МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ

ОДЕССКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ

ИМЕНИ И.И.МЕЧНИКОВА

ИНСТИТУТ МАТЕМАТИКИ, ЭКОНОМИКИ И МЕХАНИКИ

**КУРСОВАЯ РАБОТА**

**по дисциплине**

**«Системное программное обеспечение»**

**на тему:**

**«Имитация работы планировщика центрального процессора**»

|  |  |
| --- | --- |
|  | студента ІI курса  группы 1  специальности «Компьютерная инженерия»  Жужи Георгия Юрьевича  Руководитель: Лисицина Ирина Николаевна  Защищено «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_201\_ г.  с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Содержание

[ВВЕДЕНИЕ 3](#_Toc503178955)

[1. ПОСТАНОВКА ЗАДАЧИ 4](#_Toc503178956)

[2. ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ АЛГОРИТМОВ ПЛАНИРОВАНИЯ ПРОЦЕССОРА И ПАМЯТИ 6](#_Toc503178957)

[3. ОБЪЕКТНАЯ МОДЕЛЬ 7](#_Toc503178958)

[За всё время работы я написал 10 классов 7](#_Toc503178959)

[4. ИНСТРУКЦИЯ ПО ПОЛЬЗОВАНИЮ ПРОГРАММОЙ 10](#_Toc503178960)

[5. ПРИМЕР РАБОТЫ ПРОГРАММЫ 13](#_Toc503178961)

[ВЫВОДЫ 16](#_Toc503178962)

# ВВЕДЕНИЕ

Курсовая работа состоит из четырех заданий, условия большинства из которых зависят от варианта, который, в свою очередь, находится путем нахождения остатка от деления порядкового номера студента в списке на некоторое число. Мой номер в списке 20, а значит, варианты моих заданий таковы:

1. реализация приоритетной очереди к процессору – бинарная куча;
2. метод планирования центрального процессора – SJF с вытеснением и квантованием времени;
3. способ распределения памяти – битовая карта;
4. стратегия выбора свободной области памяти – стратегия следующего подходящего.

Далее будет рассмотрена постановка задач и описание реализации данных условий.

# ПОСТАНОВКА ЗАДАЧИ

Для вычислительной системы (ВС) с одним процессором и мультипрограммным режимом выполнения поступающих заданий требуется разработать программную систему для имитации процесса обслуживания заданий в вычислительных системах.

При построении модели функционирования вычислительной системы должны учитываться следующие основные моменты обслуживания заданий:

­ генерация нового задания;

­ постановка задания в очередь для ожидания момента освобождения процессора;

­ выборка задания из очереди при освобождении процессора после обслуживания очередного задания.

Считается, что в распоряжении вычислительной системы имеется N байт оперативной памяти для размещения рабочей области процесса и M (1<=m<=3) ресурсов R1,R2,…,Rm, обращение к которым переводит процесс в состояние ожидания.

Генерация нового задания (процесса) может происходить:

­ в интерактивном режиме по запросу пользователя;

­ автоматически системой как случайное событие.

Каждый процесс характеризуется:

­ именем;

­ длиной рабочей области;

­ интервалом непрерывного выполнения;

­ причиной прекращения непрерывной работы (обращение к ресурсу или завершение работы);

­ приоритетом, если он требуется используемым методом планирования процессора.

Перед постановкой задания в очередь имитируется размещение рабочей области процесса в оперативной памяти. В случае невозможности размещения процесс отвергается, в противном случае ему выделяется память и процесс помещается в очередь готовых заданий.

Размещение процессов в ОП происходит методом страничного размещения либо одним из четырех методов смежного размещения:

1. первого подходящего;

2. следующего подходящего

3. наиболее подходящего;

4. наименее подходящего;

Выборка задания из очереди готовых процессов происходит в момент, когда текущий процесс исчерпал интервал непрерывной работы и освободил центральный процессор.

В случае обращения к ресурсу процесс помещается в очередь к нему, причем время использования ресурса генерируется случайным образом.

В случае завершения процесс удаляется из очереди готовых процессов.

