

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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(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.Linq;

using Resources = System.Collections.Generic.Dictionary<string, int>;

namespace Lab4

{

class Program

{

static void OutputAll(Dictionary<string, Resources> processesAndGivenResources,

Dictionary<string, Resources> processesAndResourcesRequired,

Resources remainResources)

{

var maxStrLength = processesAndGivenResources.Keys.Select(s => s.Length).Max();

string indent = " ";

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

indent += " ";

string resGivenStr = "Предоставлено ресурсов:";

string maxResRequiredStr = "Максимальная потребность:";

Console.WriteLine(indent + resGivenStr + " " + maxResRequiredStr);

string indent2 = "";

for (int i = 0; i < resGivenStr.Length / 2; i++)

indent2 += " ";

string indent3 = "";

for (int i = 0; i < maxResRequiredStr.Length / 2; i++)

indent3 += " ";

foreach (var process in processesAndGivenResources)

{

Console.Write(process.Key + indent2);

foreach (var resource in processesAndGivenResources[process.Key])

Console.Write(resource.Value + " ");

Console.Write(indent3);

foreach (var resource in processesAndResourcesRequired[process.Key])

Console.Write(resource.Value + " ");

Console.WriteLine();

}

Console.Write("Осталось ресурсов:");

OutputRes(remainResources);

Console.WriteLine();

}

static void OutputRes(Resources resources)

{

foreach (var resource in resources)

Console.Write(resource.Value + " ");

}

static bool DoAlgorithm(Dictionary<string, Resources> processesAndGivenResources,

Dictionary<string, Resources> processesAndResourcesRequired,

Resources remainResources)

{

bool safe = true;

while (safe && processesAndGivenResources.Any() && processesAndResourcesRequired.Any())

{

safe = false;

foreach (var process in processesAndResourcesRequired)

{

var affortableRes = new List<bool>();

var requiredResources = process.Value;

foreach (var resource in requiredResources)

{

var resName = resource.Key;

var requiredResVal = resource.Value;

var givenResources = processesAndGivenResources[process.Key];

var givenResVal = givenResources[resName];

var remainResVal = remainResources[resName];

var distributableResVal = givenResVal + remainResVal;

if (distributableResVal >= requiredResVal)

affortableRes.Add(true);

else

affortableRes.Add(false);

}

if (!affortableRes.Contains(false))

{

safe = true;

Console.Write($"Процесс {process.Key} завершился и освободил: ");

var givenRes = processesAndGivenResources[process.Key];

OutputRes(givenRes);

for (int i = 0; i < remainResources.Count; i++)

{

var resourceName = remainResources.ElementAt(i).Key;

remainResources[resourceName] += givenRes[resourceName];

}

Console.Write('\t' + "Осталось ресурсов: ");

OutputRes(remainResources);

Console.WriteLine();

processesAndGivenResources.Remove(process.Key);

processesAndResourcesRequired.Remove(process.Key);

break;

}

}

}

return safe;

}

static void Main()

{

var allResourcesAvailable = new Resources

{

{ "R1", 4 },

{ "R2", 4 },

{ "R3", 4 },

{ "R4", 4 }

};

var processesAndGivenResources = new Dictionary<string, Resources>

{

{ "A", new Resources() { { "R1", 2 }, { "R2", 0 }, { "R3", 0 }, { "R4", 0 } } },

{ "B", new Resources() { { "R1", 2 }, { "R2", 2 }, { "R3", 0 }, { "R4", 0 } } },

{ "C", new Resources() { { "R1", 0 }, { "R2", 2 }, { "R3", 2 }, { "R4", 0 } } },

{ "D", new Resources() { { "R1", 0 }, { "R2", 0 }, { "R3", 2 }, { "R4", 2 } } }

};

var processesAndResourcesRequired = new Dictionary<string, Resources>

{

{ "A", new Resources() { { "R1", 2 }, { "R2", 0 }, { "R3", 2 }, { "R4", 2 } } },

{ "B", new Resources() { { "R1", 2 }, { "R2", 2 }, { "R3", 2 }, { "R4", 2 } } },

{ "C", new Resources() { { "R1", 2 }, { "R2", 4 }, { "R3", 2 }, { "R4", 4 } } },

{ "D", new Resources() { { "R1", 0 }, { "R2", 0 }, { "R3", 2 }, { "R4", 4 } } }

};

var remainRes = new Resources();

foreach (var availableRes in allResourcesAvailable)

