Лабораторная работа 05

OC, ПОИТ-3

**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:

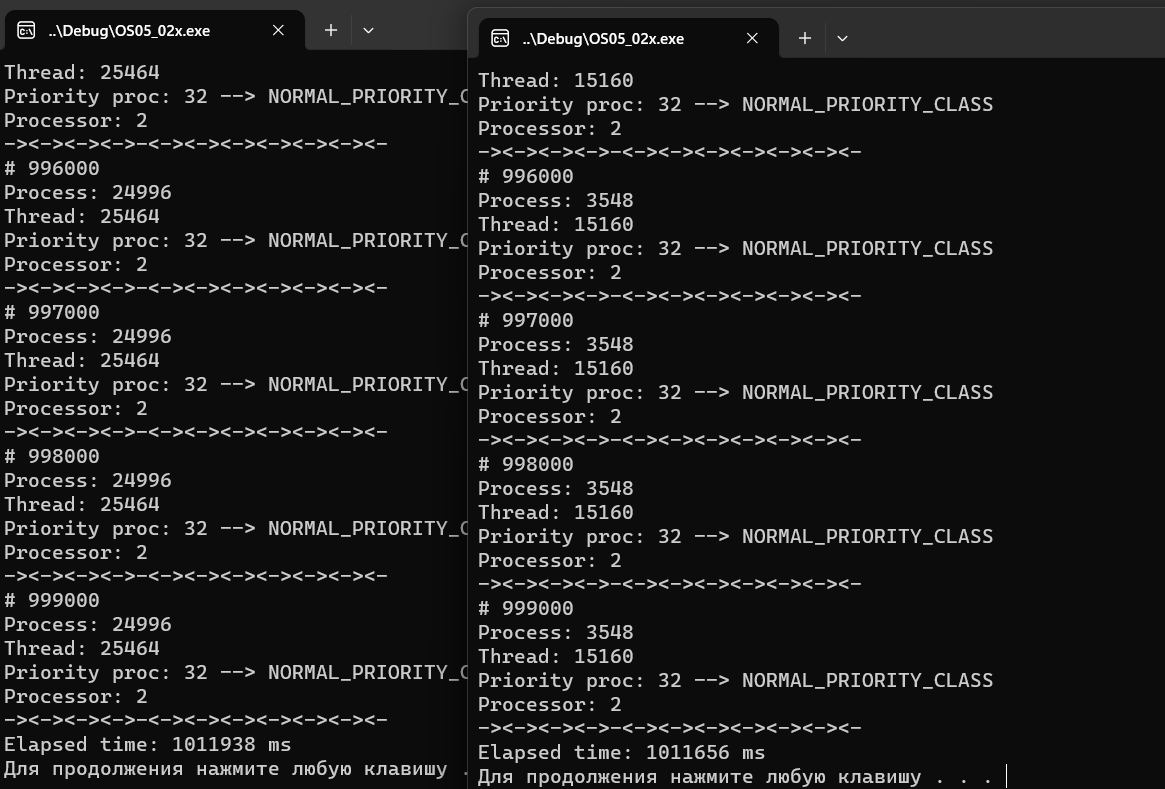
* P1: целое число, задающее маску доступности процессоров (affinity mask);
* P2: целое число, задающее класс приоритета первого дочернего процесса;
* P3: целое число, задающее класс приоритета второго дочернего процесса.

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.