Все очереди к ресурсам обслуживаются алгоритмом FCFS (в порядке поступления). Считается, что в каждый момент времени процесс может обратиться только к одному ресурсу. По окончании работы с ресурсом процесс вновь помещается в очередь готовых заданий, причем генерируется новые интервал непрерывной работы и причина ее прекращения.

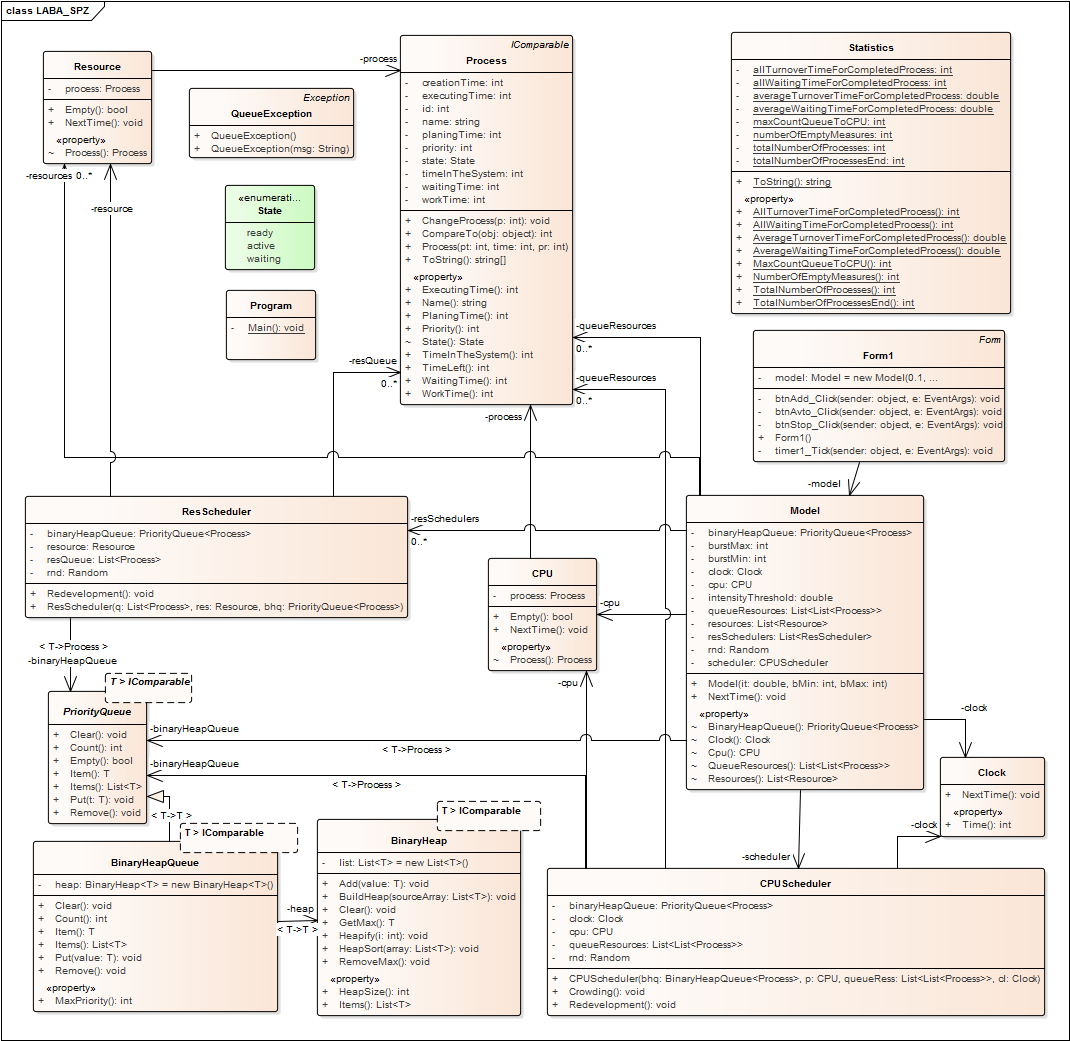
# ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ АЛГОРИТМОВ ПЛАНИРОВАНИЯ ПРОЦЕССОРА И ПАМЯТИ

Используемый алгоритм планирования процессора для моего варианта HPF с вытеснением. А это значит, что выборка процесса из очереди к процессору проводится с учетом приоритета процесса, то есть первым на обслуживание к процессору идет процесс с наибольшим приоритетом.

