Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Операционные системы

Студент: Немкович А.В.

ФИТ 3 курс 1 группа

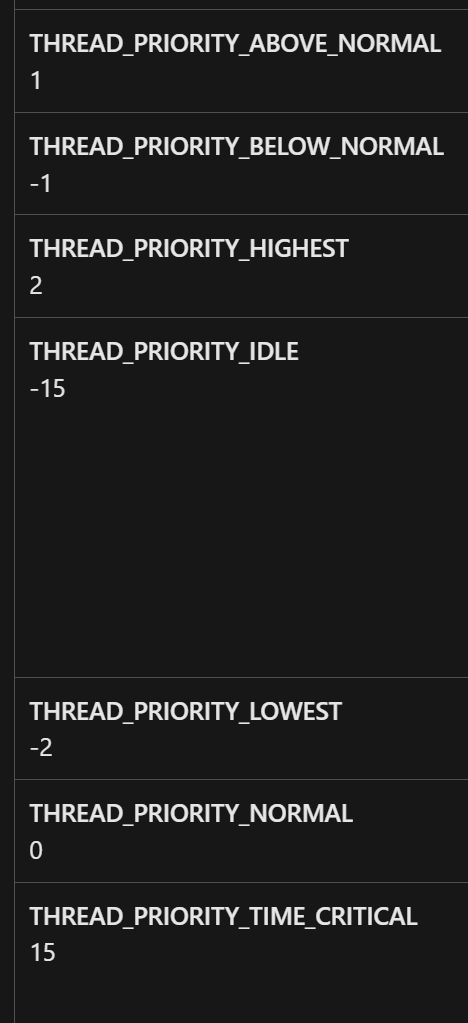
Преподаватель: Бернацкий П.В

Минск 2024

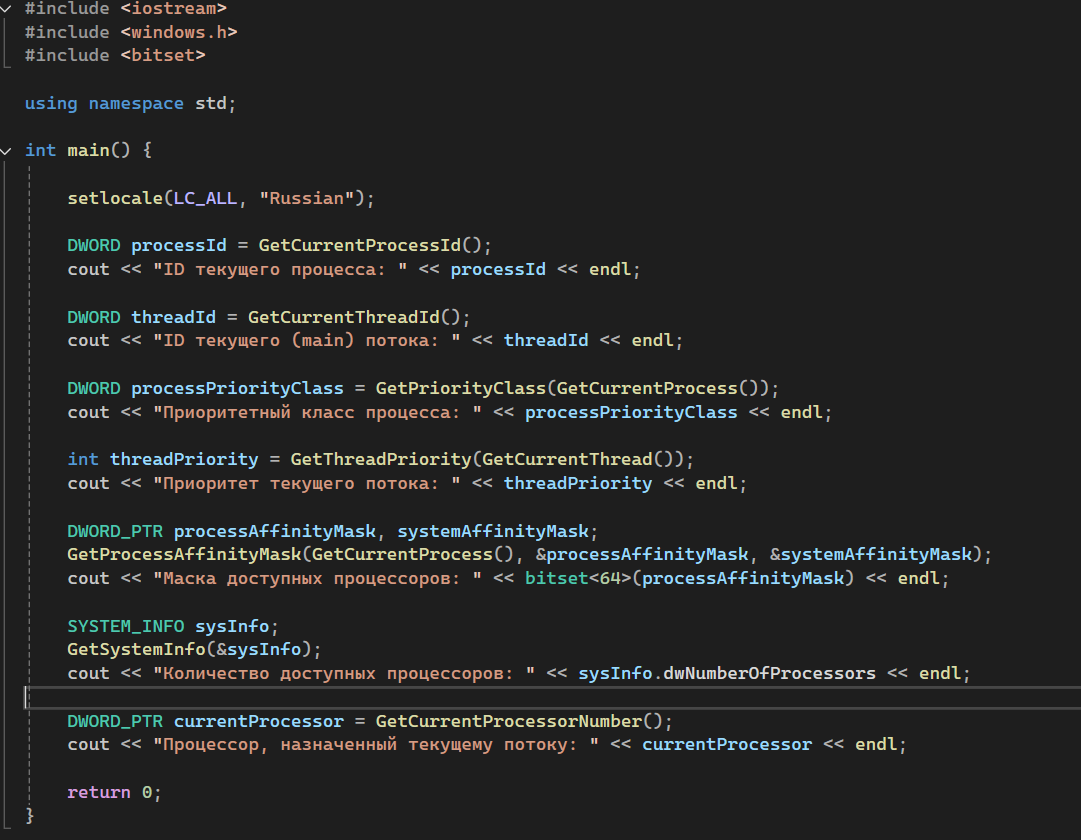
**Задание 01**

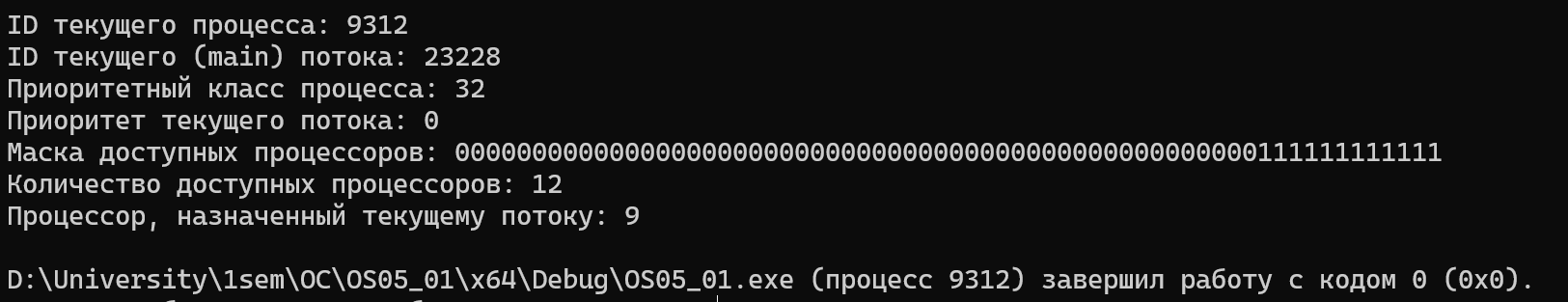
Разработайте консольное Windows-приложение OS05\_01 наязыке С++, выводящее на консоль следующую информации:

* идентификатор текущего процесса – GetCurrentProcessId()
* идентификатор текущего (main) потока – GetCurrentThreadId()
* приоритет (приоритетный класс) текущего процесса - GetPriorityClass(GetCurrentProcess())
* IDLE\_PRIORITY\_CLASS = 0x00000040 (64)
* BELOW\_NORMAL\_PRIORITY\_CLASS = 0x00004000 (16384)
* NORMAL\_PRIORITY\_CLASS = 0x00000020 (32)
* ABOVE\_NORMAL\_PRIORITY\_CLASS = 0x00008000 (32768)
* HIGH\_PRIORITY\_CLASS = 0x00000080 (128)
* REALTIME\_PRIORITY\_CLASS = 0x00000100 (256)
* приоритет текущего потока; GetThreadPriority(GetCurrentThread())

