(current_procent);
	return current_task->get_is_finish();
}
// добавление новой задачи
void processor::new_task(task *new_task)
{
	current_task = new_task;
	current_procent = current_task->get_current_procent();
}
// блокировка текущей задачи
task* processor::block_task()
{
	if (!current_task)
		return nullptr;
	current_task->block(current_procent);
	task *temp = current_task;
	current_task = nullptr;
	return temp;
}
//геттеры
int processor::get_count_completed()
{
	return count_completed;
}
int processor::get_current_procent()
{
	return current_procent;
}
task* processor::get_task()
{
	return this->current_task;
}
bool processor::is_finish_task()
{
	if (!current_task)
	{
		return true;
	}
	return current_task->get_is_finish();
}
void processor::set_number(int number)
{
	this->number = number;
}
void processor::set_is_finish()
{
	is_finish = true;
```

```
	}
	bool processor::get_is_finish()
	{
		return is_finish;
	}
	int processor::get_number()
	{
		return number;
}
```

Файл **Task.h**

```
	#pragma once
	#include<iostream>
	using namespace System;
	using namespace std;

	class task
	{
	private:
		int ID;// ID задачи
		int difficult;// сложность (время выполнения)
		bool is_start;// старт
		bool is_finish;//финиш
		int current_difficult;// оставшееся время выполнения
		bool is_block;// заблокирована ли задача
		int counter_block;// счетчки блокировки
		int const_number;// константное число, необходимое для отнимания от сложности в процессе
	обработки задачи
		int current_procent;// текущий процент выполнения при блокировке
		bool was_block;// была ли блокировка
	public:
		task(int ID, int difficult);// конструктор с параметрами
		~task();// деструктор
		bool execution();// метод моделирования обработки задачи
		void start();// старт обработки
		void block(int current_procent);// блокировка задачи
		bool time_block();// генерация времени блокировки
		//геттеры
		int get_ID();
		int get_difficult();
		bool get_is_start();
		bool get_is_finish();
		bool get_is_block();
		int get_counter_block();
		int get_current_difficult();
		void set_current_procent(int current_procent);
		int get_current_procent();
		String ^get_state();
};
```

Файл **Task.cpp**

```
	#include "Task.h"

	task::task(int ID, int difficult)// конструктор с параметрами
	{
		this->ID = ID;
		this->difficult = difficult;
		is_start = false;
		is_finish = false;
		current_difficult = difficult;
		is_block = false;
		counter_block = 0;
```

```
	const_number = 1;
	current_procent = 0;
	was_block = false;
}
task::~task()// деструктор
{

}
bool task::execution()// метод моделирования обработки задачи
{
	if (!is_start)
		is_start = true;
	current_difficult -= const_number;
	if (!current_difficult)
		return is_finish = true;
	return is_finish;
}
// старт обработки
void task::start()
{
	is_start = true;
}
// блокировка задачи
void task::block(int current_procent)
{
	Random ^rand = gcnew Random();
	counter_block = rand->Next(1, difficult);
	is_block = true;
	was_block = true;
	this->current_procent = current_procent;
}
// генерация времени блокировки
bool task::time_block()
{
	if (!is_block)
		return is_block;
	counter_block -= const_number;
	if (!counter_block)
		is_block = false;
	return is_block;
}
//геттеры
int task::get_ID()
{
	return ID;
}
int task::get_difficult()
{
	return difficult;
}
bool task::get_is_start()
{
	return is_start;
}
bool task::get_is_finish()
{
	return is_finish;
}
bool task::get_is_block()
{
	return is_block;
}
int task::get_counter_block()
{
	return counter_block;
```

```
	}
	int task::get_current_difficult()
	{
		return current_difficult;
	}
	void task::set_current_procent(int current_procent)
	{
		this->current_procent = current_procent;
	}
	int task::get_current_procent()
	{
		return current_procent;
	}
	String ^task::get_state()
	{
		String ^string;
		if (was_block)
		{
			string = gcnew String("да");
		}
		else
		{
			string = gcnew String("нет");
		}
		return string;
}
```

Файл **My_queue.h**

```
	#pragma once
	#include <iostream>
	#include "Task.h"
	using namespace std;

	class My_queue
	{
	private:
		class Node// узел очереди - класс, содержащий указатель на след элемент и задачу
		{
		public:
			Node *pNext;// указатель на след элемент
			task *data;// задача
			Node(task *data, Node *pNext = nullptr)// констркутор
			{
				this->data = data;
				this->pNext = pNext;
			}
		};
		size_t size;// кол-во элементов в списке
		Node *head;// уазатель на последний добавленный элемент в очереди
	public:
		My_queue();// конструктор
		~My_queue();// деструктор
		void push_back(task *data);// метод добавления данных в конец очереди
		task* pop_front();// удаление элемента с начала очереди
		void clear();// очистка очереди
		int Size()// метод, возвращающий кол-во элементов в списке
		{
			return size;
		}
		task* operator[](const size_t index);// перегрузка оператора []
		task *removeAt(int index);// удаление элемента по индексу
};
```

Файл **My_queue.cpp**

```
#include "My_queue.h"

My_queue::My_queue()// конструктор класса My_queue
{
	size = 0;
	head = nullptr;
}


My_queue::~My_queue()// деструктор очереди
{
	clear();
}

void My_queue::push_back(task *data)// метод добавления данных в конец очереди
{
	if (head == nullptr)// проверка на пустоту списка
	{
		head = new Node(data);
	}
	else
	{
		Node *current = this->head;// current - доп указатель для итерации по списку
		while (current->pNext != nullptr)// переход на последний элемент очереди
		{
			current = current->pNext;
		}
		current->pNext = new Node(data);
	}
	size++;
}

task* My_queue::operator[](size_t index)// перегрузка оператора []
{
	Node *current = this->head;
	for (size_t i = 0; current != nullptr; i++)// поиск нужного элемента
	{
		if (i == index)
		{
			return current->data;
		}
		current = current->pNext;
	}
	task *cr = new task(1,1);
	return cr;
}

task* My_queue::pop_front()// удаление первого элемента очереди
{
	if (size != 0)
	{
		Node *current = head;
		head = head->pNext;
		size--;
		return current->data;
	}
	else
	{
		return nullptr;
	}
}

void My_queue::clear()// очистка списка
```

```
	{
		task *current;
		while (size)
		{
			current = pop_front();
			if (current)
			{
				delete current;
			}
		}
	}
	task* My_queue::removeAt(int index)// удаление элемента по указанному индексу
	{
		task *temp;
		if (index == 0)
		{
			temp = pop_front();
			return temp;

		}
		else
		{
			Node *current = head;
			for (int i = 0; i < index - 1; i++)// поиск удаляемого элемента
			{
				current = current->pNext;
			}
			Node *temp_first = current->pNext;
			current->pNext = temp_first->pNext;
			temp = temp_first->data;
			size--;
			return temp;
		}

