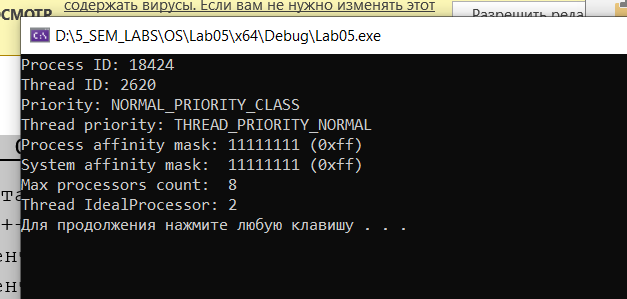
**Задание 01**

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

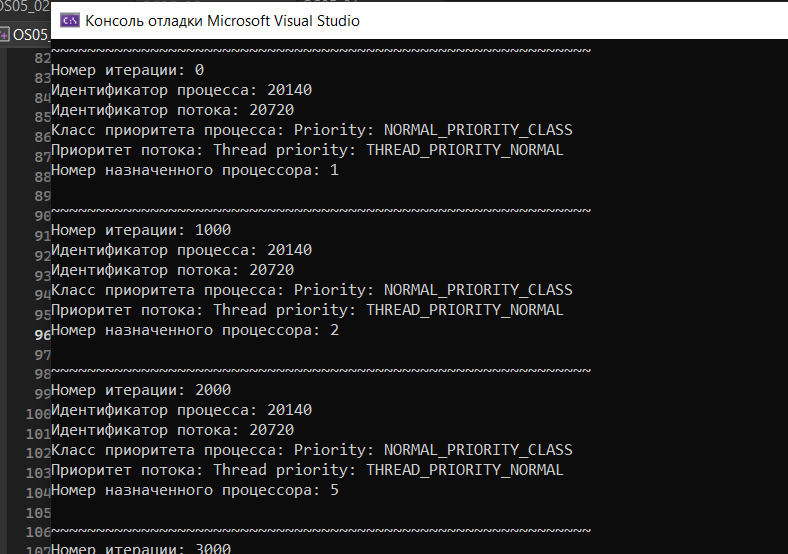
* идентификатор текущего процесса;
* идентификатор текущего (main) потока;
* приоритет (приоритетный класс) текущего процесса;
* приоритет текущего потока;
* маску (affinity mask) доступных процессу процессоров в двоичном виде;
* количество процессоров доступных процессу;
* процессор, назначенный текущему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



**Задание 02**

1. Разработайте консольное Windows-приложение **OS05\_02x,** выполняющее цикл в 1млн итераций.
2. Каждая итерация осуществляет задержку на 200 мс через каждые 1тыс итераций и выводит следующую информацию:

* номер итерации;
* идентификатор процесса;
* идентификатор потока;
* класс приоритета процесса;
* приоритет потока:
* номер назначенного процессора.



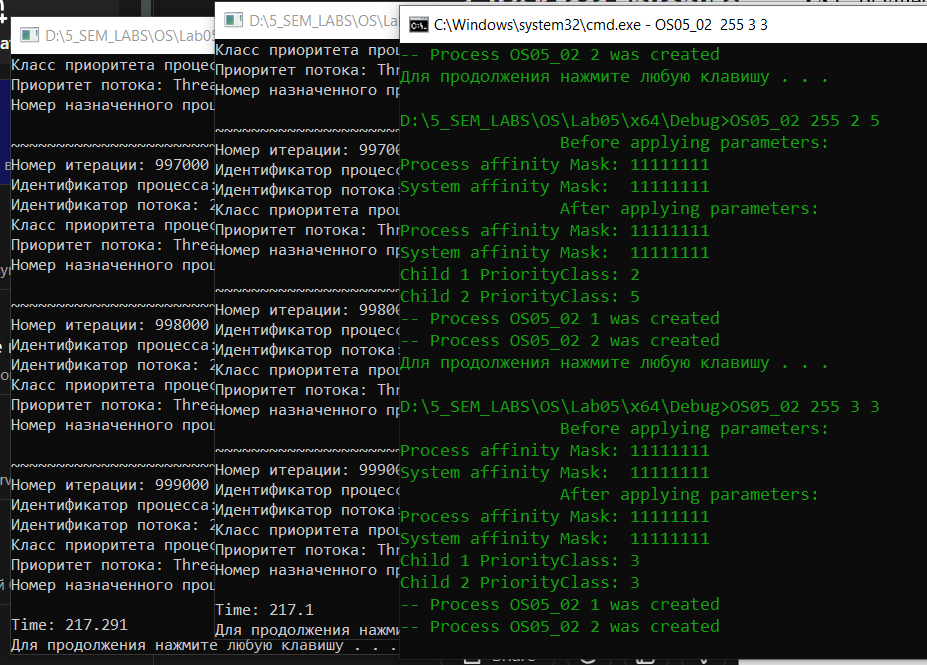
1. Разработайте консольное Windows-приложение **OS05\_02,** принимающее следующие параметры:

* P1: целое число, задающее маску доступности процессоров (affinity mask);
* P2: целое число, задающее класс приоритета первого дочернего процесса;
* P3: целое число, задающее класс приоритета второго дочернего процесса.

1. Приложение **OS05\_02** должно вывести в свое консольное окно заданные параметры и запустить два одинаковых дочерних процесса **OS05\_02x,** осуществляющих вывод в отдельные консольные окна и имеющих заданные в параметрах приоритеты.
2. Запустите приложение **OS05\_02**,принимающее следующие значения параметров:

* P1: доступны все процессоры;
* P2: Normal;
* P3: Normal.

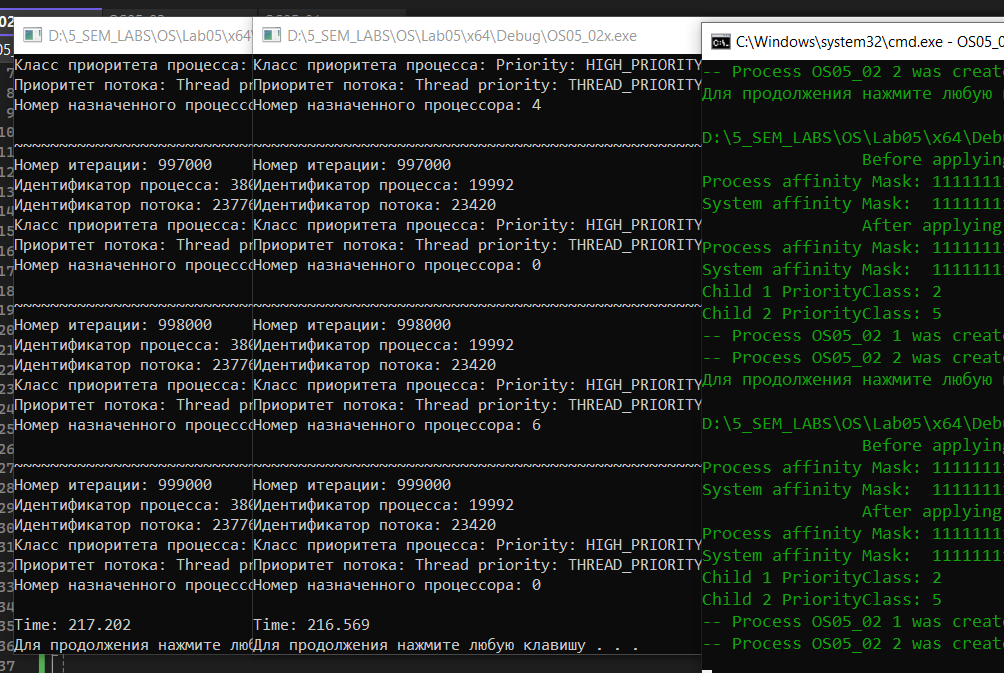
Зафиксируйте (скриншот) в момент первого окончания одного из дочерних процессов расхождение в количестве выполненных процессами итераций.



