Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

**Кафедра «Компьютерные интеллектуальные технологии»**

**Лабораторная работа № 3**

**Исследование особенностей реализации многозадачности в ОС семейства Windows**

по дисциплине «Операционные системы»

Выполнил

студент гр. з3530903/90001 А.Ю. Леснов

<*подпись*>

Руководитель

доцент, к.т.н. А.В. Сергеев

<*подпись*>

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2021 г.

Санкт-Петербург

2021

Указания по выполнению лабораторной работы.

1. Исследование сбоев и зависаний программ при работе компьютера.
   1. Для интерпретации результатов, получающихся при выполнении лабораторной работы, необходимо знать некоторые определения. Запишите в отчёт определение следующих терминов: **контекст потока**, **переключение контекста (Context Switch)**. Укажите о чём свидетельствует увеличение числа переключений контекста, а о чём наоборот отсутствие переключений контекста. Также запишите, что значит **сбой** выполнения программы. Приведите конкретные примеры ситуаций, приводящих к сбою. Напишите, что значит **зависание** программы и когда оно возникает.

**Контекст потока:** Особую роль в структурах данных, описывающих потоки, играет контекст потока. Информацию, входящую в состав контекста, необходимо периодически сохранять и восстанавливать в случае возникновения различных событий, например, при переключении потоков. Обычно сохранению и последующему восстановлению подлежат:

* программный счетчик, регистр состояния и содержимое остальных регистров процессора;
* указатели на стек ядра и пользовательский стек;
* указатели на адресное пространство, в котором выполняется поток (каталог таблиц страниц процесса).

**Переключение контекста:** при исполнении процессором некоторого процесса возникает прерывание от устройства ввода-вывода, сигнализирующее об окончании операций на устройстве. Над выполняющимся процессом производится операция приостановки. Далее ОС разблокирует процесс, инициировавший запрос на ввод-вывод и осуществляет запуск приостановленного или нового процесса, выбранного при выполнении планирования.

Для корректного переключения процессора с одного процесса на другой необходимо сохранить контекст исполнявшегося процесса и восстановить контекст процесса, на который будет переключен процессор. Такая процедура сохранения/восстановления работоспособности процессов называется переключением контекста.

В процедуру переключения контекста входит т. н. планирование задачи –

процесс принятия решения, какой задаче передать управление.

**Увеличение числа переключений контекста** может говорить о возросшей активности программы, например пользователь обращается к ней и совершает какие-либо действия. Или сама программа по некоторому триггеру перешла в режим активного использования вычислительных ресурсов.

**Уменьшение числа переключений контекста**, напротив, говорит об уменьшении активности или обращений пользователя к программе. Так же это может быть следствием возросшей нагрузки на систему, когда переключений контекста, в целом, становится больше, но для отдельных программ это число падает.

**Сбой** обычно происходит из-за выполнения программой операции, которая запрещена ОС (недостатки кода ПО)

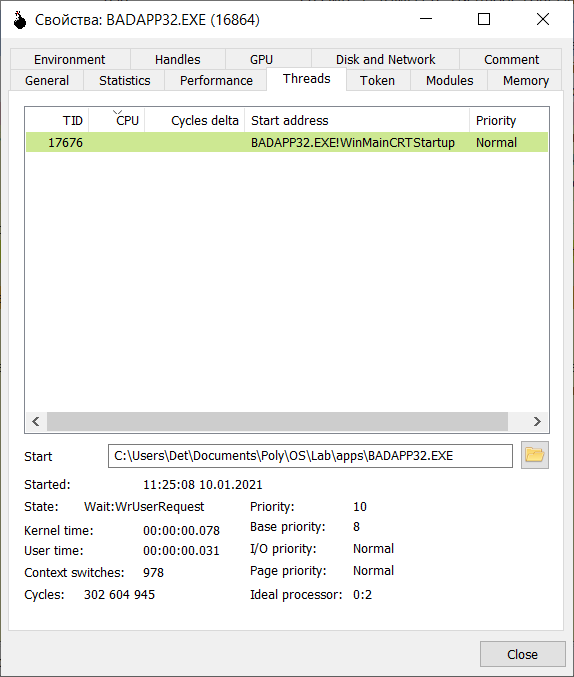
Например: попытка чтения или записи памяти, которая не предназначена для чтения или записи этой программой. Выполнение операций ввода-вывода на устройствах, к которым у неё нет доступа. Попытка получить доступ к системным ресурсам, к которым у программы нет разрешения. Выполнение машинных команды с плохими аргументами: деление на ноль, доступ к памяти в не присоединённых адресах и т. д.

Так же причиной сбоев может быть вредоносное ПО, конфликты программ, неподходящая конфигурация аппаратной части или физическое воздействие на неё (н.р. перегрев), и т. д.