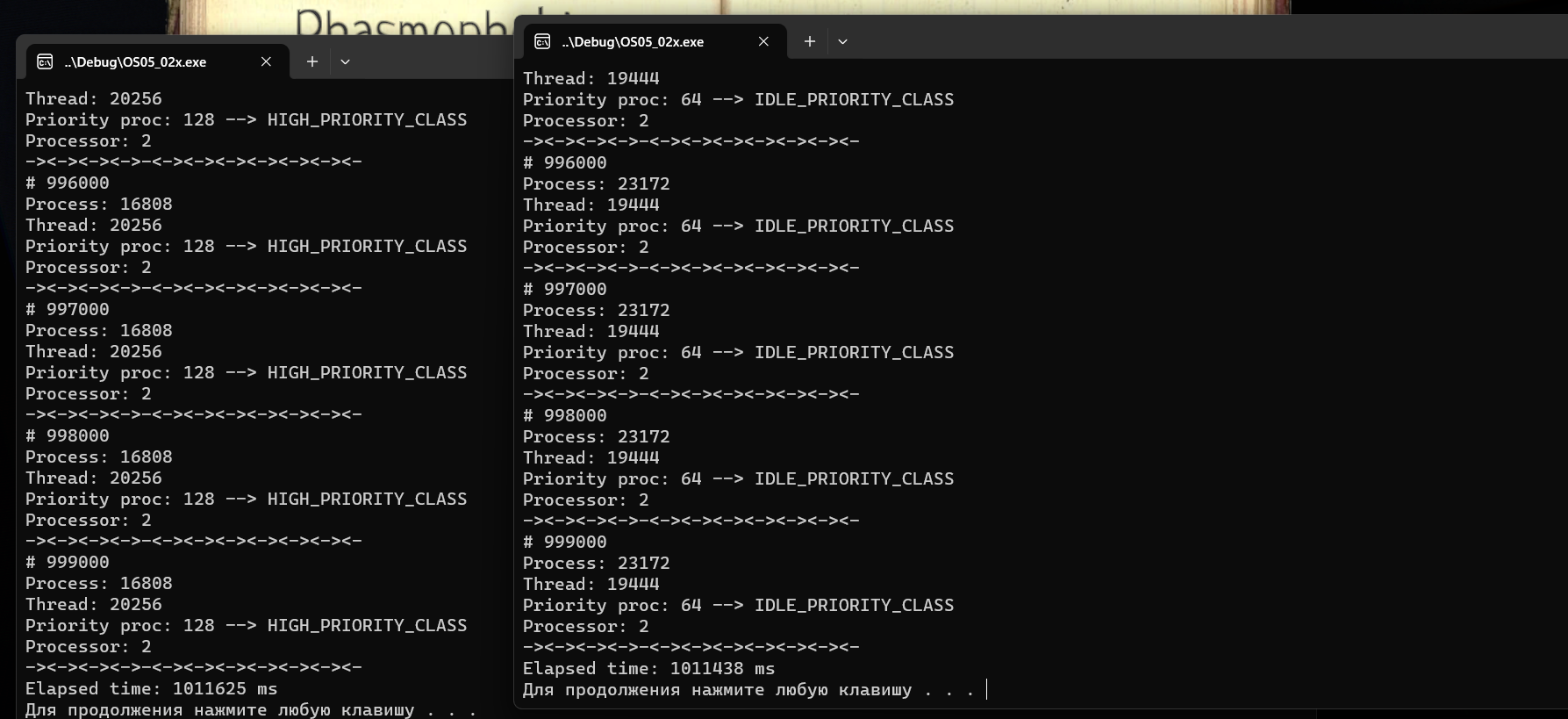
7 3 3



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.

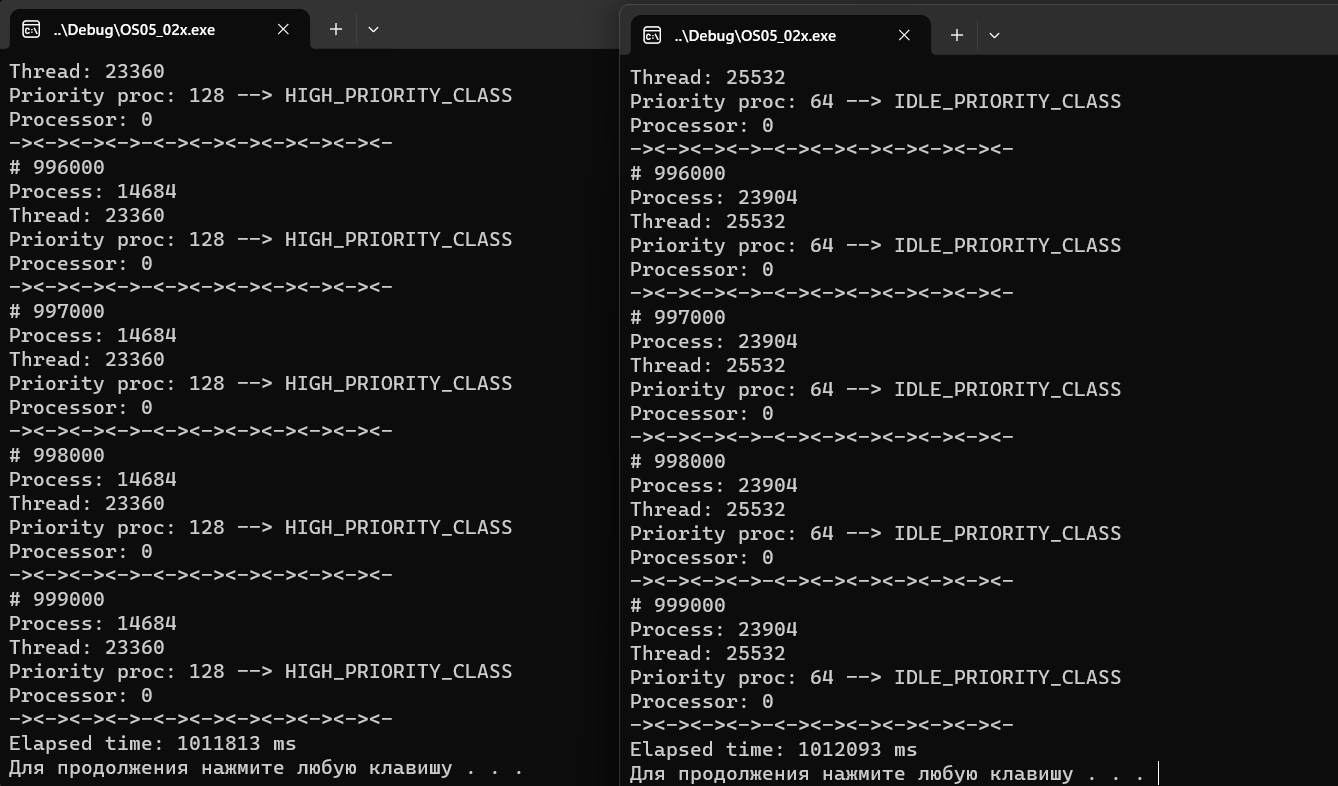


1. Запустите приложение **OS05\_02**,принимающее следующие значения параметров:

* 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 2 5



1. По зафиксированным скриншотам, объясните полученные результаты.