1. Запустите приложение **OS05\_02**,принимающее следующие значения параметров:

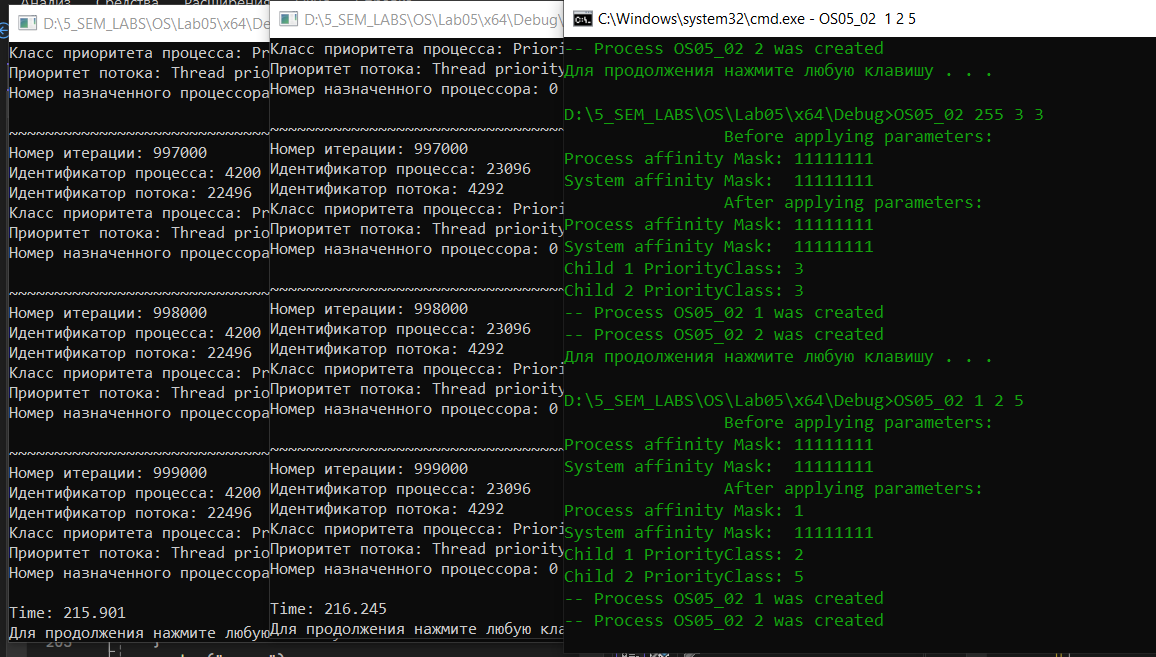
* P1: доступны все процессоры;
* P2: Below Normal;
* P3: High.

Зафиксируйте (скриншот) в момент первого окончания одного из дочерних процессов расхождение в количестве выполненных процессами итераций.



1. Запустите приложение **OS05\_02**,принимающее следующие значения параметров:

* P1: доступен один процессор;
* P2: Below Normal;
* P3: High.



Зафиксируйте (скриншот) в момент первого окончания одного из дочерних процессов расхождение в количестве выполненных процессами итераций.

1. По зафиксированным скриншотам, объясните полученные результаты.

**Задание 03**

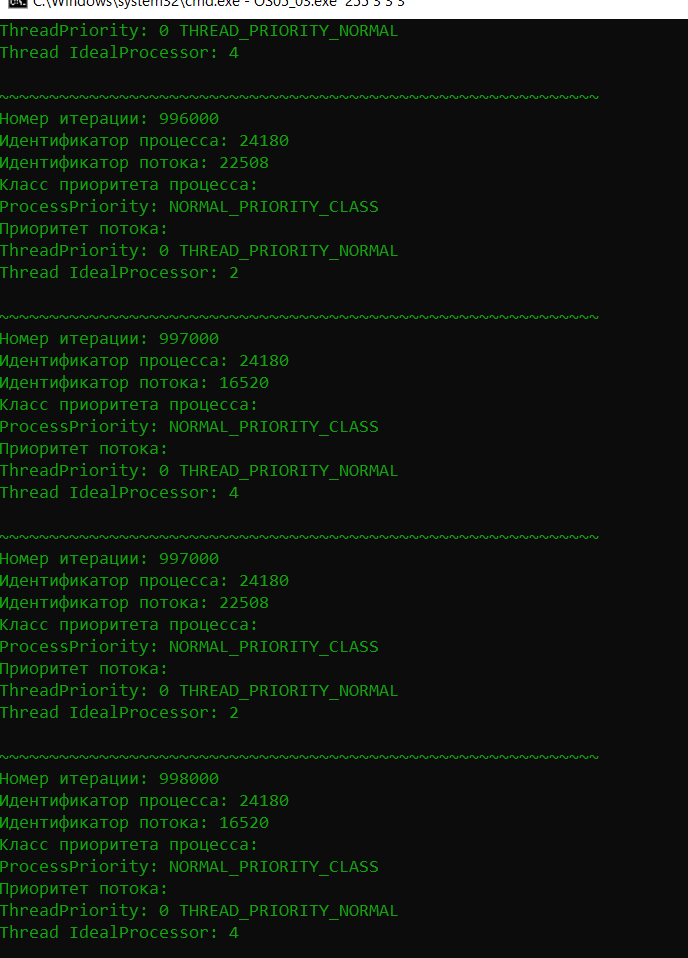
1. Разработайте консольное Windows-приложение **OS05\_03,** принимающее следующие параметры:

* P1: целое число, задающее маску доступности процессоров (affinity mask);
* P2: целое число, задающее класс приоритет процесса;
* P3: целое число, задающее приоритет первого дочернего потока;
* P4: целое число, задающее приоритет второго дочернего потока.

1. Приложение **OS05\_03**  включает в себя потоковую функцию **TA**, выполняющую цикл в 1млн итераций, аналогичный циклу в задании 02.
2. Приложение **OS05\_03** должно вывести в свое консольное окно заданные параметры и запустить два одинаковых дочерних потока (потоковая функция **TA**)**,** осуществляющих вывод консольное окно и имеющих заданные в параметрах приоритеты.
3. Запустите приложение **OS05\_03**,принимающее следующие значения параметров:

* P1: доступны все процессоры;
* P2: Normal;
* P3: Normal;
* P4: Normal;

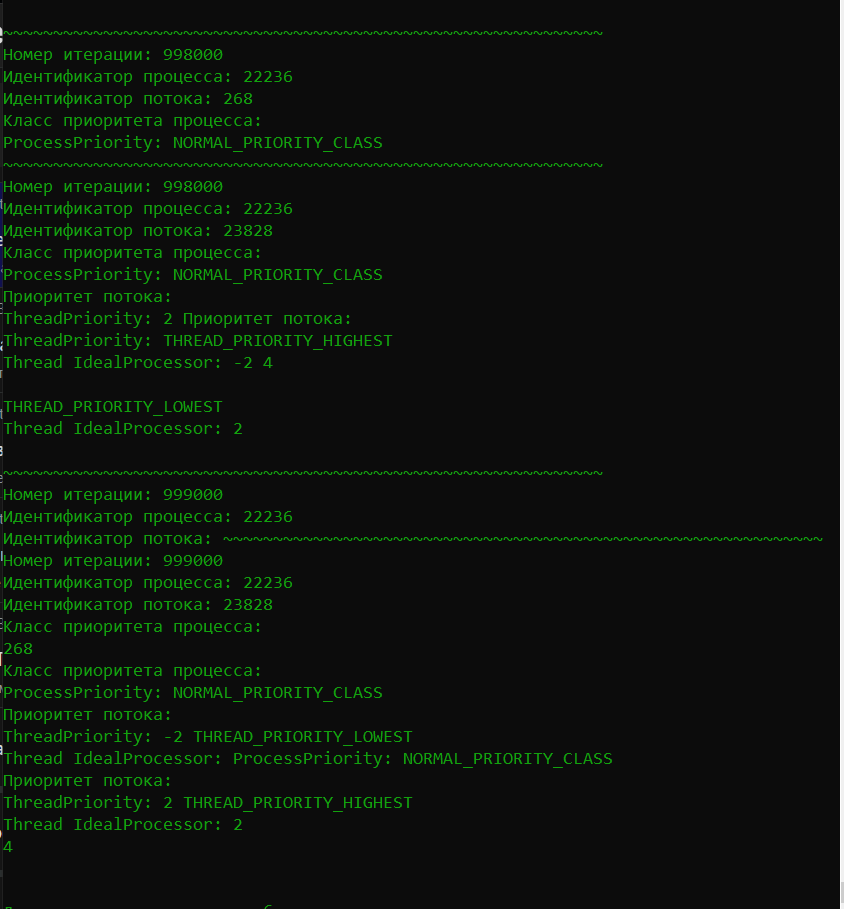
Зафиксируйте (скриншот) в момент первого окончания одного из дочерних потоков расхождение в количестве выполненных потоками итераций.



