**WINDOWS**



Базовый приоритет = приоритет класса + приоритет потока

**LINUX**

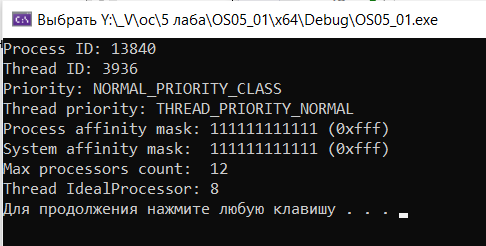
Значения nice варьируются от -20 (высший) до 19 (низший), 0 - нормальный

Приоритет = 20 – nice

**Задание 01**

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

* идентификатор текущего процесса;
* идентификатор текущего (main) потока;
* приоритет (приоритетный класс) текущего процесса;
* приоритет текущего потока;
* маску (affinity mask) доступных процессу процессоров в двоичном виде;
* количество процессоров, доступных процессу;
* процессор, назначенный текущему потоку.



Process ID: 13840 — уникальный идентификатор процесса.

Thread ID: 3936 — уникальный идентификатор потока в процессе.

Priority: NORMAL\_PRIORITY\_CLASS — стандартный приоритет процесса.

Thread priority: THREAD\_PRIORITY\_NORMAL — стандартный приоритет для потока.

Process affinity mask: 111111111111 (0xfff) — процесс может выполняться на всех 12 доступных процессорах.

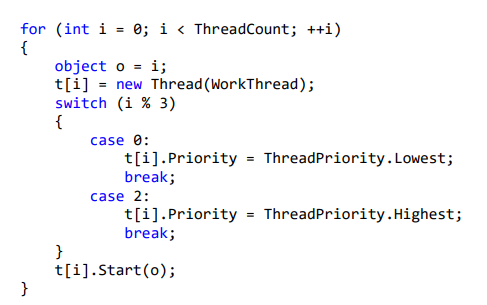
System affinity mask: 111111111111 (0xfff) — все 12 процессоров доступны для всех процессов в системе.

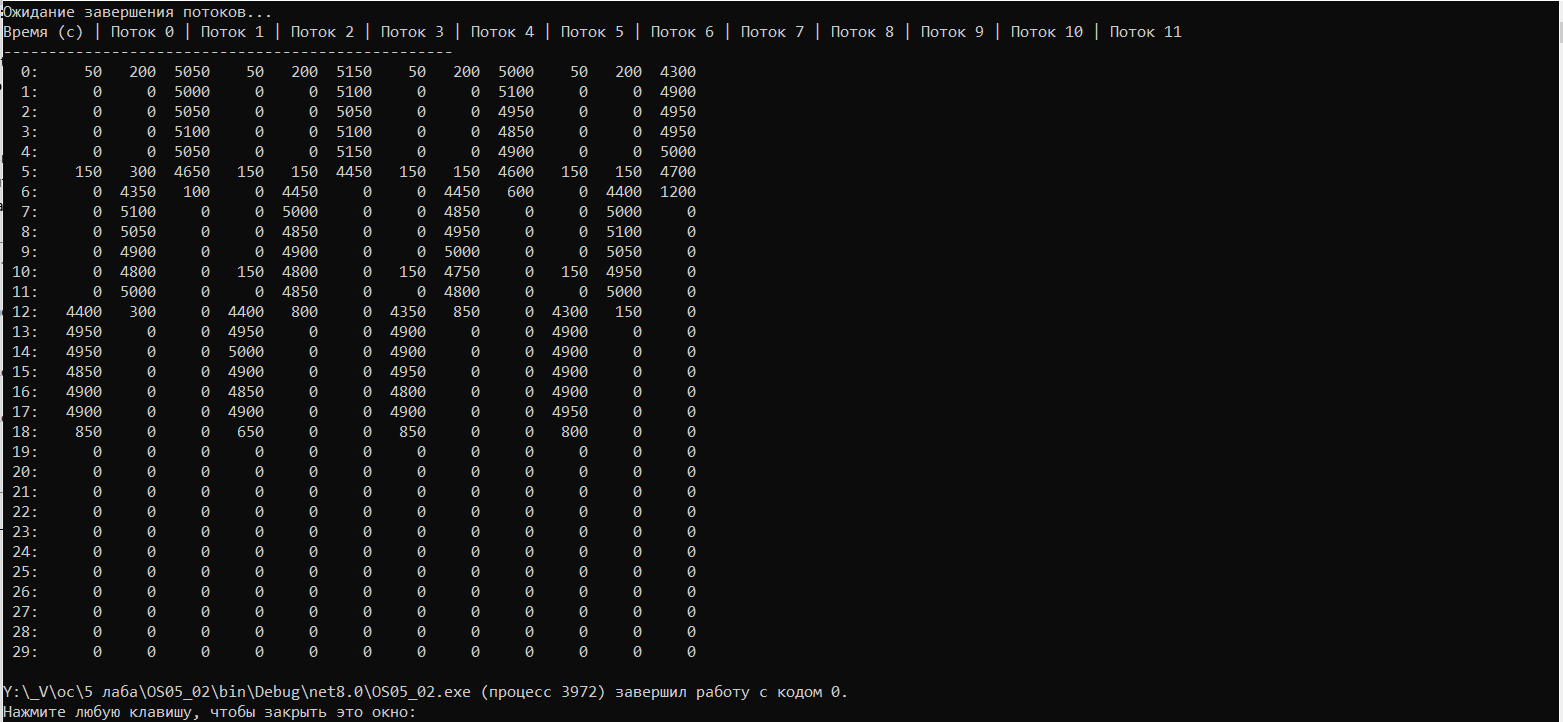
Max processors count: 12 — общее количество процессоров в системе.

