МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования
«Санкт-Петербургский государственный технологический институт
(технический университет)»
СПбГТИ(ТУ)

УГНС	09.00.00	Информатика и вычислительная техника
Направление подготовки	09.03.03	Прикладная информатика
Направленность (профиль)		Прикладная информатика в химии
Факультет		Информационных технологий и управления
Кафедра		Систем автоматизированного проектирования и управления
Учебная дисциплина		**ОПЕРАЦИОННЫЕ СИСТЕМЫ**

Курс 2 Группа 485

Отчет по лабораторной работе № 4

Тема: Управление процессами

Студент _____ Зобнин И.М.
(подпись, дата) (инициалы, фамилия)

Преподаватель _____ Макарук Р.В.
(подпись, дата) (инициалы, фамилия)

Отметка о зачете _____ _____
(подпись преподавателя)

Санкт-Петербург
2020

1. Цель работы

Получение навыков управления штатными средства ОС для исследования процессов. Также требуется разработать программу, реализующую алгоритм банкира

Исходные данные для тестирования алгоритма банкира

В контексте «алгоритма банкира» определите и обоснуйте, является ли приведенное состояние опасным или безопасным с точки зрения возникновения тупиков.

Предположим, что в системе имеются 4 одинаковых ресурсов R1, 4 одинаковых ресурсов R2, 4 одинаковых ресурсов R3 и 4 одинаковых ресурсов R4. Текущее распределение ресурсов и максимальное их количество, необходимое процессам представлено в таблице 1.

Таблица 1 – Распределение ресурсов

Процесс	**Предоставлено ресурсов** **R1 R2 R3 R4**	**Максимальная потребность** **R1 R2 R3 R4**
A	2 0 0 0	2 0 2 2
B	2 2 0 0	2 2 2 2
C	0 2 2 0	2 4 2 4
D	0 0 2 2	0 0 2 4

2. Ход работы

Код программы:

[файл Program.cs]

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

namespace Lab4
{
    class Program
    {
        static string NL = Environment.NewLine;
        static Tuple<List<Resource>, List<Process>> GetResourcesAndProcesses()
        {
            var resources = new List<Resource>();
            var processes = new List<Process>();
            using (var reader = new StreamReader("Example.txt"))
            {
                while (!reader.EndOfStream)
                {
                    string lineOfResources = reader.ReadLine();

                    lineOfResources.Split(',').ToList().ForEach(res => resources.Add(new
Resource(res)));

                    reader.ReadLine();

                    while (true)
                    {
                        string processVector = reader.ReadLine();
```

```
                    if (string.IsNullOrEmpty(processVector))
                        break;
                    processes.Add(new Process(processVector));
                }

                foreach (var process in processes)
                    process.MaxResVector = reader.ReadLine();
            }
        }
        return new Tuple<List<Resource>, List<Process>>(resources, processes);
    }

    static void OutputProcessesAndGivenAndNeededResources(List<Process> processes,
List<Resource> vectorA)
    {
        Console.WriteLine("Предоставлено ресурсов:");
        Console.Write(" ");
        foreach (var resource in vectorA)
            Console.Write(resource.Name);
        Console.WriteLine();

        foreach (var process in processes)
        {
            Console.Write(process.Name + " ");
            foreach (var givenResource in process.GivenRes)
                Console.Write(givenResource + " ");
            Console.WriteLine();
        }

        Console.WriteLine("Ресурсов необходимо:");
        Console.Write(" ");
        foreach (var resource in vectorA)
            Console.Write(resource.Name);
        Console.WriteLine();

        foreach (var process in processes)
        {
            Console.Write(process.Name + " ");
            foreach (var neededResource in process.NeededRes)
                Console.Write(neededResource + " ");
            Console.WriteLine();
        }

