ОС, ПОИТ-3, Лекция 04

1. **OS:** предыдущие лекции, повторение

* процесс – единица работы OS;
* поток – единица работы процессора, ему выделяется квант процессорного времени;
* процессы и потоки создаются с помощью системных вызовов;
* процессы и потоки – объекты ядра операционной системы;
* в рамках любого процесса, как минимум 1 поток – main;
* процессы изолированы друг от друга;
* потоки в рамках одного процесса имеют доступ ко всем общим ресурсам процесса;
* процессы и потоки имеют контекст – память, в которой сохраняются данные процесса или потока необходимые для продолжения выполнения;
* процессу соответствует исполняемый файл;
* потоку соответствует потоковая функция (фрагмент кода процесса);
* системные процессы (windows-сервисы, Linux-демоны);
* системные процессы и потоки ядра OS;
* OS API позволяет управлять процессами и потоками (создавать, удалять, приостанавливать, …);
* процессу доступны три потока данных: ввода, вывода, ошибок;
* завершение родительского процесса или потока приводит к завершению всех дочерних процессов и потоков;
* многопоточность приоритетная и кооперативная;
* в OS есть процесс инициализации - прародитель всех процессов OS.

**OS Process and Threads dispatching**

1. **OS:** циклическоепланирование

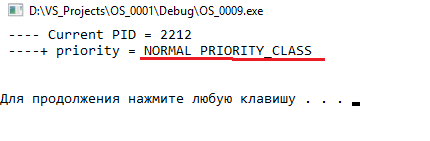


1. **OS:** приоритетное планирование



1. **OS:** кооперативное планирование
2. **OS:** гарантийное планирование, OS реального времени.
3. **OS:** автоматическое динамическое повышение приоритета.
4. **OS:** SMP(symmetric multiprocessing, одинаковые равноценные процессоры), закрепление процессов за процессорами, две задачи: повышение производительности, равномерная загрузка процессоров, переключение процесса на другой процессор приводит к снижению производительности (кэш привязан к процессору, эффект ping/pong).
5. **OS:** Windows,приоритет процесса





1. **OS:** Windows: значение приоритетов возрастающая (0-31)
2. **OS:** Windows**,** приоритеты процесса

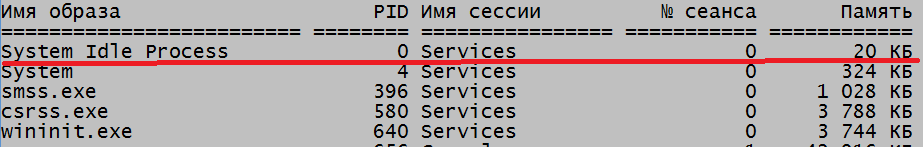
* фоновые процессы;
* процессы с нормальным приоритетом;
* процессы с высоким приоритетом;
* процессы реального времени.

1. **OS:** системы реального времени, управление, операционные системы реального времени, обратная связь, гарантированный интервал выполнения.



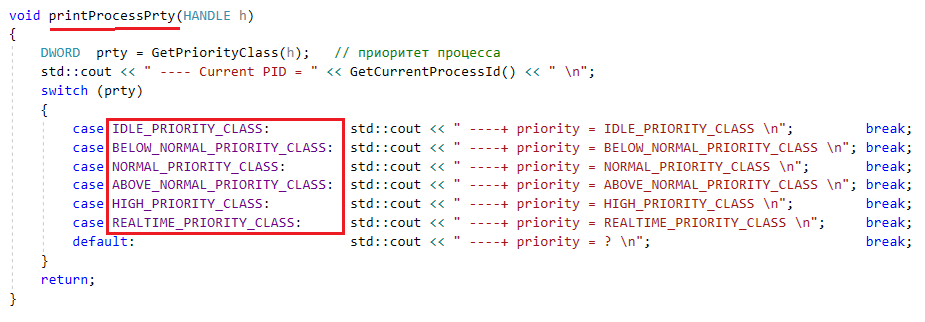


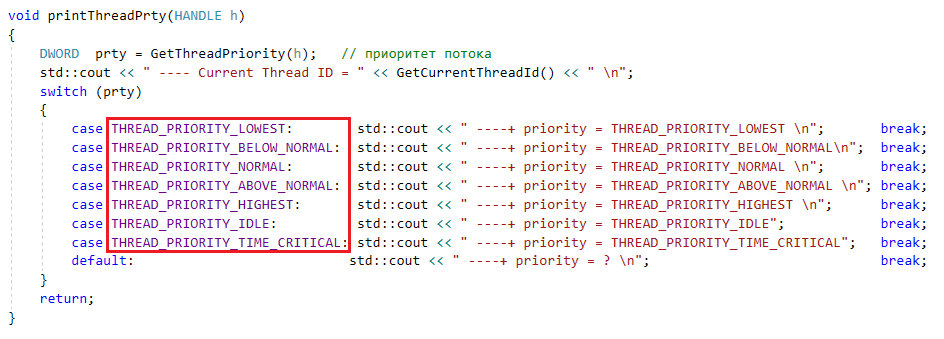
1. **OS:** Windows**,** System Idle Process

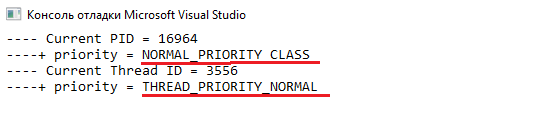


1. **OS:** Windows**,** уступить процессор Sleep(0)
2. **OS:** Windows**,** приоритет процесса и потока

****

****

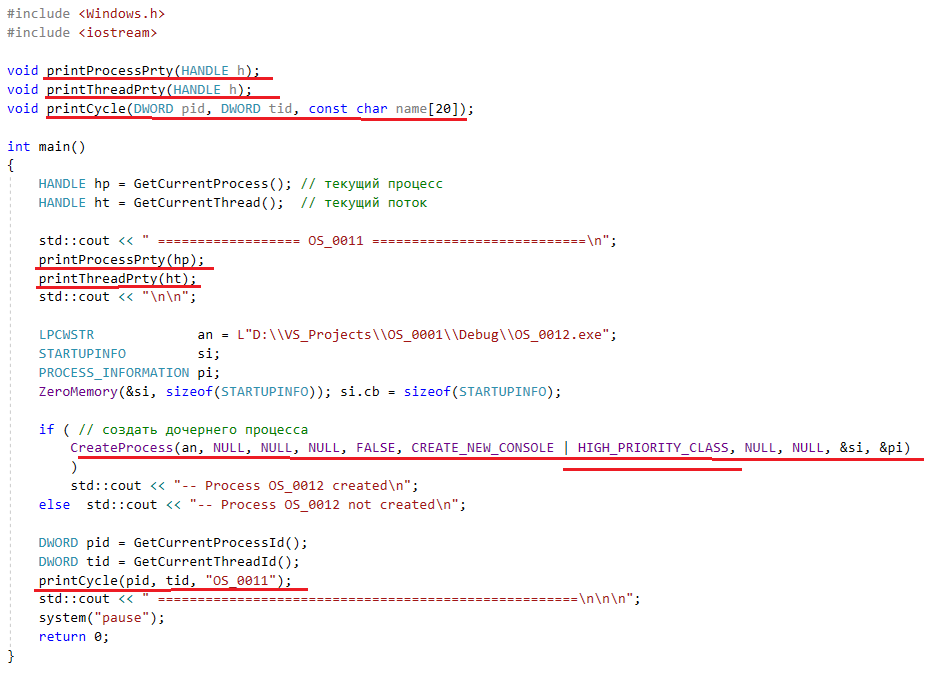
****

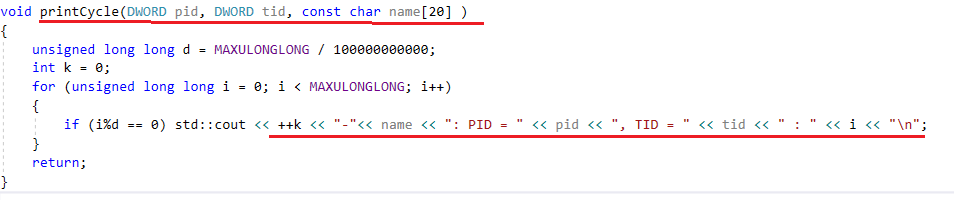
****

1. **OS:** Windows**,** базовые приоритеты потока

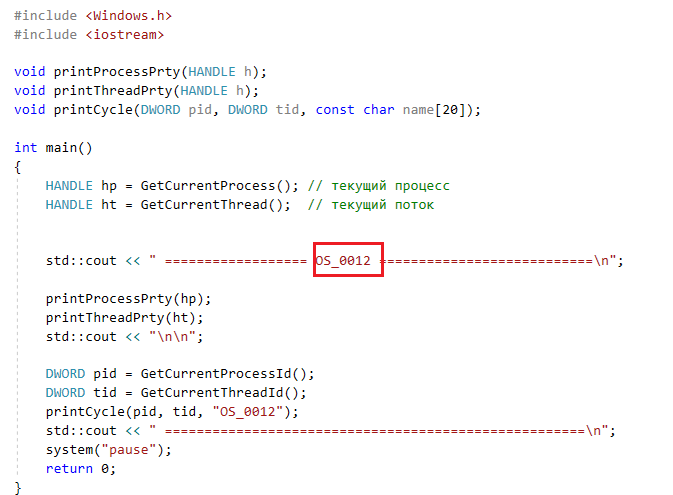
****

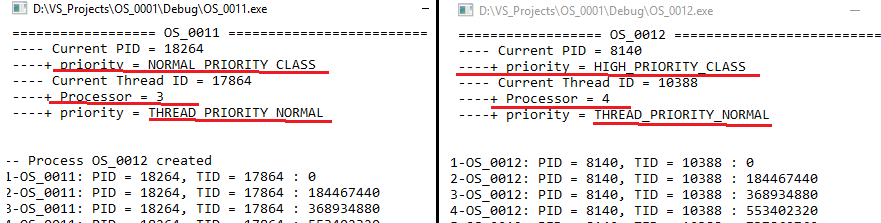
1. **OS:** Windows, CreateProcess(…HIGH\_PRIORITY\_CLASS…), GetPriorityClass(), GetThreadPriority(), SetThreadIdealProcessor(…MAXIMUM\_PROCESSORS)

