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.

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.

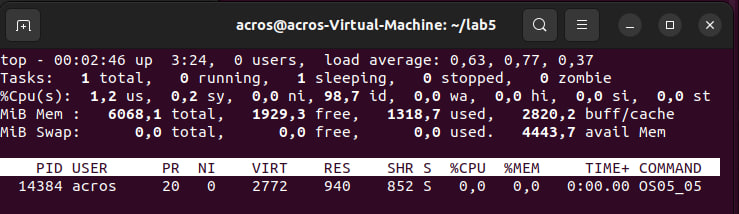
**Задание 04**

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

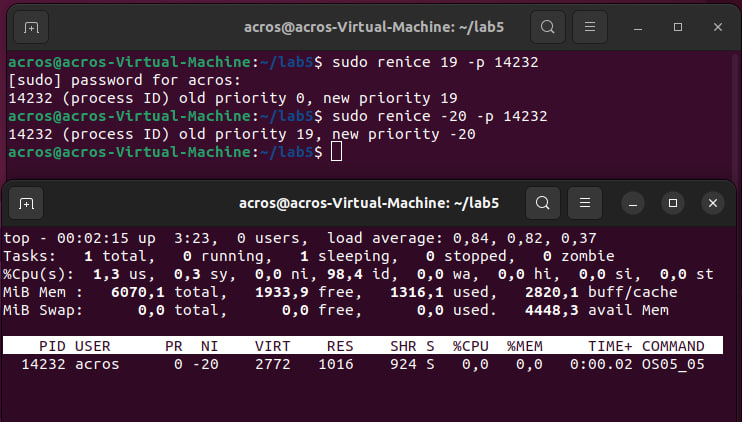
* идентификатор текущего процесса;
* идентификатор текущего (main) потока;
* приоритет (nice) текущего потока;
* номера доступных процессоров.

**Задание 05**

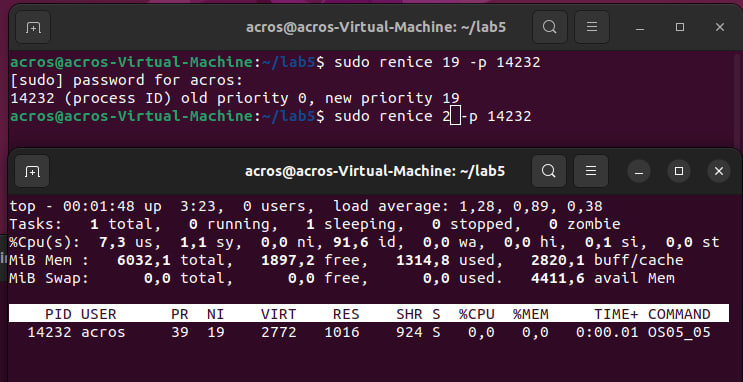
1. Разработайте консольное Linux-приложение **OS05\_05** на языке С, выполняющее длинный цикл.
2. Запустите приложение **OS05\_05.**
3. Зафиксируйте (скриншот) текущее значение **nicе**, полученное с помощью команды **top**.