**Задание 03**

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

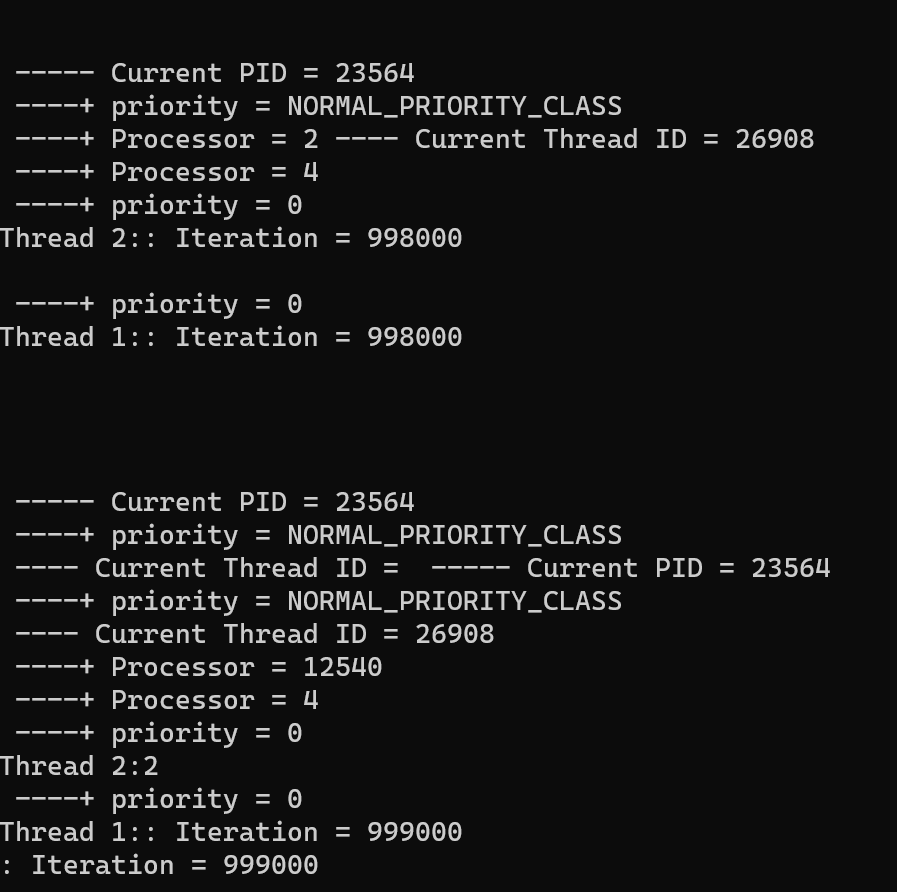
* P1: целое число, задающее маску доступности процессоров (affinity mask);
* P2: целое число, задающее класс приоритет процесса;
* P3: целое число, задающее приоритет первого дочернего потока;
* P4: целое число, задающее приоритет второго дочернего потока.

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.