Thread IdealProcessor: 8 — оптимальный процессор для выполнения данного потока (номер 8).

**Задание 02**

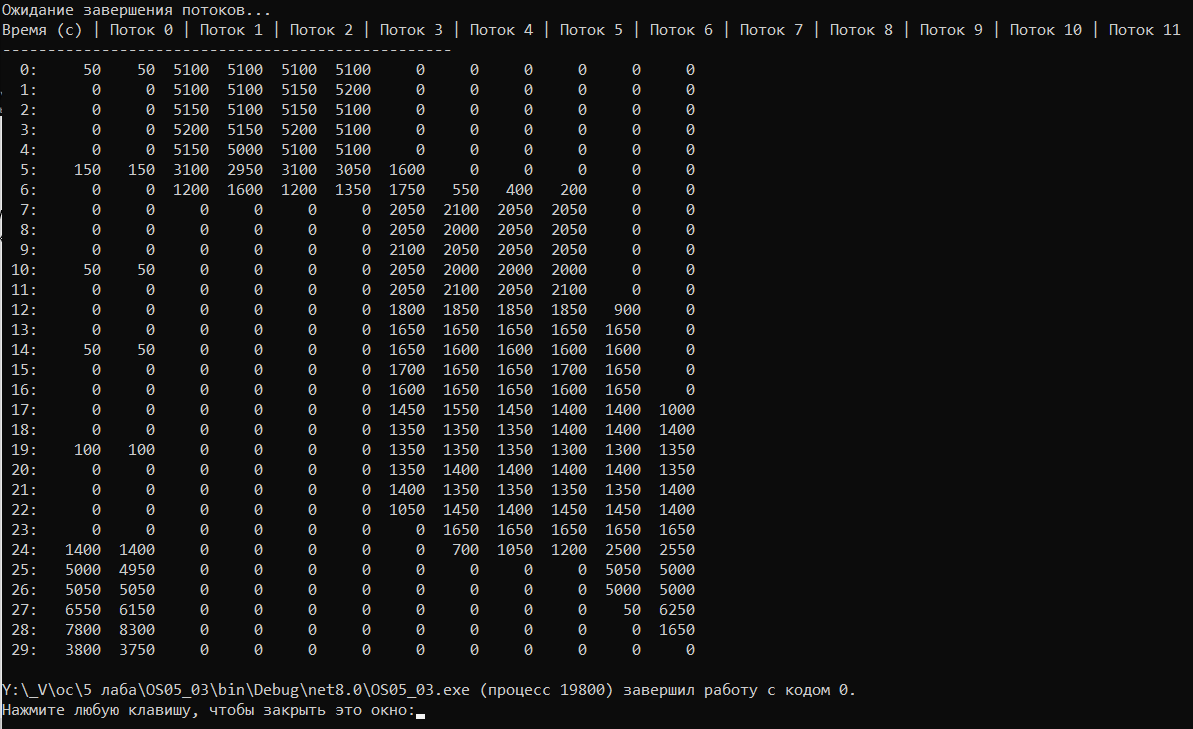
Создайте консольное Windows OS05\_02 на языке С#, взяв за основу приложение OS04\_07 из Лабораторной работы №4. Измените метод Main таким образом, чтобы потоки 0, 3, 6 и т.д. запускались с минимальным приоритетом потока, а потоки 2, 5, 8... – с максимальным. Класс приоритета процесса оставьте по умолчанию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