**Зависание** программы (не отвечает). Основными причинами являются: чрезмерная загруженность вычислительных ресурсов системы, либо недостаточная скорость ROM и/или RAM памяти (в данном случае, чаще всего, программа продолжит работу через какое-то время). Недостатки ПО (неверные условия выхода из цикла). Взаимная блокировка процессов (“кольцевое” ожидание ресурсов, занятых друг другом).

* 1. Для наблюдения за процессами и потоками запустите программу Process Explorer (**procexp.exe**).
  2. Запустите программу **badapp32.exe**. Эта программа предназначена для моделирования выполнения программой некорректной операции (сбой) и моделирования зависания программы.
  3. Запишите приоритет процесса **badapp32** и приоритет основного потока, принадлежащего этому процессу. Для этого найдите в списке процессов процесс badapp32, щёлкните по нему правой кнопкой мыши и выберите пункт меню **Properties**. В открывшемся окне выберите закладку **Threads**.

Базовый приоритет 8 (Обычный)



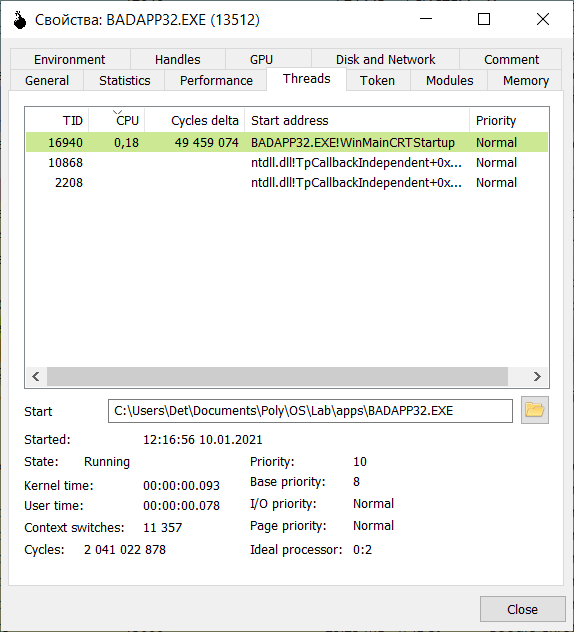
* 1. Проследите, как меняется количество изменений контекста потока (Context Switches) если программа не активна. Каков её статус? Как меняется Context Switches при наведении курсора мыши на окно программы **badapp32** и выборе какого-либо меню? В отчёте приведите объяснение наблюдаемым явлениям.

Если программа не активна:

Статус: Wait:WrUserRequest

Состояние: Выполняется

Context Switches: Очень медленный рост



При наведении курсора или выборе меню количество context switches начинает расти. Это происходит в результате пользовательского обращения к процессу (наведение курсора). Увеличивается динамический приоритет, процессор получает сигналы о наступлении событий, требующих немедленного внимания. Обработка прочих событий вытесняется, процессор начинает обрабатывать пользовательские события.

Статус периодически изменяется на Running

* 1. Промоделируйте сбой программы **badapp32**, выбрав пункт меню **Action – GPFault**. В программе Process Explorer отследите изменение состояния потока. Каков его статус? Как меняется Context Switches? Объясните произошедшие изменения.

Статус: Wait:UserRequest, n/a

Context Switches: медленный рост

Завершение потоков и процесса. Некоторое внутреннее событие, сгенерированное GPFault стало причиной сбоя и завершения программы.

Найденное по поводу статуса WrUserRequest и UserRequest:

Wait: Состояние ожидания - (Имя объекта) - Тип объекта

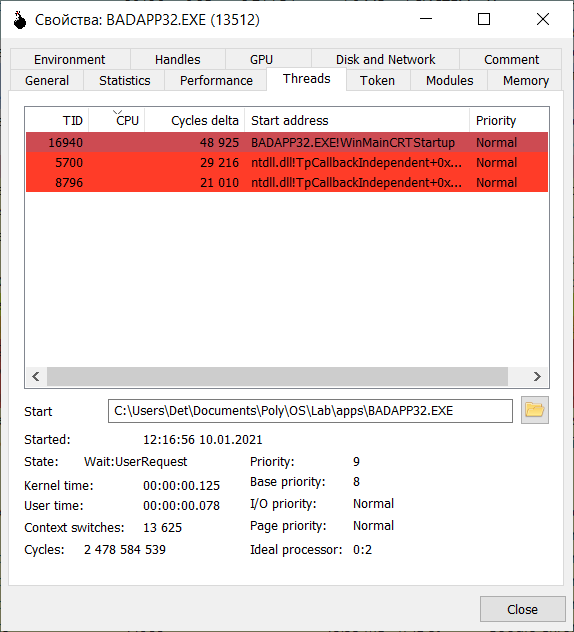
WR (WaitForWriterEvent - Событие ожидания записи): Поток ожидает снятия R-блокировок, чтобы получить W-блокировку для объекта.

(**Блокировка чтения-записи**  (r/w блокировка) — механизм разграничения блокировки на чтение и на запись некоторых данных между собой)

WrUserRequest is used by win32k.sys subsystem. Usually this is when thread calls GetMessage. So if we view WrUserRequest we can be sure that thread is waiting for window messages.