        Console.Write("Вектор A={");
        for (int i = 0; i < vectorA.Count; ++i)
        {
            Console.Write(vectorA[i].ResourceVol);
            if (i != vectorA.Count - 1)
                Console.Write(", ");
        }
        Console.WriteLine("}" + NL);
    }

    static Tuple<bool, string> DoTask(List<Process> processes, List<Resource>
vectorA)
    {
        bool unsafeCondition = false;
        string processOrder = "";
        while (processes.Count != 0 && !unsafeCondition)
        {
            int processesNum = processes.Count;
            OutputProcessesAndGivenAndNeededResources(processes, vectorA);

            foreach (var process in processes)
            {
```

```
                    bool toProvideByResource = true;
                    for (int i = 0; i < process.NeededRes.Count; ++i)
                        if (vectorA[i].ResourceVol < process.NeededRes[i])
                            toProvideByResource = false;

                    if (toProvideByResource)
                    {
                        for (int i = 0; i < process.GivenRes.Count; ++i)
                            vectorA[i].ResourceVol += process.GivenRes[i];
                        processOrder += process.Name + " ";
                        processes.Remove(process);
                        break;
                    }
                }

                if (processes.Count == processesNum)
                    unsafeCondition = false;
            }
            return new Tuple<bool, string>(unsafeCondition, processOrder);
        }

        static void Main()
        {
            var resourcesAndProcesses = GetResourcesAndProcesses();
            var resources = resourcesAndProcesses.Item1;
            var processes = resourcesAndProcesses.Item2;

            Console.Write("Имеется ресурсов: ");
            foreach (var resource in resources)
                Console.Write(resource.Name + "=" + resource.ResourceVol + " ");
            Console.WriteLine(NL);

            var vectorA = resources;

            foreach (var process in processes)
                for (int i = 0; i < process.GivenRes.Count; ++i)
                    vectorA[i].ResourceVol = vectorA[i].ResourceVol -
process.GivenRes[i];

            var result = DoTask(processes, vectorA);
            var unsafeCondition = result.Item1;
            var processesOrder = result.Item2;

            if (!unsafeCondition)
                Console.WriteLine("Состояние безопасное.");
            else
                Console.WriteLine("Состояние опасное.");
            Console.WriteLine("Последовательность процессов, которым были выделены
ресурсы: " + processesOrder);
        }
    }
}
[конец файла Program.cs]


[файл Process.cs]
using System.Collections.Generic;
using System.Linq;

namespace Lab4
{
    class Process
    {
        public string Name { get; private set; }
```

```
        public Process(string processVector)
        {
            Name = string.Join("", processVector.TakeWhile(a => a != ' '));
            processVector = processVector.Remove(0, Name.Length + 1);
            processVector.Split(' ').ToList().ForEach(a => GivenRes.Add(int.Parse(a)));
        }

        public List<int> GivenRes { get; private set; } = new List<int>();

        public List<int> NeededRes { get; private set; } = new List<int>();

        public string MaxResVector
        {
            set
            {
                var maxValues = value.Split(' ').Select(a => int.Parse(a)).ToList();
                for (int i = 0; i < maxValues.Count; ++i)
                    NeededRes.Add(maxValues[i] - GivenRes[i]);
            }
        }
    }
}
```
[конец файла Process.cs]

[файл Resource.cs]
```
using System.Linq;

namespace Lab4
{
    class Resource
    {
        public string Name { get; private set; }

        public Resource(string resource)
        {
            Name = string.Join("", resource.TakeWhile(a => a != '='));
            ResourceVol = int.Parse(resource.Substring(Name.Length + 1)); // учитывая
знак равно (+1)
        }