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

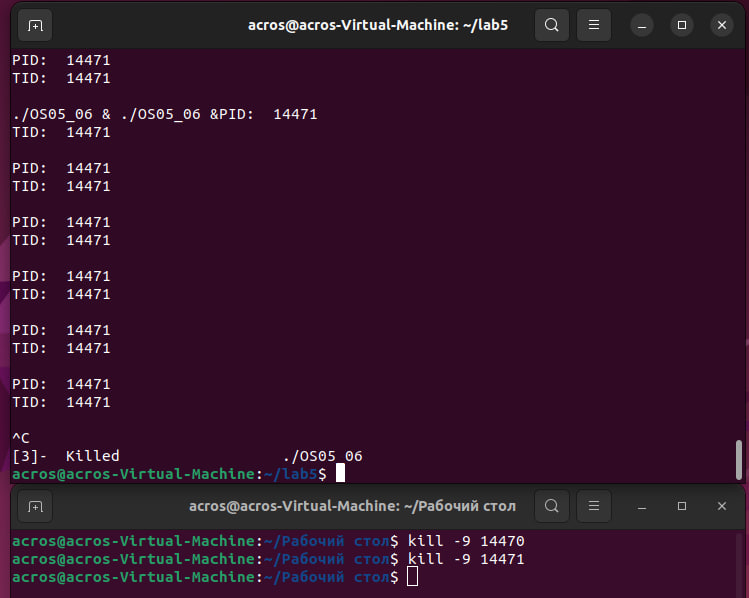


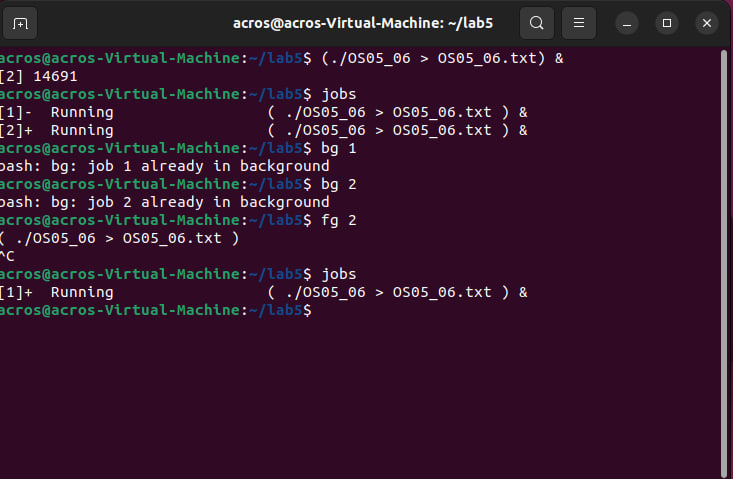
1. Уменьшите приоритет для **OS05\_05** до минимального значения (самого ничтожного). Зафиксируйте (скриншот) текущее значение **nicе**, полученное с помощью команды **top**



**Задание 06**

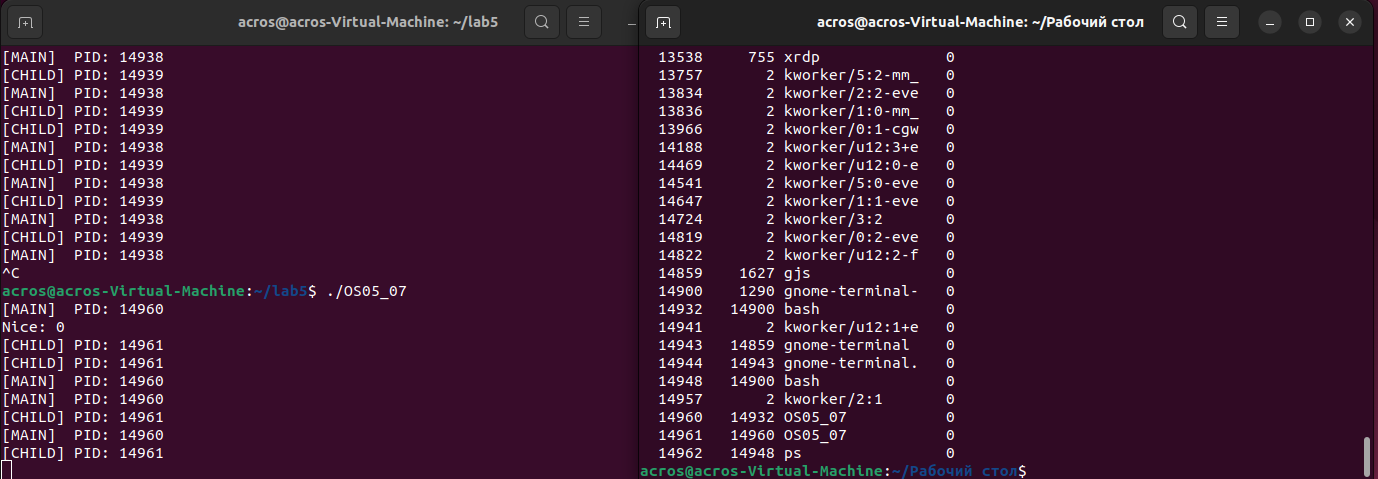
1. Разработайте консольное Linux-приложение **OS05\_06** на языке С, выполняющее длинный цикл с задержкой в 1сек в каждой итерации.
2. Продемонстрируйте запуск нескольких приложения **OS05\_06** в фоновом режиме, и команды bg, fg, jobs, Ctrl+Z, kill -9