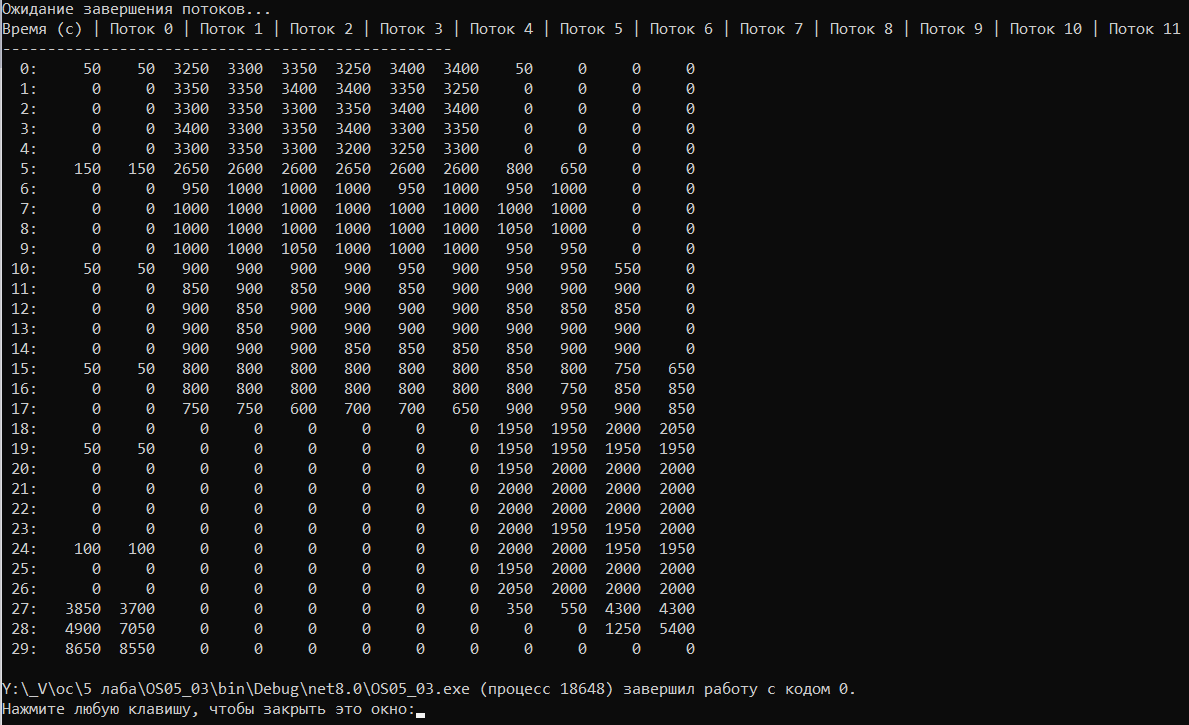


**Задание 03**

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.