Max 2 4 4

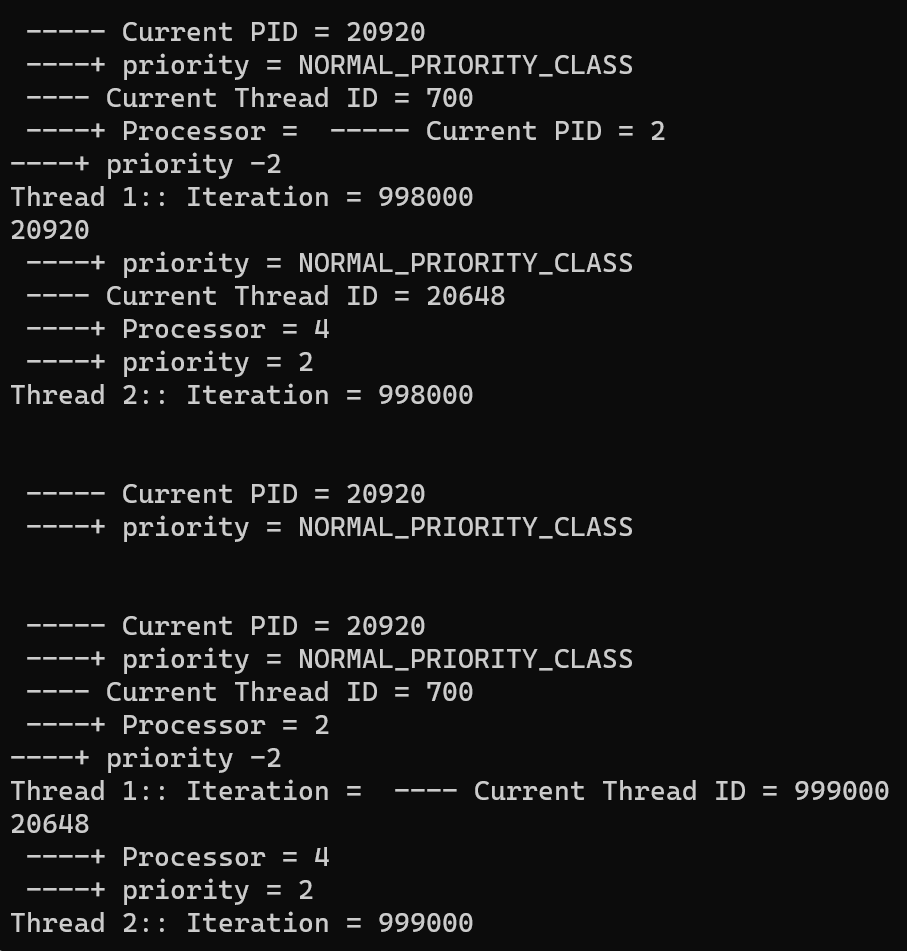


1. Запустите приложение **OS05\_03**,принимающее следующие значения параметров:

* 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.

Max 2 2 6



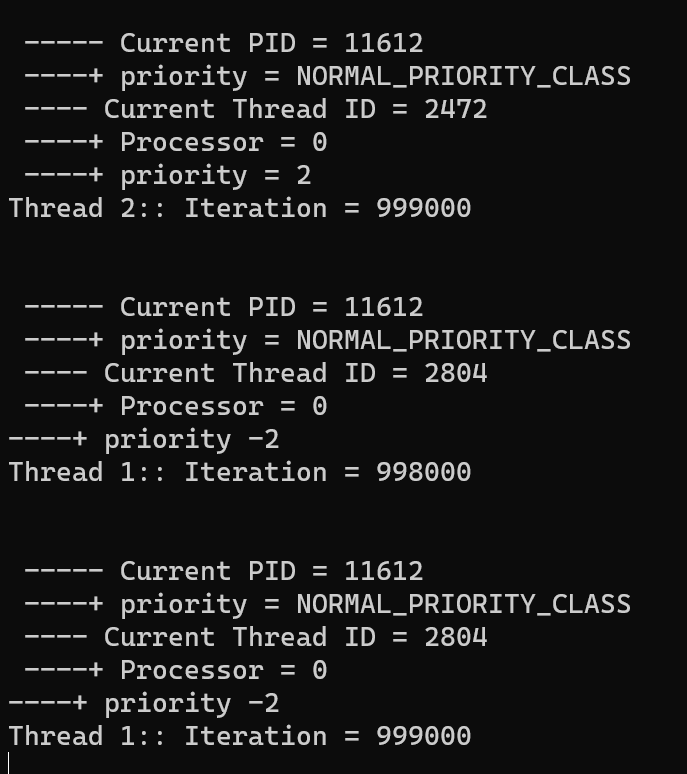
1. Запустите приложение **OS05\_03**,принимающее следующие значения параметров:

* 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.