{

var resName = availableRes.Key;

var resVal = availableRes.Value;

foreach (var process in processesAndGivenResources)

{

resVal -= process.Value[resName];

}

remainRes.Add(resName, resVal);

}

OutputAll(processesAndGivenResources, processesAndResourcesRequired, remainRes);

bool safe = DoAlgorithm(processesAndGivenResources, processesAndResourcesRequired, remainRes);

if (safe)

Console.WriteLine("Состояние безопасное.");

else

Console.WriteLine("Состояние небезопасное.");

}

}

}

[конец файла Program.cs]

**2.1 Результат работы программы**

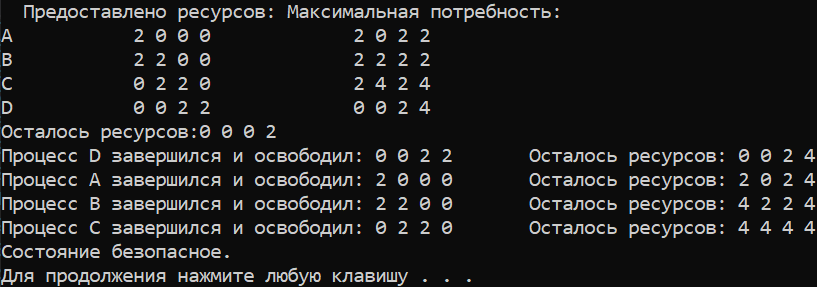


Рисунок 1 – Результат работы программы

**2.2 Штатные средства ОС для исследования процессов**

В качестве тестовой системы для выполнения практического задания использоваласьMS Windows 10.

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

В диспетчере задач MS Windows виден созданный процесс (Рисунок 2).

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.