С потоками приоритета BelowNormal и Normal



**1**: Highest

**2**: AboveNormal

**3**: Normal

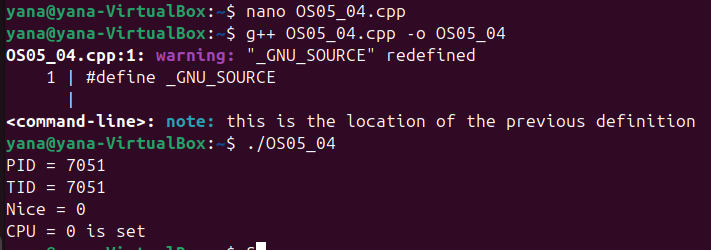
**4**: BelowNormal

**5**: Lowest

**Задание 04**

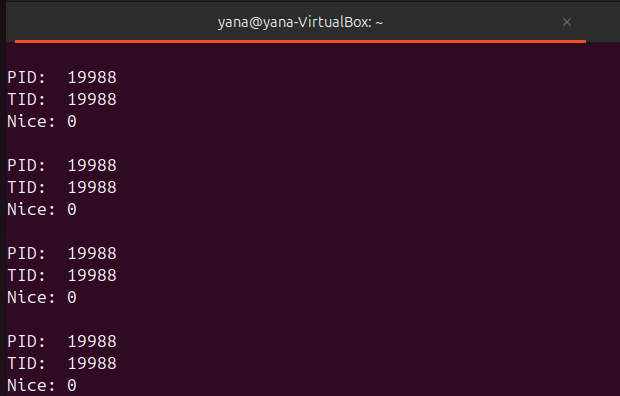
Разработайте консольное Linux-приложение OS05\_04 на языке С++, выводящее на консоль следующую информации:

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

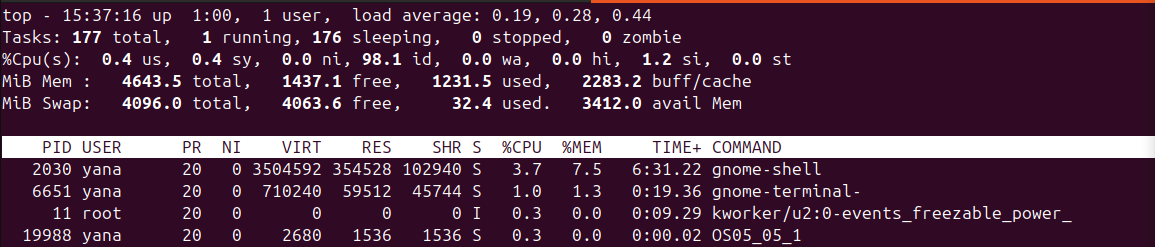


**Задание 05**

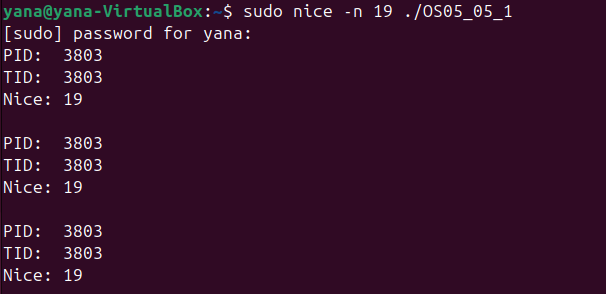
Разработайте консольное Linux-приложение OS05\_05 на языке С, выполняющее длинный цикл. Запустите приложение OS05\_05.