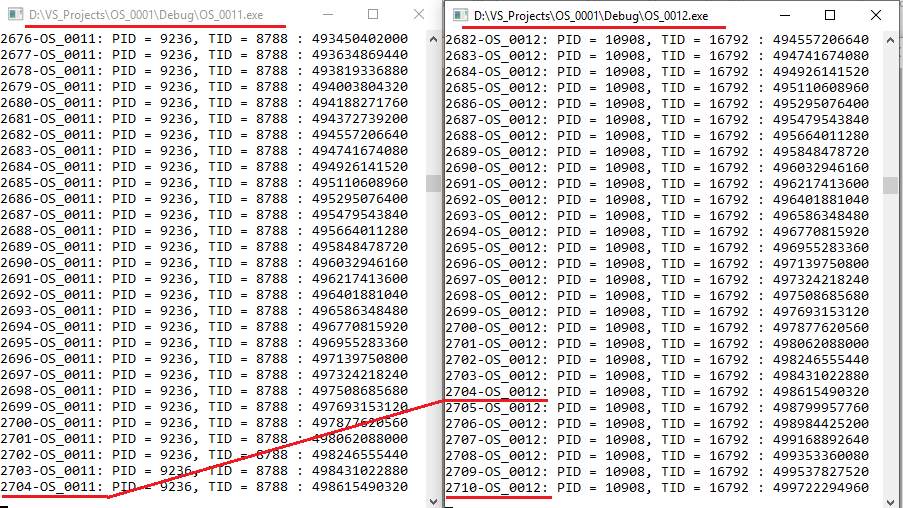


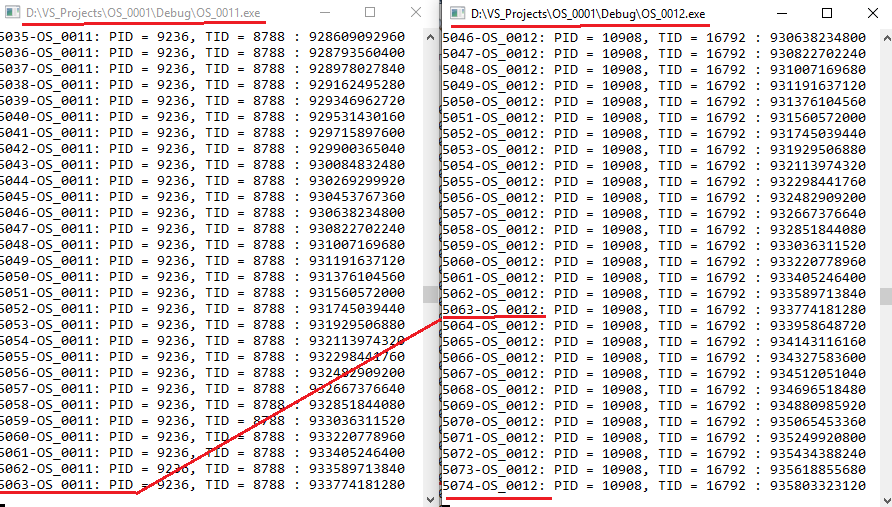






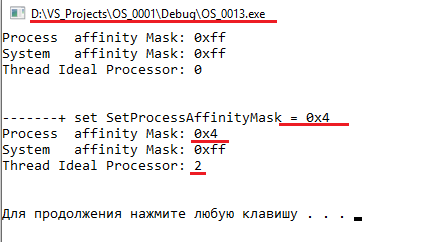


****

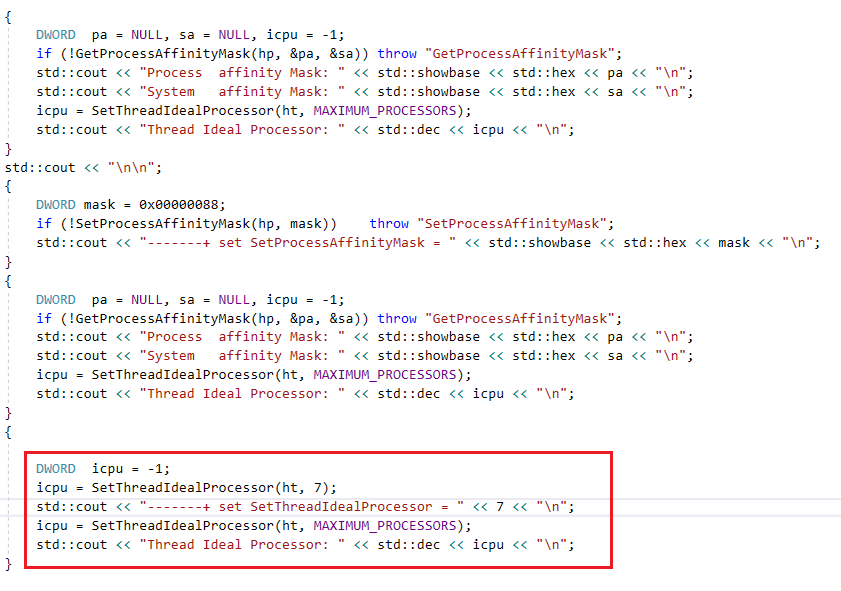


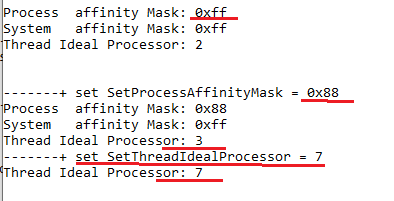
1. **OS:** GetProcessAffinityMask(), SetProcessAffinityMask(),  SetThreadIdealProcessor(…MAXIMUM\_PROCESSORS),



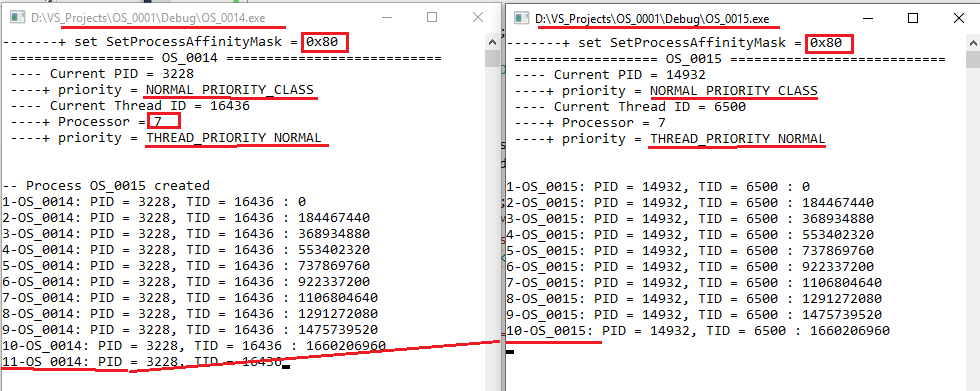


1. **OS:** Windows, SetProcessAffinityMask(), SetThreadIdealProcessor()

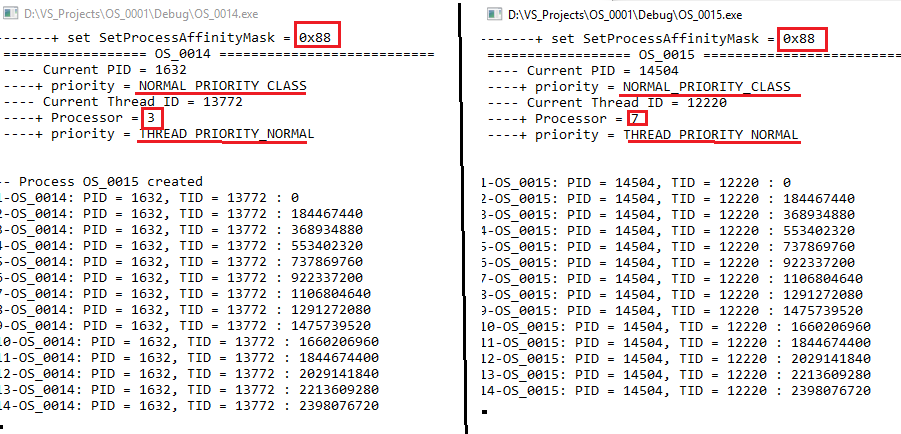
****

****

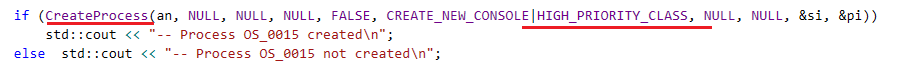
1. **OS:** Windows, один процессор, два процесса с одним приоритетом.