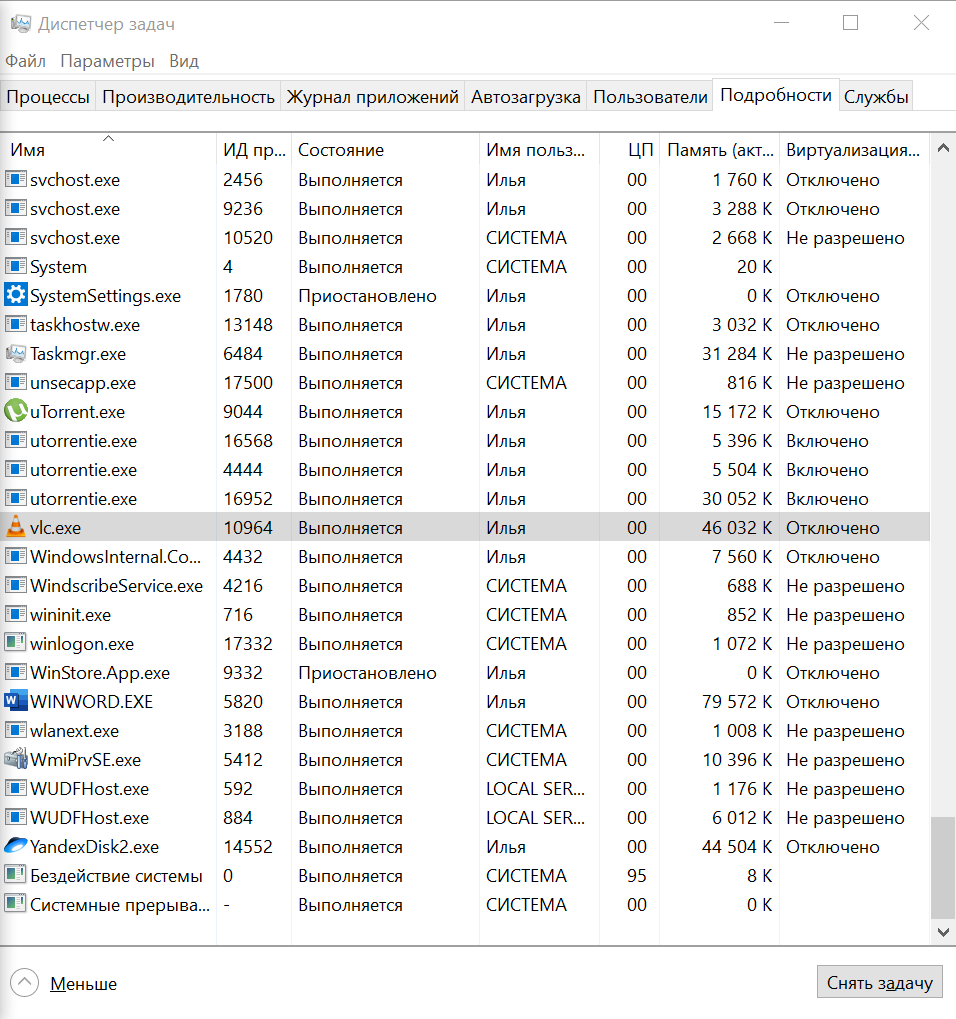


Рисунок 2 – Характеристики запущенного процесса

Просмотр (мониторинг) выполнения процесса со счётчиками представлен на Рисунке 3.