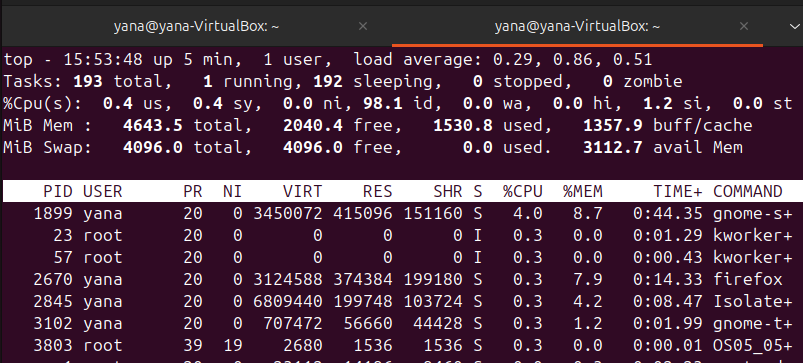


Зафиксируйте < > текущее значение nicе, полученное с помощью команды top.

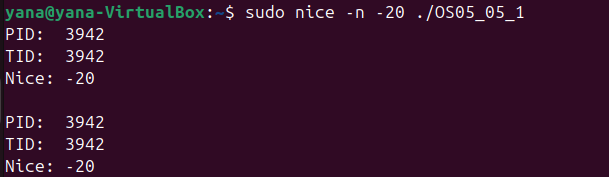


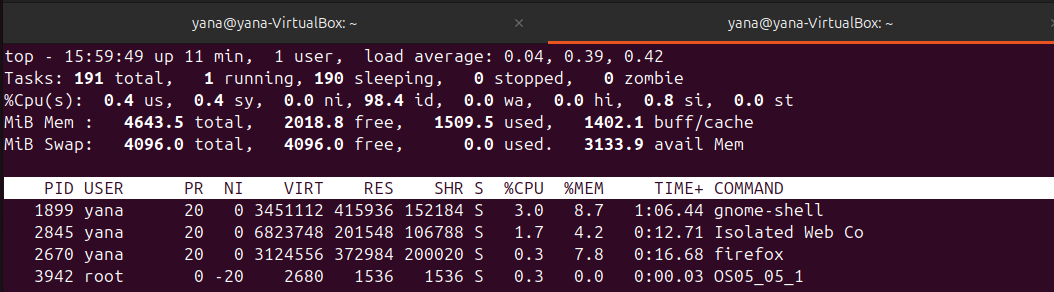
Уменьшите приоритет для OS05\_05 до минимального значения (самого ничтожного). Зафиксируйте < > текущее значение nicе, полученное с помощью команды top





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





**Вопросы**

1. Поясните понятие «мультизадачная OS с вытеснением»

Мультизадачная OS с вытеснением - это операционная система, которая поддерживает выполнение нескольких процессов или потоков (задач) одновременно и использует алгоритмы планирования, которые могут вытеснять текущий процесс или поток с ЦП, даже если он еще не завершил свою работу. Это обеспечивает равномерное распределение ресурсов процессора между задачами и гарантирует, что ни одна задача не блокирует выполнение других задач.

1. Поясните понятие «циклическое планирование».

Циклическое планирование - это метод планирования, при котором каждая задача или процесс получает фиксированный квант времени для выполнения на процессоре. По истечении кванта времени процесс переходит в конец очереди планирования, и следующий процесс начинает выполнение. Этот метод обеспечивает справедливое распределение процессорного времени между задачами.

1. Поясните понятие «приоритетное планирование».

Приоритетное планирование - это метод планирования, при котором каждая задача или процесс имеет приоритет, который определяет, насколько часто она получает доступ к процессору. Задачи с более высоким приоритетом будут исполняться чаще, чем задачи с более низким приоритетом.

1. Поясните понятие «кооперативное планирование».

Это такой алгоритм планирования, при котором процесс получает столько процессорного времени, сколько он считает нужным. Таким образом, все процессы делят процессорное время, периодически передавая управление следующей задаче.

1. Поясните понятие «OS реального времени».

Система, которая гарантирует фиксированное время для выполнения задачи.

Если операция *должна* быть проведена точно в срок (или в определенный период времени), то мы имеем дело с **системой жесткого реального времени**. Другой разновидностью подобных систем является **система мягкого реального времени**, в которой хотя и нежелательно, но вполне допустимо несоблюдение срока какого-нибудь действия, что не наносит непоправимого вреда.