        public int ResourceVol { get; set; }
    }
}
```
[конец файла Resource.cs]

2.1 Результат работы программы

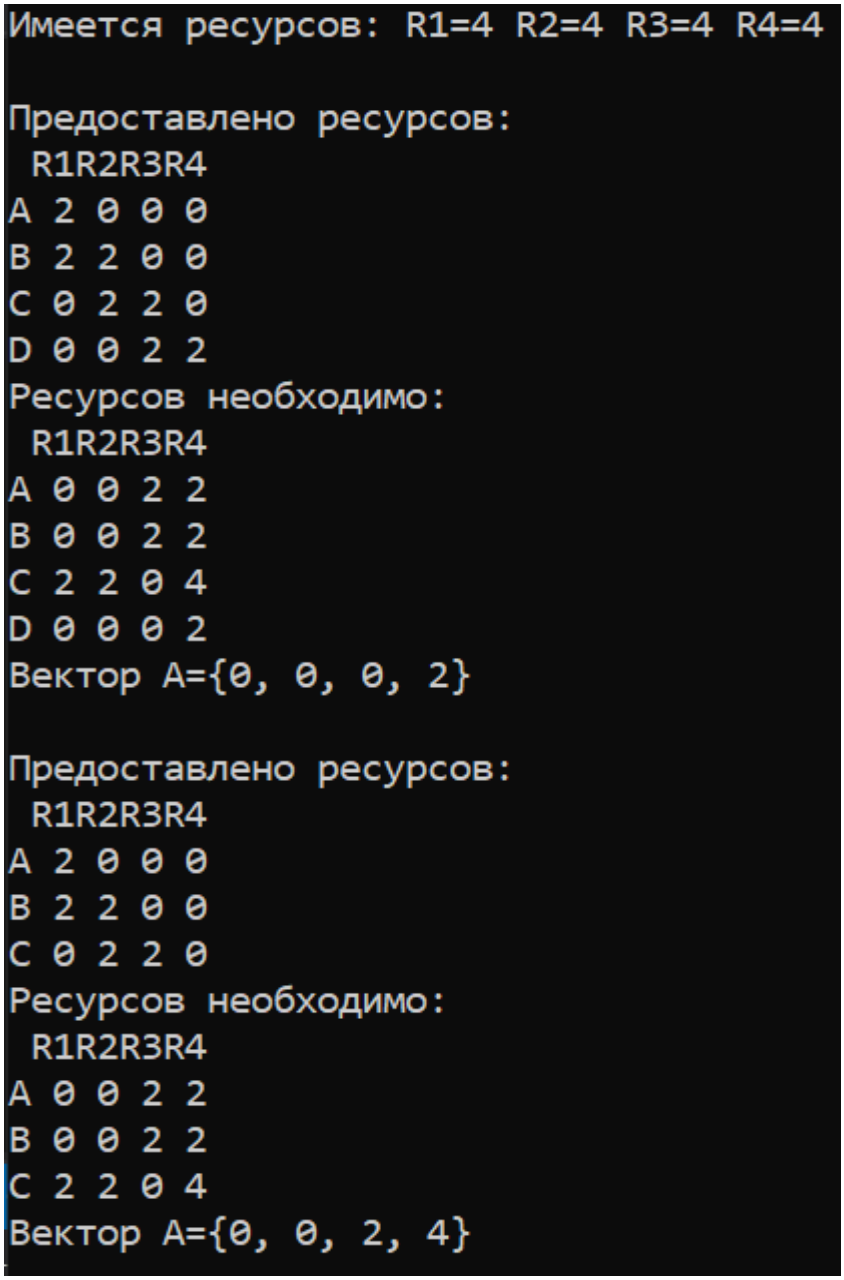

Рисунок 1 – Результат работы программы

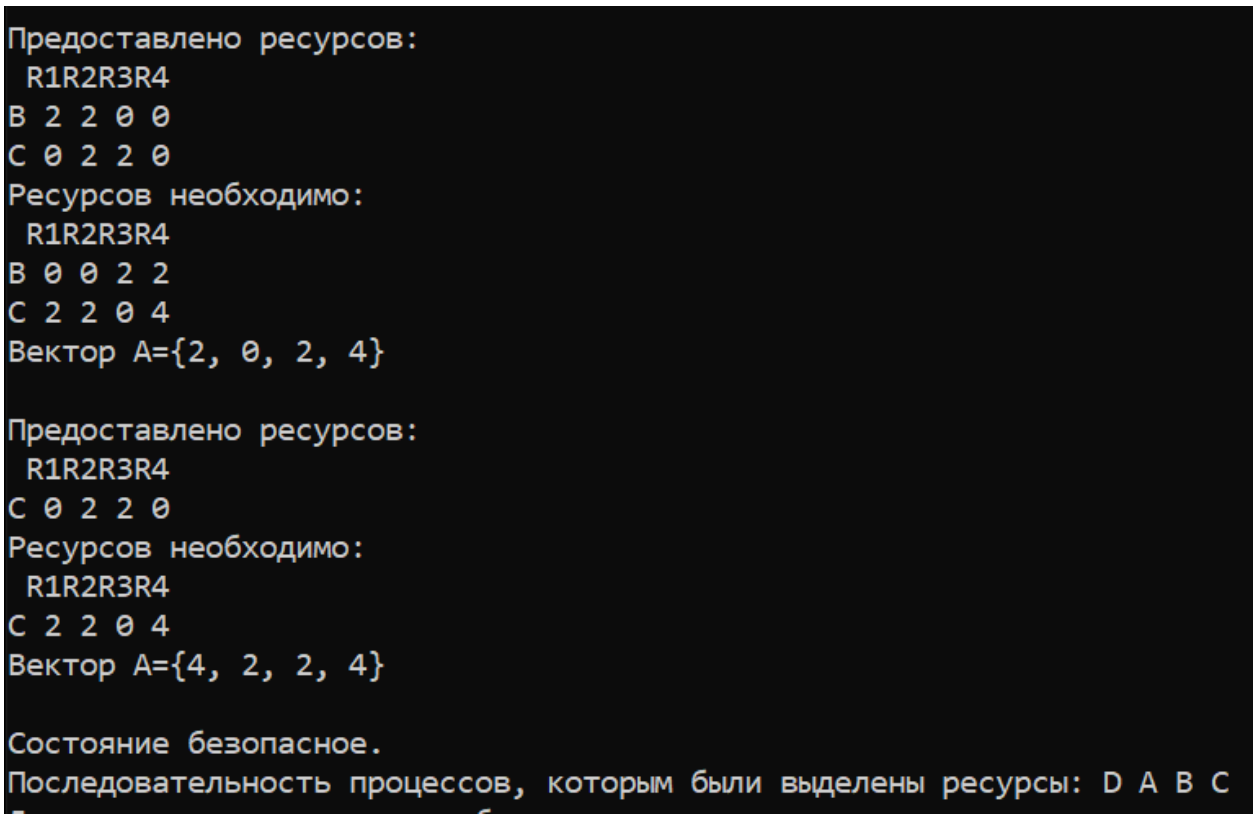

Рисунок 2 – Результат работы программы

2.2 Штатные средства ОС для исследования процессов

В качестве тестовой системы для выполнения практического задания использовалась MS Windows 10.

В качестве исследуемого приложения был выбран проигрыватель VLC. Выполнимая операция – воспроизведение мультимедиа.

В диспетчере задач MS Windows виден созданный процесс (Рисунок 3).

Диспетчер задач позволяет получить обобщенную информацию об использовании основных ресурсов компьютера: общее количество процессов и потоков, участвующих в системе и т.д.