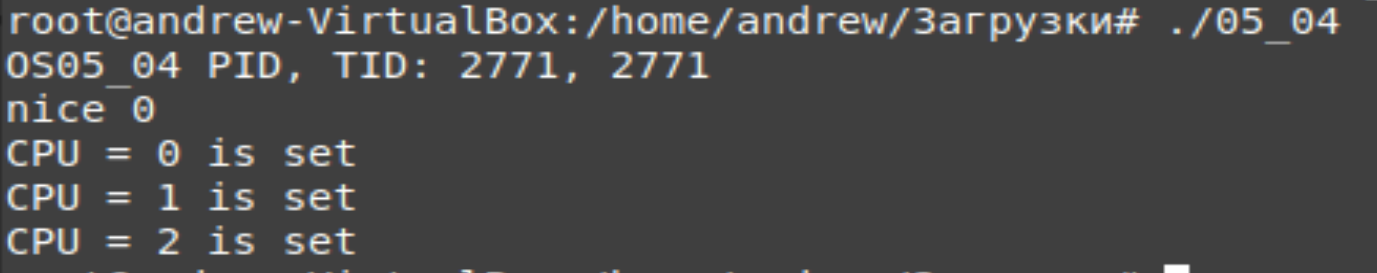
1 2 2 6



**Задание 04**

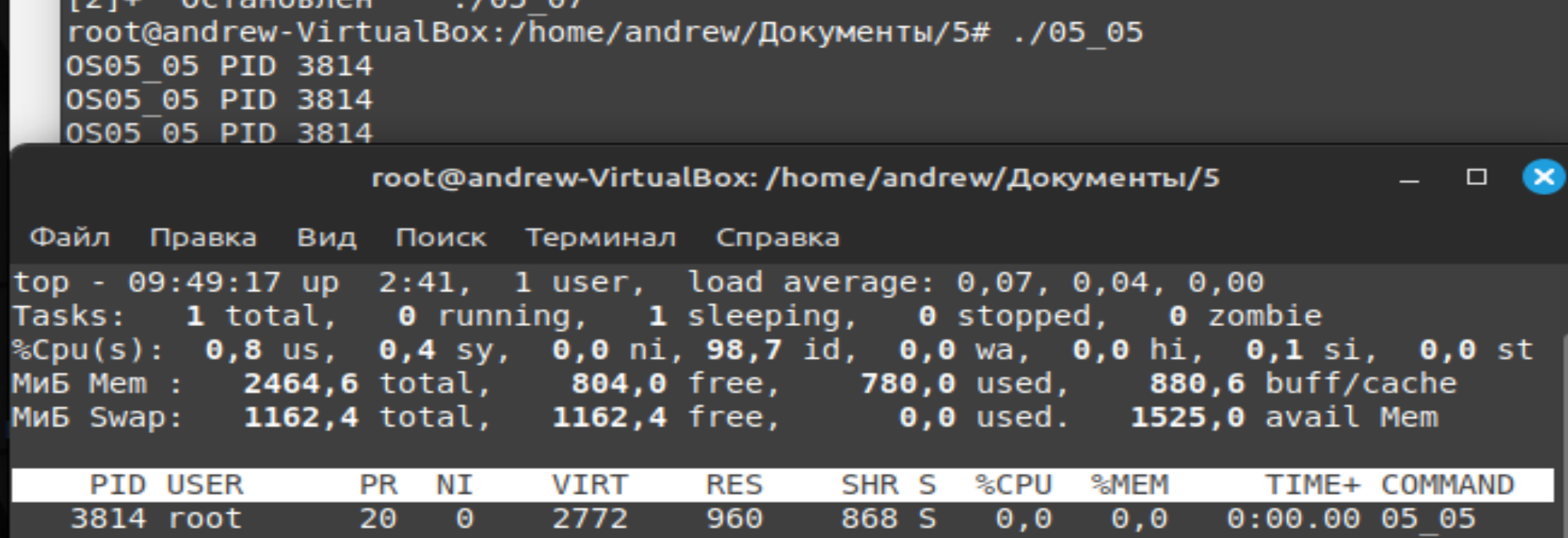
1. Разработайте консольное Linux-приложение **OS05\_04** на языке С++, выводящее на консоль следующую информации:

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

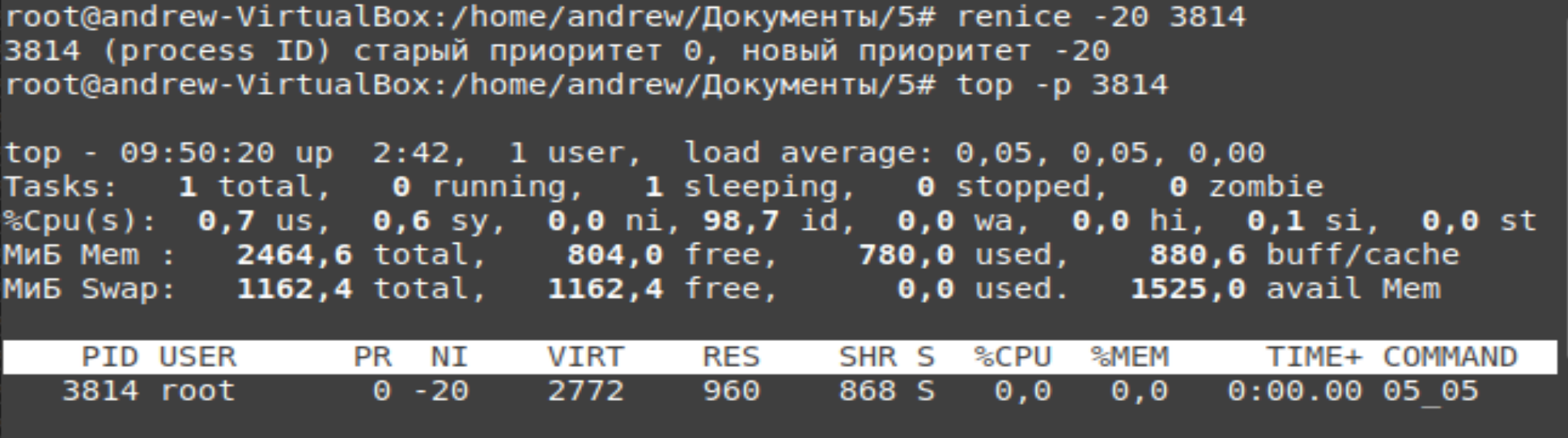


**Задание 05**

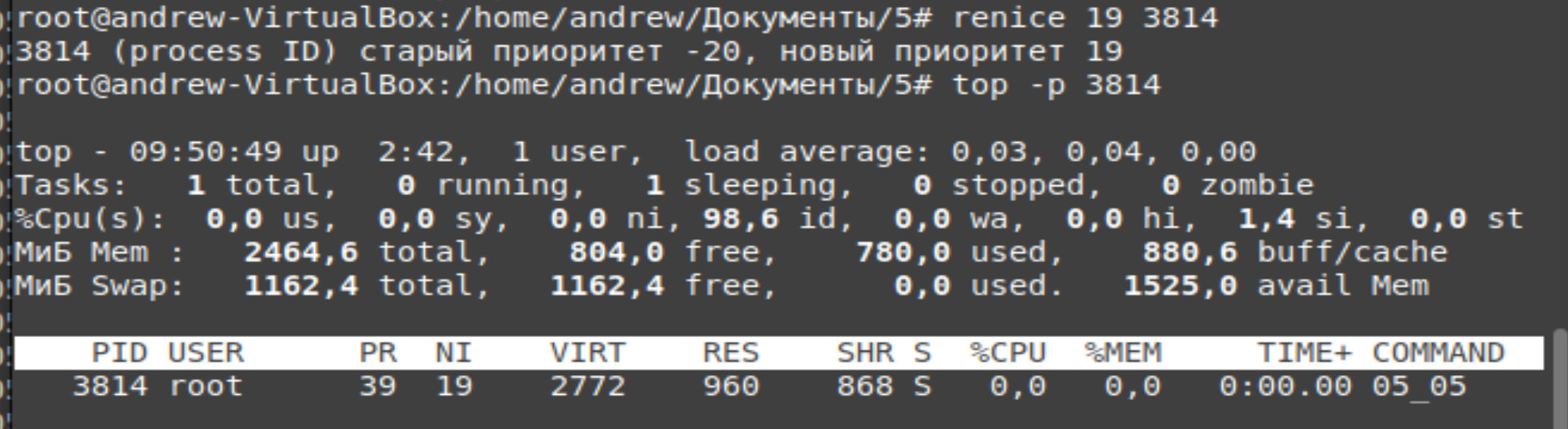
1. Разработайте консольное Linux-приложение **OS05\_05** на языке С, выполняющее длинный цикл.
2. Запустите приложение **OS05\_05.**
3. Зафиксируйте (скриншот) текущее значение **nicе**, полученное с помощью команды **top**.



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

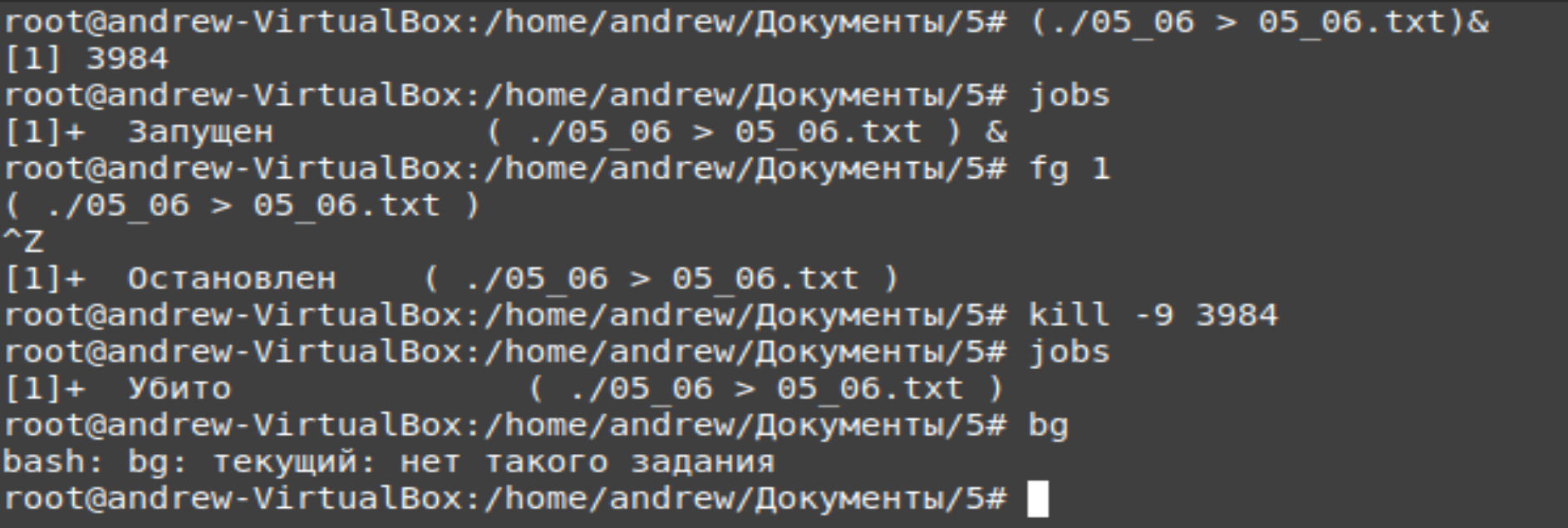


1. Уменьшите приоритет для **OS05\_05** до минимального значения (самого ничтожного). Зафиксируйте (скриншот) текущее значение **nicе**, полученное с помощью команды **top**



**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

bg: Запустит последнюю приостановленную фоновую задачу в фоновом режиме.

fg: Переместит фоновую задачу в передний план.