****

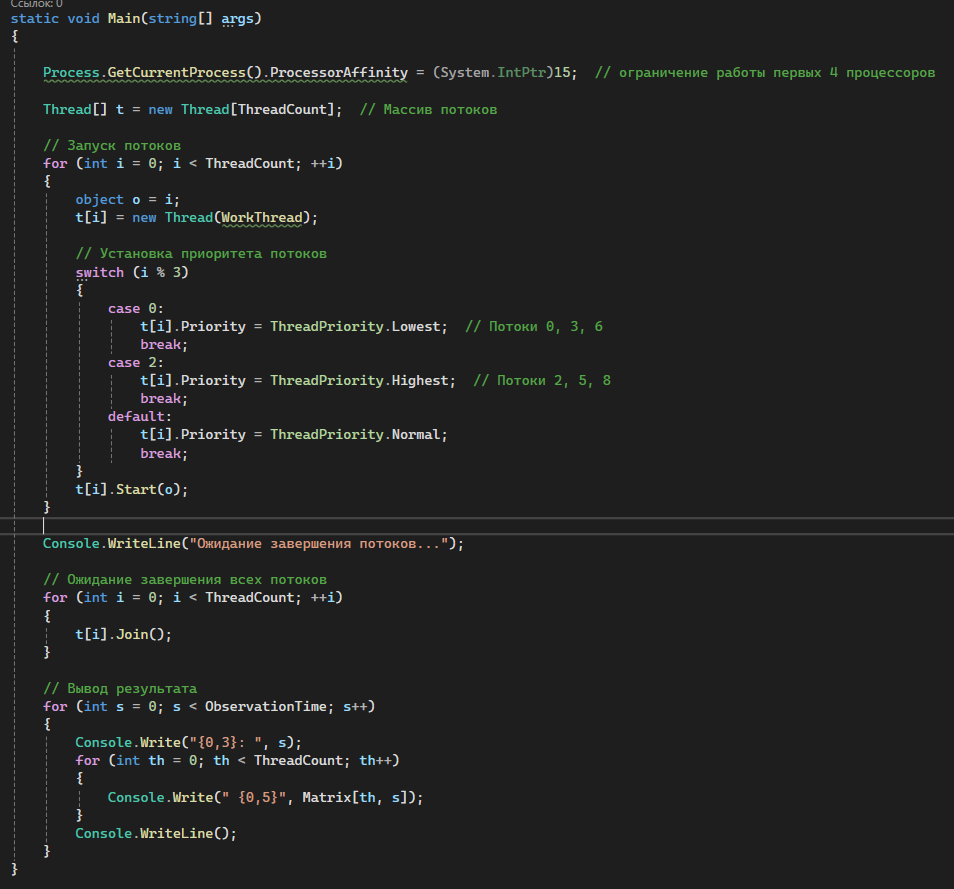
* маску (affinity mask) доступных процессу процессоров в двоичном виде. GetProcessAffinityMask — это функция Windows API, которая определяет, к каким процессорам может быть прикреплен процесс. Она возвращает две маски: processAffinityMask — маска доступных процессоров для данного процесса. systemAffinityMask — маска доступных процессоров для всей операционной системы на этом компьютере.
* количество процессоров, доступных процессу - sysInfo.dwNumberOfProcessors
* процессор, назначенный текущему потоку. GetCurrentProcessorNumber

****

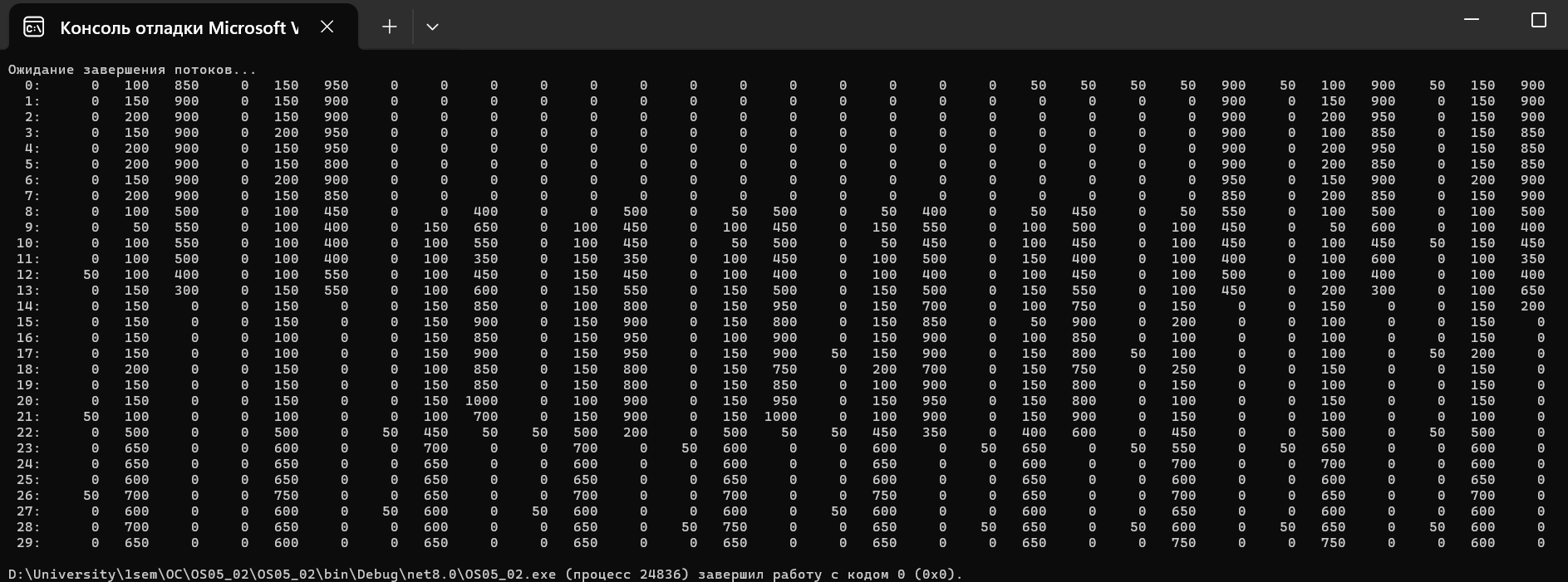
****

**Задание 02**

Создайте консольное Windows OS05\_02 на языке С#, взяв за основу приложение OS04\_07 из Лабораторной работы №4. Измените метод Main таким образом, чтобы потоки 0, 3, 6 и т.д. запускались с минимальным приоритетом потока, а потоки 2, 5, 8... – с максимальным. Класс приоритета процесса оставьте по умолчанию (Normal).



Выполните приложение, не забудьте про персонификацию вывода результатов. По зафиксированным скриншотам объясните полученные результаты.