}
```

Файл **Start_form.h**

```
	#pragma once
	#include "My_queue.h"
	#include "Task.h"
	#include "Processor.h"
	#include "System.h"
	#include "Main_form.h"

	namespace Planning_process {

		using namespace System;
		using namespace System::ComponentModel;
		using namespace System::Collections;
		using namespace System::Windows::Forms;
		using namespace System::Data;
		using namespace System::Drawing;

		/// <summary>
		/// Сводка для Start_form
		/// </summary>
		public ref class Start_form : public System::Windows::Forms::Form
		{
		public:
			Start_form(void)
			{
				InitializeComponent();
				count_cpu = 0;
				count_tasks = 0;
```

```
                        is_next = false;
                        step = 0;
                        queue = new My_queue();
                }

        protected:
                /// <summary>
                /// Освободить все используемые ресурсы.
                /// </summary>
                ~Start_form()
                {
                        if (components)
                        {
                                delete components;
                        }
                }
        private: System::Windows::Forms::Label^  label1;
        private: System::Windows::Forms::ComboBox^  set_cpu;

        private: System::Windows::Forms::Label^  label2;
        private: System::Windows::Forms::TextBox^  set_tasks;
        private: System::Windows::Forms::Label^  check_automatic_difficult;
        private: System::Windows::Forms::Label^  check_manual_difficult;
        private: System::Windows::Forms::Button^  next_button;
        private: System::Windows::Forms::Label^  number_task_label;
        private: System::Windows::Forms::TextBox^  difficult_text;
        private: System::Windows::Forms::Label^  difficult_label;
        private:
                size_t count_cpu;// кол-во процессоров
                size_t count_tasks;// кол-во задач
                size_t step;// шаг инициализации
                bool is_next;
                My_queue *queue;// очередь задач
                task *current_task;// текущая задача
                processor *cpu;// указатель на массив процессоров
        private: System::Windows::Forms::Label^  ID_label;
        private: System::Windows::Forms::Label^  label5;
        private: System::Windows::Forms::TextBox^  ID_text;
        private: System::Windows::Forms::Button^  exit_button;
                        System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
                /// <summary>
                /// Обязательный метод для поддержки конструктора - не изменяйте
                /// содержимое данного метода при помощи редактора кода.
                /// </summary>
                void InitializeComponent(void)
                {
                        this->label1 = (gcnew System::Windows::Forms::Label());
                        this->set_cpu = (gcnew System::Windows::Forms::ComboBox());
                        this->label2 = (gcnew System::Windows::Forms::Label());
                        this->set_tasks = (gcnew System::Windows::Forms::TextBox());
                        this->check_automatic_difficult = (gcnew
System::Windows::Forms::Label());
                        this->check_manual_difficult = (gcnew System::Windows::Forms::Label());
                        this->next_button = (gcnew System::Windows::Forms::Button());
                        this->number_task_label = (gcnew System::Windows::Forms::Label());
                        this->difficult_text = (gcnew System::Windows::Forms::TextBox());
                        this->difficult_label = (gcnew System::Windows::Forms::Label());
                        this->ID_label = (gcnew System::Windows::Forms::Label());
                        this->label5 = (gcnew System::Windows::Forms::Label());
                        this->ID_text = (gcnew System::Windows::Forms::TextBox());
                        this->exit_button = (gcnew System::Windows::Forms::Button());
                        this->SuspendLayout();
                        //
```

```
                    // label1
                    //
                    this->label1->AutoSize = true;
                    this->label1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                           static_cast<System::Byte>(204)));
                    this->label1->Location = System::Drawing::Point(334, 60);
                    this->label1->Name = L"label1";
                    this->label1->Size = System::Drawing::Size(333, 24);
                    this->label1->TabIndex = 0;
                    this->label1->Text = L"Выберите количество процессоров";
                    //
                    // set_cpu
                    //
                    this->set_cpu->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                           static_cast<System::Byte>(204)));
                    this->set_cpu->FormattingEnabled = true;
                    this->set_cpu->Items->AddRange(gcnew cli::array< System::Object^  >(12)
{
                           L"1", L"2", L"3", L"4", L"5", L"6", L"7", L"8", L"9",
                                  L"10", L"11", L"12"
                    });
                    this->set_cpu->Location = System::Drawing::Point(411, 94);
                    this->set_cpu->Name = L"set_cpu";
                    this->set_cpu->Size = System::Drawing::Size(177, 32);
                    this->set_cpu->TabIndex = 1;
                    //
                    // label2
                    //
                    this->label2->AutoSize = true;
                    this->label2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                           static_cast<System::Byte>(204)));
                    this->label2->Location = System::Drawing::Point(379, 172);
                    this->label2->Name = L"label2";
                    this->label2->Size = System::Drawing::Size(254, 24);
                    this->label2->TabIndex = 2;
                    this->label2->Text = L"Введите количество задач";
                    //
                    // set_tasks
                    //
                    this->set_tasks->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                           static_cast<System::Byte>(204)));
                    this->set_tasks->Location = System::Drawing::Point(454, 199);
                    this->set_tasks->Name = L"set_tasks";
                    this->set_tasks->Size = System::Drawing::Size(100, 29);
                    this->set_tasks->TabIndex = 3;
                    //
                    // check_automatic_difficult
                    //
                    this->check_automatic_difficult->AutoSize = true;
                    this->check_automatic_difficult->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 14.25F, System::Drawing::FontStyle::Regular,
                           System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
                    this->check_automatic_difficult->Location = System::Drawing::Point(315,
297);
                    this->check_automatic_difficult->Name = L"check_automatic_difficult";
                    this->check_automatic_difficult->Size = System::Drawing::Size(378, 24);
                    this->check_automatic_difficult->TabIndex = 4;
                    this->check_automatic_difficult->Text = L"1. Автоматическая установка
сложности";
```

```
                    this->check_automatic_difficult->Click += gcnew
System::EventHandler(this, &Start_form::check_automatic_difficult_Click);
                    //
                    // check_manual_difficult
                    //
                    this->check_manual_difficult->AutoSize = true;
                    this->check_manual_difficult->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 14.25F, System::Drawing::FontStyle::Regular,
                            System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
                    this->check_manual_difficult->Location = System::Drawing::Point(315,
332);
                    this->check_manual_difficult->Name = L"check_manual_difficult";
                    this->check_manual_difficult->Size = System::Drawing::Size(247, 24);
                    this->check_manual_difficult->TabIndex = 5;
                    this->check_manual_difficult->Text = L"2. Ручной ввод сложности";
                    this->check_manual_difficult->Click += gcnew System::EventHandler(this,
&Start_form::check_manual_difficult_Click);
                    //
                    // next_button
                    //
                    this->next_button->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(204)));
                    this->next_button->Location = System::Drawing::Point(805, 480);
                    this->next_button->Name = L"next_button";
                    this->next_button->Size = System::Drawing::Size(167, 49);
                    this->next_button->TabIndex = 6;
                    this->next_button->Text = L"Далее";
                    this->next_button->UseVisualStyleBackColor = true;
                    this->next_button->Click += gcnew System::EventHandler(this,
&Start_form::next_button_Click);
                    //
                    // number_task_label
                    //
                    this->number_task_label->AutoSize = true;
                    this->number_task_label->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 14.25F, System::Drawing::FontStyle::Regular,
                            System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
                    this->number_task_label->Location = System::Drawing::Point(12, 60);
                    this->number_task_label->Name = L"number_task_label";
                    this->number_task_label->Size = System::Drawing::Size(90, 24);
                    this->number_task_label->TabIndex = 7;
                    this->number_task_label->Text = L"Задача 1";
                    this->number_task_label->Visible = false;
                    //
                    // difficult_text
                    //
                    this->difficult_text->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 14.25F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(204)));
                    this->difficult_text->Location = System::Drawing::Point(129, 135);
                    this->difficult_text->Name = L"difficult_text";
                    this->difficult_text->Size = System::Drawing::Size(57, 29);
                    this->difficult_text->TabIndex = 8;
                    this->difficult_text->Visible = false;
                    //
                    // difficult_label
                    //
                    this->difficult_label->AutoSize = true;
                    this->difficult_label->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 14.25F, System::Drawing::FontStyle::Regular,
```