UserRequest - this means that thread waits on some object[s] via WaitForSingleObject[Ex] or WaitForMultipleObjects[Ex] or MsgWaitForMultipleObjects[Ex] (or it equivalents)

Вывод: различия между WrUserRequest и UserRequest для данного вопроса несущественны.



* 1. Объясните, почему сбой одной программы не влияет на другие программы?

В случае с **badapp32.exe** возникает ситуация, когда процессор не может выполнить некоторую инструкцию исполняющегося потока. Происходит обработка исключений на процессоре (сбой, завершение работы процесса) и процессор переключается на выполнение других задач.

В некоторых случаях сбой отдельной программы может приводить к зависанию или сбою ОС. Например, если программа является важной частью ОС

* 1. Закройте программу **badapp32.exe** и запустите её вновь. Промоделируйте зависание программы, выбрав пункт меню **Action – Hang**. Запишите какой приоритет стал у процесса и потока. Запишите также, какой стал статус у потока и как меняется Context Switches? Объясните получившиеся результаты.

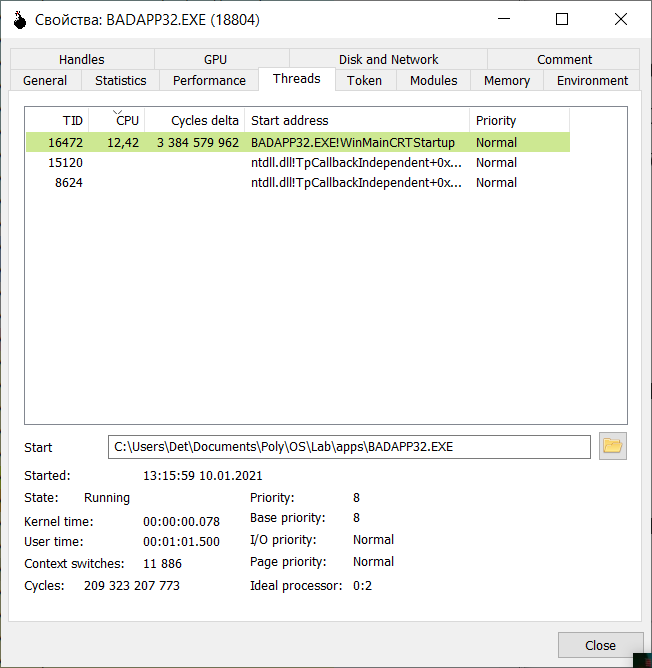
Статус: Running

Состояние: Не отвечает

Context Switches: быстрый рост

Base & Dynamic priority: 8

Зависание программы, предположительно: повторяющийся цикл выполнения (огромный рост Cycles). ОС снизила динамический приоритет программы до установленного минимума в результате адаптивного планирования (приоритет потока, исчерпавшего свой квант времени, падает)



* 1. Объясните, почему зависание одной программы не влияет на другие программы?

Это снова следствие адаптивного планирования. Зависшему процессу постоянно выделяются время процессора и ресурсы, однако ОС не позволяет ему занять всё возможное время, постоянно вытесняя и выполняя задачи других программ. Context switches отражает количество переключений между зависшим процессом и остальными задачами.

“Квант времени считает таймер реального времени. Он расположен на системной плате. По истечении кванта времени, выделенного на выполнение потока, таймер создаёт сигнал прерывания. Этот сигнал поступает на специальный вход процессора. Возникает аппаратное прерывание, процессор останавливает выполнение потока.”

К тому же процесс завис лишь на одном из ядер, вырабатывая его ресурсы. Остальные ядра процессора выполняют прочие задачи в штатном режиме.

1. Исследование свойств кооперативной (невытесняющей) и вытесняющей многозадачности Win32.
   1. Попробуйте запустить программу **badapp16.exe** в 64-х разрядной операционной системе.Почему не удалось её запустить?

64-разрядная версия Windows не поддерживает запуск 16-разрядных приложений, основная причина в том, что дескрипторы имеют 32 значащих бита в 64-битной Windows.