1. Запустите приложение **OS05\_03**,принимающее следующие значения параметров:

* P1: доступны все процессоры;
* P2: Normal;
* P3: Lowest;
* P4: Highest.

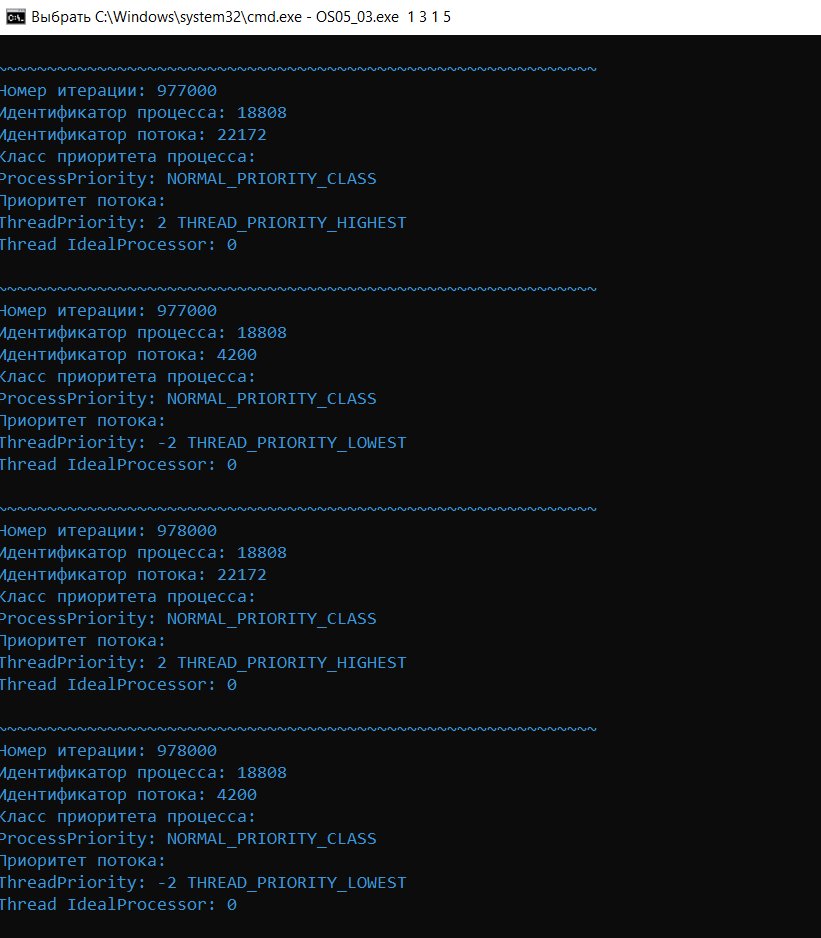
Зафиксируйте (скриншот) в момент первого окончания одного из дочерних потоков расхождение в количестве выполненных потоками итераций.



1. Запустите приложение **OS05\_03**,принимающее следующие значения параметров:

* P1: доступен один процессор;
* P2: Normal;
* P3: Lowest;
* P4: Highest.

Зафиксируйте (скриншот) в момент первого окончания одного из дочерних потоков расхождение в количестве выполненных потоками итераций.



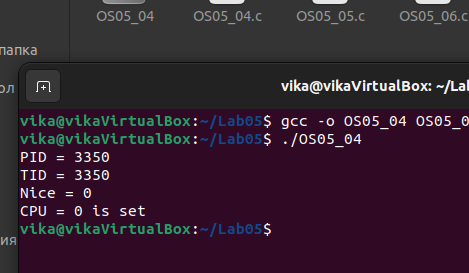
1. По зафиксированным скриншотам объясните полученные результаты.