В диспетчере задач отображаются сведения о процессах, выполняемых на компьютере. Кроме того, там можно просмотреть наиболее часто используемые показатели быстродействия процессов.

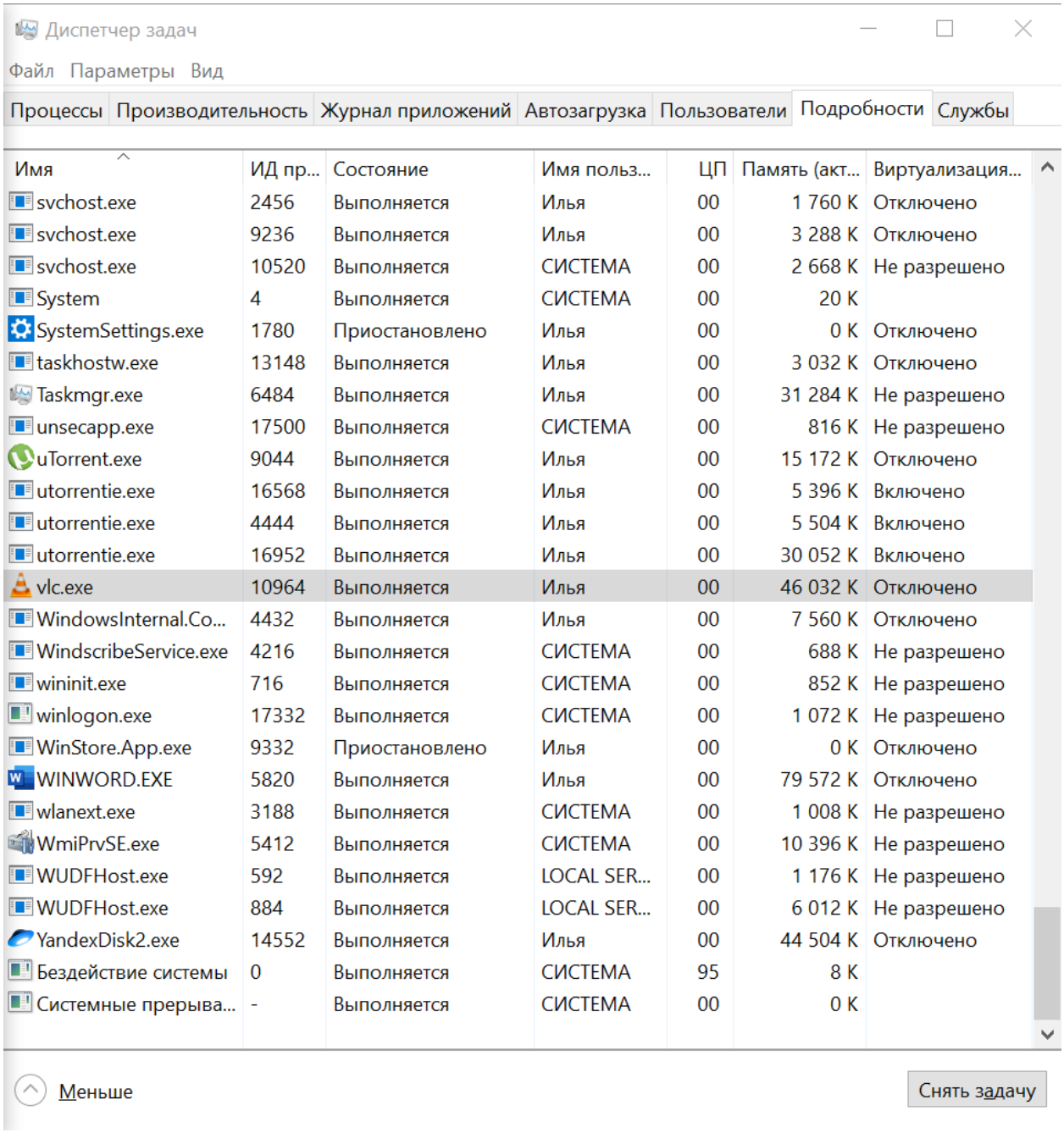

Рисунок 3 – Характеристики запущенного процесса

Просмотр (мониторинг) выполнения процесса со счётчиками представлен на Рисунке 4.

Рисунок 4 – Выполнение операции при среднем приоритете

Системный монитор служит для сбора и просмотра в реальном времени данных памяти, диска, процессора, сети и других параметров в виде графика, гистограммы или отчета.

Анализ данных наблюдения позволяет обнаружить такие явления, как избыточный спрос на определенные ресурсы, приводящий к возникновению узкого места в работе системы.

Перед выполнением исследуемой задачи устанавливаем сначала минимальный приоритет процесса (Рисунок 5)

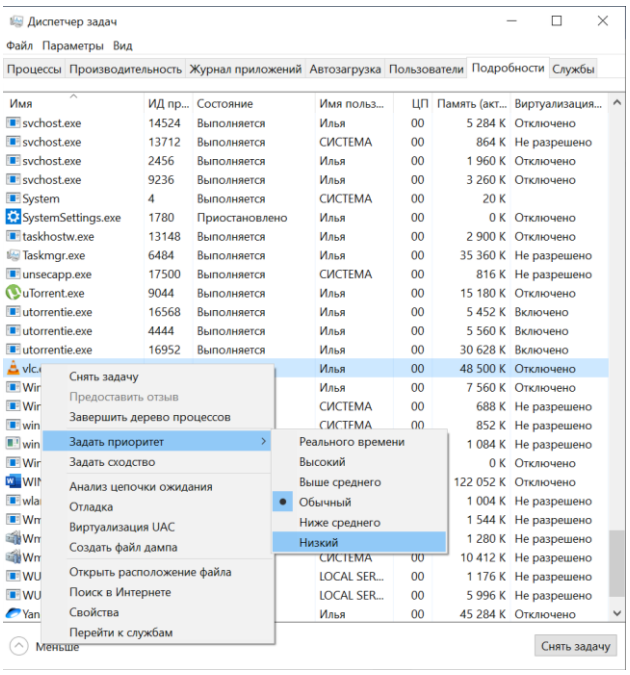