```
                                System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
                        this->difficult_label->Location = System::Drawing::Point(12, 138);
                        this->difficult_label->Name = L"difficult_label";
                        this->difficult_label->Size = System::Drawing::Size(111, 24);
                        this->difficult_label->TabIndex = 9;
                        this->difficult_label->Text = L"Сложность";
                        this->difficult_label->Visible = false;
                        //
                        // ID_label
                        //
                        this->ID_label->AutoSize = true;
                        this->ID_label->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(204)));
                        this->ID_label->Location = System::Drawing::Point(12, 100);
                        this->ID_label->Name = L"ID_label";
                        this->ID_label->Size = System::Drawing::Size(27, 24);
                        this->ID_label->TabIndex = 10;
                        this->ID_label->Text = L"ID";
                        this->ID_label->Visible = false;
                        //
                        // label5
                        //
                        this->label5->AutoSize = true;
                        this->label5->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(204)));
                        this->label5->Location = System::Drawing::Point(450, 262);
                        this->label5->Name = L"label5";
                        this->label5->Size = System::Drawing::Size(111, 24);
                        this->label5->TabIndex = 11;
                        this->label5->Text = L"Сложность";
                        this->label5->Visible = false;
                        //
                        // ID_text
                        //
                        this->ID_text->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(204)));
                        this->ID_text->Location = System::Drawing::Point(129, 97);
                        this->ID_text->Name = L"ID_text";
                        this->ID_text->Size = System::Drawing::Size(57, 29);
                        this->ID_text->TabIndex = 12;
                        this->ID_text->Visible = false;
                        //
                        // exit_button
                        //
                        this->exit_button->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(204)));
                        this->exit_button->Location = System::Drawing::Point(12, 480);
                        this->exit_button->Name = L"exit_button";
                        this->exit_button->Size = System::Drawing::Size(167, 49);
                        this->exit_button->TabIndex = 13;
                        this->exit_button->Text = L"Выход";
                        this->exit_button->UseVisualStyleBackColor = true;
                        this->exit_button->Click += gcnew System::EventHandler(this,
&Start_form::exit_button_Click);
                        //
                        // Start_form
                        //
                        this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                        this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                        this->ClientSize = System::Drawing::Size(984, 541);
```

```
                    this->Controls->Add(this->exit_button);
                    this->Controls->Add(this->ID_text);
                    this->Controls->Add(this->label5);
                    this->Controls->Add(this->ID_label);
                    this->Controls->Add(this->difficult_label);
                    this->Controls->Add(this->difficult_text);
                    this->Controls->Add(this->number_task_label);
                    this->Controls->Add(this->next_button);
                    this->Controls->Add(this->check_manual_difficult);
                    this->Controls->Add(this->check_automatic_difficult);
                    this->Controls->Add(this->set_tasks);
                    this->Controls->Add(this->label2);
                    this->Controls->Add(this->set_cpu);
                    this->Controls->Add(this->label1);
                    this->FormBorderStyle =
System::Windows::Forms::FormBorderStyle::FixedSingle;
                    this->MaximizeBox = false;
                    this->MinimizeBox = false;
                    this->Name = L"Start_form";
                    this->StartPosition =
System::Windows::Forms::FormStartPosition::CenterScreen;
                    this->Text = L"Установка параметров";
                    this->ResumeLayout(false);
                    this->PerformLayout();

             }
#pragma endregion
             //обработка события при выборе автоматической генерации сложности задачи
       private: System::Void check_automatic_difficult_Click(System::Object^  sender,
System::EventArgs^  e) {
                           this->check_automatic_difficult->ForeColor =
System::Drawing::Color::Blue;
                           this->check_manual_difficult->ForeColor =
System::Drawing::Color::Black;
       }
                    //обработка события при выборе ручного ввода сложности задачи
private: System::Void check_manual_difficult_Click(System::Object^  sender,
System::EventArgs^  e) {
                    this->check_automatic_difficult->ForeColor =
System::Drawing::Color::Black;
                    this->check_manual_difficult->ForeColor =
System::Drawing::Color::Blue;
}
              // //обработка события перехода на главную форму
private: System::Void next_button_Click(System::Object^  sender, System::EventArgs^  e) {
                    int ID;
                    int difficult;
                    if (!is_next)
                    {
                           is_next = true;
                           //конвертирование и инициализация введеных данных пользователем
                           try
                           {
                                  count_cpu = Convert::ToInt16(this->set_cpu->Text);
                                  count_tasks = Convert::ToInt16(this->set_tasks->Text);
                                  if (check_automatic_difficult->ForeColor !=
System::Drawing::Color::Blue&&
                                         check_manual_difficult->ForeColor !=
System::Drawing::Color::Blue || count_cpu == 0 || count_tasks == 0)
                                  {
                                         throw;// бросается исклбчение в случае
некорректности введенных данных
                                  }
                           }
                           catch (Exception ^ex)
```

```
                                {
                                        MessageBox::Show("Не верна введена информация\nВведите
еще раз!", "Ошибка");
                                        return;
                                }

                                cpu = new processor[count_cpu];// массив процессоров
                                for (size_t i = 0; i < count_cpu; i++)
                                {
                                        cpu[i].set_number(i);
                                }
                                if (this->check_automatic_difficult->ForeColor ==
System::Drawing::Color::Blue)
                                {
                                        // автоматическая генерация сложности задачи
                                        Random ^rand = gcnew Random();
                                        for (size_t i = 0; i < count_tasks; i++)
                                        {
                                                ID = i + 1;
                                                difficult = rand->Next(1, 50);
                                                current_task = new task(ID, difficult);
                                                queue->push_back(current_task);
                                        }
                                        My_system *my_system = new My_system(queue, count_cpu,
cpu);
                                        Main_form ^form = gcnew Main_form(my_system);
                                        // запуск главной формы после инициализации данных
                                        this->Hide();
                                        form->ShowDialog();
                                        delete form;
                                        //delete my_system;
                                        this->Show();
                                }
                                else
                                {
                                        //ручной ввод сложности
                                        set_cpu->Visible = false;
                                        set_tasks->Visible = false;
                                        label1->Visible = false;
                                        label2->Visible = false;
                                        check_automatic_difficult->Visible = false;
                                        check_manual_difficult->Visible = false;
                                        number_task_label->Visible = true;
                                        ID_label->Visible = true;
                                        difficult_label->Visible = true;
                                        ID_text->Visible = true;
                                        difficult_text->Visible = true;
                                }
                        }
                        else
                        {
                                try
                                {
                                        ID = Convert::ToInt32(this->ID_text->Text);
                                        difficult =
Convert::ToInt32(this->difficult_text->Text);
                                }
                                catch (Exception ^ex)
                                {
                                        MessageBox::Show("Не верна введена информация\nВведите
еще раз!", "Ошибка");
                                        return;
                                }
                                step++;// переход к след шагу
```