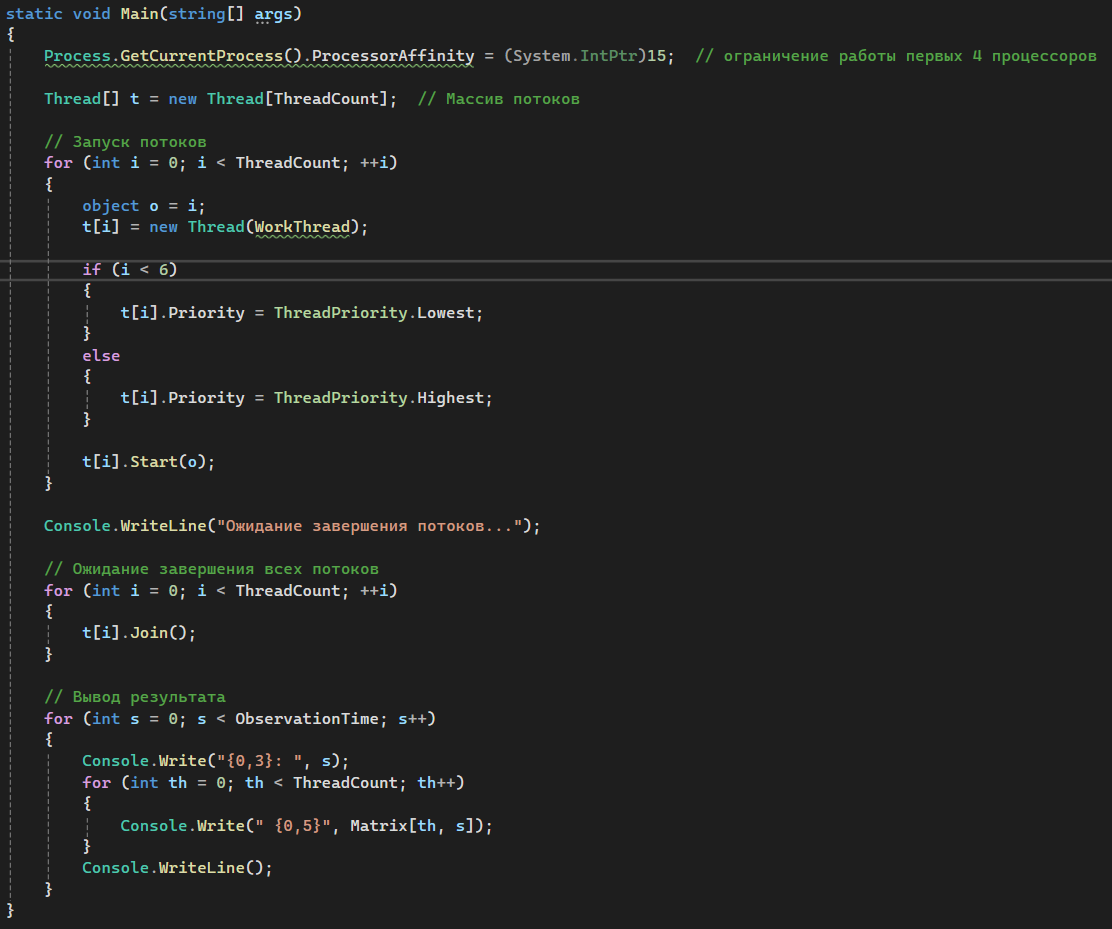


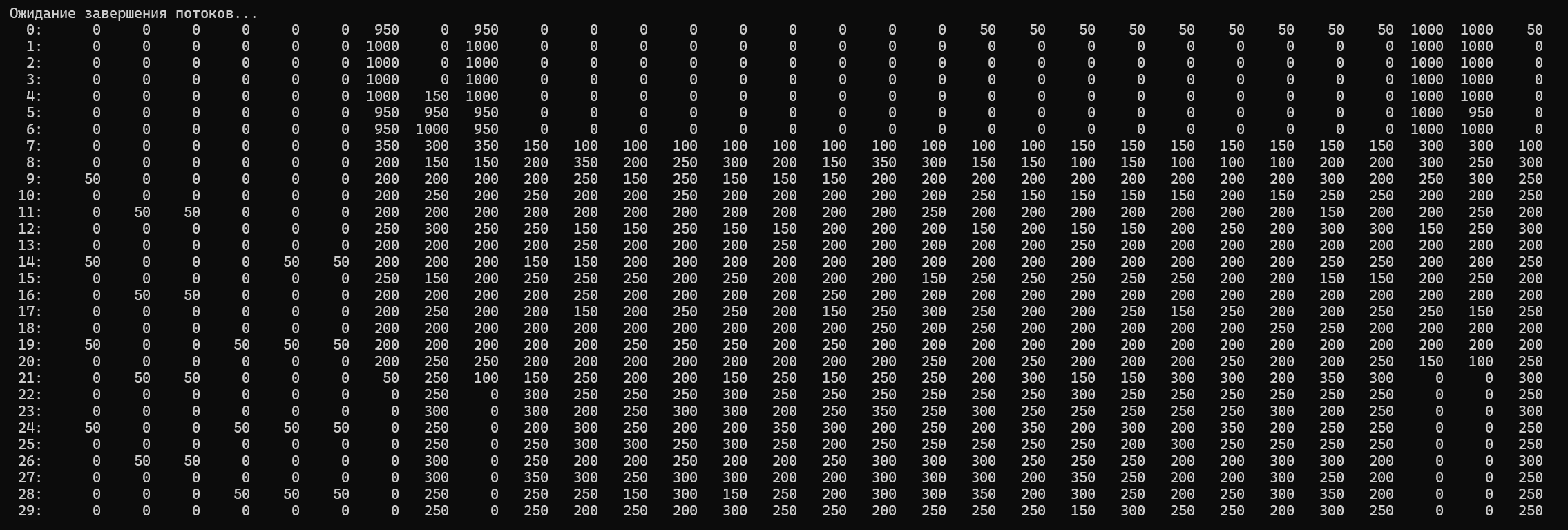
ThreadPriority.Lowest - Низкоприоритетные потоки имеют значительно меньшие значения активности. Они получают меньше процессорного времени, и их результат существенно ниже.

ThreadPriority.Highest - Высокоприоритетные потоки выполняются намного чаще, их значения в матрице возрастают быстрее и достигают более высоких значений в сравнении с потоками с низким приоритетом.

**Задание 03**

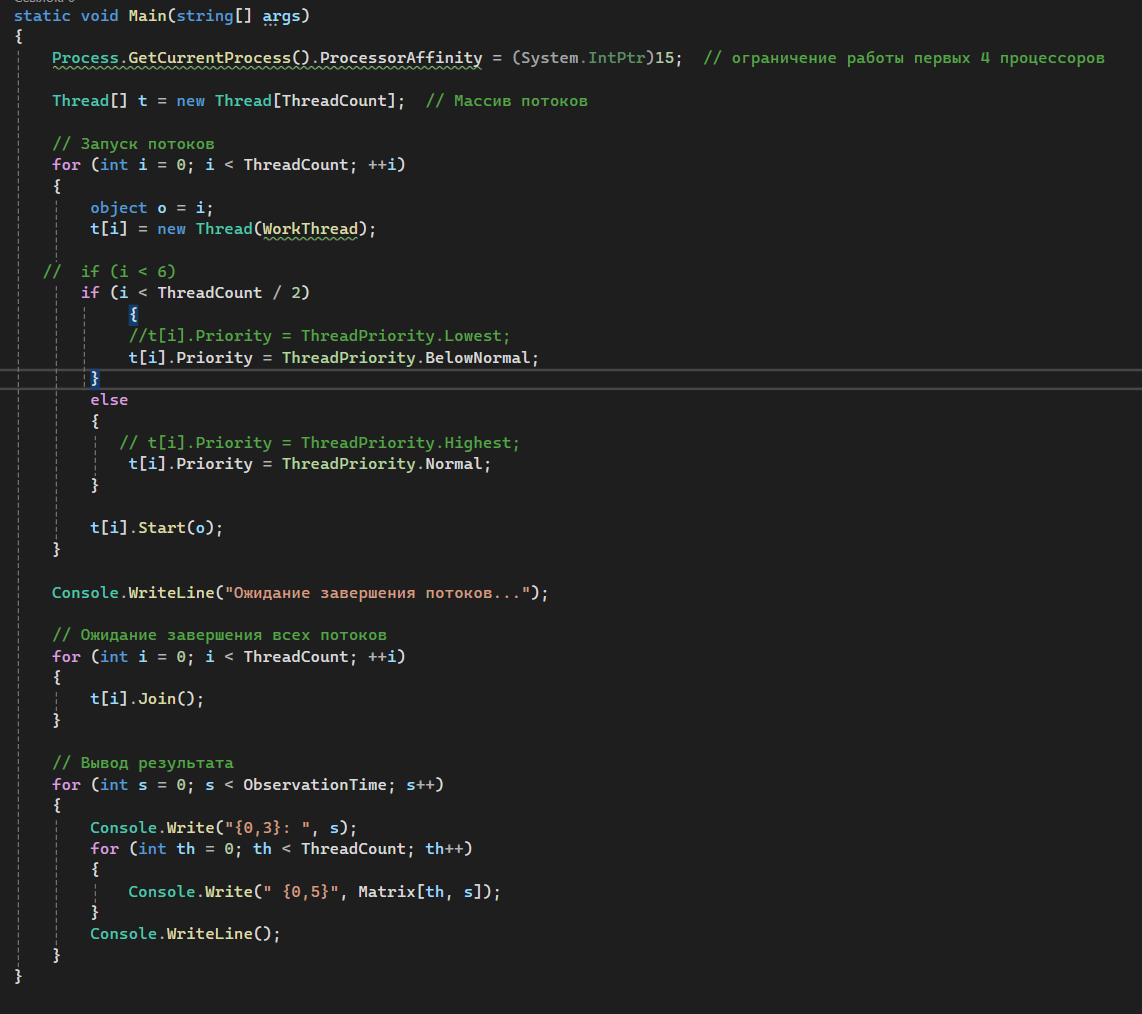
Создайте консольное Windows OS05\_03 на языке С#, взяв за основу приложение OS05\_02 из настоящей работы. На этот раз только несколько потоков запустите на наименьшем приоритете потока, а остальные – на наибольшем.