Рисунок 5 – Понижение базового приоритета

На Рисунке 6 можно наблюдать ход выполнения самой емкой операции при низком приоритете.

Рисунок 6 – Выполнение операции при пониженном приоритете

Изменим приоритет запущенного процесса (Рисунок 7)

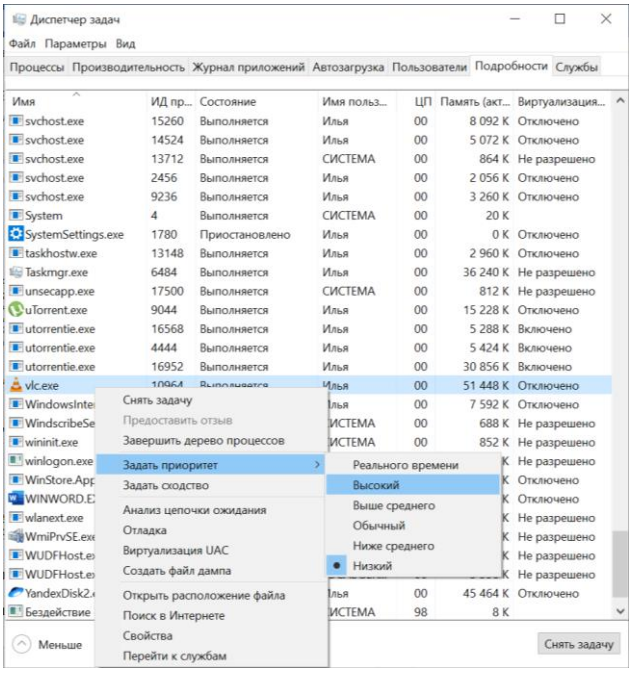

Рисунок 7 – Повышение базового приоритета

На Рисунке 8 показано выполнение той же операции с повышением приоритета.

Рисунок 8 – Процесс выполняется с повышенным приоритетом

Из представленных рисунков очевидно, что при изменении приоритета процесса резко изменяется характер работы процесса в пользовательском режиме в много процессной системе - задача выполняется быстрее.

Диспетчер задач может показать количество потоков, созданных в конкретном процессе. В рассматриваемом процессе было создано 24 потока. Эта многопоточность в рамках одного процесса положительно влияет на производительность задачи.