```
                                current_task = new task(ID, difficult);// выделение памяти под
        новую задачу
                                queue->push_back(current_task);
                                number_task_label->Text = "Задача " + Convert::ToString(step +
        1);
                                ID_text->Text = "";
                                difficult_text->Text = "";
                                if (step == count_tasks)
                                {
                                        My_system *my_system = new My_system(queue, count_cpu,
        cpu);
                                        Main_form ^form = gcnew Main_form(my_system);
                                        // запуск главной формы
                                        this->Hide();
                                        form->ShowDialog();
                                        if (form)
                                        {
                                                delete form;
                                        }
                                        Application::Exit();
                                }
                        }
        }
                        // обработка события нажатия на кнопку выхода
        private: System::Void exit_button_Click(System::Object^  sender, System::EventArgs^  e) {
                        if (queue)
                        {
                                delete queue;
                        }
                        if (cpu)
                        {
                                delete[] cpu;
                        }
                        Application::Exit();
        }
        };
}
```

Файл **Main_form.h**

```
        #pragma once
        #include "My_queue.h"
        #include "Task.h"
        #include "Processor.h"
        #include "System.h"
        #include "Queue_form.h"

        namespace Planning_process {

                using namespace System;
                using namespace System::ComponentModel;
                using namespace System::Collections;
                using namespace System::Windows::Forms;
                using namespace System::Data;
                using namespace System::Drawing;

                /// <summary>
                /// Сводка для Main_form
                /// </summary>
                public ref class Main_form : public System::Windows::Forms::Form
                {
                public:
                        Main_form(My_system *my_system)
                        {
                                this->my_system = my_system;
```

```
                        InitializeComponent();
                        this->my_system->setup();
                        is_finish = false;
                        minut = 0;
                        sekond = 0;
                        form_general_task = gcnew Queue_form(my_system, 1);
                        form_block_task = gcnew Queue_form(my_system, 2);
                        form_finish_task = gcnew Queue_form(my_system, 3);
                        //
                        //TODO: добавьте код конструктора
                        //
                }

        protected:
                /// <summary>
                /// Освободить все используемые ресурсы.
                /// </summary>
                ~Main_form()
                {
                        if (components)
                        {
                                delete components;
                        }
                }

        private:
                /// <summary>
                /// Требуется переменная конструктора.
                /// </summary>
                bool is_finish;
                My_system *my_system;//  указатель на объект системы
                int minut;// минуты
                int sekond;// секунды
                Queue_form ^form_general_task;// указатель на форму с главной очередью задач
                Queue_form ^form_block_task;// указатель на форму с заблокированных задач
                Queue_form ^form_finish_task;// указатель на форму с завершенных задач

        private: System::Windows::Forms::DataGridView^  dataGridView;

        private: System::Windows::Forms::Label^  state_system_label;
        private: System::Windows::Forms::Label^  time_work_label;
        private: System::Windows::Forms::Button^  exit_button;

        private: System::Windows::Forms::Button^  pause_button;
        private: System::Windows::Forms::Button^  block_task_button;


        private: System::Windows::Forms::Button^  queue_button;
        private: System::Windows::Forms::DataGridViewTextBoxColumn^  cpu_number;
        private: System::Windows::Forms::DataGridViewTextBoxColumn^  procent_finish;
        private: System::Windows::Forms::DataGridViewTextBoxColumn^  ID_task;
        private: System::Windows::Forms::DataGridViewTextBoxColumn^  difficult;
        private: System::Windows::Forms::DataGridViewTextBoxColumn^  count_finish_tasks;
        private: System::Windows::Forms::DataGridViewButtonColumn^  block_task;
        private: System::Windows::Forms::Timer^  timer;
        private: System::Windows::Forms::Label^  count_finish_tasks_label;
        private: System::Windows::Forms::Label^  count_task_label;
        private: System::Windows::Forms::Button^  finish_task_button;
        private: System::ComponentModel::IContainer^  components;

#pragma region Windows Form Designer generated code
                /// <summary>
                /// Обязательный метод для поддержки конструктора - не изменяйте
                /// содержимое данного метода при помощи редактора кода.
                /// </summary>
```

```
		void InitializeComponent(void)
		{
			this->components = (gcnew System::ComponentModel::Container());
			System::Windows::Forms::DataGridViewCellStyle^  dataGridViewCellStyle1
= (gcnew System::Windows::Forms::DataGridViewCellStyle());
			System::Windows::Forms::DataGridViewCellStyle^  dataGridViewCellStyle2
= (gcnew System::Windows::Forms::DataGridViewCellStyle());
			System::Windows::Forms::DataGridViewCellStyle^  dataGridViewCellStyle3
= (gcnew System::Windows::Forms::DataGridViewCellStyle());
			System::Windows::Forms::DataGridViewCellStyle^  dataGridViewCellStyle4
= (gcnew System::Windows::Forms::DataGridViewCellStyle());
			this->dataGridView = (gcnew System::Windows::Forms::DataGridView());
			this->cpu_number = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
			this->procent_finish = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
			this->ID_task = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
			this->difficult = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
			this->count_finish_tasks = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
			this->block_task = (gcnew
System::Windows::Forms::DataGridViewButtonColumn());
			this->state_system_label = (gcnew System::Windows::Forms::Label());
			this->time_work_label = (gcnew System::Windows::Forms::Label());
			this->exit_button = (gcnew System::Windows::Forms::Button());
			this->pause_button = (gcnew System::Windows::Forms::Button());
			this->block_task_button = (gcnew System::Windows::Forms::Button());
			this->queue_button = (gcnew System::Windows::Forms::Button());
			this->timer = (gcnew System::Windows::Forms::Timer(this->components));
			this->count_finish_tasks_label = (gcnew
System::Windows::Forms::Label());
			this->count_task_label = (gcnew System::Windows::Forms::Label());
			this->finish_task_button = (gcnew System::Windows::Forms::Button());