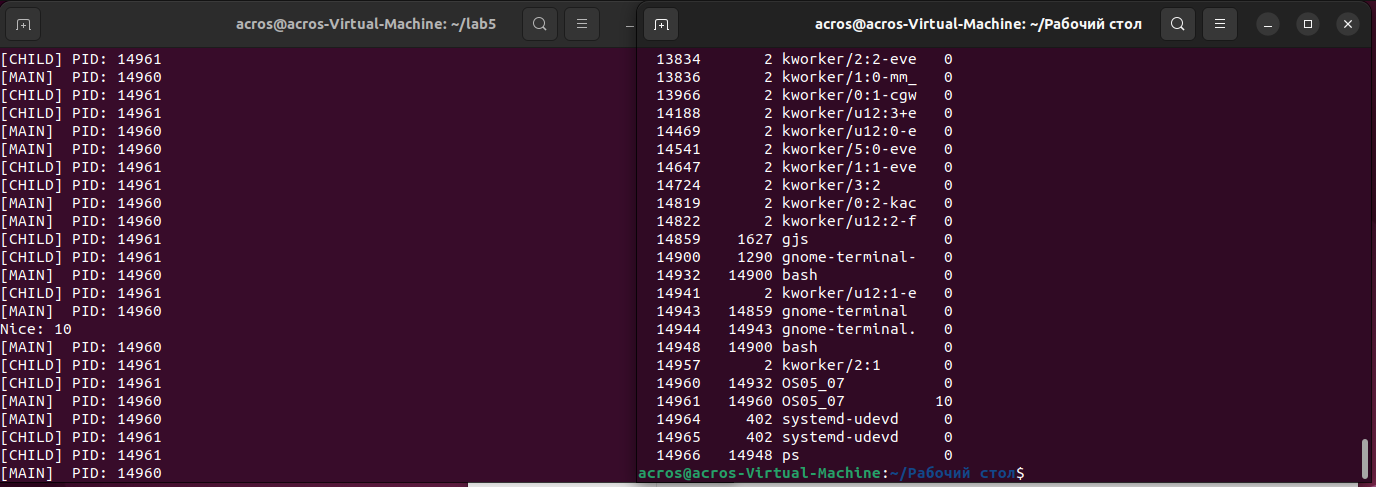


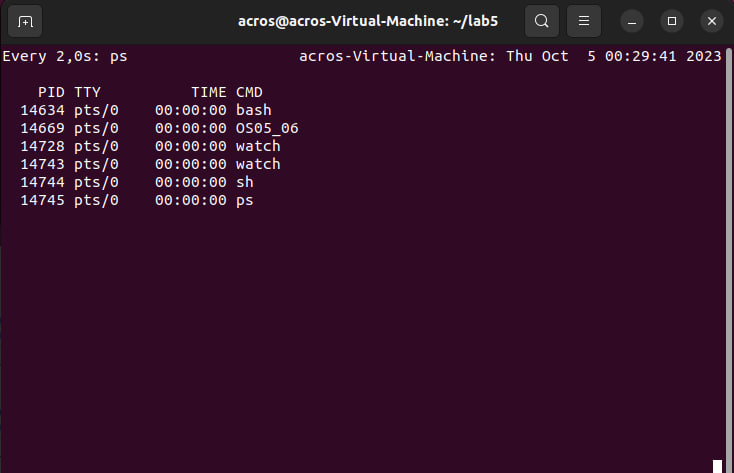


**Задание 07**

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

ОС, позволяющая выполнять много задач, выталкивая их из процессора для смены задач.