Алгоритм с вытеснением означает, что если процессор занят процессом, а в очереди появляется новый процесс с меньшим временем на выполнение, процессор освобождает исполняемый в данный момент процесс и начинает работу с процессом, обладающим наименьшим временем на выполнение. Освобожденный процесс возвращается обратно в очередь.

Что касается способа реализации метода распределения памяти, в лабораторных работах используется битовая карта, согласно варианту. При работе с битовым массивом память разделяется на блоки размером от нескольких слов до нескольких килобайтов. В битовой карте каждому свободному блоку соответствует один нулевой бит, а каждому занятому блоку — бит, установленный в 1.

В каждый момент времени в памяти присутствует несколько свободных областей различных размеров. Когда появляется новый процесс, нуждающийся в памяти, планировщик ищет среди них свободную область, достаточную для размещения процесса.

1. ОБЪЕКТНАЯ МОДЕЛЬ

За всё время работы я написал 10 классов

1. Класс BinaryHeap – структура очереди под названием бинарная куча, упорядочивает элементы очереди для дальнейшей передачи на процессор. Этот класс имеет следующие методы:
   1. Clear() – метод очищающий очередь
   2. HeapSize() – метод возвращающий размер очереди
   3. Items() – метод возвращающий максимальный элемент очереди
   4. RemoveMax() – метод удаляющий элемент с максимальным приоритетом
   5. Add() – метод добавляющий новый элемент в очередь
   6. Heapify() – метод сортирующий очередь методом бинарной кучи
   7. ToArray() – метод для вывода очереди в виде массива
   8. BuildHeap() – метод строящий бинарную кучу из неупорядоченного массива
   9. GetMax() – извлекает элемент с макс. приоритетом, т.е. корень и восстанавливает кучу
   10. HeapSort() – метод сортирующий массив с помощью бинарной кучи
2. Класс Model – нужен для имитации операционной системы.
   1. NextTime() – описывает шаг системы
   2. Model() – конструктор
3. Класс Process – описывает процессы. Отвечает за поступление новых процессов, создание процессов, различные операции типа ожидания и т.д.
   1. Process() – конструктор
   2. ToString() – выводит информацию о процессе
   3. CompareTo() – сравнение двух процессов
4. Класс CPU – необходим, что б организовывать процессы, поступающие на процессор.
   1. Empty() – оповещает, есть ли на процессоре процесс
   2. NextTime() – шаг работы процессора
5. Класс Clock – считает время работы процессов.
   1. NextTime() – переход к следующему такту
6. Класс CPUScheduler – планировщик процессов.
   1. CPUScheduler() - конструктор
   2. Crowding() – вытеснение процесса с процессора, если у первого больший приоритет
   3. Redevelopment() – перепланировка, если процессор пустой, или запланированное время = время исполнение на процессоре
7. Класс Statistics – выводит статистику по параметрам
   1. ToString() – выводит информацию о статистику
8. Класс Resource – описывает ресурс
   1. NextTime() – метод увеличивающий время исполнения процесса на CPU
   2. Empty() – метод проверяющий на пустоту ресурса
9. Класс ReScheduler – перепланировщик работы ресурса
   1. ReScheduler() – конструктор
   2. Redevelopment() – перепланировка ресурса, если он не пустой и запланированное время = время исполнения

Вот примеры некоторых методов:

public void Redevelopment()

{

if (!resource.Empty() && resource.Process.PlaningTime == resource.Process.WorkTime)

{

resource.Process.State = State.ready;

resource.Process.ChangeProcess(rnd.Next(1, 20));

binaryHeapQueue.Put(resource.Process);

resource.Process = null;

}

if (resource.Empty() && resQueue.Count != 0)

{

resource.Process = resQueue[0];

resQueue.RemoveAt(0);

}

}

public void Crowding()