	(cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView))->B
eginInit();
			this->SuspendLayout();
			//
			// dataGridView
			//
			this->dataGridView->BackgroundColor =
System::Drawing::SystemColors::Control;
			dataGridViewCellStyle1->Alignment =
System::Windows::Forms::DataGridViewContentAlignment::MiddleCenter;
			dataGridViewCellStyle1->BackColor =
System::Drawing::SystemColors::Control;
			dataGridViewCellStyle1->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 12, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
				static_cast<System::Byte>(204)));
			dataGridViewCellStyle1->ForeColor =
System::Drawing::SystemColors::WindowText;
			dataGridViewCellStyle1->SelectionBackColor =
System::Drawing::SystemColors::Highlight;
			dataGridViewCellStyle1->SelectionForeColor =
System::Drawing::SystemColors::HighlightText;
			dataGridViewCellStyle1->WrapMode =
System::Windows::Forms::DataGridViewTriState::True;
			this->dataGridView->ColumnHeadersDefaultCellStyle =
dataGridViewCellStyle1;
			this->dataGridView->ColumnHeadersHeightSizeMode =
System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;
			this->dataGridView->Columns->AddRange(gcnew cli::array<
System::Windows::Forms::DataGridViewColumn^  >(6) {
```

```
                            this->cpu_number,
                                   this->procent_finish, this->ID_task, this->difficult,
this->count_finish_tasks, this->block_task
                     });
                     dataGridViewCellStyle2->Alignment =
System::Windows::Forms::DataGridViewContentAlignment::MiddleLeft;
                     dataGridViewCellStyle2->BackColor =
System::Drawing::SystemColors::Window;
                     dataGridViewCellStyle2->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 12, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(204)));
                     dataGridViewCellStyle2->ForeColor =
System::Drawing::SystemColors::ControlText;
                     dataGridViewCellStyle2->SelectionBackColor =
System::Drawing::SystemColors::Highlight;
                     dataGridViewCellStyle2->SelectionForeColor =
System::Drawing::SystemColors::HighlightText;
                     dataGridViewCellStyle2->WrapMode =
System::Windows::Forms::DataGridViewTriState::False;
                     this->dataGridView->DefaultCellStyle = dataGridViewCellStyle2;
                     this->dataGridView->GridColor = System::Drawing::SystemColors::Control;
                     this->dataGridView->Location = System::Drawing::Point(0, -1);
                     this->dataGridView->Name = L"dataGridView";
                     dataGridViewCellStyle3->Alignment =
System::Windows::Forms::DataGridViewContentAlignment::MiddleCenter;
                     dataGridViewCellStyle3->BackColor =
System::Drawing::SystemColors::Control;
                     dataGridViewCellStyle3->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 8.25F, System::Drawing::FontStyle::Regular,
                            System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
                     dataGridViewCellStyle3->ForeColor =
System::Drawing::SystemColors::WindowText;
                     dataGridViewCellStyle3->SelectionBackColor =
System::Drawing::SystemColors::Highlight;
                     dataGridViewCellStyle3->SelectionForeColor =
System::Drawing::SystemColors::HighlightText;
                     dataGridViewCellStyle3->WrapMode =
System::Windows::Forms::DataGridViewTriState::True;
                     this->dataGridView->RowHeadersDefaultCellStyle =
dataGridViewCellStyle3;
                     this->dataGridView->RowHeadersVisible = false;
                     dataGridViewCellStyle4->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 12, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(204)));
                     this->dataGridView->RowsDefaultCellStyle = dataGridViewCellStyle4;
                     this->dataGridView->RowTemplate->DefaultCellStyle->Alignment =
System::Windows::Forms::DataGridViewContentAlignment::MiddleCenter;
                     this->dataGridView->RowTemplate->DefaultCellStyle->WrapMode =
System::Windows::Forms::DataGridViewTriState::True;
                     this->dataGridView->RowTemplate->Height = 30;
                     this->dataGridView->RowTemplate->Resizable =
System::Windows::Forms::DataGridViewTriState::False;
                     this->dataGridView->Size = System::Drawing::Size(983, 410);
                     this->dataGridView->TabIndex = 0;
                     this->dataGridView->CellContentClick += gcnew
System::Windows::Forms::DataGridViewCellEventHandler(this,
&Main_form::dataGridView_CellContentClick);
                     //
                     // cpu_number
                     //
                     this->cpu_number->AutoSizeMode =
System::Windows::Forms::DataGridViewAutoSizeColumnMode::None;
                     this->cpu_number->Frozen = true;
                     this->cpu_number->HeaderText = L"CPU №";
```

```
                    this->cpu_number->Name = L"cpu_number";
                    this->cpu_number->ReadOnly = true;
                    this->cpu_number->Resizable =
System::Windows::Forms::DataGridViewTriState::True;
                    this->cpu_number->SortMode =
System::Windows::Forms::DataGridViewColumnSortMode::NotSortable;
                    //
                    // procent_finish
                    //
                    this->procent_finish->AutoSizeMode =
System::Windows::Forms::DataGridViewAutoSizeColumnMode::None;
                    this->procent_finish->Frozen = true;
                    this->procent_finish->HeaderText = L"% завершения задачи";
                    this->procent_finish->Name = L"procent_finish";
                    this->procent_finish->ReadOnly = true;
                    this->procent_finish->Resizable =
System::Windows::Forms::DataGridViewTriState::True;
                    this->procent_finish->SortMode =
System::Windows::Forms::DataGridViewColumnSortMode::NotSortable;
                    this->procent_finish->Width = 250;
                    //
                    // ID_task
                    //
                    this->ID_task->AutoSizeMode =
System::Windows::Forms::DataGridViewAutoSizeColumnMode::None;
                    this->ID_task->Frozen = true;
                    this->ID_task->HeaderText = L"ID задачи";
                    this->ID_task->Name = L"ID_task";
                    this->ID_task->ReadOnly = true;
                    this->ID_task->Resizable =
System::Windows::Forms::DataGridViewTriState::True;
                    this->ID_task->SortMode =
System::Windows::Forms::DataGridViewColumnSortMode::NotSortable;
                    this->ID_task->Width = 150;
                    //
                    // difficult
                    //
                    this->difficult->AutoSizeMode =
System::Windows::Forms::DataGridViewAutoSizeColumnMode::None;
                    this->difficult->Frozen = true;
                    this->difficult->HeaderText = L"Сложность";
                    this->difficult->Name = L"difficult";
                    this->difficult->ReadOnly = true;
                    this->difficult->Resizable =
System::Windows::Forms::DataGridViewTriState::True;
                    this->difficult->SortMode =
System::Windows::Forms::DataGridViewColumnSortMode::NotSortable;
                    this->difficult->Width = 120;
                    //
                    // count_finish_tasks
                    //
                    this->count_finish_tasks->AutoSizeMode =
System::Windows::Forms::DataGridViewAutoSizeColumnMode::None;
                    this->count_finish_tasks->Frozen = true;
                    this->count_finish_tasks->HeaderText = L"количество выполненных задач";
                    this->count_finish_tasks->Name = L"count_finish_tasks";
                    this->count_finish_tasks->ReadOnly = true;
                    this->count_finish_tasks->Resizable =
System::Windows::Forms::DataGridViewTriState::True;
                    this->count_finish_tasks->SortMode =
System::Windows::Forms::DataGridViewColumnSortMode::NotSortable;
                    this->count_finish_tasks->Width = 200;
                    //
                    // block_task
                    //
```