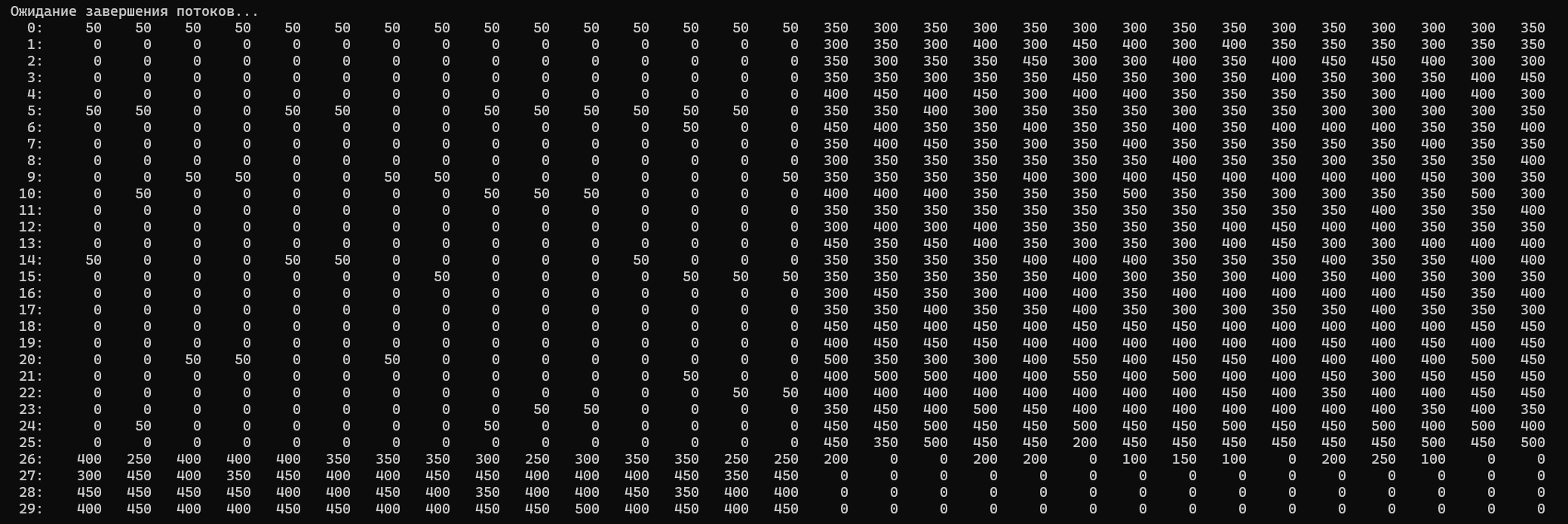


Выполните приложение OS05\_03. Удалось ли поработать низкоприоритетным потокам? 

Низкоприоритетные потоки практически не получили времени на выполнение, особенно на первых этапах работы программы.

Выполните приложение OS05\_03 с другими парами приоритетов, например, BelowNormal и Normal. Изменился ли характер работы потоков?





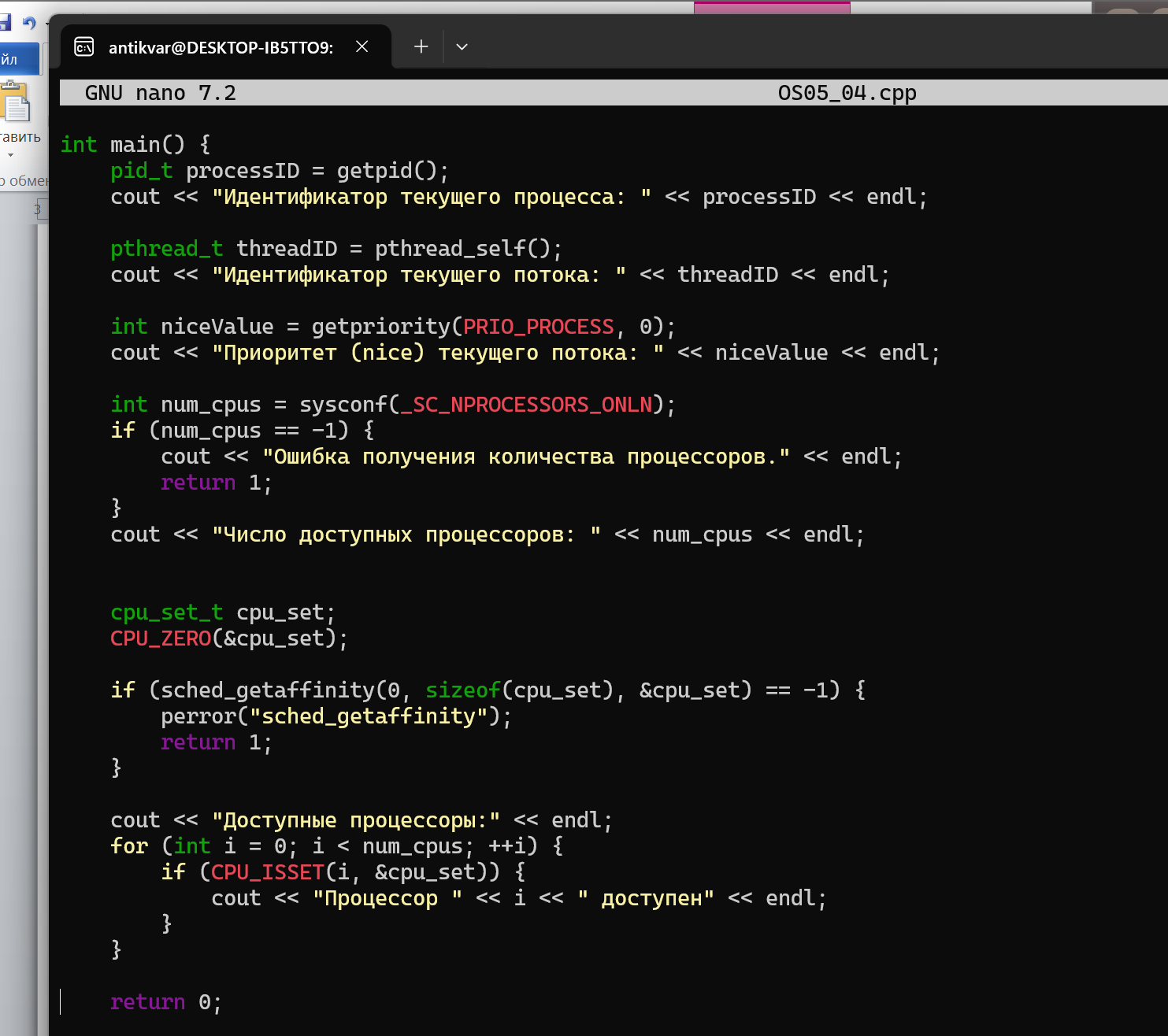
По зафиксированным скриншотам объясните полученные результаты. При этом укажите числовые значения приоритетов потоков.

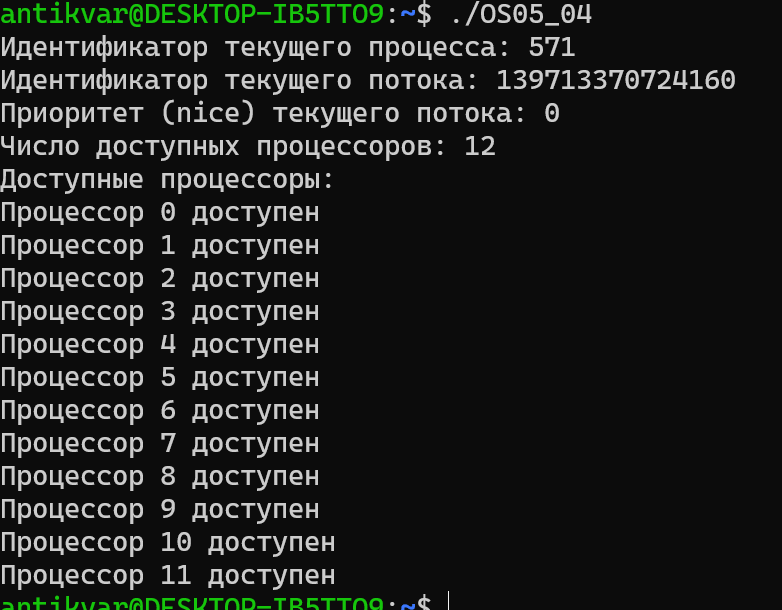
Потоки с низким приоритетом могут активно работать только тогда, когда ресурсы свободны, но если есть потоки с более высоким приоритетом, они захватывают процессорное время и вытесняют менее приоритетные потоки.