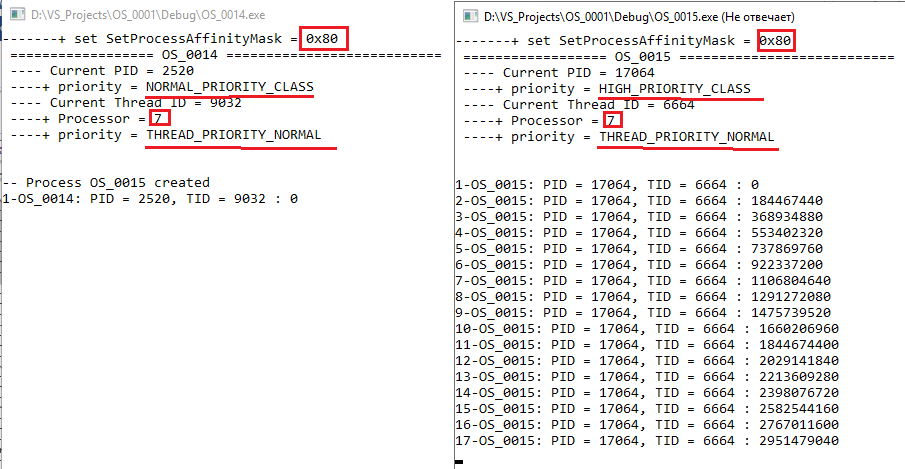


1. **OS:** Windows, два процессора, два процесса с одним приоритетом.

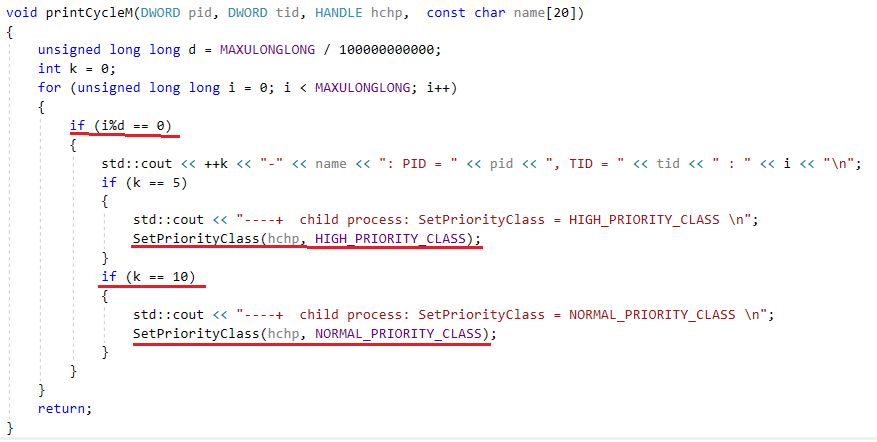


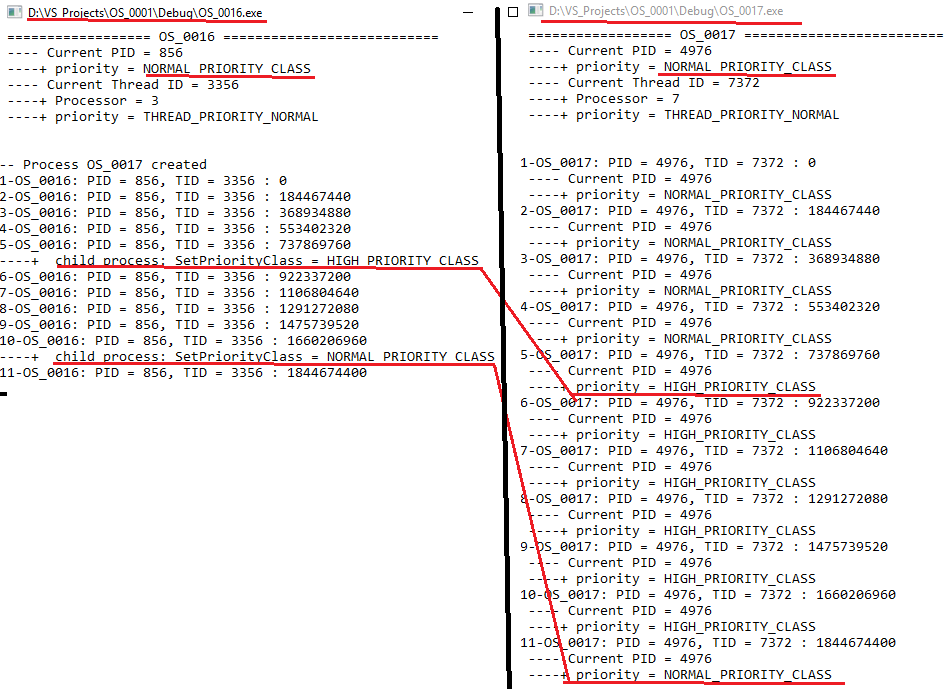
1. **OS:** Windows, один процессор, два процесса с разными приоритетами.



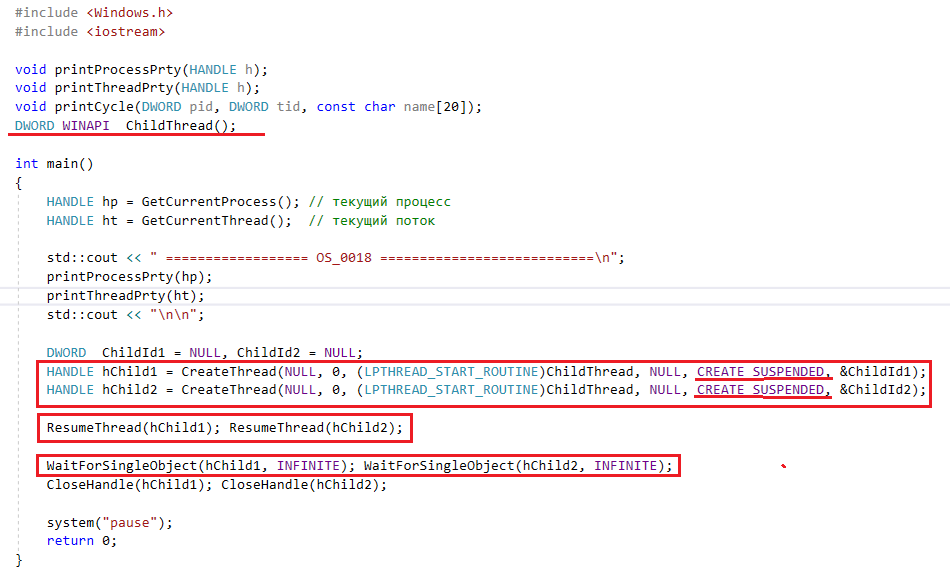


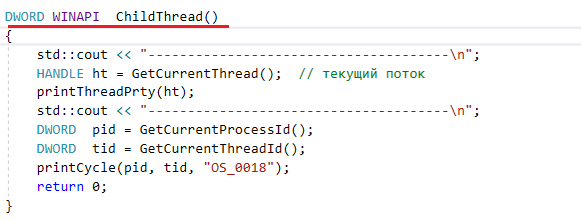
1. **OS:** Windows, SetPriorityClass()

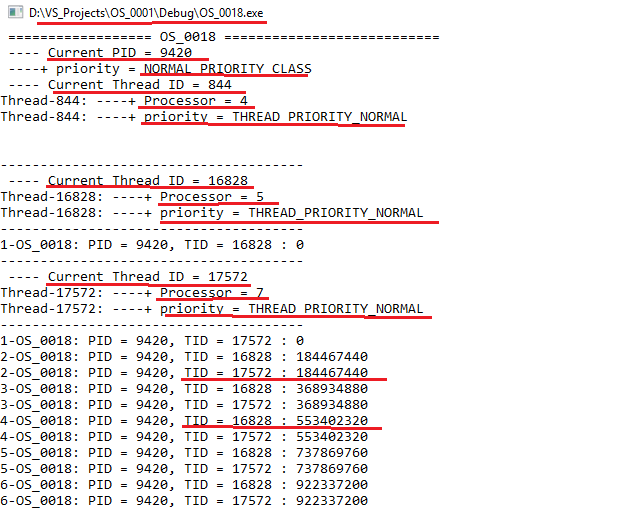




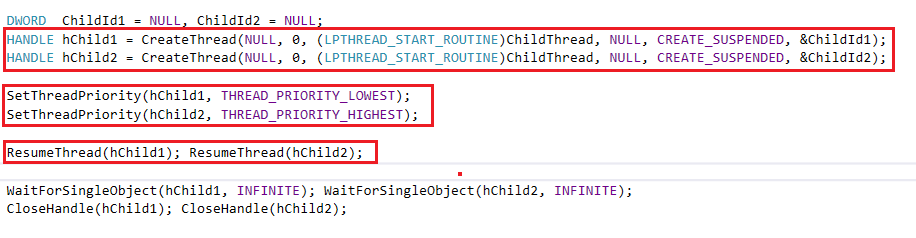
1. **OS:** Windows, CreateThread, ResumeThread, WaitForSingleObject

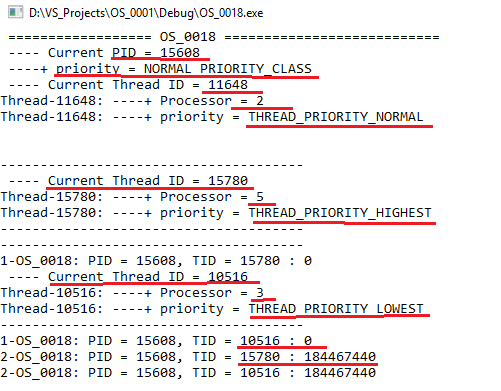






1. **OS:** Windows, CreateThread, ResumeThread, SetThreadPriority, WaitForSingleObject