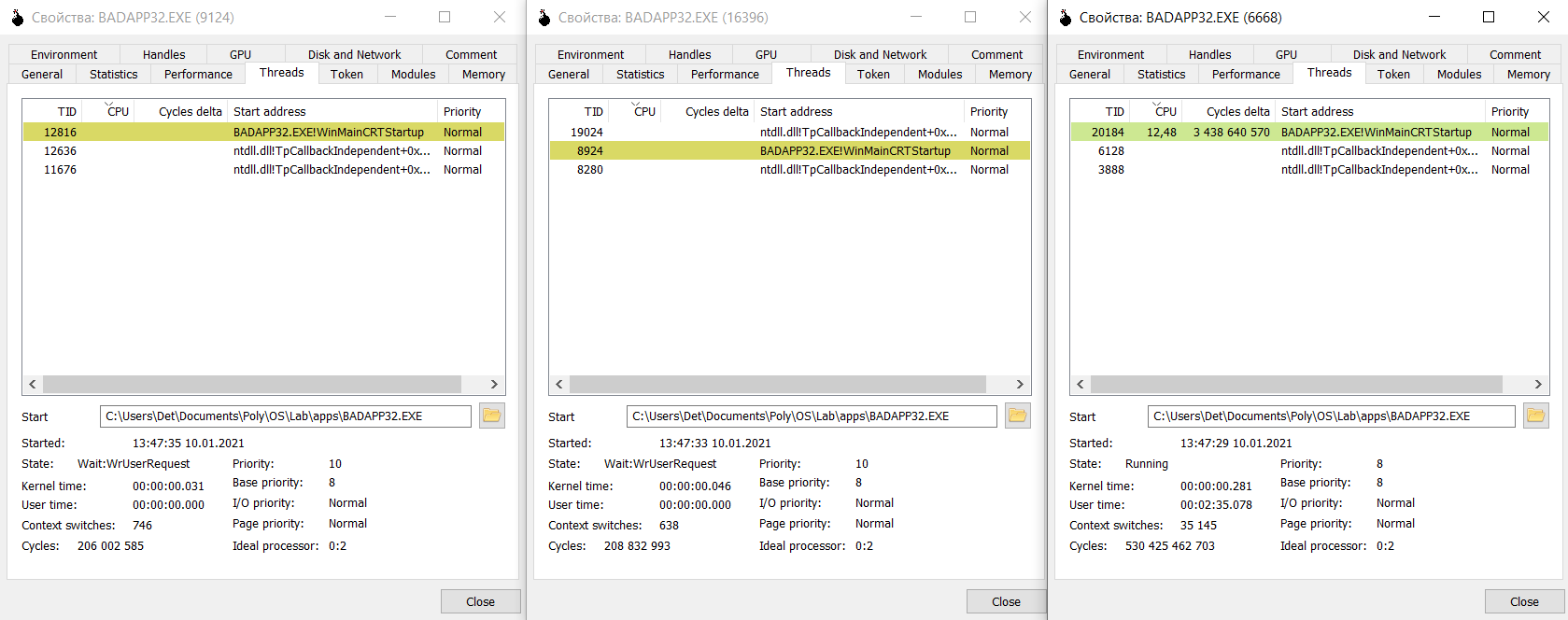
Поэтому дескрипторы не могут быть усечены и переданы 16-разрядным приложениям без потери данных.

* 1. Запустите 3 экземпляра приложения **badapp16** в 32-х разрядной операционной системе и произведите операцию искусственного зависания (Hang) одного из них. Зафиксируйте поведение остальных экземпляров программы. Сделайте вывод и запишите его в отчёт.

К сожалению, не имею возможности выполнить пункт 2.2

* 1. Запустите 3 экземпляра приложения **badapp32** и произведите операцию искусственного зависания (Hang) одного из них. Зафиксируйте поведение остальных экземпляров программы. Объясните, получившиеся различия в поведении 16-разрядных и 32разрядных приложений.

Зависание одной программы в 64-разрядной ОС не влияет на поведение остальных экземпляров



1. Исследование поведения процесса в режиме реального времени.
   1. Переведите программу **badapp32** в режим зависания (Hang).
   2. Используя **Диспетчер задач**, задайте процессу badapp32.exe приоритет реального времени.
   3. Используя программу Process Explorer, убедитесь, что приоритет потока управления равен 24.
   4. Запишите, какой статус у потока? Как меняется Context Switches? На каком ядре запущен поток, и какой процент процессорного времени ядра занимает поток? Объясните получившиеся результаты.