Теперь посмотрим на работу VLC в Windows XP. При обычном приоритете Рисунок 9.

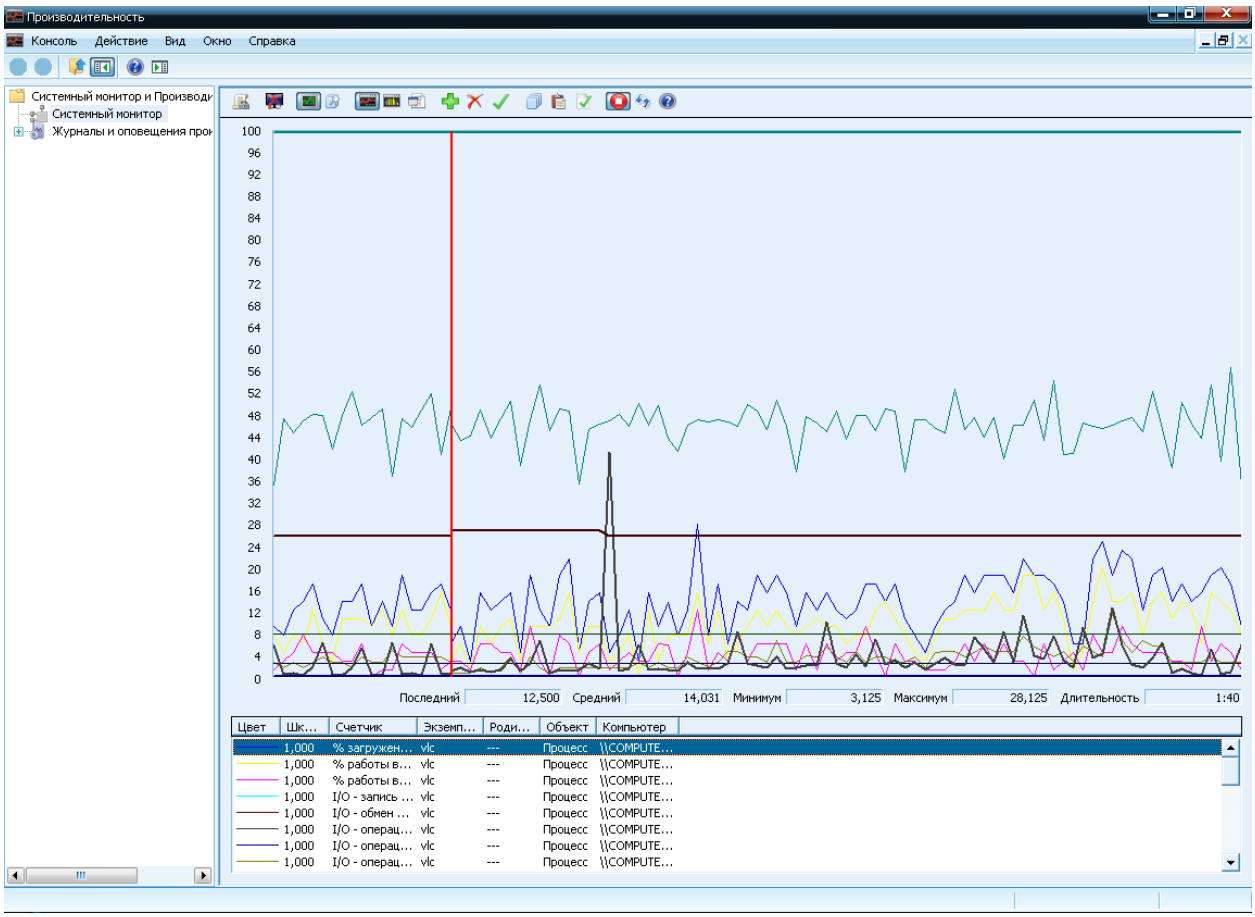

Рисунок 9 – Выполнение операции в обычном приоритете.

Теперь понизим приоритет и взглянем на работу (Рисунок 10).

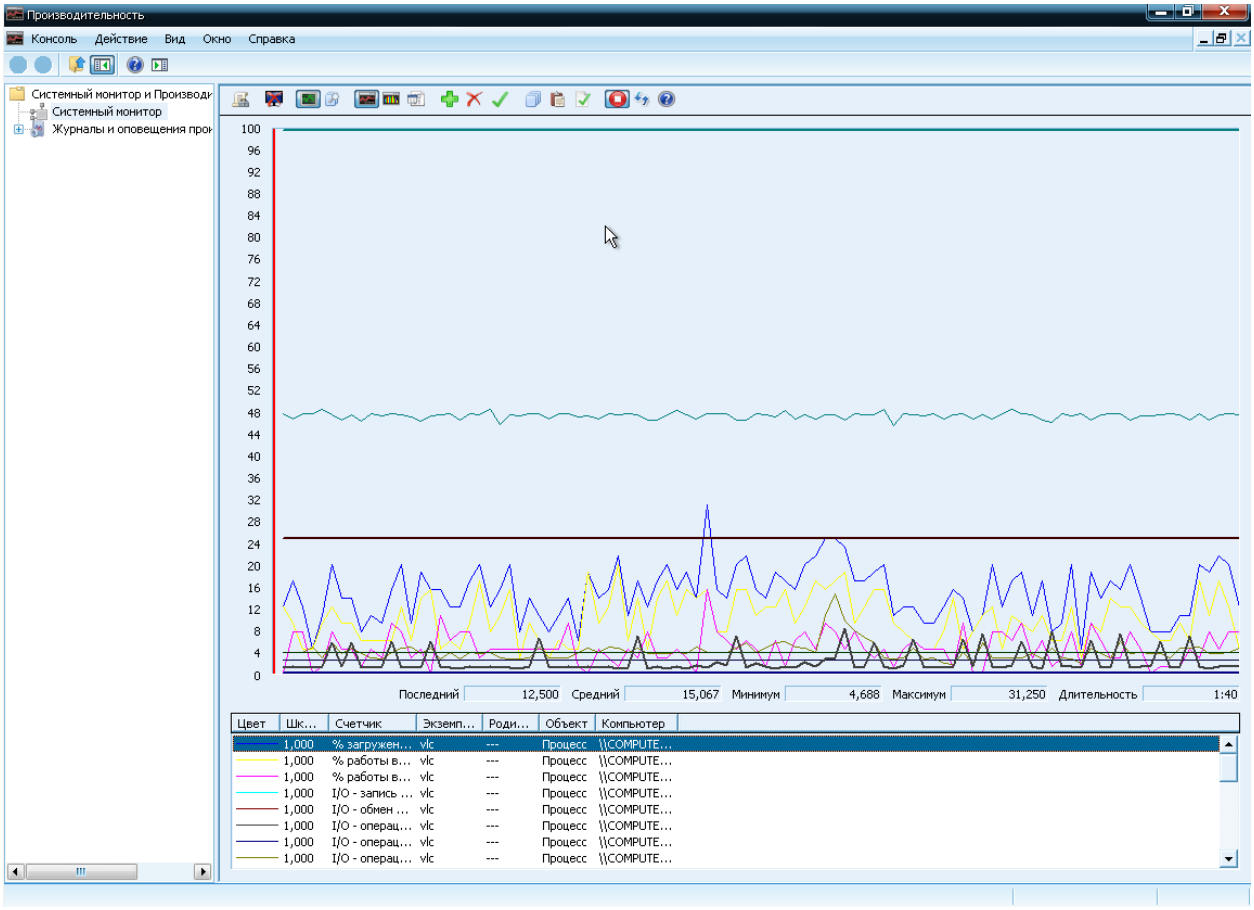

Рисунок 10 – Выполнение операции при пониженном приоритете

Теперь проследим за работой процесса при высоком приоритете (Рисунок 11).

Рисунок 11 – Выполнение операции при повышенном приоритете.

Как и в Windows 10, процессы в Windows XP положительно реагируют на повышение приоритета, система выделяет на их выполнение больше ресурсов, что ускоряет их работу.

3 Ответы на вопросы

Концепция дискретных состояний процесса.