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

Метод планирования задач, где каждая задача имеет ограниченное время выполнения и они выполняются по кругу.

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

Метод, где задачам назначают приоритеты, и задачи с более высоким приоритетом выполняются первыми.

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

Метод, где задачи сами передают управление друг другу для выполнения, что требует сотрудничества между задачами.

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

ОС, гарантирующая выполнение задач в строго определенные сроки, важно в системах, где задержки недопустимы.

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

Число, определяющее важность выполнения процесса; процессы с более высоким приоритетом выполняются раньше.

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

"Поток уступает процессор другому потоку" означает, что текущий поток в многозадачной системе освобождает процессор и позволяет другому потоку начать выполнение.

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

В Windows поток может уступить процессор, используя функцию Sleep или вызов Wait функции для событий или объектов синхронизации.

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

Базовый приоритет потока в Windows определяется системой и может изменяться в диапазоне от 1 до 31(8). Он используется для определения порядка выполнения потоков.

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

Системный вызов SetThreadIdealProcessor в Windows позволяет установить предпочтительный процессор для выполнения определенного потока, что может помочь управлять распределением нагрузки на процессоры.

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

Системный вызов ResumeThread в Windows используется для возобновления выполнения приостановленного потока.

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

Системный вызов WaitForSingleObject в Windows позволяет потоку ждать завершения операции над объектом синхронизации, прерывая выполнение до завершения операции.

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

Системные вызовы GetProcessPriorityBoost, GetThreadPriorityBoost, SetProcessPriorityBoost и SetThreadPriorityBoost в Windows используются для управления приоритетом процессов и потоков и их увеличением или уменьшением важности в системе.

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

В Linux процессы и потоки идентифицируются уникальными числами, называемыми идентификаторами процесса (PID) и идентификаторами потоков (TID).

Ядро не различает процессы и потоки, общая системы нумерации, main-поток TID = PID;

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

Планировщик потоков в Linux - это часть ядра, отвечающая за распределение процессорного времени между потоками. Он определяет, какие потоки получают доступ к CPU и в каком порядке.

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

Значение nice в Linux - это числовая метка, определяющая приоритет процесса. Оно может изменяться от -20 до 19, где -20 означает самый высокий приоритет.

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

В Linux существует несколько политик планирования, включая SCHED\_OTHER, SCHED\_FIFO, SCHED\_RR и SCHED\_BATCH. Политика SCHED\_OTHER действует по умолчанию.

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

Для выяснения действующей политики планирования процесса можно использовать файл /proc/<PID>/sched. Например, cat /proc/12345/sched отобразит информацию о политике планирования для процесса с PID 12345.

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

Поток в Linux может уступить процессор, вызвав системный вызов sched\_yield(). Это позволяет ядру операционной системы принять решение о перераспределении процессорного времени между потоками.

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

В Linux, системный вызов nice используется для изменения приоритета процесса, в то время как вызов setpriority позволяет устанавливать приоритет для процесса и его потоков. setpriority более гибкий, так как он может управлять приоритетами на более гранулированном уровне.

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

Планировщик ввода/вывода в Linux отвечает за управление операциями ввода/вывода. Чтобы узнать доступные планировщики ввода/вывода, можно просмотреть содержимое директории /sys/block/<device>/queue/scheduler, где <device> - это имя блочного устройства.

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

Некоторые известные планировщики ввода/вывода в Linux включают CFQ, NOOP и Deadline.

* CFQ обеспечивает справедливое распределение времени на доступ к диску между процессами.
* NOOP просто обрабатывает запросы в порядке поступления.
* Deadline ставит сроки выполнения для запросов и пытается минимизировать задержки.

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

Для определения типа планировщика, используемого для блокового устройства, можно прочитать содержимое файла /sys/block/<device>/queue/scheduler. Этот файл указывает на текущий активный планировщик ввода/вывода для устройства.