```
                    this->block_task->AutoSizeMode =
System::Windows::Forms::DataGridViewAutoSizeColumnMode::None;
                    this->block_task->Frozen = true;
                    this->block_task->HeaderText = L"Блокировка задачи";
                    this->block_task->Name = L"block_task";
                    this->block_task->Width = 163;
                    //
                    // state_system_label
                    //
                    this->state_system_label->AutoSize = true;
                    this->state_system_label->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 14.25F, System::Drawing::FontStyle::Regular,
                            System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
                    this->state_system_label->Location = System::Drawing::Point(8, 412);
                    this->state_system_label->Name = L"state_system_label";
                    this->state_system_label->Size = System::Drawing::Size(251, 24);
                    this->state_system_label->TabIndex = 1;
                    this->state_system_label->Text = L"Состояние системы: пауза";
                    //
                    // time_work_label
                    //
                    this->time_work_label->AutoSize = true;
                    this->time_work_label->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 14.25F, System::Drawing::FontStyle::Regular,
                            System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
                    this->time_work_label->Location = System::Drawing::Point(8, 436);
                    this->time_work_label->Name = L"time_work_label";
                    this->time_work_label->Size = System::Drawing::Size(256, 24);
                    this->time_work_label->TabIndex = 2;
                    this->time_work_label->Text = L"Время работы системы: 0:0";
                    //
                    // exit_button
                    //
                    this->exit_button->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(204)));
                    this->exit_button->Location = System::Drawing::Point(822, 488);
                    this->exit_button->Name = L"exit_button";
                    this->exit_button->Size = System::Drawing::Size(150, 41);
                    this->exit_button->TabIndex = 3;
                    this->exit_button->Text = L"Выход";
                    this->exit_button->UseVisualStyleBackColor = true;
                    this->exit_button->Click += gcnew System::EventHandler(this,
&Main_form::exit_button_Click);
                    //
                    // pause_button
                    //
                    this->pause_button->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(204)));
                    this->pause_button->Location = System::Drawing::Point(625, 488);
                    this->pause_button->Name = L"pause_button";
                    this->pause_button->Size = System::Drawing::Size(150, 41);
                    this->pause_button->TabIndex = 4;
                    this->pause_button->Text = L"Запуск";
                    this->pause_button->UseVisualStyleBackColor = true;
                    this->pause_button->Click += gcnew System::EventHandler(this,
&Main_form::pause_button_Click);
                    //
                    // block_task_button
                    //
```

```
                    this->block_task_button->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 12, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(204)));
                    this->block_task_button->Location = System::Drawing::Point(220, 488);
                    this->block_task_button->Name = L"block_task_button";
                    this->block_task_button->Size = System::Drawing::Size(150, 41);
                    this->block_task_button->TabIndex = 5;
                    this->block_task_button->Text = L"Блок. процессы";
                    this->block_task_button->UseVisualStyleBackColor = true;
                    this->block_task_button->Click += gcnew System::EventHandler(this,
&Main_form::button3_Click);
                    //
                    // queue_button
                    //
                    this->queue_button->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(204)));
                    this->queue_button->Location = System::Drawing::Point(12, 488);
                    this->queue_button->Name = L"queue_button";
                    this->queue_button->Size = System::Drawing::Size(150, 41);
                    this->queue_button->TabIndex = 6;
                    this->queue_button->Text = L"Очередь";
                    this->queue_button->UseVisualStyleBackColor = true;
                    this->queue_button->Click += gcnew System::EventHandler(this,
&Main_form::queue_button_Click);
                    //
                    // timer
                    //
                    this->timer->Interval = 1000;
                    this->timer->Tick += gcnew System::EventHandler(this,
&Main_form::timer_Tick);
                    //
                    // count_finish_tasks_label
                    //
                    this->count_finish_tasks_label->AutoSize = true;
                    this->count_finish_tasks_label->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 14.25F, System::Drawing::FontStyle::Regular,
                            System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
                    this->count_finish_tasks_label->Location = System::Drawing::Point(575,
436);
                    this->count_finish_tasks_label->Name = L"count_finish_tasks_label";
                    this->count_finish_tasks_label->Size = System::Drawing::Size(268, 24);
                    this->count_finish_tasks_label->TabIndex = 7;
                    this->count_finish_tasks_label->Text = L"Число выполненных задач: 0";
                    //
                    // count_task_label
                    //
                    this->count_task_label->AutoSize = true;
                    this->count_task_label->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 14.25F, System::Drawing::FontStyle::Regular,
                            System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
                    this->count_task_label->Location = System::Drawing::Point(575, 412);
                    this->count_task_label->Name = L"count_task_label";
                    this->count_task_label->Size = System::Drawing::Size(125, 24);
                    this->count_task_label->TabIndex = 8;
                    this->count_task_label->Text = L"Число задач:";
                    //
                    // finish_task_button
                    //
                    this->finish_task_button->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 9.75F, System::Drawing::FontStyle::Regular,
```

```
                            System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
                        this->finish_task_button->Location = System::Drawing::Point(422, 488);
                        this->finish_task_button->Name = L"finish_task_button";
                        this->finish_task_button->Size = System::Drawing::Size(150, 41);
                        this->finish_task_button->TabIndex = 9;
                        this->finish_task_button->Text = L"Выполненные процессы";
                        this->finish_task_button->UseVisualStyleBackColor = true;
                        this->finish_task_button->Click += gcnew System::EventHandler(this,
&Main_form::finish_task_button_Click);
                        //
                        // Main_form
                        //
                        this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                        this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                        this->ClientSize = System::Drawing::Size(984, 541);
                        this->Controls->Add(this->finish_task_button);
                        this->Controls->Add(this->count_task_label);
                        this->Controls->Add(this->count_finish_tasks_label);
                        this->Controls->Add(this->queue_button);
                        this->Controls->Add(this->block_task_button);
                        this->Controls->Add(this->pause_button);
                        this->Controls->Add(this->exit_button);
                        this->Controls->Add(this->time_work_label);
                        this->Controls->Add(this->state_system_label);
                        this->Controls->Add(this->dataGridView);
                        this->FormBorderStyle =
System::Windows::Forms::FormBorderStyle::FixedSingle;
                        this->MaximizeBox = false;
                        this->MinimizeBox = false;
                        this->Name = L"Main_form";
                        this->StartPosition =
System::Windows::Forms::FormStartPosition::CenterScreen;
                        this->Text = L"Обработка процессов";
                        this->Load += gcnew System::EventHandler(this,
&Main_form::Main_form_Load);

       (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView))->E
ndInit();
                        this->ResumeLayout(false);
                        this->PerformLayout();

                }
#pragma endregion
                //вывод формы с очередью задач, готовых к выполнению
       private: System::Void queue_button_Click(System::Object^  sender, System::EventArgs^
e) {
                                        form_general_task->Show();