Статус: Running

Состояние: Не отвечает

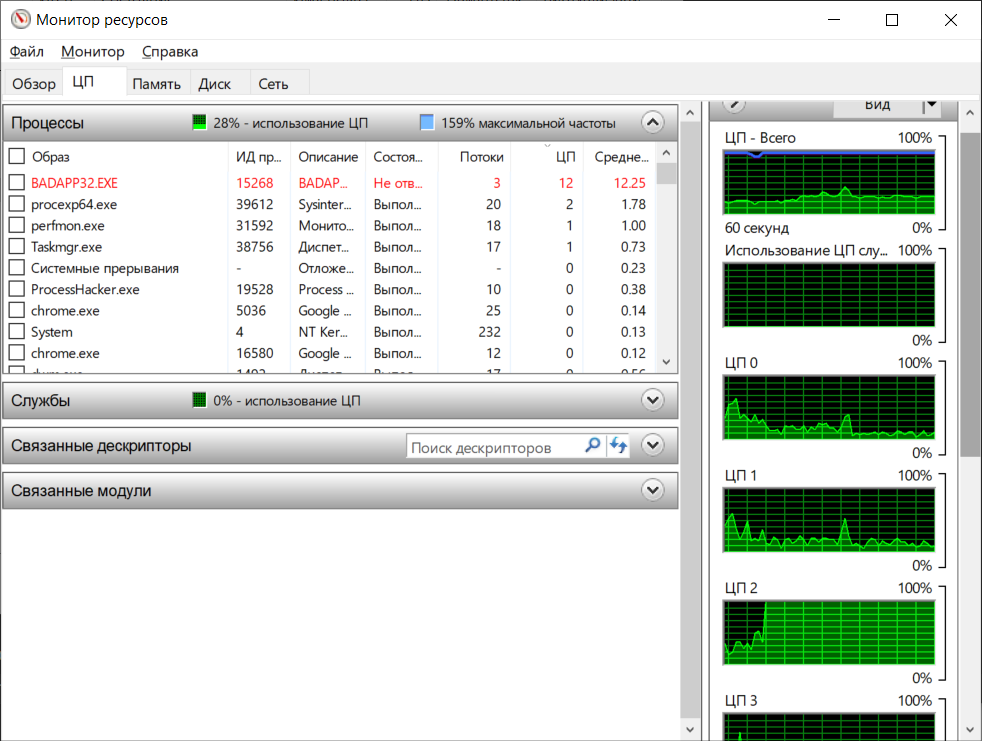
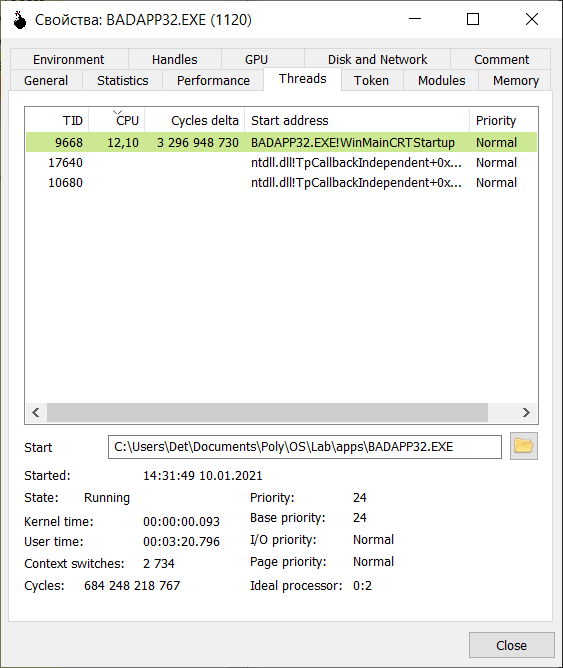
Context Switches: незначительный рост

Ядро: Ideal processor 2 (3-е ядро, загрузка 100%)

Незначительный рост Context Switches связан с приоритетом реального времени процесса (над процессом не производится операция приостановки)

“Задача планировщика систем реального времени — обеспечить приоритетное выполнение отдельных критических процессов, причем не важно, насколько жесткими накладными расходами для всей остальной системы в целом это обойдется.”

Во время выполнения этого задания у меня умер тачпад. Интересно, что не меняя приоритет **badapp32** на «Реального времени»**,** тачпад остаётся жив.



1. Определение влияния приоритета процесса на выделяемое этому процессу процессорное время. Для дальнейшей работы надо использовать программу counter. Для того, чтобы программа заработала, надо зарегистрировать в операционной системе библиотеку VB40032.DLL. Для этого надо обладать правами администратора. Если у Вас нет таких прав, то вместо counter можно использовать программу badapp32. В этом случае надо перейти к п. 5. В Windows 7 при запуске программы counter пользователем с правами администратора библиотека зарегистрируется автоматически. В Windows 10 для регистрации библиотеки надо нажать правой кнопкой мыши на файл counter.exe и в контекстном меню выбрать пункт «Запуск от имени администратора». После этого библиотека зарегистрируется, и программа counter запустится. Если программа counter запустилась, то п. 5 выполнять не надо. Запустите два экземпляра приложения **counter.** Определите при помощи утилиты «Диспетчер задач» доли процессорного времени, выделяемого каждому из полученных процессов. Процент процессорного времени, отведённому каждому процессу, представлен в колонке ЦП «Диспетчера задач».

Каждый процесс занимает до 10-12.5% процессорного времени

* 1. Измените приоритет одного процесса **counter** на **Высокий**, а другого на **Низкий** и определите доли процессорного времени, выделяемого каждому из полученных процессов. Объясните получившийся результат.