{

if (!cpu.Empty() && !binaryHeapQueue.Empty() && cpu.Process.CompareTo(binaryHeapQueue.Item()) < 0)

{

cpu.Process.State = State.ready;

binaryHeapQueue.Put(cpu.Process);

cpu.Process = binaryHeapQueue.Item();

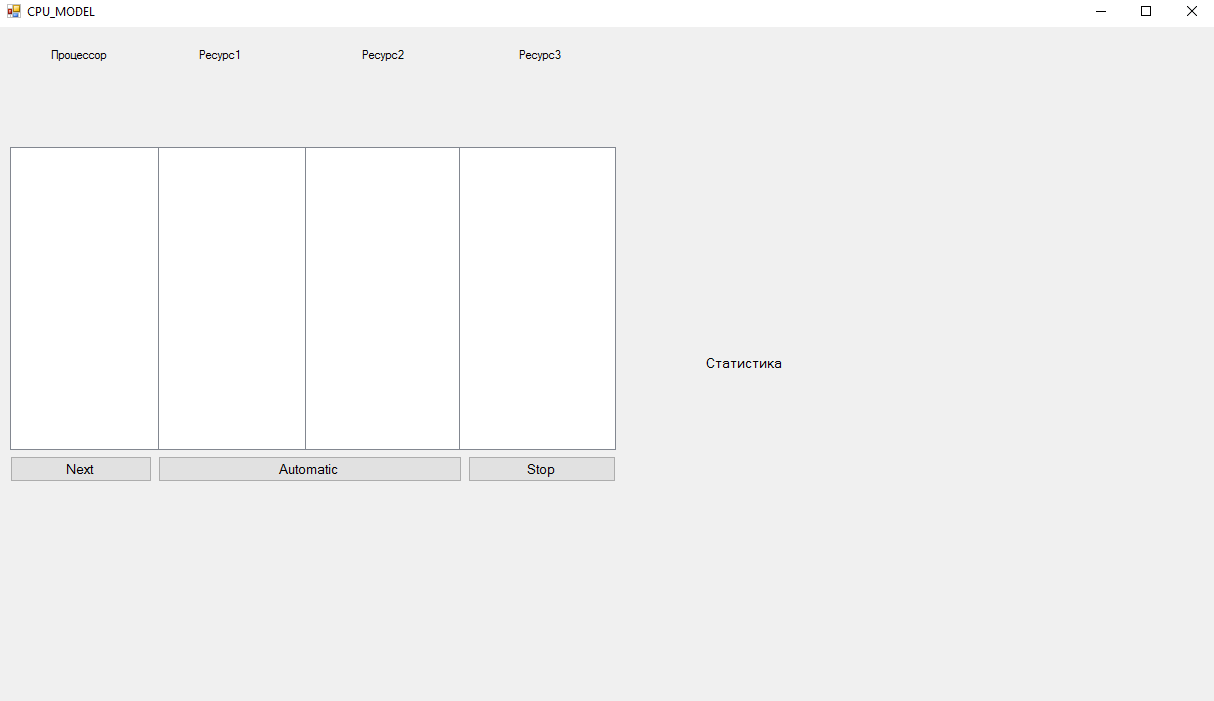
cpu.Process.State = State.active;

binaryHeapQueue.Remove();

}

}

# ИНСТРУКЦИЯ ПО ПОЛЬЗОВАНИЮ ПРОГРАММОЙ



(Рис.1) Общий вид формы

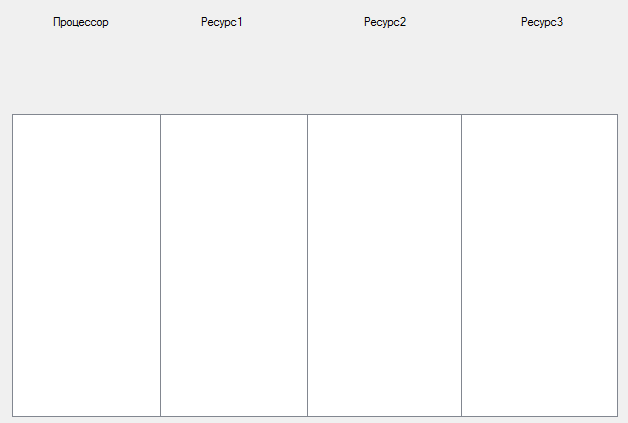
Так выглядит форма.