**Задание 04**

Разработайте консольное Linux-приложение OS05\_04 на языке С++, выводящее на консоль следующую информации:

* идентификатор текущего процесса;
* идентификатор текущего (main) потока;
* приоритет (nice) текущего потока;
* номера доступных процессоров.



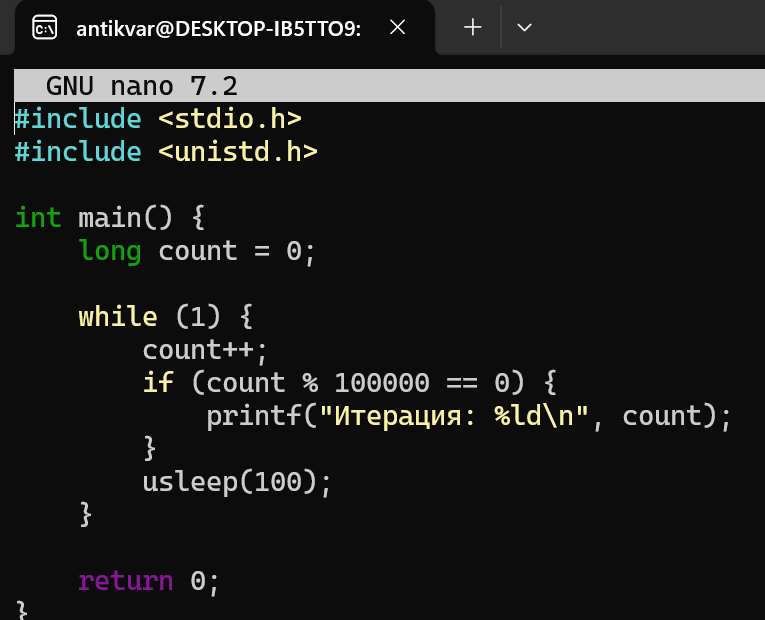


Используется функция pthread\_self(), которая определена в библиотеке Pthreads, поэтому флаг -lpthread необходим при компиляции.

Число доступных процессоров — это количество логических процессоров (или потоков), доступных в системе.

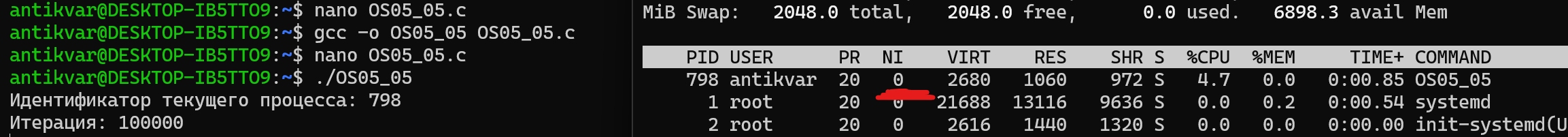
**Задание 05**

Разработайте консольное Linux-приложение OS05\_05 на языке С, выполняющее длинный цикл.

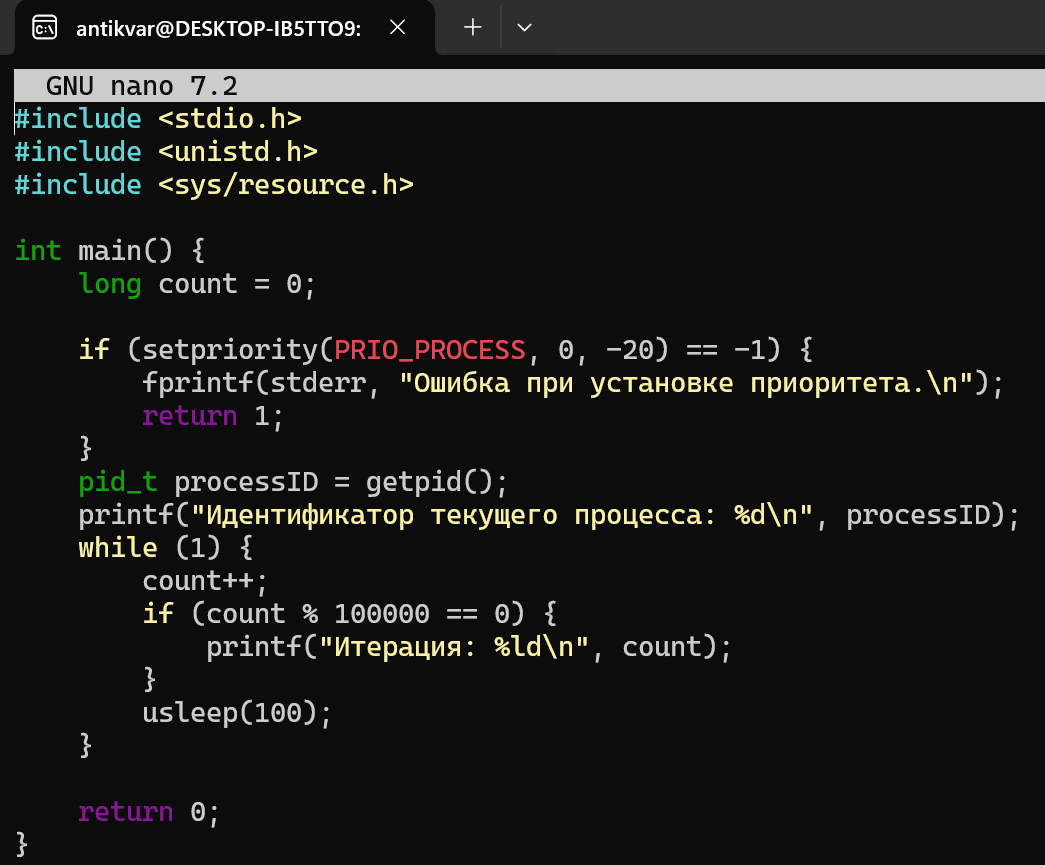


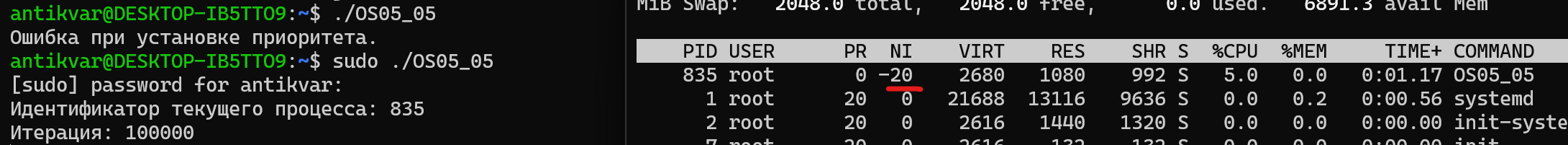
Запустите приложение OS05\_05.

Зафиксируйте текущее значение nicе, полученное с помощью команды top.