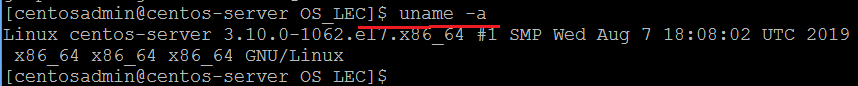


1. **OS:** Windows, только для prty = [0, 15], GetProcessPriorityBoost, SetProcessPriorityBoost, GetThreadPriorityBoost, SetThreadPriorityBoost, **TRUE – запретить boost, FALSE – разрешить boost**

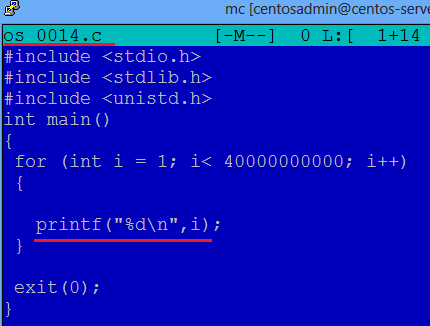


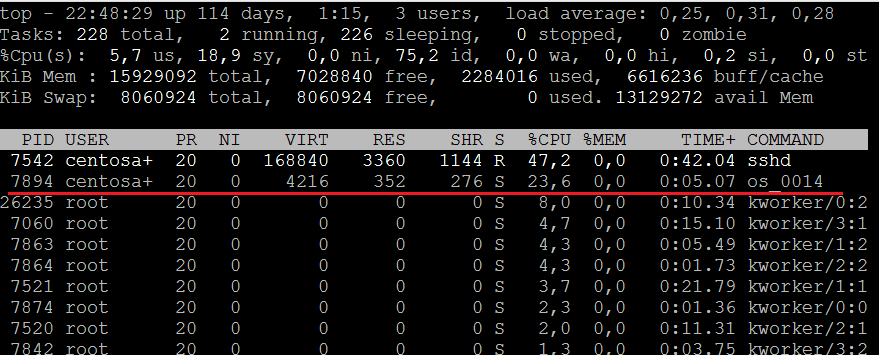


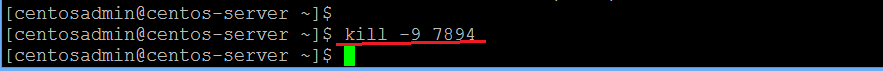
1. **OS:** Linux – мультизадачная OS с вытеснением (приоритетами).

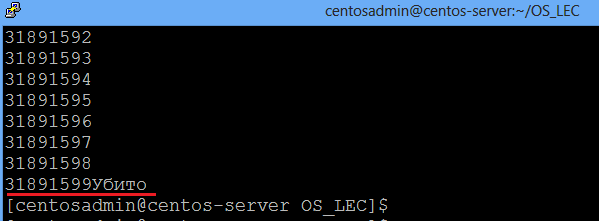


1. **OS:** Linux, top, NI - nice, kill

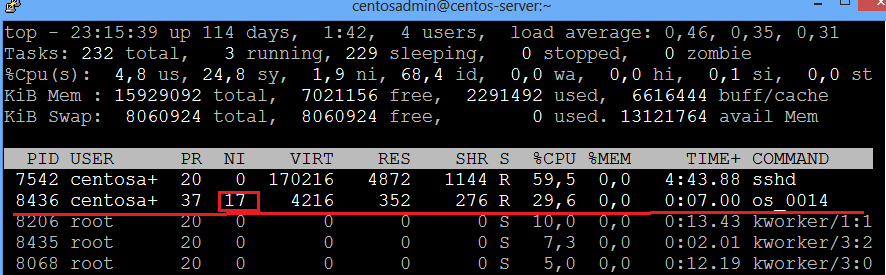




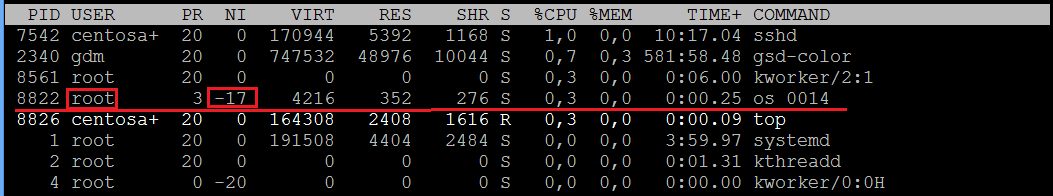






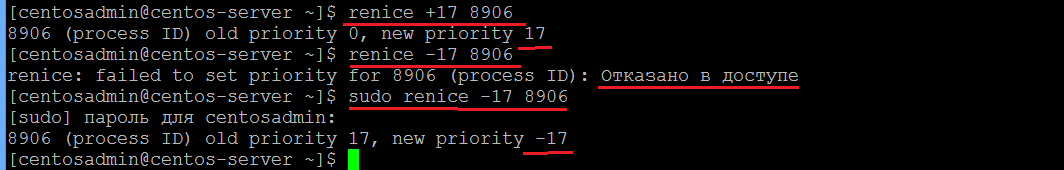


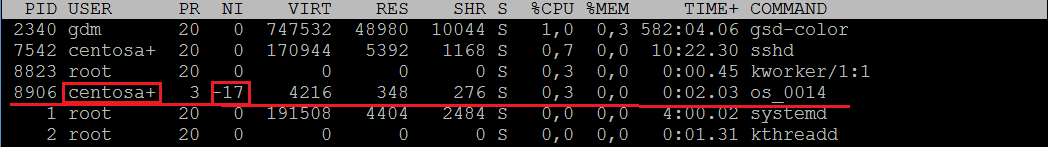




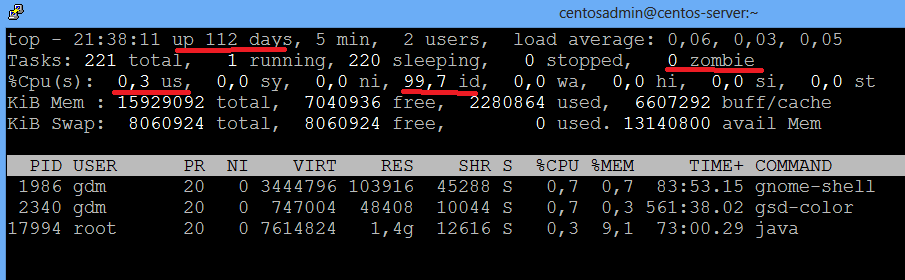


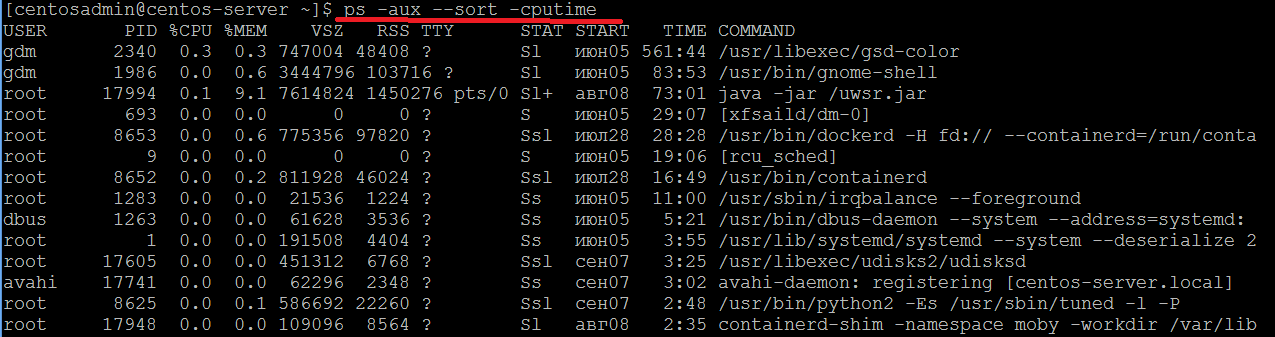
1. **OS:** Linux, top, renice



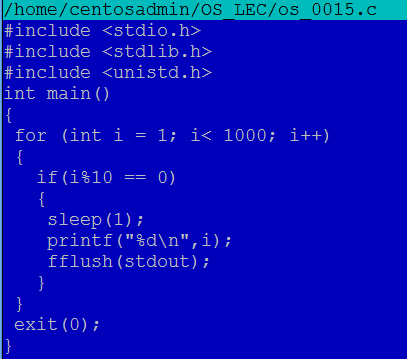


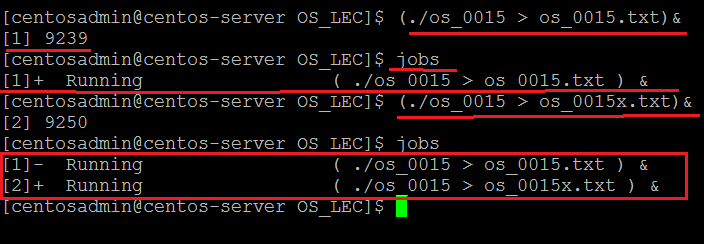
1. **OS:** Linux, потоки в Linux – единица выполнения процесса; ядро Linux не знает ничего о потоках, для ядра поток – это процесс; для ядра нет разницы между двумя изолированными процессами и двумя потоками в рамках одного процесса; два потока в одном процессе для ядра – это два процесса использующих общие ресурсы ядра.
2. **OS:** process scheduler – планировщик процессов в Linux, компонент ядра, выбирает процесс для выполнения, квантует процессорное время, ˜20мс., O(1)-планировщик(shed.c): Линус Торвальдс, 1993 – 254стр., 1996 – 6000стр., 2002 – 7000стр.
3. **OS:** Linux/Centos 7**,** top –i, ps aus –sort -cputime