Внизу расположены кнопки для ручной регулировки:



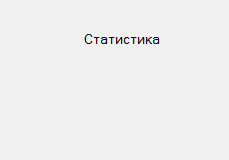
(Рис. 2) Кнопки для ручного ввода

Немалую часть формы занимает область для наблюдения за работой процессов на процессоре. Какими ресурсами пользуются, сколько работают и т.д.:



(Рис. 4) Панель работы процессоров и ресурсов

Справа находится область для сбора статистических данных:



(Рис. 5) Область для сбора статистических данных

1. ПРИМЕР РАБОТЫ ПРОГРАММЫ

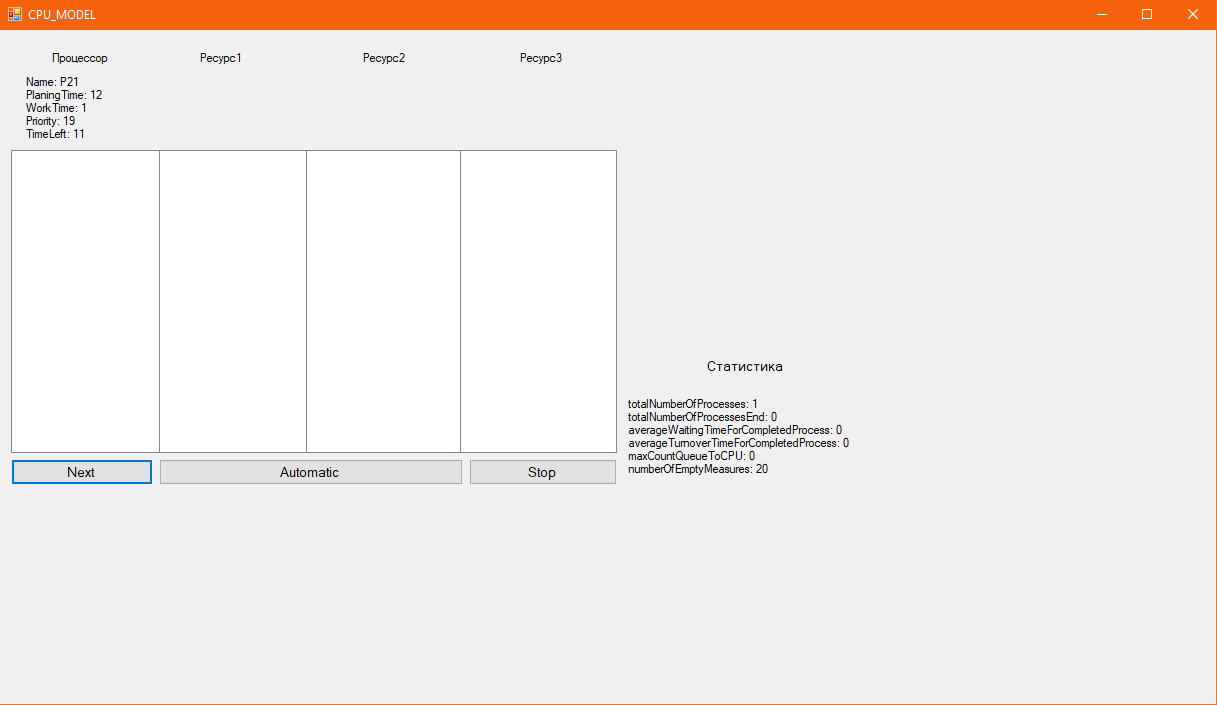
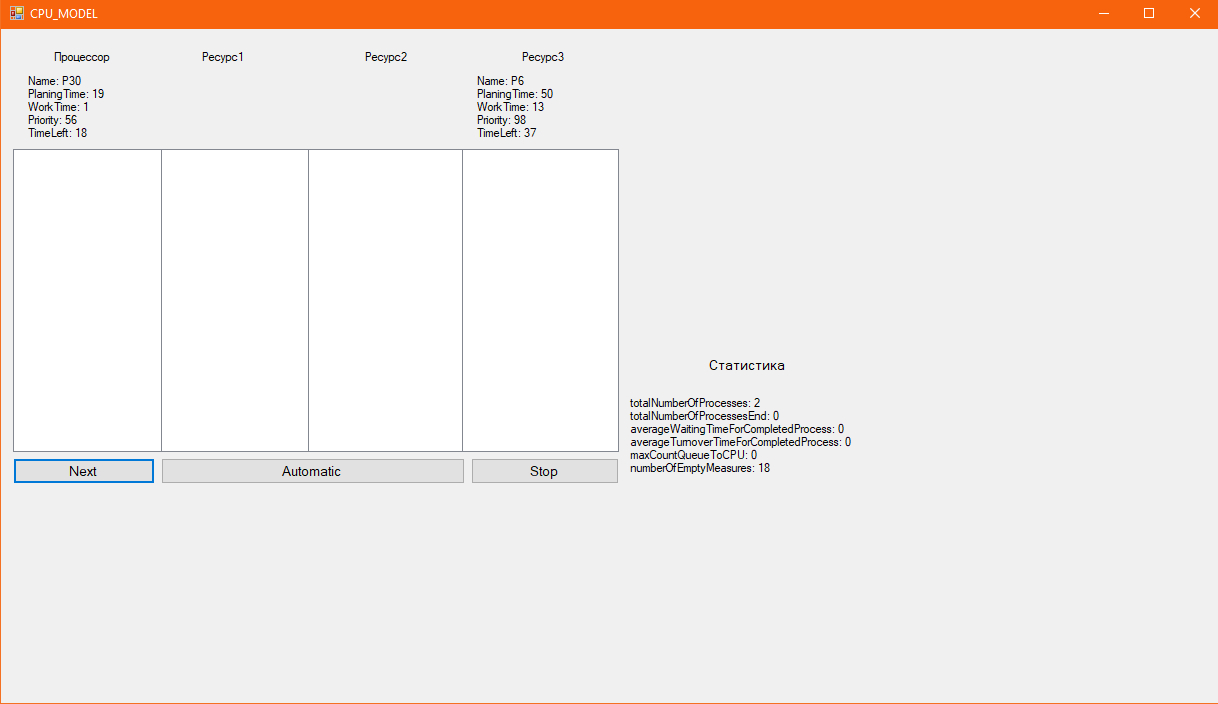
При создании первого процесса он автоматически попадает на CPU для дальнейшей обработки. У процесса имеются поля: «Имя», «Планируемое время работы», «Время работы», «Приоритет» и «Оставшееся время».