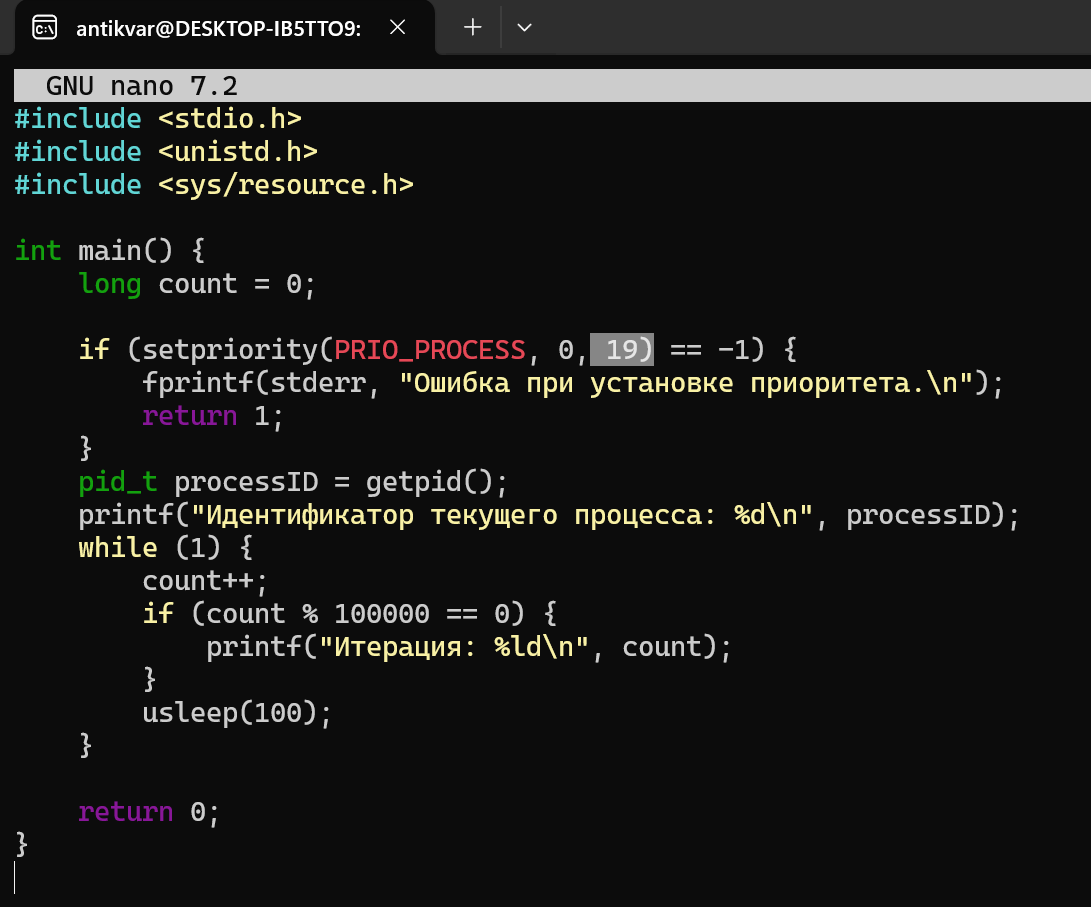


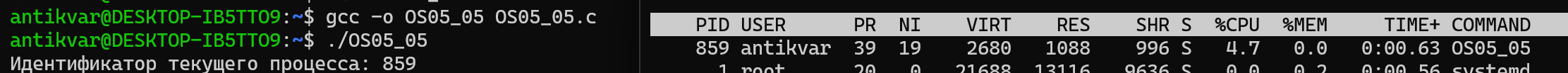
Увеличьте приоритет для OS05\_05 до максимального значения (самого привилегированного). Зафиксируйте текущее значение nicе, полученное с помощью команды top.





Уменьшите приоритет для OS05\_05 до минимального значения (самого ничтожного). Зафиксируйте текущее значение nicе, полученное с помощью команды top





**Задание 06**

1. **Поясните понятие «мультизадачная OS с вытеснением».**

Мультизадачная OS с вытеснением (Preemptive Multitasking OS) – это операционная система, которая способна одновременно выполнять несколько задач, при этом планировщик может прервать выполнение одного процесса или потока и передать управление другому, более приоритетному, без разрешения от текущего процесса. Вытеснение происходит автоматически по таймеру или в случае появления более высокого приоритета.

1. **Поясните понятие «циклическое планирование».**

Циклическое планирование (Round-Robin Scheduling). Каждому процессу назначается определенный интервал времени, называемый его квантом, в течение которого ему предоставляется возможность выполнения. Если процесс к завершению кванта времени все еще выполняется, то ресурс центрального процессора у него отбирается и передается другому процессу. Разумеется, если процесс переходит в заблокированное состояние или завершает свою работу до истечения кванта времени, то переключение центрального процессора на другой процесс происходит именно в этот момент. От планировщика требуется всего лишь вести список процессов, готовых к выполнению. Когда процесс исчерпает свой квант времени, он помещается в конец списка.

1. **Поясните понятие «приоритетное планирование».**

Приоритетное планирование (Priority Scheduling). Каждому процессу присваивается значение приоритетности и запускается тот процесс, который находится в состоянии готовности и имеет наивысший приоритет.

Чтобы предотвратить бесконечное выполнение высокоприоритетных процессов, планировщик должен понижать уровень приоритета текущего выполняемого процесса с каждым сигналом таймера (то есть с каждым его прерыванием). Если это действие приведет к тому, что его приоритет упадет ниже приоритета следующего по этому показателю процесса, произойдет переключение процессов.

Можно выбрать и другую альтернативу: каждому процессу может быть выделен максимальный квант допустимого времени выполнения. Когда квант времени будет исчерпан, шанс запуска будет предоставлен другому процессу, имеющему наивысший приоритет.

Приоритеты могут присваиваться процессам в статическом или в динамическом режиме.

1. **Поясните понятие «кооперативное планирование».**

Кооперативное планирование (Cooperative Scheduling) – это тип планирования, при котором операционная система передает управление процессам или потокам, но не прерывает их принудительно. Процессы сами должны добровольно уступать процессор, когда они завершают свою работу или готовы сделать это. В такой системе ОС не может принудительно прервать выполнение задачи.

1. **Поясните понятие «OS реального времени».**

OS реального времени (Real-Time OS) – это операционная система, которая гарантирует выполнение задач в строго заданные временные рамки. Такая ОС используется в системах, где важна своевременность реакции, например, в медицинских устройствах или системах управления транспортом.

1. **Поясните понятие «приоритет процесса».**

Приоритет процесса (Process Priority) это значение, которое определяет важность или срочность выполнения процесса в операционной системе, на основании которого происходит планирование. Процессы с более высоким приоритетом получают доступ к процессору раньше процессов с низким приоритетом.

1. **Поясните выражение «поток уступает процессор другому потоку».**

Поток уступает процессор другому потоку – это ситуация, когда текущий поток либо завершает выполнение, либо принудительно вытесняется планировщиком, чтобы другой поток мог получить доступ к ресурсу центрального процессора.

1. **Windows: как поток может уступить процессор?**

В Windows поток может уступить процессор в следующих случаях:

* Истекает квант времени, выделенный для его выполнения.(квант)
* Он выполняет системный вызов Sleep или SwitchToThread, которые явно передают управление другому потоку. (приостанавливабт выполнение потока)
* Операционная система вытесняет поток из-за более высокого приоритета у другого потока.
* Поток может ожидать завершения определенного события или объекта синхронизации (например, мьютекса, семафора или объекта события). В этом случае поток переводится в состояние ожидания и освобождает процессор для выполнения других потоков.

1. **Windows: что такое базовый приоритет потока, как он вычисляется и диапазон его изменения?**

Базовый приоритет потока – это начальный приоритет, с которым поток запускается. Он вычисляется на основе приоритета процесса и классов приоритетов (Idle, Normal, High, Real-time). Диапазон значений базового приоритета в Windows варьируется от 0 до 31, где 0 – самый низкий приоритет, а 31 – самый высокий.

1. **Windows: поясните назначение и принцип применения системного вызова ResumeThread.**

Вызов ResumeThread используется для возобновления выполнения приостановленного потока. Когда поток находится в состоянии ожидания (suspended), вызов этой функции уменьшает его счетчик приостановки. Если счетчик достигает нуля, поток возобновляет выполнение.

1. **Windows: поясните назначение и принцип применения системного вызова WaitForSingleObject.**

Вызов WaitForSingleObject используется для ожидания завершения объекта (процесса, потока, события и т. д.). Он блокирует выполнение текущего потока, пока указанный объект не перейдет в сигнальное состояние или пока не истечет заданное время ожидания.

1. **Windows: поясните назначение и принцип применения системных вызовов GetProcessPriorityBoost, GetThreadPriorityBoost, SetProcessPriorityBoost, SetThreadPriorityBoost.**

Эти вызовы управляют механизмом автоматического повышения приоритета процессов и потоков в Windows

GetProcessPriorityBoost и GetThreadPriorityBoost возвращают информацию о том, включено ли автоматическое повышение приоритета для процесса или потока.