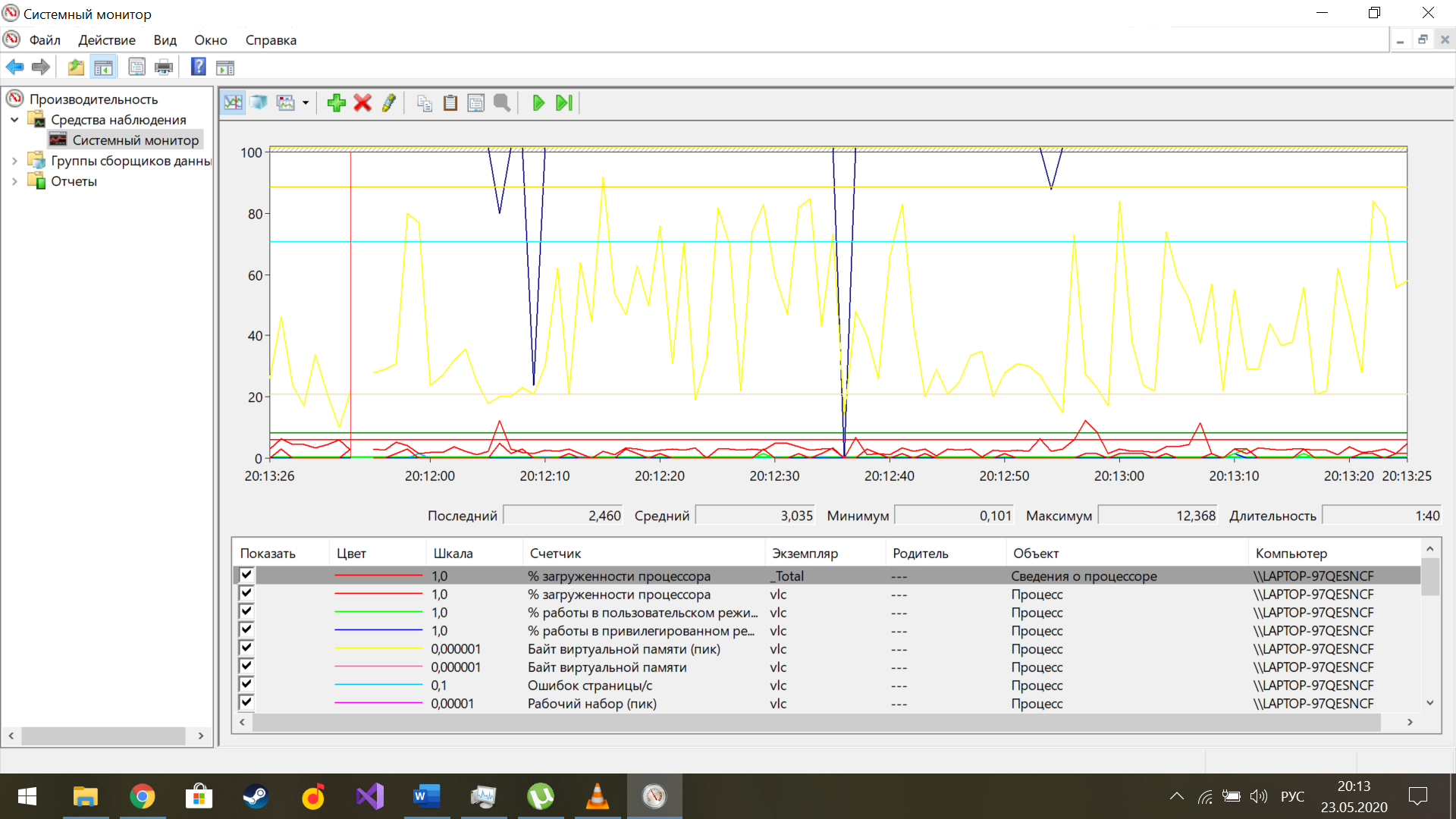


Рисунок 3 – Выполнение операции при среднем приоритете

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 4)