Рис.5.1

Во время работы генерируются новые процессы и может возникнуть ситуация, когда текущий процесс вытесняется более приоритетным процессом.

 Рис.5.2

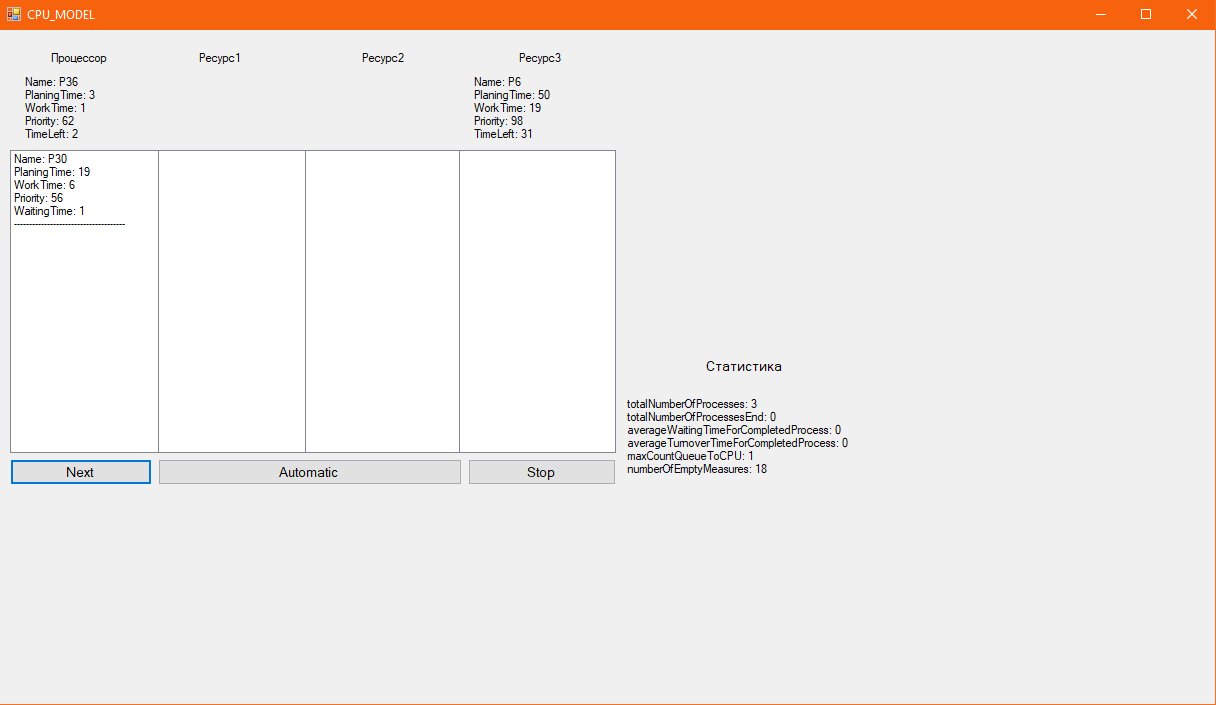


Рис.5.3

На рисунке 5.2 и 5.3 можно увидеть, как процесс <<P30>> закончил свою работу на процессоре и заместился процессом <<P36>>.