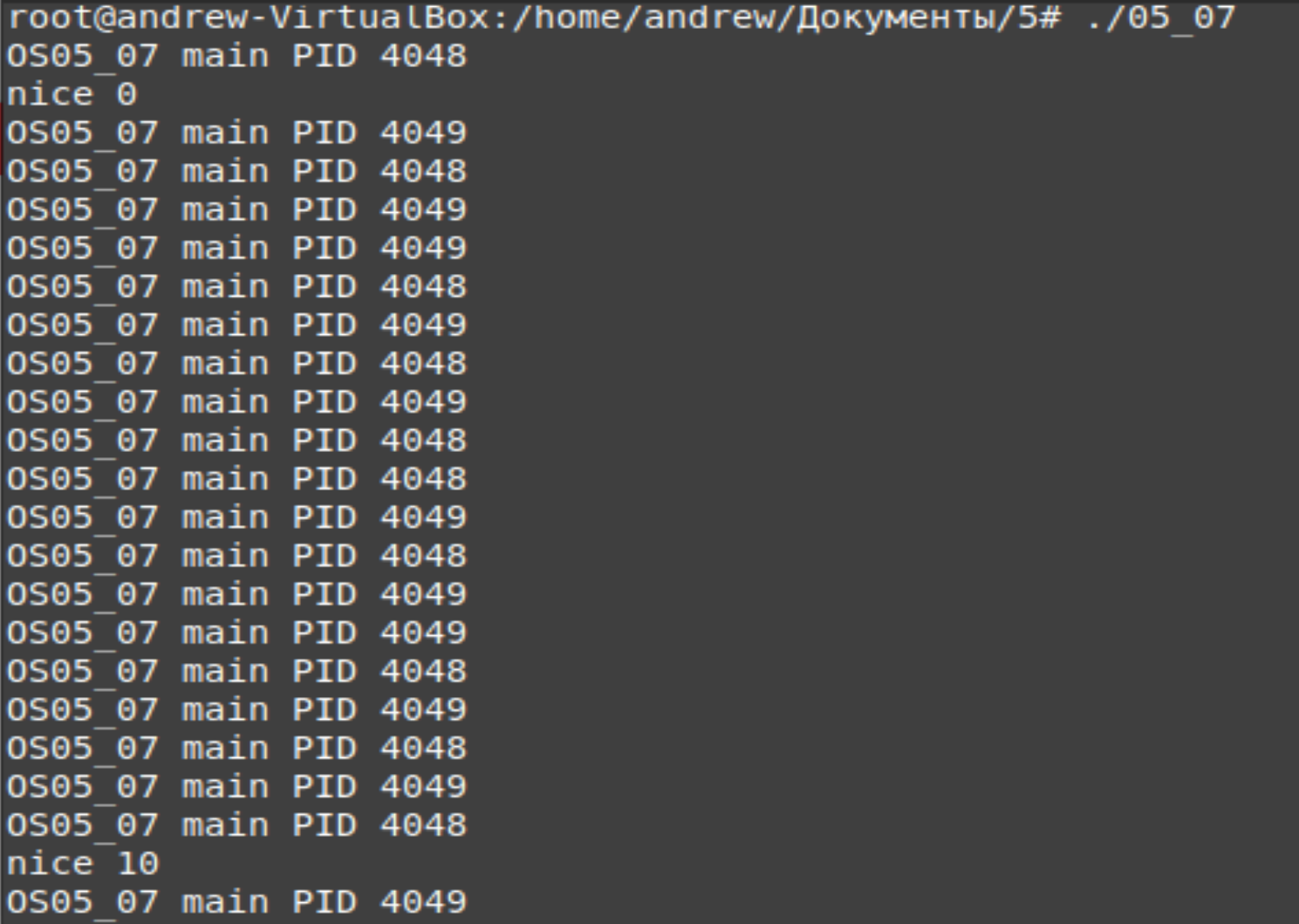
jobs: Отобразит список активных задач и их статус.

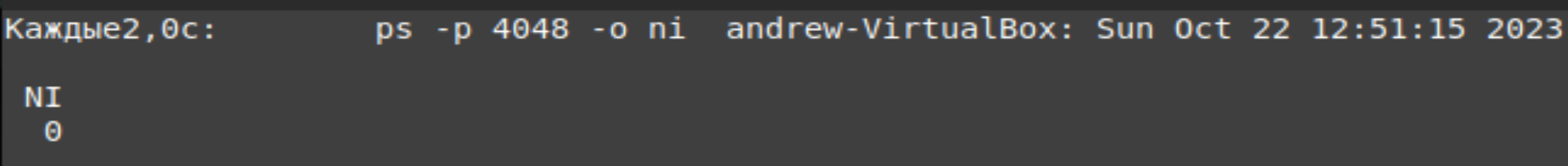
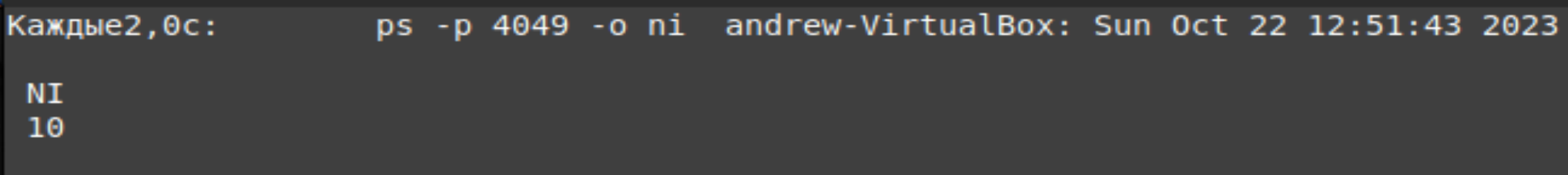
Ctrl+Z: Приостановит выполнение текущей задачи.

kill -9 PID: Завершит задачу с указанным идентификатором процесса (PID) жестким завершением.