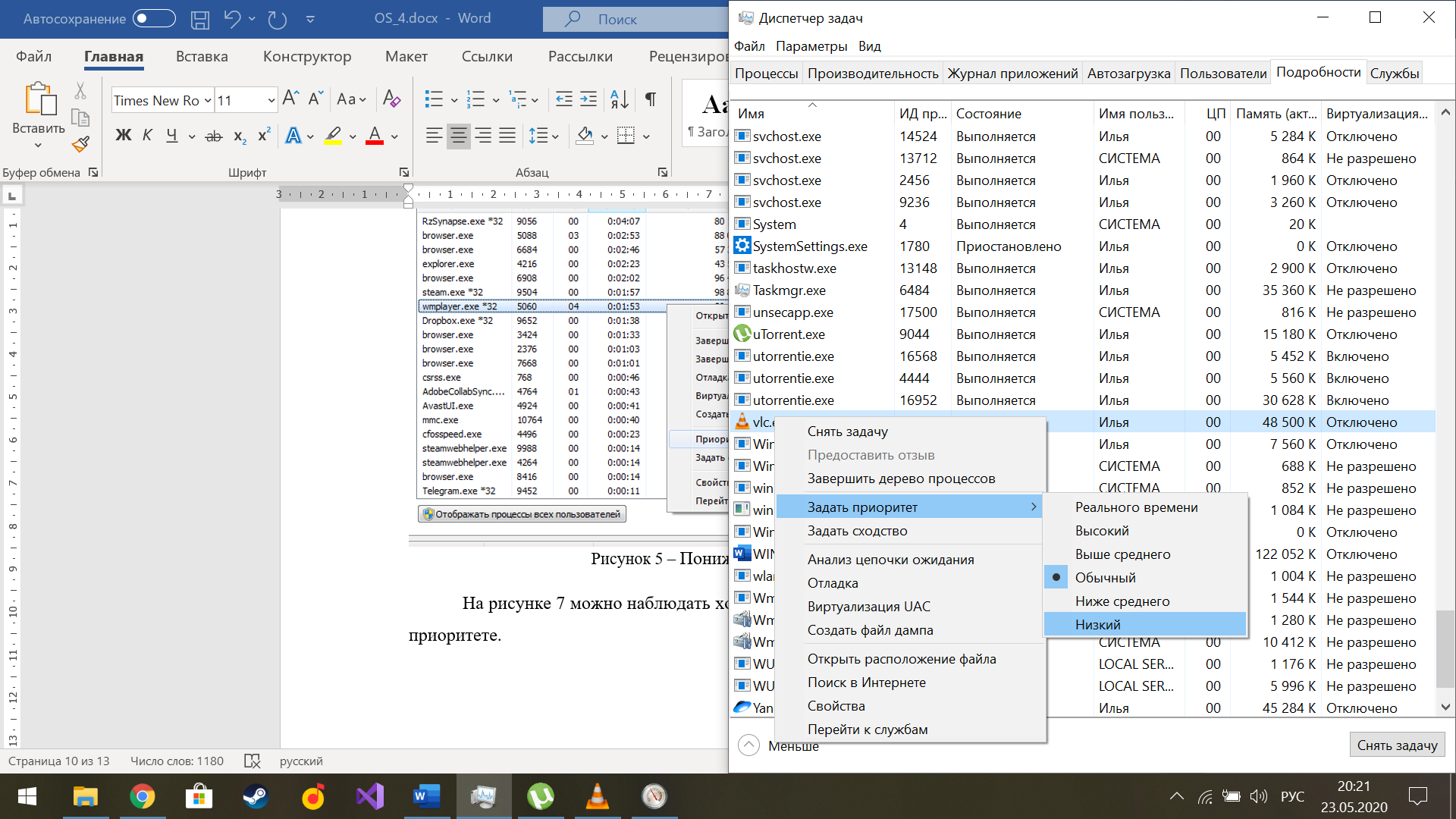


Рисунок 4 – Понижение базового приоритета

На Рисунке 5 можно наблюдать ход выполнения самой емкой операции при низком приоритете.