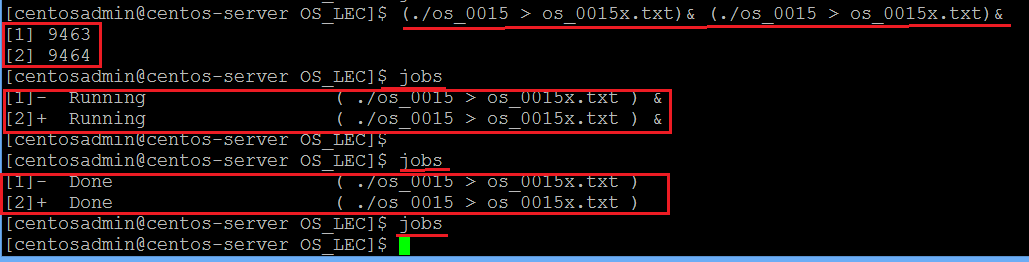


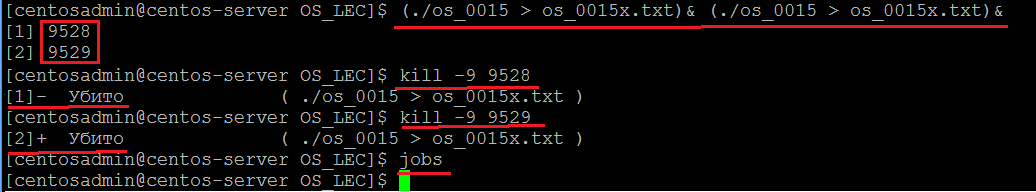


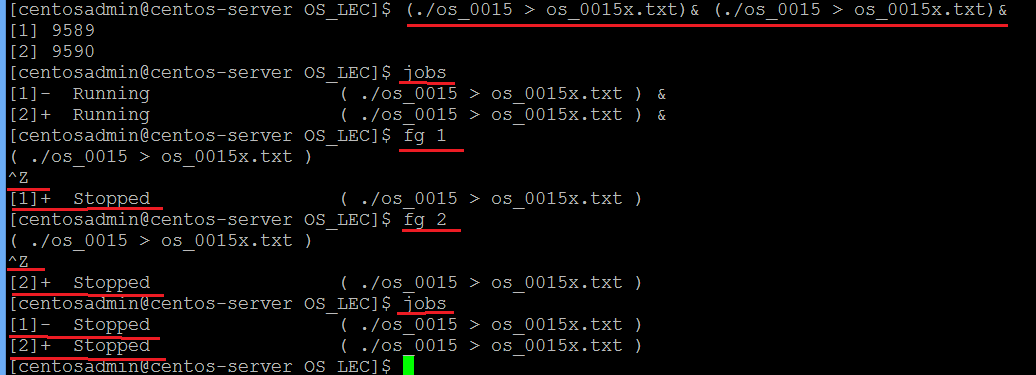
1. **OS:** Linux: &-запуск в фоновом (отсоединенном) режиме, Ctrl+Z, bg, fg.



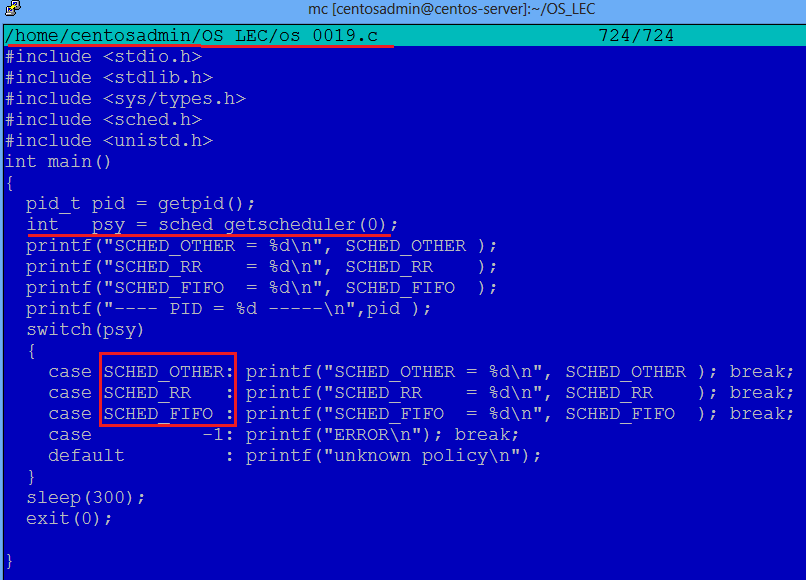


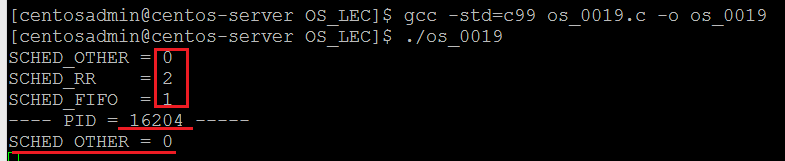


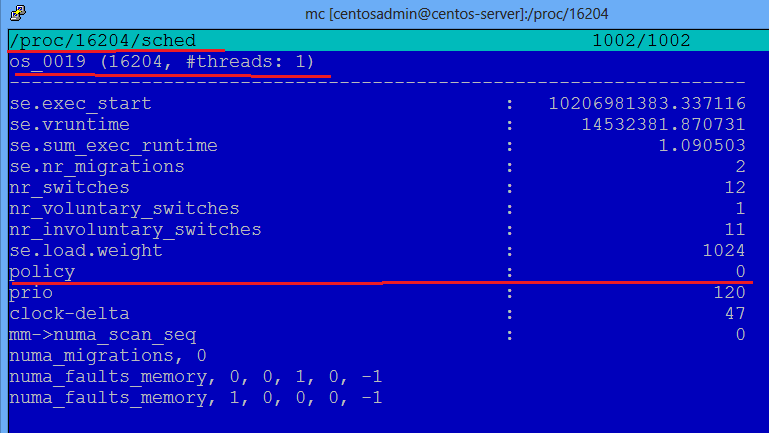




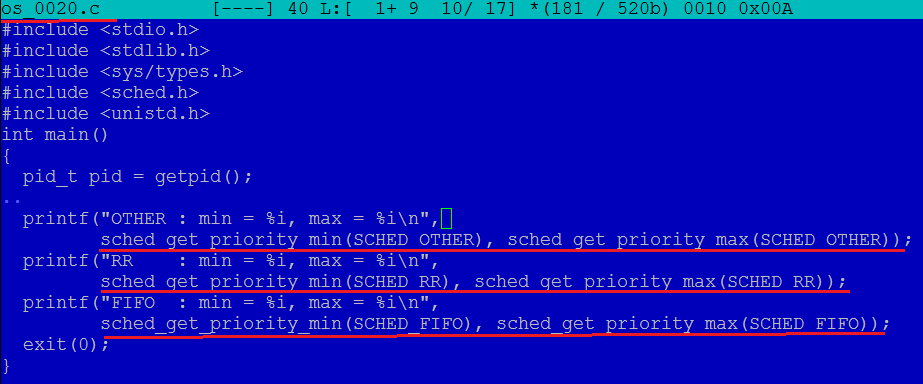
1. **OS:** Linux, политики планирования процесса: 1) стандартная (OTHER, разделения времени); 2) FIFO-политика (FIFO, реального времени); 3) карусельная (round-robin) политика (RR); 4) пакетная политика(BATCH).

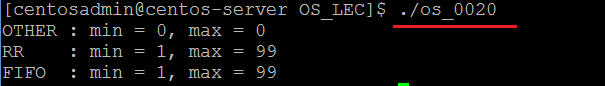


****

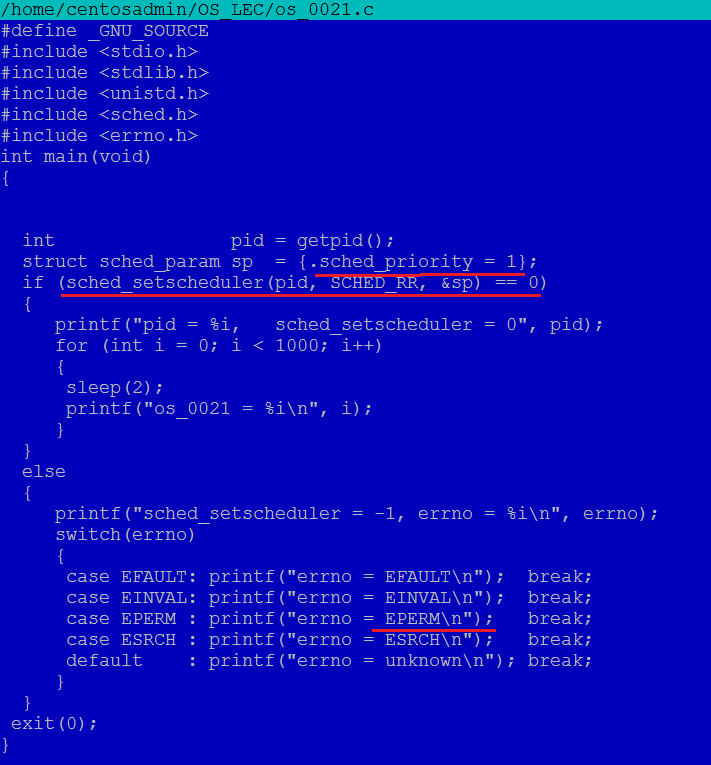


1. **OS:** Linux, политика OTHER(0), статический приоритет равен 0, приоритетность определяются nice. Процессы с политикой FIFO и PR подавляют процессы OTHER.
2. **OS:** Linux, политика FIFO(1), процесс всегда выполняется, если у него наивысший статический приоритет; подавляет OTHER; работает до блокировки или sheld\_yeld(); после блокировки становится в конец своей приоритетной очереди; FIFO-процесс с большим приоритетом подавляет процесс с более низким приоритетом, подавляемый процесс остается на своем месте в очереди; новый процесс становится в начало своей приоритетной очереди и подавляет текщий.
3. **OS:** Linux, политика RR(2), похож на FIFO, выполнение квантуется; после окончания кванта – в конец приоритетной очереди.
4. **OS:** статические приоритеты для политик FIFO, RR, get\_priority\_min, get\_priority\_max