       }
                         // метод загрузки главной формы и инициализации объектов
private: System::Void Main_form_Load(System::Object^  sender, System::EventArgs^  e) {
                         dataGridView->AllowUserToAddRows = false;
                         String ^ID_task=gcnew String("NULL");;
                         String ^difficult_task = gcnew String("NULL");
                         // вывод данных в таблицу
                         for (size_t i = 0; i < my_system->get_count_cpu(); i++)
                         {
                                if (my_system->get_cpu(i)->get_task())
                                {
                                        ID_task =
Convert::ToString(my_system->get_cpu(i)->get_task()->get_ID());
                                        difficult_task =
Convert::ToString(my_system->get_cpu(i)->get_task()->get_difficult());
                                }
```

```
                                else
                                {
                                        ID_task = "NULL";
                                        difficult_task = "NULL";
                                }

dataGridView->Rows->Add(Convert::ToString(my_system->get_cpu(i)->get_number() + 1),

Convert::ToString(my_system->get_cpu(i)->get_current_procent())+"%",
                                        ID_task,
                                        difficult_task,

Convert::ToString(my_system->get_cpu(i)->get_count_completed()), "Блокировка");
                        }
                        delete ID_task;
                        delete difficult_task;
                        this->count_task_label->Text = "Число задач: " +
Convert::ToString(my_system->get_count_tasks());
}
                // нажатие на кнопку паузы
private: System::Void pause_button_Click(System::Object^  sender, System::EventArgs^  e) {
                        if (this->pause_button->Text == "Запуск")
                        {
                                this->timer->Enabled = true;
                                my_system->start();
                                this->pause_button->Text = "Пауза";
                                state_system_label->Text = "Состояние системы: работа";
                        }
                        else
                        {
                                this->timer->Enabled = false;
                                my_system->pause();
                                this->pause_button->Text = "Запуск";
                                state_system_label->Text = "Состояние системы: пауза";
                        }
}
                // обработка события на тик таймера
private: System::Void timer_Tick(System::Object^  sender, System::EventArgs^  e) {
                        // вызов метода work и проверка на завершение работы
                        if (my_system->work())
                        {
                                is_finish = true;
                        }
                        dataGridView->Rows->Clear();
                        String ^ID_task = gcnew String("NULL");;
                        String ^difficult_task = gcnew String("NULL");
                        // вывод новых данных в таблицу
                        for (size_t i = 0; i < my_system->get_count_cpu(); i++)
                        {
                                if (my_system->get_cpu(i)->get_task())
                                {
                                        ID_task =
Convert::ToString(my_system->get_cpu(i)->get_task()->get_ID());
                                        difficult_task =
Convert::ToString(my_system->get_cpu(i)->get_task()->get_difficult());
                                }
                                else
                                {
                                        ID_task = "NULL";
                                        difficult_task = "NULL";
                                }

dataGridView->Rows->Add(Convert::ToString(my_system->get_cpu(i)->get_number() + 1),

Convert::ToString(my_system->get_cpu(i)->get_current_procent())+"%",
```

```
                                        ID_task,
                                        difficult_task,

        Convert::ToString(my_system->get_cpu(i)->get_count_completed()), "Блокировка");
                                }
                                delete ID_task;
                                delete difficult_task;
                                // обработка секундомера
                                sekond++;
                                if (sekond == 60)
                                {
                                        minut++;
                                        sekond = 0;
                                }
                                time_work_label->Text = "Время работы системы: " +
        Convert::ToString(minut) + " мин : " + Convert::ToString(sekond)+" сек";
                                count_finish_tasks_label->Text = "Число выполненных задач: " +
        Convert::ToString(my_system->get_count_finish_tasks());
                                form_general_task->time_tick();
                                form_block_task->time_tick();
                                form_finish_task->time_tick();
                                //если работа завершена
                                if (is_finish)
                                {
                                        this->timer->Enabled = false;
                                        state_system_label->Text = "Состояние системы: работа
        завершена";
                                        form_finish_task->Hide();
                                        form_finish_task->Show();
                                }
        }
                        // обработка нажатия на кнопку выхода
        private: System::Void exit_button_Click(System::Object^  sender, System::EventArgs^  e) {
                                //delete my_system;
                                delete form_general_task, form_block_task, form_finish_task;
                                delete this;
        }
               private: System::Void button3_Click(System::Object^  sender, System::EventArgs^  e) {
                                        form_block_task->Show();
               }
        private: System::Void finish_task_button_Click(System::Object^  sender, System::EventArgs^
        e) {
                                        form_finish_task->Show();
        }
                        // блокировка выбранной задачи
        private: System::Void dataGridView_CellContentClick(System::Object^  sender,
        System::Windows::Forms::DataGridViewCellEventArgs^  e) {
                                if (e->ColumnIndex == 5)
                                {
                                        my_system->block_task(e->RowIndex);
                                }
        }
        };
}
```

Файл **Queue_form.h**

```
        #pragma once
        #include"System.h"

        namespace Planning_process {

               using namespace System;
               using namespace System::ComponentModel;
               using namespace System::Collections;
```

```
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;

/// <summary>
/// Сводка для Queue_form
/// </summary>
public ref class Queue_form : public System::Windows::Forms::Form
{
public:
	Queue_form(My_system *my_system, int choice)
	{
		InitializeComponent();
		this->my_system = my_system;
		this->choice = choice;
		//
		//TODO: добавьте код конструктора
		//
	}

protected:
	/// <summary>
	/// Освободить все используемые ресурсы.
	/// </summary>
	~Queue_form()
	{
		if (components)
		{
			delete components;
		}
	}
private: System::Windows::Forms::DataGridView^  dataGridView;
protected:

protected:




private:
	/// <summary>
	/// Требуется переменная конструктора.
	/// </summary>
	My_system *my_system;
	int choice;

private: System::Windows::Forms::DataGridViewTextBoxColumn^  ID_column;
private: System::Windows::Forms::DataGridViewTextBoxColumn^  Difficult_column;
private: System::Windows::Forms::DataGridViewTextBoxColumn^  Current_procent;
private: System::Windows::Forms::DataGridViewTextBoxColumn^  State_column;
private: System::Windows::Forms::Label^  close_label;




		System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
	/// <summary>
	/// Обязательный метод для поддержки конструктора - не изменяйте
	/// содержимое данного метода при помощи редактора кода.
	/// </summary>
```

```
            void InitializeComponent(void)
            {
                    System::Windows::Forms::DataGridViewCellStyle^  dataGridViewCellStyle1
= (gcnew System::Windows::Forms::DataGridViewCellStyle());
                    System::Windows::Forms::DataGridViewCellStyle^  dataGridViewCellStyle2
= (gcnew System::Windows::Forms::DataGridViewCellStyle());
                    this->dataGridView = (gcnew System::Windows::Forms::DataGridView());
                    this->ID_column = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
                    this->Difficult_column = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
                    this->Current_procent = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
                    this->State_column = (gcnew
System::Windows::Forms::DataGridViewTextBoxColumn());
                    this->close_label = (gcnew System::Windows::Forms::Label());