SetProcessPriorityBoost и SetThreadPriorityBoost позволяют включать или отключать эту функцию. Автоматическое повышение приоритета временно увеличивает приоритет потока, чтобы избежать его блокировки при взаимодействии с вводом-выводом.

1. **Linux: поясните принцип идентификации процессов и потоков и поясните, почему он такой.**

В Linux процесс и поток идентифицируются с помощью уникального идентификатора — PID (Process ID) для процессов и TID (Thread ID) для потоков. Для процессов PID является уникальным, но в Linux потоки — это "легковесные" процессы (lightweight processes), и каждому потоку также присваивается свой уникальный TID. Все потоки, созданные внутри одного процесса, имеют общий PID, но каждый поток имеет свой TID. Это позволяет ядру Linux одинаково обращаться как с процессами, так и с потоками, делая планирование более унифицированным и упрощенным.

1. **Linux: Поясните понятие «планировщик потоков».**

В Linux понятие «планировщик потоков» относится к компоненту операционной системы, который отвечает за распределение процессорного времени между потоками и процессами. Планировщик определяет, какие потоки или процессы будут выполняться в данный момент, сколько времени они будут использовать процессор и в каком порядке они будут выполняться.

Completely Fair Scheduler (CFS)CFS является планировщиком по умолчанию в современных версиях Linux. Он стремится обеспечить «справедливое» распределение процессорного времени.

Логика:

* Каждому потоку назначается виртуальное время, которое определяет, сколько процессорного времени он должен получить.
* Потоки с меньшим временем ожидания получают более высокий приоритет.

Политики реального времени: Для задач, требующих строгих временных ограничений, Linux поддерживает несколько политик планирования:

* FIFO (First In, First Out): Потоки с этой политикой выполняются в порядке их поступления. Поток с более высоким приоритетом может прервать поток с более низким приоритетом.
* Round Robin: Потоки делят процессорное время поровну, но при этом потоки с более высоким приоритетом получают большее количество временных квантов.

Логика:

* Потоки реального времени имеют более высокий приоритет по сравнению с обычными потоками.
* Приоритеты могут задаваться вручную через системные вызовы, такие как sched\_setscheduler.

Алгоритмы динамического приоритета: В большинстве алгоритмов, включая CFS, приоритеты могут изменяться в зависимости от поведения потоков:

Если поток долгое время не использует процессор (например, находится в состоянии ожидания), его приоритет может повышаться, чтобы избежать «голодания». Активные потоки могут понижать свои приоритеты, чтобы дать возможность менее активным потокам получить доступ к процессору.

Системные вызовы для управления приоритетами

* setpriority() и nice(): Пользователи и приложения могут изменять приоритеты процессов через системные вызовы, такие как setpriority или nice, которые позволяют увеличить значение «nice», что в свою очередь понижает приоритет процесса.
* sched\_setscheduler(): Этот вызов позволяет установить конкретную политику планирования и приоритет для процесса или потока.

Приоритеты могут изменяться на основе времени, проведенного потоком в ожидании: Потоки, которые долго ждут, могут получать повышенные приоритеты для предотвращения ситуации, когда они не могут работать.

1. **Linux: поясните принцип использования значения nice – процесса, диапазон его изменения, для какого режима работы планировщика это значение применяется?**

Приоритет nice и приоритет планировщика процессов ядра ОС — разные числа. Число nice — приоритет, который пользователь хотел бы назначить процессу. Приоритет планировщика — действительный приоритет, назначенный процессу планировщиком.

Планировщик процессов ядра ОС Linux поддерживает приоритеты от 0 (реальное время) до 139 включительно. Приоритеты -20…+19 утилиты или команды nice соответствуют приоритетам 100…139 планировщика процессов.

nice — это параметр, который определяет приоритет выполнения процесса.

Значение nice определяет базовый приоритет процесса в системе. Чем ниже значение nice, тем выше приоритет процесса, и наоборот.

Значение nice может быть изменено пользователями и процессами с помощью команд, таких как nice и renice:

* nice <значение> <команда>: Запускает новую команду с указанным значением nice.
* renice <новое значение> <PID>: Изменяет значение nice для уже запущенного процесса.

Значение nice может изменяться в диапазоне от -20 до 19:

* -20: Наивысший приоритет (максимально низкий уровень «nice»). Статический приоритет равен 100.
* 0: Нормальный уровень приоритета (значение по умолчанию).
* 19: Наинизший приоритет (максимально высокий уровень «nice»). Статический приоритет равен 100.

Completely Fair Scheduler (CFS). Этот алгоритм учитывает значение nice для определения, сколько времени процесс должен получить на процессоре.

Реальное время: Значение nice не влияет на процессы, запущенные с политиками реального времени. Процессы реального времени имеют более высокий приоритет, и их выполнение не зависит от значения nice.

1. **Linux: перечислите политики планирования, какая действует по умолчанию?**

1. SCHED\_OTHER (или CFS) Это стандартная политика планирования, которая используется для обычных процессов. Она основана на алгоритме CompletelyFair Scheduler (CFS), который обеспечивает «справедливое» распределение процессорного времени между всеми процессами.

Используется: для обычных (не привилегированных) процессов.

Статический приоритет равен 0. Для политики OTHER важнее не абсолютный приоритет, а "nice" — параметр, который определяет относительный приоритет процесса. Чем выше значение nice, тем меньше приоритет у процесса (он будет меньше конкурировать за процессорное время).

Процессы с политикой OTHER будут выполняться, если нет процессов с более высокоприоритетными политиками, такими как FIFO или RR.

2. SCHED\_FIFO Это политика реального времени, которая использует алгоритм First In, First Out(FIFO). Процессы с этой политикой выполняются в порядке их поступления, а потоки с более высоким приоритетом могут прерывать потоки с более низким приоритетом. Используется для процессов реального времени, которым нужно более строгое планирование.

У каждого процесса FIFO фиксированный статический приоритет, который определяет порядок его выполнения. Процессы с более высоким приоритетом подавляют процессы с более низким приоритетом.

Если процесс с политикой FIFO имеет самый высокий приоритет, он будет выполняться до тех пор, пока либо не блокируется (например, ожидая ввод-вывод), либо не уступит процессор по вызову sched\_yield().

Когда процесс блокируется, он помещается в конец своей очереди, и очередь продвигается.

Новый процесс с политикой FIFO добавляется в начало своей очереди, то есть начинает исполняться сразу, если у него более высокий приоритет.

3. SCHED\_RR (Round Robin): Эта политика также относится к реальному времени и основана на алгоритме Round Robin. Процессы с этой политикой получают фиксированный временной квант, после чего планировщик переключается на следующий поток с тем же приоритетом. Используется также для процессов реального времени, но с квантованием времени.

Приоритет также определяется статическим приоритетом, как и в FIFO, и процессы с более высоким приоритетом подавляют более низкие.

Отличие от FIFO в том, что здесь процесс получает "квант времени" — определенное количество времени, в течение которого он может выполняться.

Когда квант времени заканчивается, процесс помещается в конец своей очереди, и CPU переходит к следующему процессу с тем же приоритетом.

Такая политика позволяет всем процессам с одинаковым приоритетом работать по очереди, предотвращая бесконечное выполнение одного процесса.

Для установки и получения политики планирования процесса используются функции sched\_setscheduler и sched\_getscheduler, соответственно.

4. SCHED\_BATCH: политика для выполнения процессов, не требующих интерактивности. BATCH от OTHER отличается тем, что BATCH снижает частоту переключений задач, что делает её лучше для длительных фоновых вычислений. OTHER же оптимизирована для обычных задач, требующих более быстрого отклика, и распределяет ресурсы с учётом интерактивности процессов.

Задачи с BATCH запускаются, когда нет активных интерактивных задач

5. SCHED\_IDLE: Эта политика используется для фоновых процессов, которые должны выполняться только тогда, когда нет других активных процессов. Потоки с этой политикой имеют очень низкий приоритет.