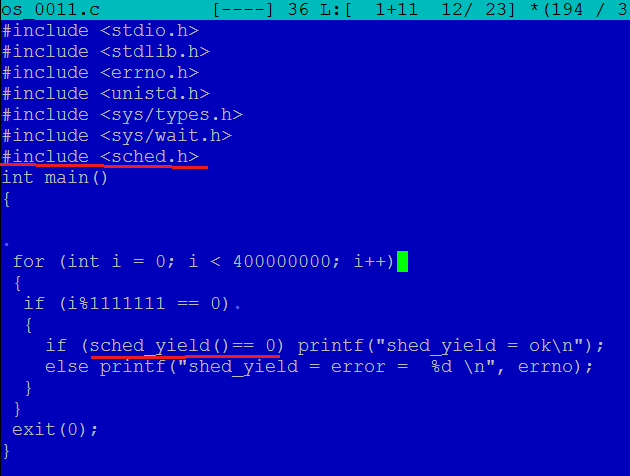


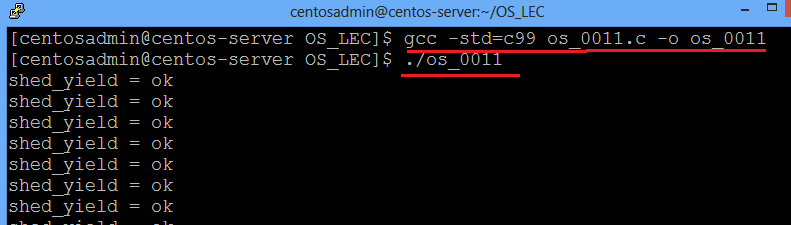
****

1. **OS:** Linux, sched\_setscheduler, EPERM

****

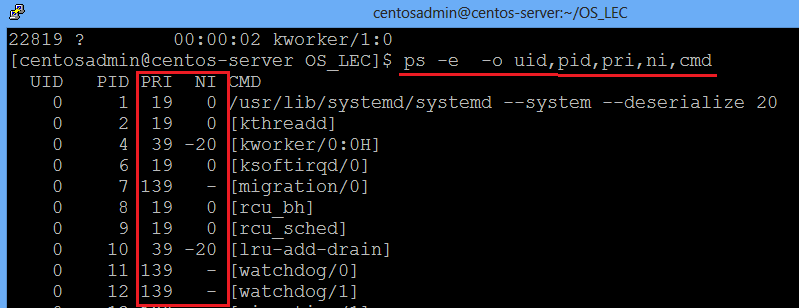
1. **OS:** Linux, политика BATCH, пакетная политика = политика планирования бездействия (idle scheduling policy), выполнятся когда других процессов в OS нет.
2. **OS:**Linux, уступить процессор **shed\_yield(),** эффект ping/pong(аппаратный кэш, L1, L2…)

****

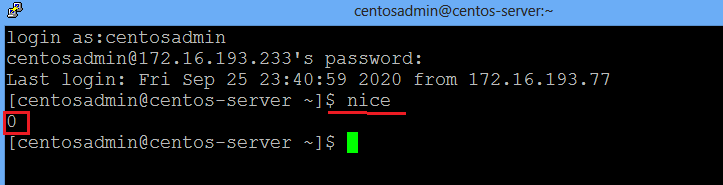


1. **OS:**Linux, при политике OTHER значения приоритетов убывающие:

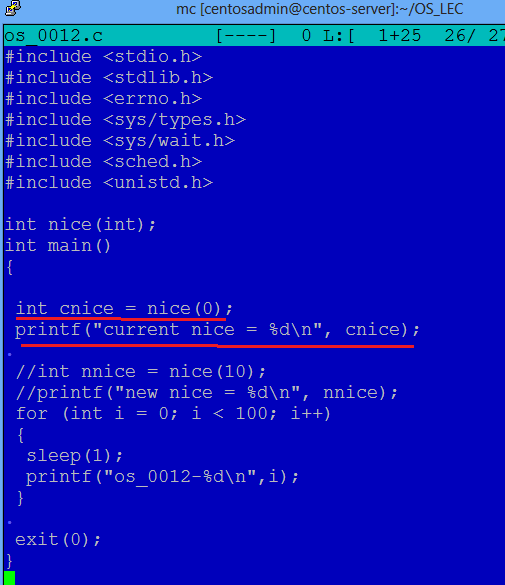
* nice – диапазон приоритетов [-20, 19], default = 0;
* real time – диапазон приоритетов, default =[0,99].

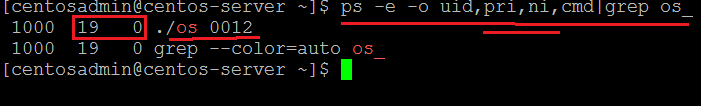


1. **OS:**Linux: nice – значение любезности, чем меньше, тем выше приоритет, значение nice – минимальное значение приоритета (лучшее значение).
2. **OS:** Linux, nice по умолчанию, устанавливается оболочка

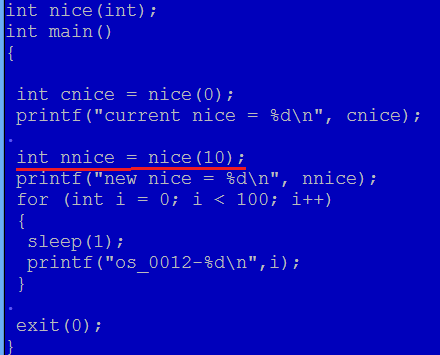


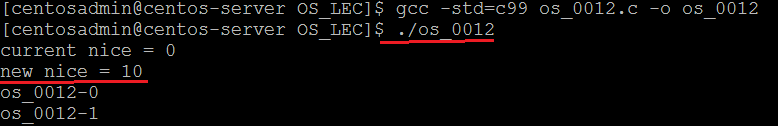
1. **OS:** Linux: функция nice

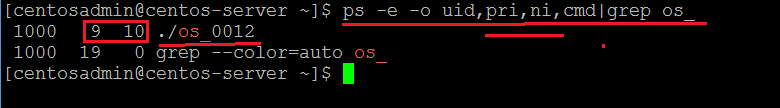


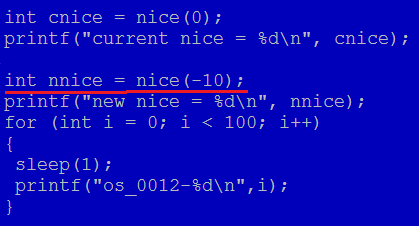


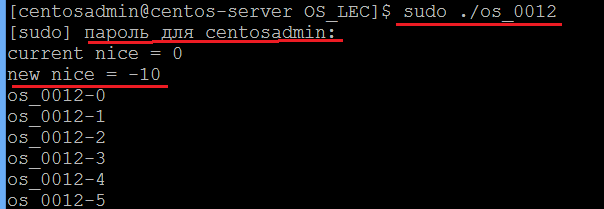
1. **OS:** Linux: nice(10) +=10, nice(-10) +=(-10) – необходима привилегия