**Задание 04**

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

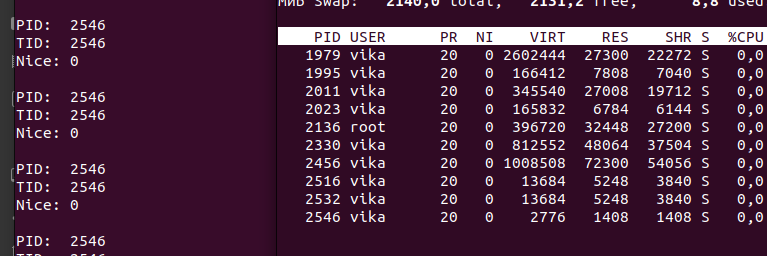
* идентификатор текущего процесса;
* идентификатор текущего (main) потока;
* приоритет (nice) текущего потока;
* номера доступных процессоров.

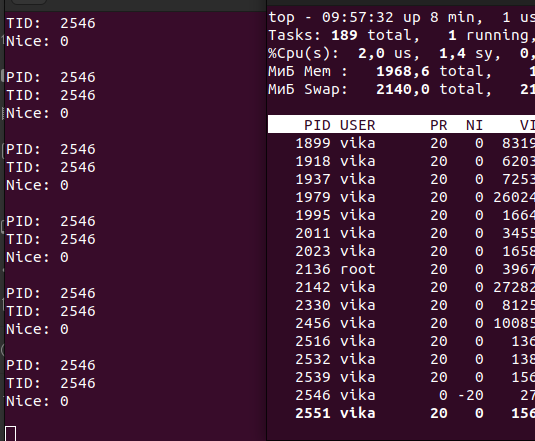
nice -n -10 ./OS05 OR PID sudo renice -20 -p pid

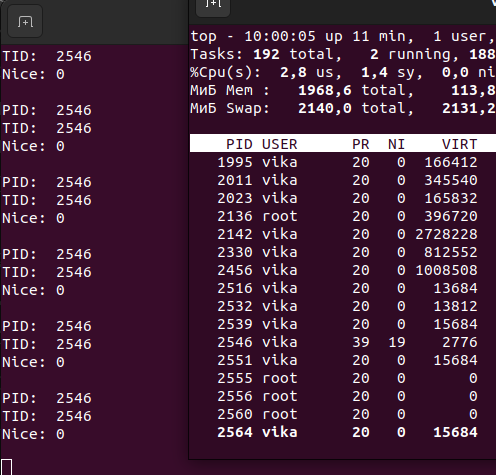


**Задание 05**

1. Разработайте консольное Linux-приложение **OS05\_05** на языке С, выполняющее длинный цикл.
2. Запустите приложение **OS05\_05.**
3. Зафиксируйте (скриншот) текущее значение **nicе**, полученное с помощью команды **top**.
4. Увеличьте приоритет для **OS05\_05** до максимального значения (самого привилегированного). Зафиксируйте (скриншот) текущее значение **nicе**, полученное с помощью команды **top**.
5. Уменьшите приоритет для **OS05\_05** до минимального значения (самого ничтожного). Зафиксируйте (скриншот) текущее значение **nicе**, полученное с помощью команды **top**

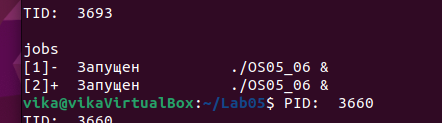


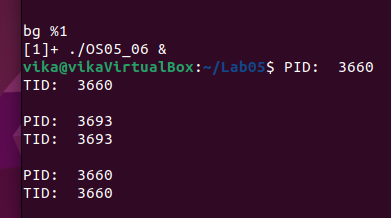




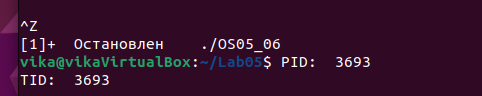
**Задание 06**

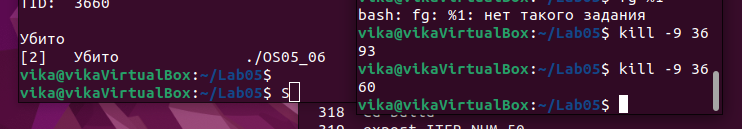
1. Разработайте консольное Linux-приложение **OS05\_06** на языке С, выполняющее длинный цикл с задержкой в 1сек в каждой итерации.
2. Продемонстрируйте запуск нескольких приложения **OS05\_06** в фоновом режиме, и команды bg, fg, jobs, Ctrl+Z, kill -9
3. Ctrl+Z: Stop process
4. bg: Move process to background
5. fg [%n]: Move process [%n] to foreground
6. jobs: All processes in this bash console
7. kill -9: Terminate process
8. <cmd> &: Start command in bg