1. Поясните понятие «приоритет процесса».

Приоритет процесса - это числовое значение, которое определяет, насколько важен процесс для операционной системы. Процессы с более высоким приоритетом получают больше процессорного времени и выполняются чаще.

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

Поток уступает процессор другому потоку - это означает, что поток, который в данный момент выполняется на процессоре, добровольно переходит в состояние ожидания, чтобы другой поток мог получить доступ к процессору. Это осуществляется с целью справедливого распределения процессорного времени между потоками.

1. Windows: как поток может уступить процессор?

В Windows поток может уступить процессор, вызвав функцию Sleep(0) или SwitchToThread(), которые позволяют другим потокам выполниться.

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

Базовый приоритет потока – сочетание класса приоритета процесса и приоритета потока, изменяется в пределах [1,31], по умолчанию – 8, приоритеты возрастающие;

так вычисляется:



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

Функция ResumeThread используется для возобновления выполнения потока, который был приостановлен с помощью SuspendThread. Она позволяет продолжить выполнение приостановленного потока.

DWORD ResumeThread (

HANDLE hThread // дескриптор потока

);

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

Функция WaitForSingleObject используется для ожидания завершения выполнения объекта, такого как поток, процесс, событие или мьютекс. Она блокирует текущий поток до тех пор, пока объект не завершит свою работу.

DWORD WaitForSingleObject(

HANDLE **hHandle**, //Дескриптор потока.

DWORD **dwMilliseconds** //Интервал ожидания в миллисекундах.(если INFINITE - то функция вернет только тогда , когда объект сигнализируется.)

);

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

Эти системные вызовы используются для управления приоритетом и "приоритетным ускорением" процессов и потоков в Windows. GetProcessPriorityBoost, GetThreadPriorityBoost возвращает true/false в зависимости от того, разрешено ли динамическое изменение приоритета процесса/потока.

SetProcessPriorityBoost, SetThreadPriorityBoost задают это разрешение.

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

В Linux идентификация процессов и потоков выполняется с использованием идентификаторов процессов (PID) и идентификаторов потоков (TID). Каждый процесс имеет уникальный PID, и каждый поток внутри процесса имеет уникальный TID, но PID и TID могут совпадать в разных процессах. Это позволяет операционной системе однозначно идентифицировать процессы и потоки.

Номер процесса всегда равен номеру главного потока, потому что ядро Линукса не различает потоки и процессы.

1. Linux: Поясните понятие «планировщик потоков».

Планировщик потоков (или планировщик задач) в Linux - это часть операционной системы, отвечающая за распределение процессорного времени между процессами и потоками. Он решает, какие процессы или потоки будут выполняться, и в каком порядке. В Linux существует несколько политик планирования, таких как SCHED\_FIFO, SCHED\_RR и SCHED\_OTHER, которые определяют алгоритмы планирования и приоритеты.

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

nice – чем меньше, тем выше приоритет

значение nice – минимальное значение приоритета =лучшее значение = самый высокий приоритет.

- nice – диапазон приоритетов [-20, 19], default = 0

Чтобы установить значение nice ниже нуля, требуются права суперпользователя

РЕЖИМ РАБОТЫ: вытесняющий(other) (Планировщик может принудительно забирать управление у потока (например по таймеру или при появлении потока с большим приоритетом))

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

В Linux существует несколько политик планирования, включая:

SCHED\_OTHER (по умолчанию) - политика для обычных процессов

SCHED\_FIFO - политика с жесткими временными ограничениями

SCHED\_RR (карусельная (round-robin)) - политика с жесткими временными ограничениями и квази-раунд-робин планированием

SCHED\_BATCH - политика для процессов, выполняющих задачи с низким приоритетом

SCHED\_IDLE - политика с наименьшим приоритетом

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

Поток может уступить процессор с помощью системного вызова sched\_yield().