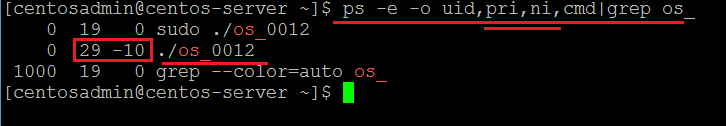




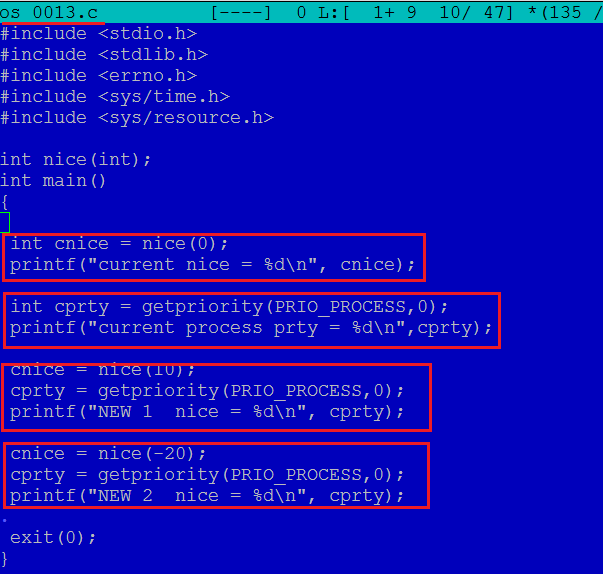


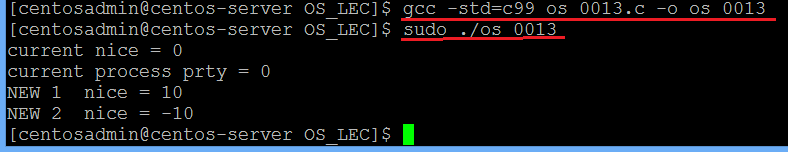


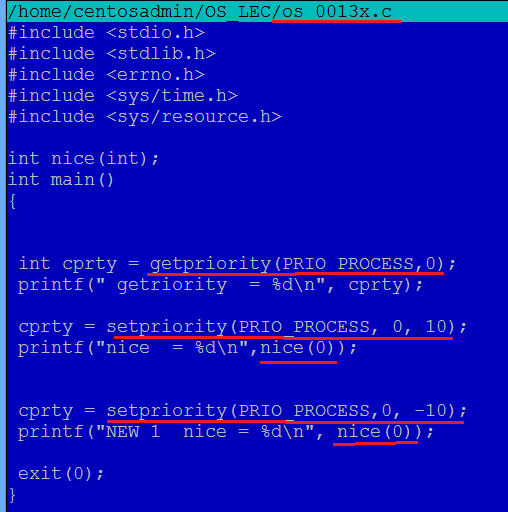


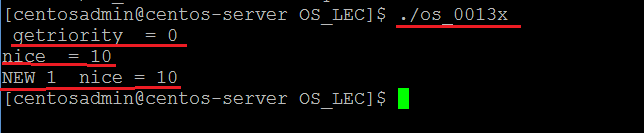


1. **OS:** Linux: nice() , getpriority(), setpriority()

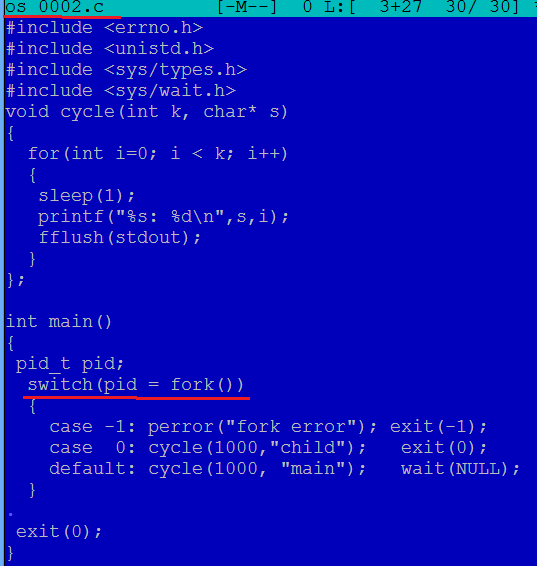


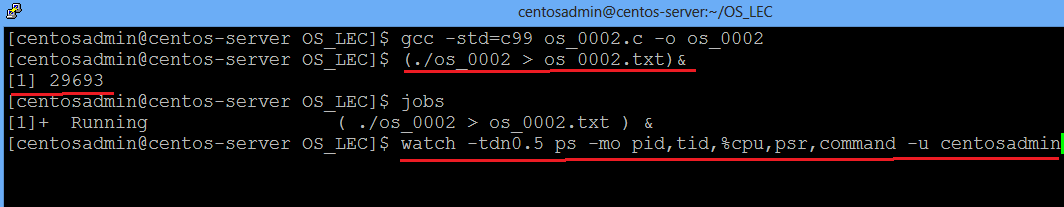


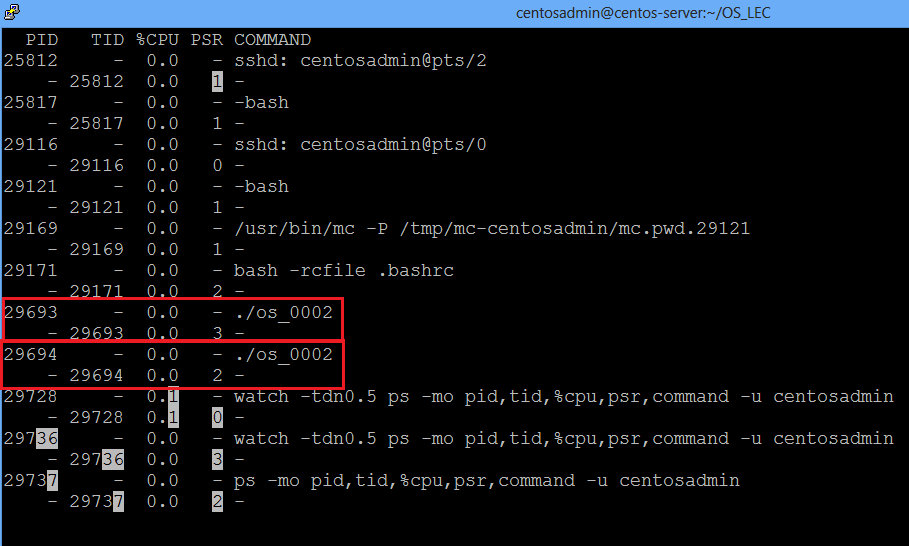


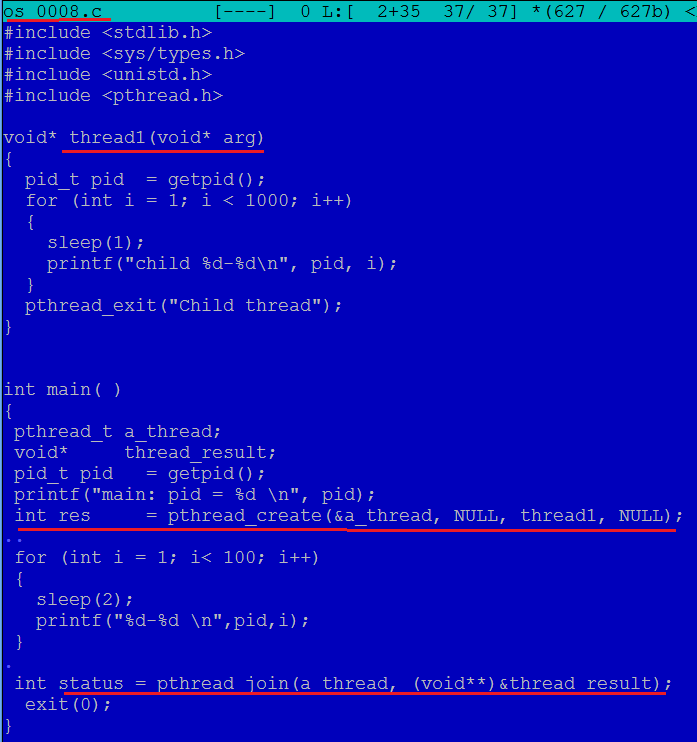


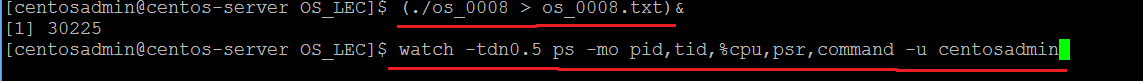
1. **OS:** Linux: привязка процессов(потоков) к процессору.