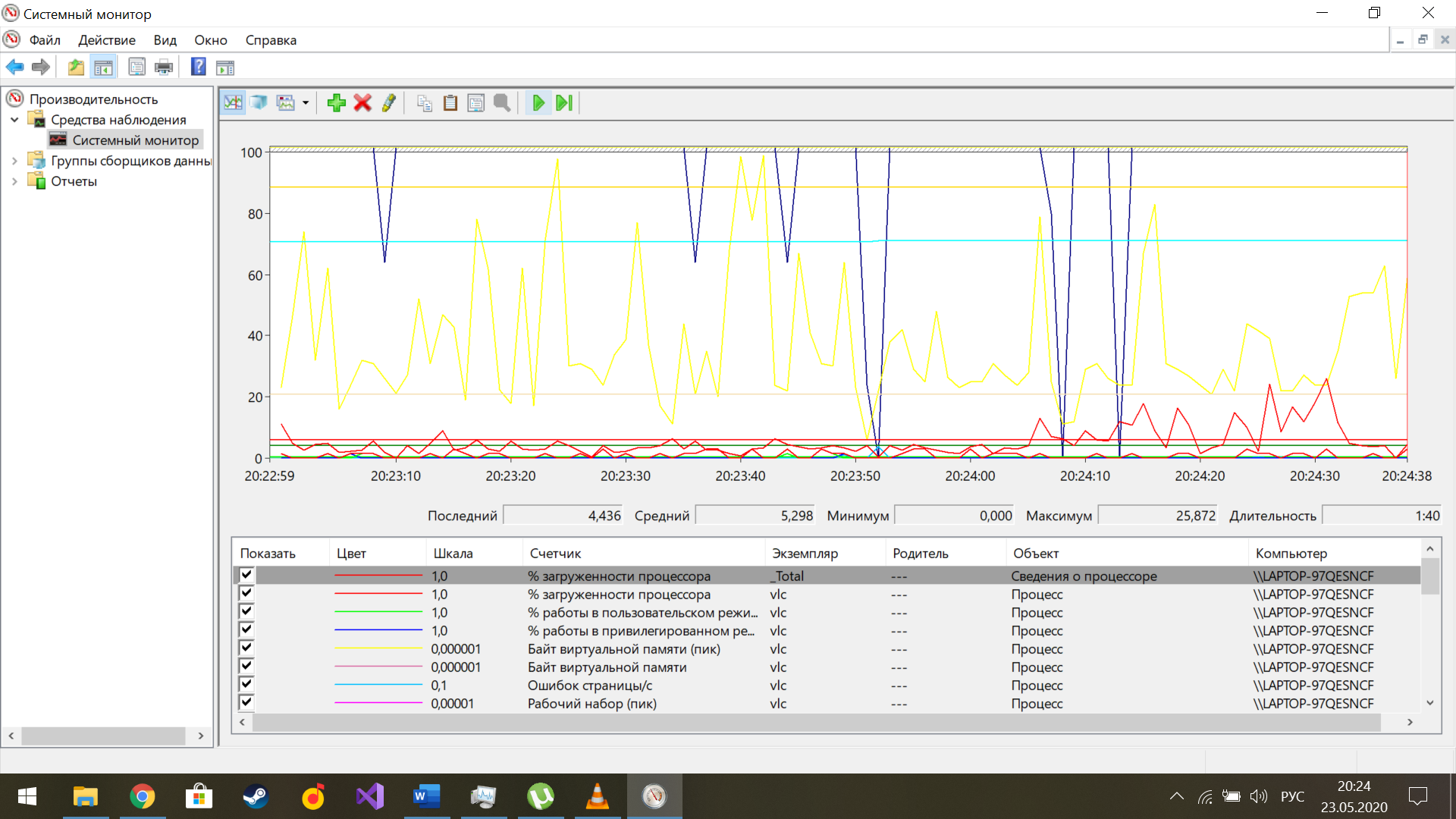


Рисунок 5 – Выполнение операции при пониженном приоритете

Изменим приоритет запущенного процесса (Рисунок 6)