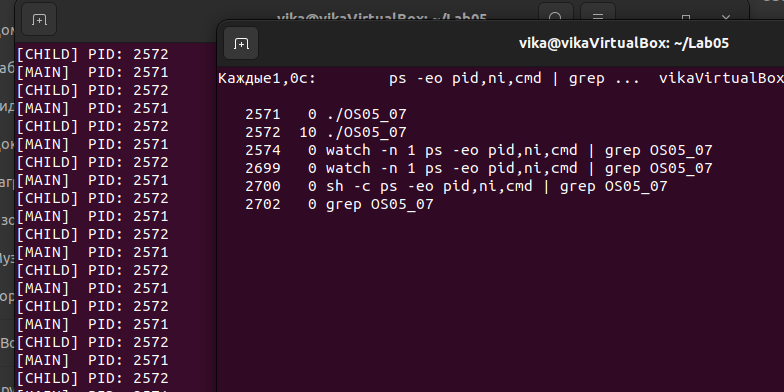


**Задание 07**

1. Разработайте консольное Linux-приложение **OS05\_07** на языке С, выполняющее длинный цикл с задержкой в 1сек в каждой итерации. Приложение с помощью **OS05\_07** системного вызова **fork** вызывает дочерний поток который понижает свой приоритет на 10.

С помощью команды **watch ps** продемонстрируйте работу этих потоков и их значение **nice**.

watch -n 1 "ps -eo pid,ni,cmd | grep OS05\_07"



cat /sys/block/<device>/queue/scheduler

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 SetThreadIdealProcessor.

Функция SetThreadIdealProcessor в Windows используется для установки предпочитаемого процессора для потока. Это позволяет программе указать, на каком процессоре должен выполняться поток. Это может быть полезно для оптимизации производительности в многопроцессорных системах.

DWORD SetThreadIdealProcessor(

HANDLE hThread, // дескриптор потока

DWORD dwIdealProcessor // номер идеального процессора

);

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 – значение любезности, 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: как выяснить действующую политику планирования для процесса с помощью файловой системы proc?

Вы можете узнать текущую политику планирования для процесса, просмотрев файл **/proc/[PID]/sched** или **/proc/[PID]/status**, где **[PID]** - это идентификатор процесса.

0- OTHER

1-FIFO

2-RR

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

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

1. Linux: чем отличается системный вызов nice от вызова setpriority

Системный вызов nice используется для изменения приоритета процесса и принимает значение от -20 (высший приоритет) до +19 (низший приоритет). Системный вызов setpriority используется для установки приоритета процесса и потока и принимает в качестве аргументов идентификаторы процесса и потока, а также значения приоритета, включая их диапазон.

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

Планировщик ввода/вывода (I/O scheduler) в Linux отвечает за управление доступом к блоковым устройствам, таким как жесткие диски. Он определяет порядок выполнения операций ввода/вывода, чтобы оптимизировать производительность и уменьшить фрагментацию данных.

???

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

Некоторые известные планировщики ввода/вывода в Linux:

NOOP – простой I/O-планировщик, общая FIFO- очередь read/write-запросов, объединяет однотипные запросы для сокращения операций.

CFQ (Completely Fair Queueing, наиболее справедливая очередь) – 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-приоритетов.

BFQ (Budget Fair Queueing) – базируется на CFQ, каждой CFQ-очереди выделяется бюджет, который растет для процессов с интенсивным вводом/выводом.

Deadline - 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».

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

Тип планировщика, используемого для конкретного блокового устройства, можно узнать, просмотрев содержимое файла /sys/block/[device]/queue/scheduler, где [device] - это имя блокового устройства.