      (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView))->B
eginInit();
                    this->SuspendLayout();
                    //
                    // dataGridView
                    //
                    this->dataGridView->BackgroundColor =
System::Drawing::SystemColors::Control;
                    this->dataGridView->ColumnHeadersBorderStyle =
System::Windows::Forms::DataGridViewHeaderBorderStyle::Single;
                    dataGridViewCellStyle1->Alignment =
System::Windows::Forms::DataGridViewContentAlignment::MiddleCenter;
                    dataGridViewCellStyle1->BackColor =
System::Drawing::SystemColors::Control;
                    dataGridViewCellStyle1->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 11.25F, System::Drawing::FontStyle::Regular,
                            System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
                    dataGridViewCellStyle1->ForeColor =
System::Drawing::SystemColors::WindowText;
                    dataGridViewCellStyle1->SelectionBackColor =
System::Drawing::SystemColors::Highlight;
                    dataGridViewCellStyle1->SelectionForeColor =
System::Drawing::SystemColors::HighlightText;
                    dataGridViewCellStyle1->WrapMode =
System::Windows::Forms::DataGridViewTriState::True;
                    this->dataGridView->ColumnHeadersDefaultCellStyle =
dataGridViewCellStyle1;
                    this->dataGridView->ColumnHeadersHeightSizeMode =
System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;
                    this->dataGridView->Columns->AddRange(gcnew cli::array<
System::Windows::Forms::DataGridViewColumn^  >(4) {
                            this->ID_column,
                                    this->Difficult_column, this->Current_procent,
this->State_column
                    });
                    this->dataGridView->Location = System::Drawing::Point(1, 24);
                    this->dataGridView->Name = L"dataGridView";
                    this->dataGridView->ReadOnly = true;
                    this->dataGridView->RowHeadersVisible = false;
                    dataGridViewCellStyle2->Alignment =
System::Windows::Forms::DataGridViewContentAlignment::MiddleCenter;
                    dataGridViewCellStyle2->Font = (gcnew System::Drawing::Font(L"Microsoft
Sans Serif", 11.25F, System::Drawing::FontStyle::Regular,
                            System::Drawing::GraphicsUnit::Point,
static_cast<System::Byte>(204)));
                    this->dataGridView->RowsDefaultCellStyle = dataGridViewCellStyle2;
                    this->dataGridView->Size = System::Drawing::Size(484, 492);
```

```
                        this->dataGridView->TabIndex = 0;
                        //
                        // ID_column
                        //
                        this->ID_column->HeaderText = L"ID";
                        this->ID_column->Name = L"ID_column";
                        this->ID_column->ReadOnly = true;
                        this->ID_column->Width = 80;
                        //
                        // Difficult_column
                        //
                        this->Difficult_column->HeaderText = L"Сложность";
                        this->Difficult_column->Name = L"Difficult_column";
                        this->Difficult_column->ReadOnly = true;
                        this->Difficult_column->Width = 130;
                        //
                        // Current_procent
                        //
                        this->Current_procent->HeaderText = L"% завершения";
                        this->Current_procent->Name = L"Current_procent";
                        this->Current_procent->ReadOnly = true;
                        this->Current_procent->Width = 140;
                        //
                        // State_column
                        //
                        this->State_column->HeaderText = L"Была ли блокировка";
                        this->State_column->Name = L"State_column";
                        this->State_column->ReadOnly = true;
                        this->State_column->Width = 138;
                        //
                        // close_label
                        //
                        this->close_label->AutoSize = true;
                        this->close_label->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 14.25F, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(204)));
                        this->close_label->ForeColor = System::Drawing::Color::Red;
                        this->close_label->Location = System::Drawing::Point(418, -3);
                        this->close_label->Name = L"close_label";
                        this->close_label->Size = System::Drawing::Size(77, 24);
                        this->close_label->TabIndex = 1;
                        this->close_label->Text = L"Скрыть";
                        this->close_label->Click += gcnew System::EventHandler(this,
&Queue_form::close_label_Click);
                        //
                        // Queue_form
                        //
                        this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                        this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                        this->ClientSize = System::Drawing::Size(497, 516);
                        this->ControlBox = false;
                        this->Controls->Add(this->close_label);
                        this->Controls->Add(this->dataGridView);
                        this->FormBorderStyle =
System::Windows::Forms::FormBorderStyle::FixedSingle;
                        this->MaximizeBox = false;
                        this->Name = L"Queue_form";
                        this->StartPosition =
System::Windows::Forms::FormStartPosition::CenterScreen;
                        this->Text = L"Очередь";
                        this->Load += gcnew System::EventHandler(this,
&Queue_form::Queue_form_Load);

                (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->dataGridView))->E
ndInit();
```

```
                    this->ResumeLayout(false);
                    this->PerformLayout();

             }
#pragma endregion
private: System::Void Queue_form_Load(System::Object^  sender, System::EventArgs^  e) {
                     time_tick();
                     dataGridView->AllowUserToAddRows = false;
}
public: void time_tick()
{
                    dataGridView->AllowUserToAddRows = false;
                    dataGridView->Rows->Clear();
                    if (choice == 1)
                    {
                           this->Text = "Общая очередь задач";
                           for (size_t i = 0; i < my_system->get_count_general_queue_tasks();
i++)
                           {
                                  String ^state = my_system->get_task(i)->get_state();

       dataGridView->Rows->Add(Convert::ToString(my_system->get_task(i)->get_ID()),

       Convert::ToString(my_system->get_task(i)->get_difficult()),

       Convert::ToString(my_system->get_task(i)->get_current_procent()) + "%",
                                         state);
                                  delete state;
                           }

                    }
                    else if (choice == 2)
                    {
                           this->Text = "Заблокированные задачи";
                           dataGridView->Columns[3]->HeaderText = "Ост. время блокировки";
                           for (size_t i = 0; i < my_system->get_count_block_tasks(); i++)
                           {

       dataGridView->Rows->Add(Convert::ToString(my_system->get_block_task(i)->get_ID()),

       Convert::ToString(my_system->get_block_task(i)->get_difficult()),

       Convert::ToString(my_system->get_block_task(i)->get_current_procent()) + "%",

       Convert::ToString(my_system->get_block_task(i)->get_counter_block() + "сек"));
                           }
                    }
                    else
                    {
                           this->Text = "Выполненные задачи";
                           for (size_t i = 0; i < my_system->get_count_finish_tasks(); i++)
                           {
                                  String ^state =
my_system->get_finish_task(i)->get_state();

       dataGridView->Rows->Add(Convert::ToString(my_system->get_finish_task(i)->get_ID()),

       Convert::ToString(my_system->get_finish_task(i)->get_difficult()),

       Convert::ToString(my_system->get_finish_task(i)->get_current_procent()) + "%",
                                         state);
                                  delete state;
                           }
                    }
}
```

```
        private: System::Void close_label_Click(System::Object^  sender, System::EventArgs^  e) {
                                this->Hide();
        }
        };
}
```