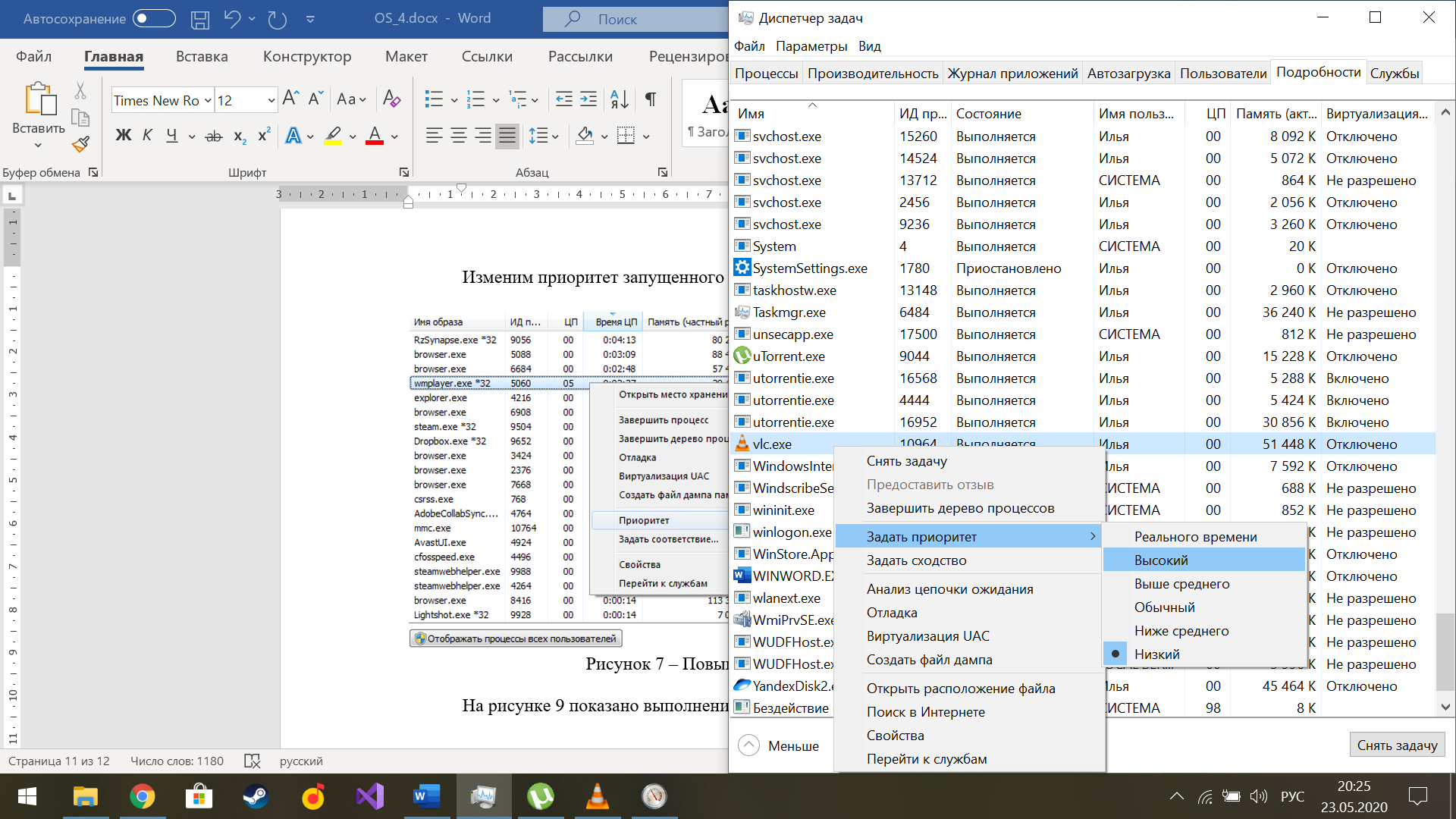


Рисунок 6 – Повышение базового приоритета

На Рисунке 7 показано выполнение той же операции с повышением приоритета.