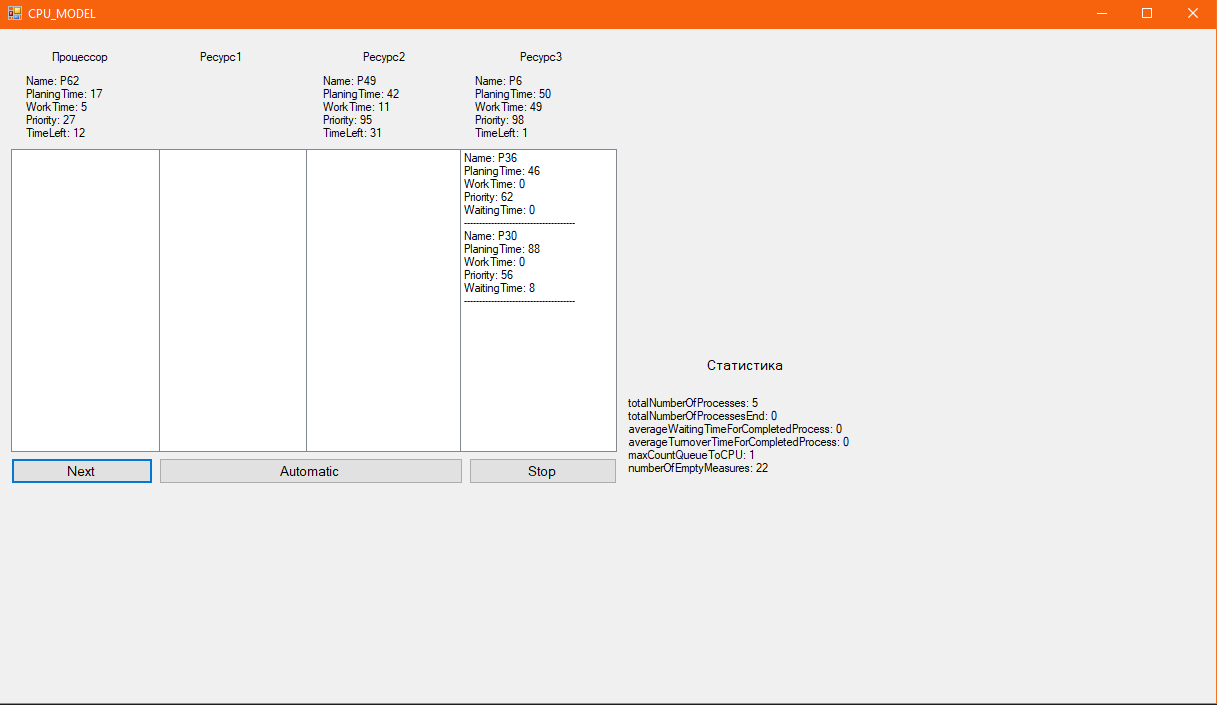
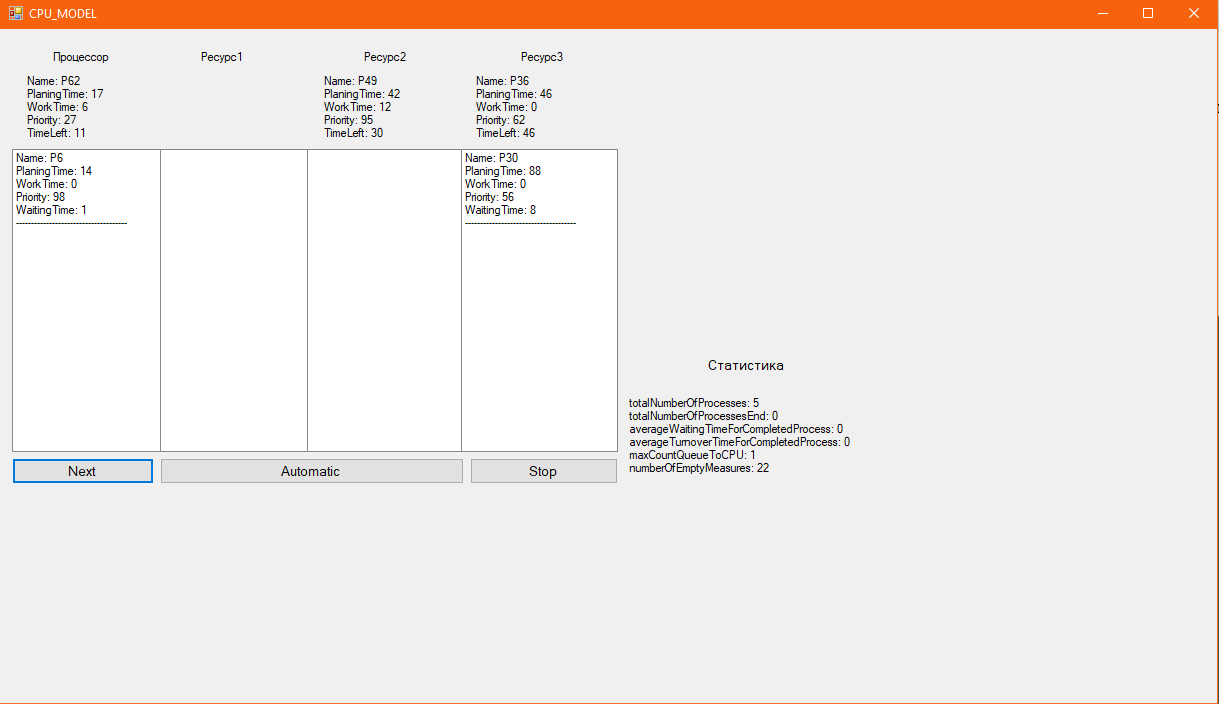


Рис.5.4

Рис.5.5

На рисунках 5.4 и 5.5 видно, как процесс <<P6>> на ресурсе 3 закончил работу и заместился процессом <<P36>>.

Справа от «Ресурса 3» присутствует постоянно обновляющаяся статистика.

# ВЫВОДЫ

Итак, модель, демонстрирующая алгоритм работы центрального процессора, работает правильно. В ходе выполнения лабораторных работ я разобрался со всеми алгоритмами планирования процессора и способами распределения памяти, а также написал класс приоритетной очереди, основанной на бинарном дереве поиска.