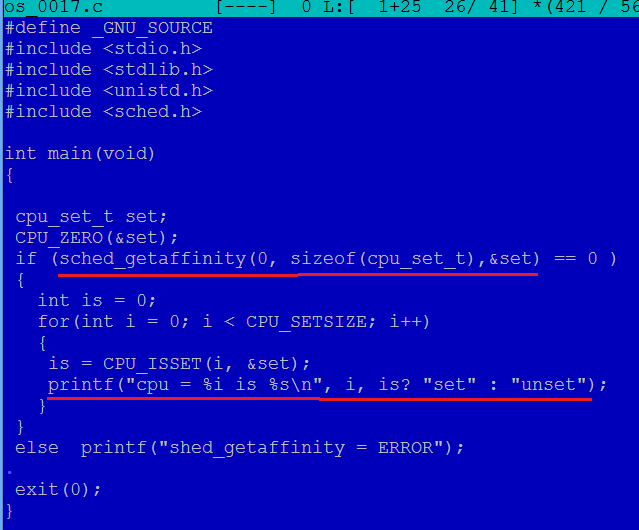


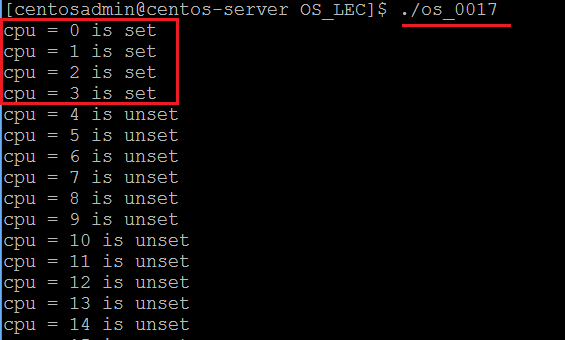


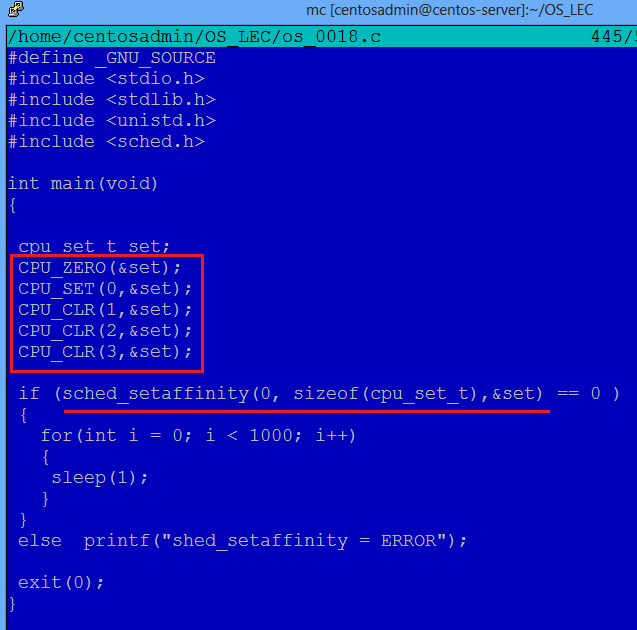


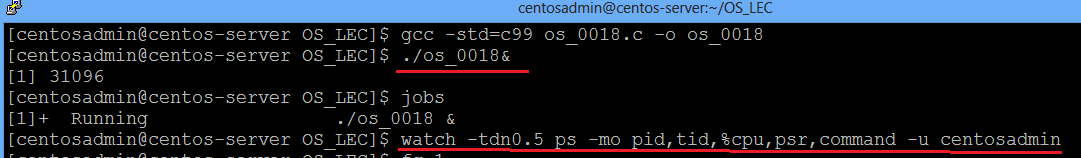


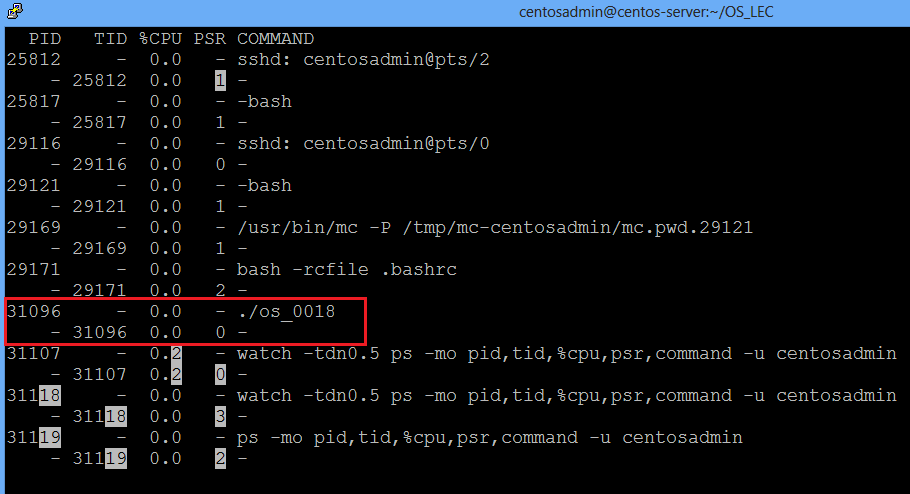
1. **OS:** Linux: процессоры доступные процессору, sched\_getaffinity



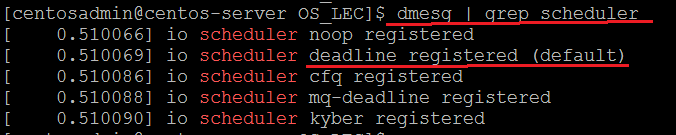
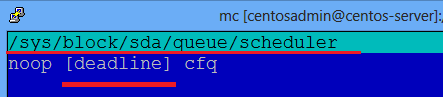




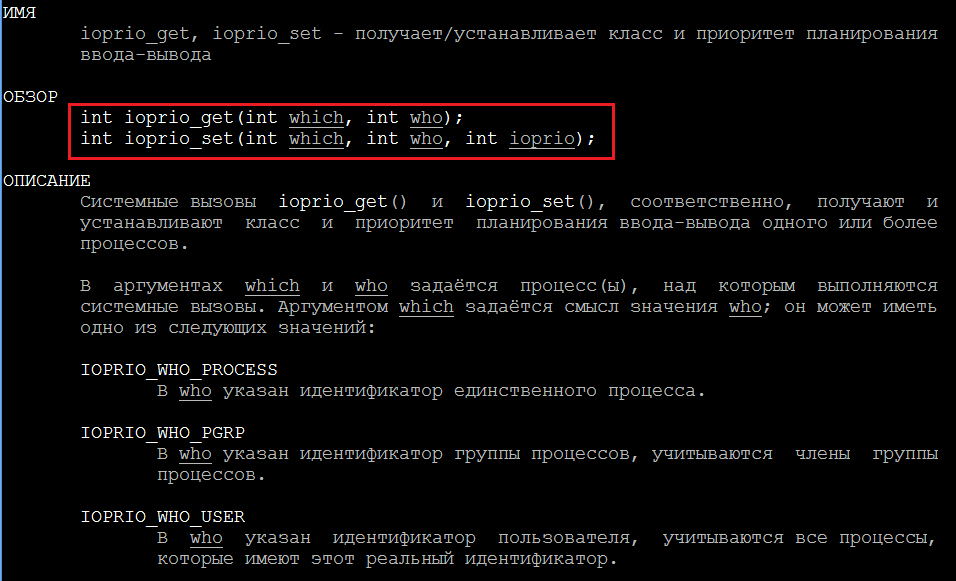


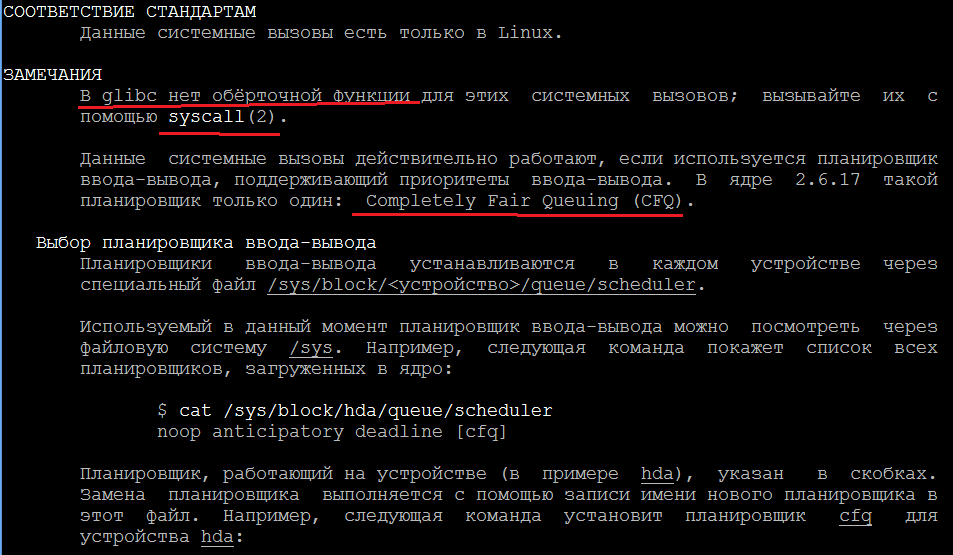


1. **OS:** Linux: планировщики ввода/вывода (I/O) – программная прослойка между блочными устройствами (дисковые устройства) и низкоуровневыми драйверами ввода/вывода. Задача планировщика – оптимизация доступа процесса к дисковому устройству(уменьшение времени поиска данных, обеспечение приоритетности, гарантировать данные за заданное время), лифтовые алгоритмы (elevator) – операции в порядке чтения (записи) ближайших секторов.

1. **OS:** Linux: **NOOP** – простой I/O-планировщик, общая FIFO- очередь read/write-запросов, объединяет однотипные запросы для сокращения операций.
2. **OS:** Linux: **CFQ** (Completely Fair Queueing, наиболее справедливая очередь) – 2003 год, у каждого процесса своя очередь, у каждой очереди свой квант времени, планировщик циклически обходит очереди, обслуживает очередь в течении кванта, чтение в очереди имеет приоритет. Поддержка i/o-приоритов.
3. **OS:** Linux: **Deadline** -2002 год, I/O-планировщик пытается выполнить запрос в указанное время, две очереди read и write, read-очередь приоритетнее, запросы объединяются в пакеты по принципу «алгоритма лифта».
4. **OS:** Linux: **BFQ** (Budget Fair Queueing) –базируется на CFQ, каждой CFQ-очереди выделяется бюджет, который растет для процессов с интенсивным вводом/выводом.
5. **OS:** Linux: **MQ-Deadline** –модификация Deadlineдля новых устройств.
6. **OS:** Linux: **Kyber** - для работы с быстрыми устройствами, две очереди read и write, read-очередь приоритетнее? измеряетcя время завершения каждого запроса и корректирует фактический размер очереди для достижения установленных в настройках задержек.
7. **OS:** Linux: ioprio\_get, ioprio\_set – системные вызовы позволяющие получать и изменять i/o-приоритеты.
8. **OS:** Linux: **man ioprio\_set**





1. **OS:** Linux:
2. Ии
3. **ПРОДОЛЖЕНИЕ**
4. https://habr.com/ru/company/clrium/blog/488260/