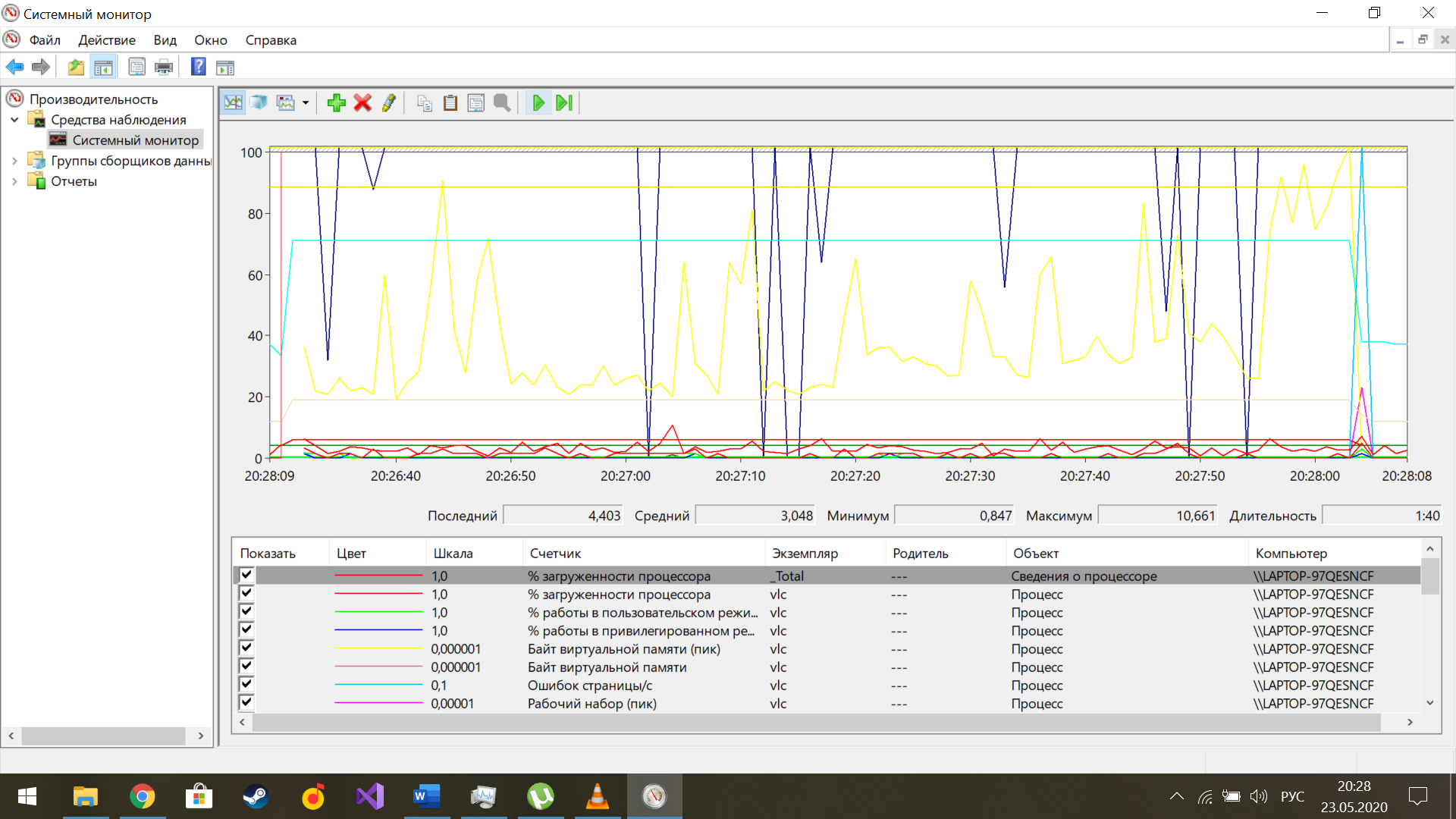


Рисунок 7 – Процесс выполняется с повышенным приоритетом

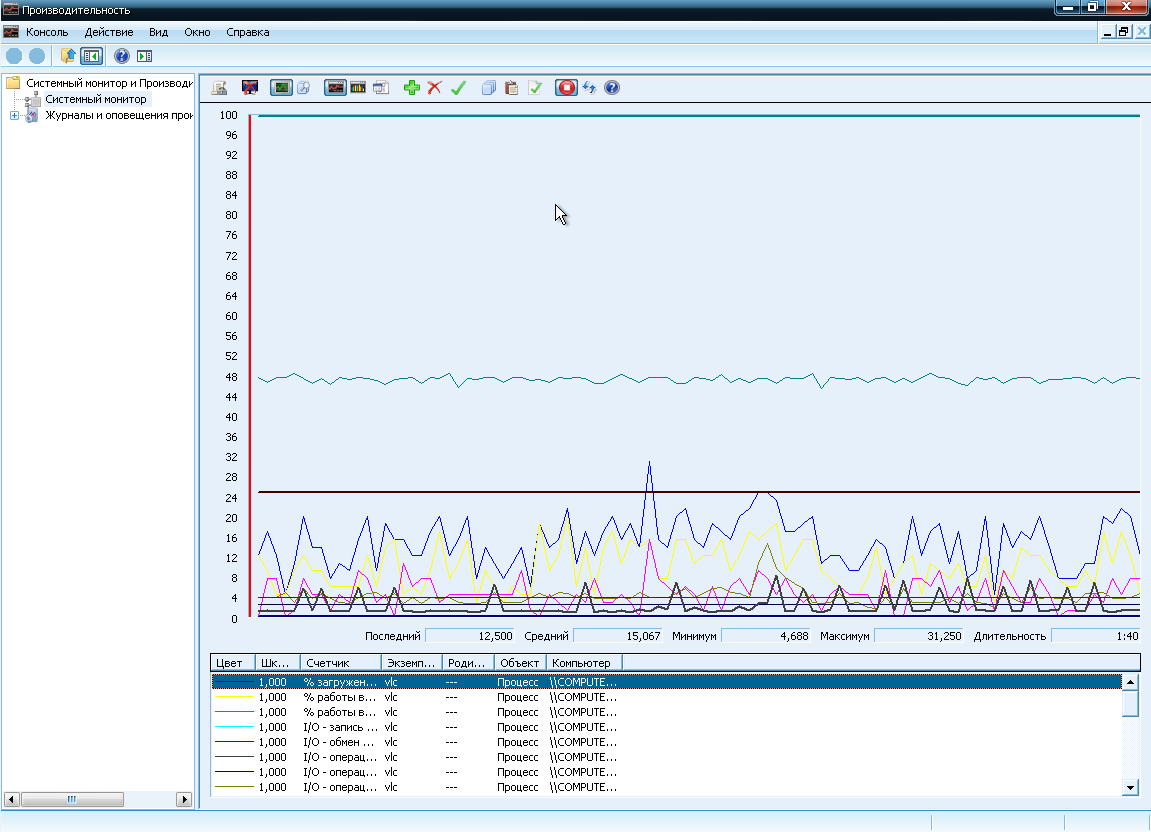
Из представленных рисунков очевидно, что при изменении приоритета процесса резко изменяется характер работы процесса в пользовательском режиме в много процессной системе - задача выполняется быстрее.

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 8.

Рисунок 8 – Выполнение операции в обычном приоритете.

Теперь понизим приоритет и взглянем на работу (Рисунок 9).

Рисунок 9 – Выполнение операции при пониженном приоритете

Теперь проследим за работой процесса при высоком приоритете (Рисунок 10).

Рисунок 10 – 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.