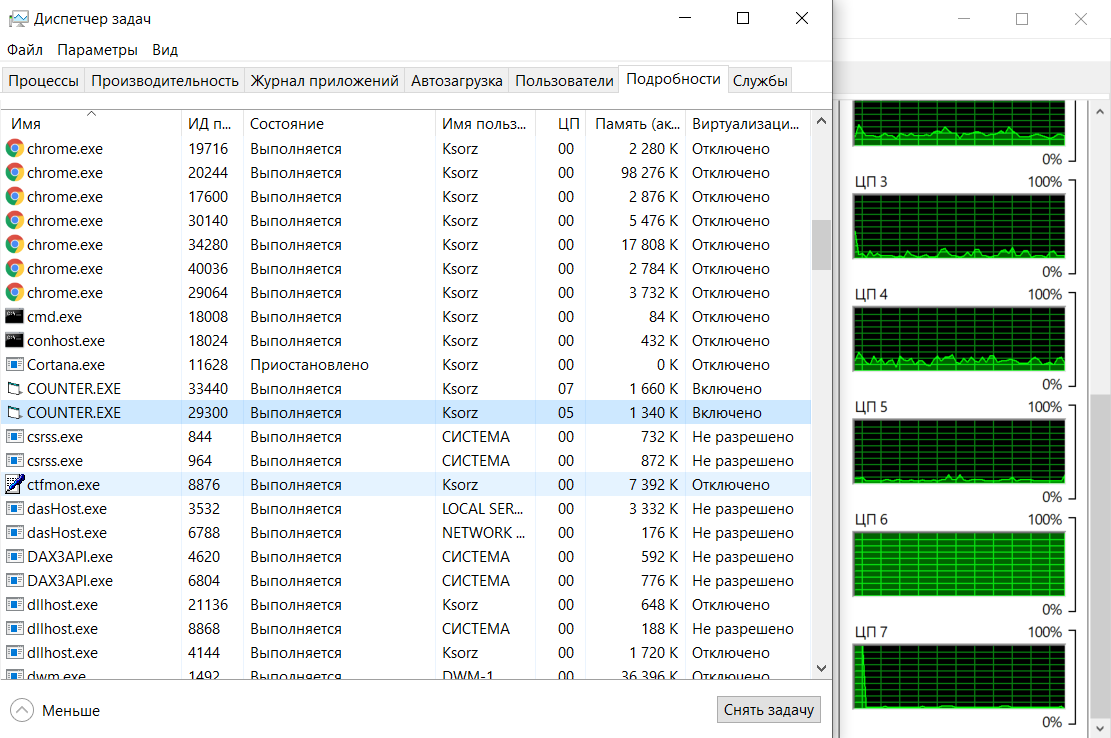
Процессы по-прежнему занимают до 10-12.5% процессорного времени. Это связано с тем, что процессы выполняются на разных ядрах.

* 1. Запустите оба процесса **counter** на одном ядре процессора. Для этого в «Диспетчере задач» надо нажать правой кнопкой мыши на процесс **counter** и выбрать из контекстного меню «**Задать соответствие»** (**Задать сходство**). Как изменился процент процессорного времени, отводимый каждому процессу counter? Почему?

Процессорное время приоритета **Высокий**: 6-7%

Процессорное время приоритета **Низкий**: 4-5%

**counter** пытается занять 100% вычислительного времени одного из ядер процессора, этим объясняется максимальное значение 12.5% из предыдущих вопросов (на 8-ядерном процессоре). Запуская 2 таких процесса на одном и том же ядре, мы получаем «борьбу за ресурсы» между этими процессами, большую часть которых ОС отдаёт процессу с наивысшим приоритетом.



* 1. Запустите третий экземпляр программы **counter** на том же процессорном ядре, что и остальные процессы **counter**. Как изменилась доля процессорного времени, отводимого каждому процессу counter? Почему?