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



**Задание 08.**Ответьте на следующие вопросы

1. 1 Поясните понятие «мультизадачная OS с вытеснением».
2. 2 Поясните понятие «циклическое планирование».
3. 3 Поясните понятие «приоритетное планирование».
4. 4 Поясните понятие «кооперативное планирование».
5. 5 Поясните понятие «OS реального времени».
6. 6 Поясните понятие «приоритет процесса».
7. 7 Поясните выражение «поток уступает процессор другому потоку».
8. 8 Windows: как поток может уступить процессор?
9. 9 Windows: что такое базовый приоритет потока, как он вычисляется и диапазон его изменения?
10. 10 Windows: поясните назначение и принцип применения системного вызова SetThreadIdealProcessor.
11. 11 Windows: поясните назначение и принцип применения системного вызова ResumeThread.
12. 12 Windows: поясните назначение и принцип применения системного вызова WaitForSingleObject.
13. 13 Windows: поясните назначение и принцип применения системных вызовов GetProcessPriorityBoost, GetThreadPriorityBoost, SetProcessPriorityBoost, SetThreadPriorityBoost.
14. 14 Linux: поясните принцип идентификации процессов и потоков и поясните почему он такой.
15. 15 Linux: Поясните понятие «планировщик потоков».
16. 16 Linux: поясните принцип использования значения nice –процесса, диапазон его изменения, для какого режима работы планировщика это значение применяется?
17. 17 Linux: перечислите политики планирования, какая действует по умолчанию?
18. 18 Linux: как выяснить действующую политику планирования для процесса с помощью файловой системы proc?
19. 19 Linux: с помощью какого системного вызова поток может уступить процессор.
20. 20 Linux: чем отличается системный вызов **nice** от вызова **setpriority**.
21. 21 Linux: поясните понятие «планировщик ввода вывода», каким образом можно выяснить какие планировщики ввода/ вывода доступны?
22. 22 Linux: перечислите известные вам планировщики ввода/ вывода, кратко охарактеризуйте их.

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

1. Поясните понятие «мультизадачная OS с вытеснением».

ОС, которая выполняет несколько задач одновременно, выделяет квант времени, определяет, когда это время истекает и переключает задачу

2. Поясните понятие «циклическое планирование».

Алгоритм циклического планирования. 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. 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.

Недостаток: Если частые переключения (квант - 4мс, а время переключения равно 1мс), то происходит уменьшение производительности.

3. Поясните понятие «приоритетное планирование».

каждому процессу присваивается значение приоритетности и запускается тот процесс, который находится в состоянии готовности и имеет наивысший приоритет

4. Поясните понятие «кооперативное планирование».

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.

5. 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 это система 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.

6. Поясните понятие «приоритет процесса».

Приоритет процесса — число, ориентируясь на значение которого планировщик процессов может выдавать процессу больше или меньше процессорного времени. Это значение, которое влияет на выбор процесса, который будет выполняться следующим.

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

Это значит, что текущий поток прерывает свое выполнение. При этом управление процессором передается следующему из очереди потоку.

8. Windows: как поток может уступить процессор?

Sleep(0) – процесс становится в конец очереди.

9. 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;

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

SetThreadIdealProcessor устанавливает предпочтительный процессор для потока, т.е. процессор на котором он будет работать

Если функция завершается успешно, величина возвращаемого значения - предшествующий привилегированный процессор или MAXIMUM\_PROCESSORS, если поток не имеет такового.

Если функция завершается с ошибкой, величина возвращаемого значения равна - (минус) 1

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

Запускает поток, после его системного вызова SuspendThread.

Уменьшает счетчик приостановленных потоков. Когда счетчик приостановки уменьшается до нуля, выполнение потока возобновляется.

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

Ожидает, пока указанный объект не перейдет в сигнальное состояние или пока не истечет интервал времени ожидания.

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

GetProcessPriorityBoost, GetThreadPriorityBoost возвращает true/false в зависимости от того, разрешено ли динамическое изменение приоритета процесса/потока.

SetProcessPriorityBoost, SetThreadPriorityBoost задают это разрешение.

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

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

15. Linux: Поясните понятие «планировщик потоков».

Это алгоритм, который определяет порядок выполнения потоков.

Планировщик — это часть ядра, которая решает какая запущенная нить будет выполняться процессором следующей.

Планировщик потоков ядра отвечает за то, какие потоки выполняются на процессорах системы.

16. 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))

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

Linux, политики планирования процесса:

1) стандартная (OTHER, разделения времени); - по умолчанию

2) FIFO-политика (FIFO, реального времени);

3) карусельная (round-robin) политика (RR);

4) пакетная политика(BATCH).

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

/proc/PID/sched (в строчке police будет)

0-OTHER

1-FIFO

2-RR

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

уступить процессор shed\_yield()

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

nice прибавляет к текущему значению любезности nice аргумент, setpriority устанавливает значение nice

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

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.

dmesg | grep scheduler

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

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

CFQ (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 – 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».

MQ-Deadline – модификация Deadline для новых устройств.

Kyber - для работы с быстрыми устройствами, две очереди read и write, read-очередь приоритетнее измеряется время завершения каждого запроса и корректирует фактический размер очереди для достижения установленных в настройках задержек.

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

cat /sys/block/sda/queue/scheduler