В период существования процесс проходит через ряд дискретных состояний. Смену состояний могут вызвать различные события. Базовые состояния процессов в системе:

1. Готов (Ready) - находится в состоянии готовности (т.е. может выполняться). Процесс, которому выделены все ресурсы кроме ЦП носит название готового к исполнению. Когда процессу выделяется ЦП - происходит смена состояний. Предоставление процессу ЦП называется запуском. Одновременно в системе может находиться несколько процессов (список готовых к исполнению процессов).
2. Выполняется (Running) - продолжающиеся (в состоянии выполнения). Говорит, что процесс выполняется, если ему выделен ЦП. В одном процессоре только один процесс может находиться в исполнении.
3. Блокирован (Blocked) - процесс блокирован или в состоянии блокирования, если не может выполняться пока не получен необходимый ему ресурс или сообщение от другого процесса.

Стратегии разрешения тупиков.

Методы предотвращения тупиков ориентированы главным образом на нарушение первых трех условий возникновения тупиков (условие взаимоисключения, условие ожидания ресурсов и условие неперераспределяемости) путем введения ряда ограничений на поведение процессов и способы распределения ресурсов. Эти методы обнаружения и устранения сводятся к поиску и разрыву цикла ожидания ресурсов.

Основные стратегии борьбы с тупиками:

- Игнорирование проблемы в целом
- Предотвращение тупиков
- Обнаружение тупиков
- Восстановление после тупиков

4.Вывод

В ходе выполнения данной работы, на примере MS Windows 10, были получены навыки управления средствами ОС для изучения работы процессов, были отмечены основные стратегии борьбы с тупиками, а также была разработана программа, реализующая алгоритм банкира.