Процессорное время приоритета **Высокий**: 6-7%

Процессорное время приоритета **Обычный**: 4-5%

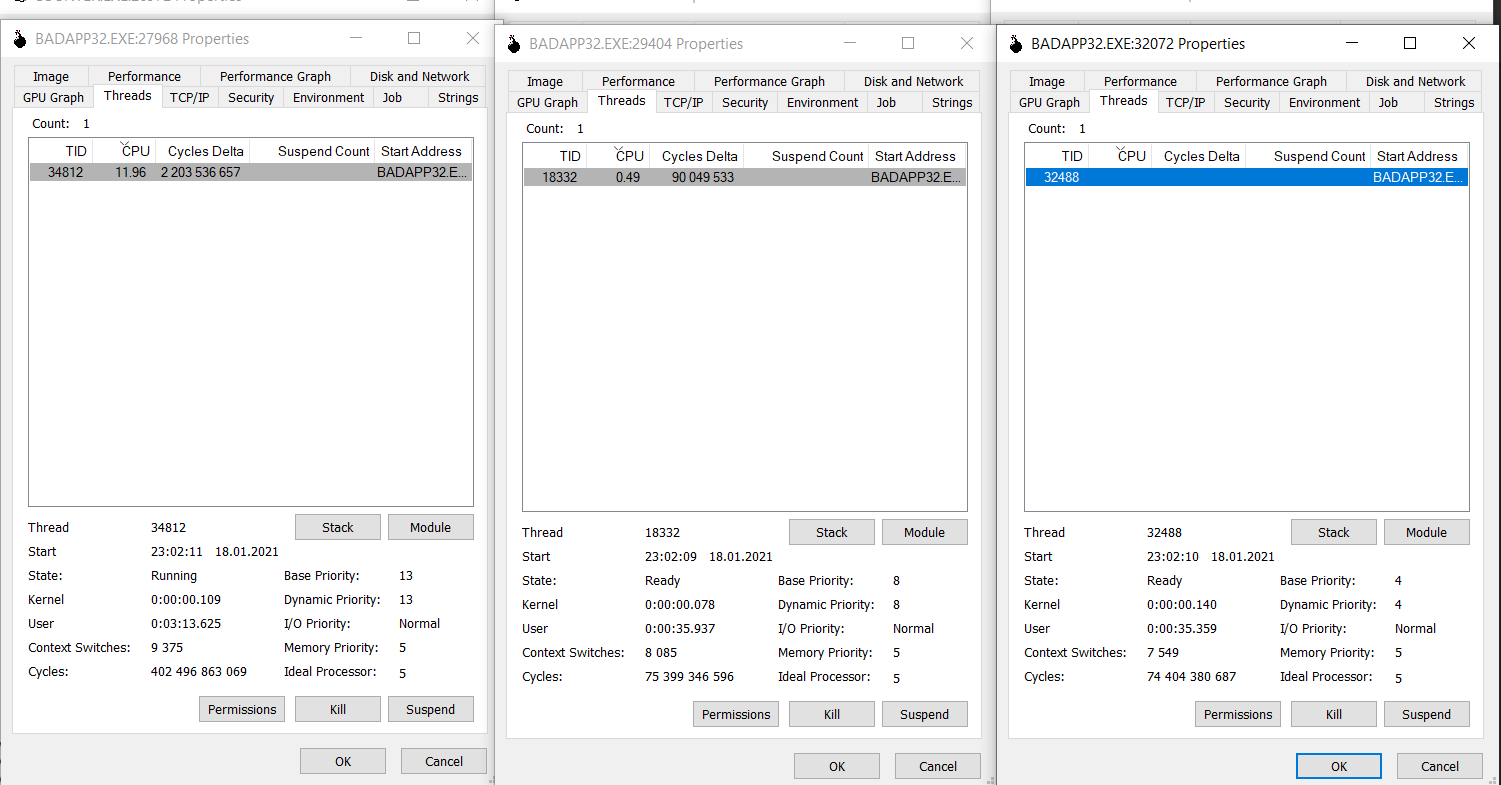
Процессорное время приоритета **Низкий**: 0-1%

Следуя логике работы из примера 4.3, я ожидал увидеть плавное снижение потребляемых ресурсов от высокого приоритета к низкому. Однако в ситуации с 3 и более процессами, только два наивысших приоритета потребляют все ресурсы (плавное разделение происходит только если задать всем процессам одинаковый приоритет).

Очевидно, данная картина связана с адаптивным планированием и «голодающими» потоками. Обнаруживая не выполняемый на протяжении нескольких секунд поток ОС поднимает его приоритет до динамического максимума (15), после короткой сессии использования ресурсов приоритет потока снова падает до базового значения и цикл повторяется. Именно этим объясняется крайне скудный доступ ко времени ядра низкоприоритетным потоком.

Однако я так и не смог понять почему поток с приоритетом «Обычный» получает значительную часть ресурсов ядра несмотря на пониженный приоритет. Следуя логике, он так же должен быть «голодающим» и получать доступ к ресурсам лишь изредка.

Далее я сделал то же самое с 3 процессами **badapp.exe** (Hang), тут результат оказался предсказуемым:



Лишь один процесс badapp с наивысшим приоритетом потреблял практически все ресурсы.

Возможно, в случае с **counter.exe** реализуется некоторая логика программных прерываний и изменений внутренних приоритетов, в то время как с зависшими экземплярами **badapp.exe** всё было отдано логике ОС.

*Поправьте, пожалуйста в комментарии, интересно знать верный ответ*

* 1. Установите у нового процесса приоритет **Высокий.** Как изменилась доля процессорного времени, отводимого каждому процессу **counter**? Почему?

Процессорное время приоритета **Высокий**: 5-6%

Процессорное время приоритета **Высокий**: 5-6%

Процессорное время приоритета **Низкий**: 0-1%

У двух процессов с приоритетами **Высокий** доля процессорного времени выровнялась. В данном случае вполне логично, что ОС разделяет поровну процессорное время между двумя идентичными процессами с наивысшим динамическим приоритетом

* 1. Оцените качественно возможность реакции процессов **counter** на интерактивное событие – перетаскивание окна. Есть ли разница при перетаскивании разных окон **counter**? Объясните результат. Объясните, почему при перетаскивании окна **counter** останавливается счётчик.