1. **Linux: с помощью какого системного вызова поток может уступить процессор.**

Поток в Linux может уступить процессор с помощью системного вызова sched\_yield(). Этот вызов позволяет текущему потоку добровольно отдать процессорное время другому потоку, который ожидает выполнения, согласно политике планирования.

**Вопросы у Смелова (которых у нас в лабе нету)**

1.Windows: поясните назначение и принцип применения системного вызова SetThreadIdealProcessor.

Системный вызов SetThreadIdealProcessor в Windows используется для указания предпочтительного (или "идеального") процессора для выполнения определённого потока (thread). Этот вызов помогает операционной системе понять, на каком процессоре лучше всего исполнять данный поток, что может улучшить производительность за счёт уменьшения частоты переключения процессора и эффективного использования кэша. SetThreadIdealProcessor лишь указывает предпочтительный процессор, но не гарантирует, что поток будет выполняться именно на нём.

2. Linux: как выяснить действующую политику планирования для процесса с помощью файловой системы proc?

В Linux для определения политики планирования конкретного процесса можно воспользоваться файловой системой /proc. Каждый процесс в системе имеет свой подкаталог в /proc, содержащий различные файлы с информацией о процессе. Для определения политики планирования процесса можно использовать файл status в каталоге процесса. Он находится по пути /proc/<pid>/status, где <pid> — идентификатор процесса.

**4. Linux: чем отличается системный вызов nice от вызова setpriority.**

nice изменяет приоритет процесса относительно стандартного значения приоритета (обычно 0). Этот вызов сдвигает "nice-значение" процесса, что влияет на его приоритет планирования.

nice принимает значение, называемое "nice-уровнем", от -20 до 19. Чем выше значение nice, тем ниже приоритет процесса (например, 19 означает "наименьший" приоритет, а -20 — наивысший). Команда nice может быть использована при запуске процесса, чтобы сразу задать ему приоритет. Только у текущего процееса.

setpriority: Этот системный вызов позволяет изменять приоритет процесса, группы процессов или всех процессов пользователя, но только если процесс, для которого вы пытаетесь установить новый приоритет, принадлежит вам, или если вы работаете с процессами другого пользователя, имея привилегии суперпользователя (root).

Для повышения приоритета процесса (уменьшение nice-значения) требуются права суперпользователя.

Если вы пытаетесь изменить приоритет чужого процесса (потока), не имея прав root, вызов setpriority завершится с ошибкой (например, EPERM — операция не разрешена).

5. Linux: поясните понятие «планировщик ввода вывода», каким образом можно выяснить какие планировщики ввода/ вывода доступны?

Они постоянно взаимодействуют со своими пользователями и поэтому тратят много времени на ожидание нажатий клавиш и операций мыши. При получении ввода процесс должен быть быстро разбужен, иначе пользователь обнаружит, что система не отвечает. Обычно средняя задержка должна находиться в пределах от 50 до 150 миллисекунд. Дисперсия такой задержки также должна быть ограничена, иначе пользователь обнаружит, что система работает нестабильно. Типичные интерактивные программы — это командные оболочки, текстовые редакторы и графические приложения.

cat /sys/block/устройство/queue/scheduler

6. Linux: перечислите известные вам планировщики ввода/ вывода, кратко охарактеризуйте их.

CFQ (Completely Fair Queuing) — планировщик, обеспечивающий справедливое распределение времени процессора для операций ввода-вывода, с учетом приоритетов и пропорциональности.

Deadline — планировщик, который ориентируется на сроки выполнения операций. Он пытается минимизировать задержки, управляя очередями запросов с учетом их дедлайнов.

Noop — очень простой планировщик, который просто передает запросы устройству без какой-либо сортировки или сложной логики. Он подходит для быстрых устройств хранения, таких как SSD, где порядок операций имеет минимальное значение.

Anticipatory — старый планировщик, который предсказывает, что если одно устройство ввода-вывода было выполнено, скорее всего, следующий запрос будет также связан с тем же устройством, и планирует операции с учетом этого.

BFQ (Budget Fair Queueing) — более сложный планировщик, ориентированный на справедливое распределение ресурсов I/O между потоками с учетом их «бюджетов», что улучшает взаимодействие с многозадачными приложениями.

7. Linux: каким образом можно выяснить тип планировщика действующего для блокового устройства?

При назначении приоритетов потокам и процессам в операционной системе Windows ключевыми аспектами являются:

1. Тип задачи и её важность

Критичность задачи: Например, задачи, связанные с системными процессами, драйверами, или обработкой ввода от пользователя, требуют более высокого приоритета. Такие задачи должны реагировать быстрее, поскольку их задержка может напрямую повлиять на пользовательский опыт и стабильность системы.

Тип процесса: Например, фоновая задача резервного копирования может иметь низкий приоритет, чтобы не замедлять активные задачи пользователя.

2. Время отклика и требование к интерактивности

Интерактивные приложения (например, текстовые редакторы, браузеры) требуют низкого времени отклика, чтобы пользователи ощущали мгновенную реакцию на свои действия. Поэтому приоритет их потоков обычно повышается.

Фоновые процессы: Процессы, выполняемые в фоновом режиме (например, индексация файлов или обновления), могут быть отложены на более позднее время и получать меньшее внимание.

3. Ресурсоемкость процесса

Процессы с высоким потреблением ресурсов (например, видеообработка, 3D-рендеринг) могут иметь повышенный приоритет, чтобы быстрее завершить ресурсоемкие задачи.

Энергоэффективность: Важно учитывать также влияние на батарею (для ноутбуков) и нагрузку на систему.

4. Наличие ввода-вывода и событий ожидания

Задачи с интенсивным вводом-выводом (например, операции чтения и записи на диск) получают временное повышение приоритета. Это позволяет минимизировать простои, когда потоки ждут данных и не используют процессор.

Ожидание завершения событий: Задачи, которые ждут завершения какого-то события, могут временно получать более высокий приоритет, чтобы сразу возобновиться, как только событие произойдет.

5. Взаимозависимость процессов

Координация с другими процессами: Если один процесс зависит от данных другого, важно, чтобы они работали с согласованными приоритетами, избегая взаимного блокирования или значительных задержек.

Общие ресурсы: Например, если несколько потоков работают с одними и теми же данными, их приоритеты должны учитывать очередность доступа к данным для повышения общей производительности.

6. Уровень приоритета пользователя

Указания пользователя: В Windows пользователь может вручную изменить приоритет конкретного процесса через диспетчер задач. Это учитывается системой, хотя динамические алгоритмы могут при этом скорректировать настройки при необходимости.

Режимы энергосбережения и мощности: Например, в режиме экономии энергии Windows может автоматически понижать приоритет некоторых процессов для снижения энергопотребления.

7. Вытесняемость и многозадачность

Поддержка вытеснительной многозадачности: В Windows приоритет может меняться в зависимости от текущей нагрузки и наличия других процессов с более высоким приоритетом. Если активируется поток с более высоким приоритетом, текущий поток может быть вытеснен, чтобы критически важные задачи завершались быстрее.

8. Пользовательская активность и простаивающее время

Бустинг при простаивающем времени: Windows может временно повышать приоритет простаивающих потоков, если они долгое время не выполнялись, чтобы поддерживать общий баланс нагрузки.

Понижение при активном использовании процессора: Если поток активно использует процессор, система может снизить его приоритет, чтобы он не блокировал другие процессы, особенно в системах с ограниченным количеством ядер.

9. Политика безопасности

Ограничение прав доступа: Потоки и процессы с низким уровнем доверия (например, работающие в "песочнице" или с ограниченными правами) часто имеют ограниченный приоритет. Это помогает защитить систему от чрезмерного потребления ресурсов менее безопасными или потенциально вредоносными процессами.

10. Аппаратные и системные ограничения

Число доступных процессоров и ядер: В многопроцессорных системах (например, с многоядерными CPU) приоритеты назначаются так, чтобы оптимально распределить нагрузку на все доступные ядра.

Прерывания и драйверы: Прерывания аппаратных устройств имеют высокий приоритет и могут прерывать обычные процессы для немедленной обработки.