Статус экземпляров **counter.exe**, если их не трогать изменяется в диапазоне трех стадий:

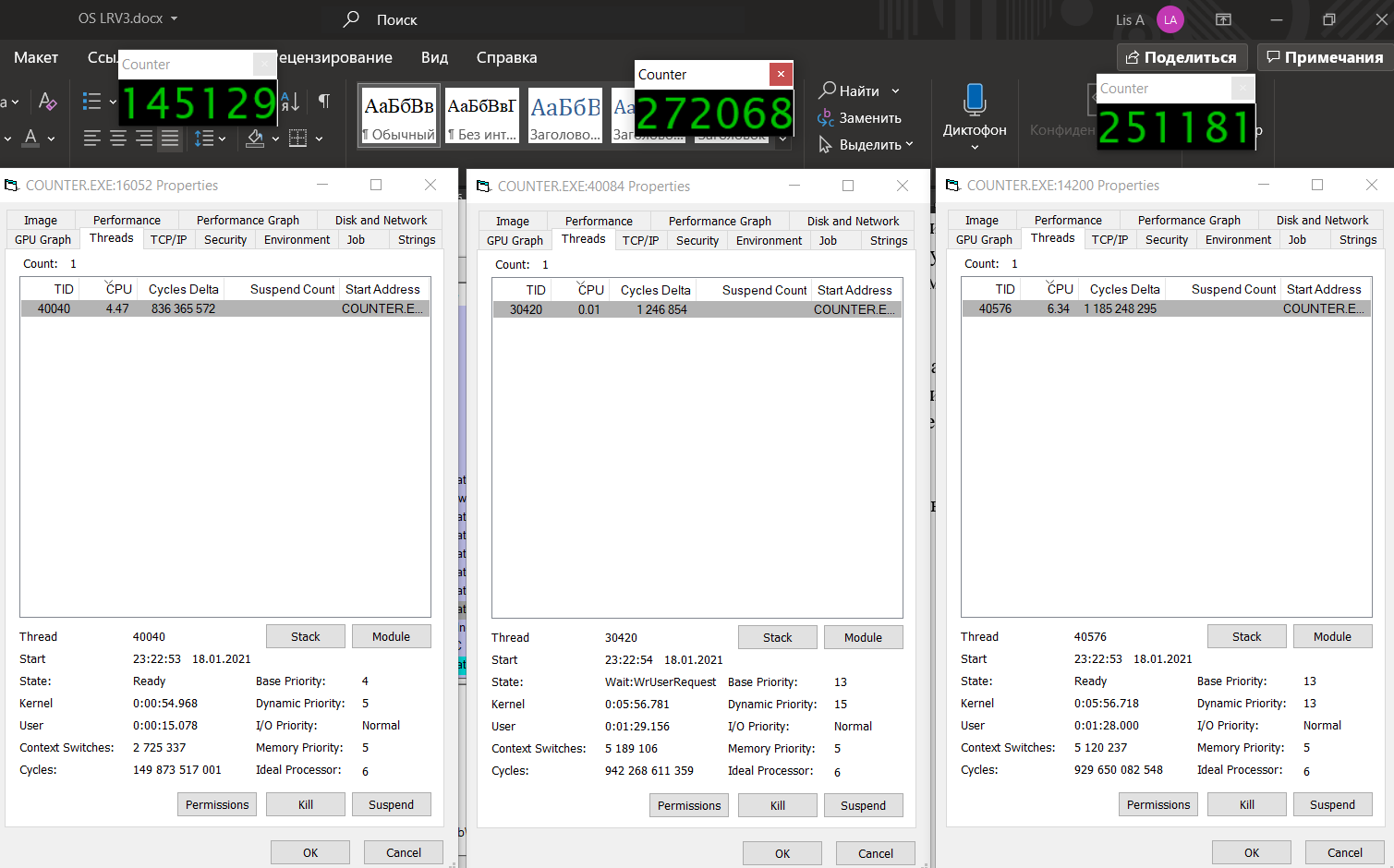
**Ready, Wait:WrResource, Running.**

Что является циклом, в котором находится программа.

Статус процесса, окно которого мы перетаскиваем:

**Wait:WrUserRequest**

Программа ожидает действия пользователя, приоритет повышается, процессорное время практически не используется.



Можно сделать вывод, что действия пользователя (н.р. перетаскивание окна) являются одним из триггеров выхода программы из цикла, следовательно счётчик отключен.

Разницы при перетаскивании различных окон нет, независимо от приоритета интерактивный процесс прекращает свой основной цикл, остальные 2 процесса начинают активно делить оставшийся ресурс ядра (поведение как в пункте 4.3). Динамический приоритет интерактивного процесса увеличивается.

Активное перетаскивание окна одного из процессов повышает его процессорное время вплоть до 2% (перемещая его активнее можно ещё увеличить этот процент).

Однако здесь увеличение процессорного времени не связано с активностью самой программы, а лишь с обработкой интерактивных процессов пользователя, который таскает несчастное окно по монитору чтобы сделать лабу)