ОТЧЁТ

Операционные Системы

Беласин Д. А.

Лабораторная №5

**Задание 01**

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

- идентификатор текущего процесса;

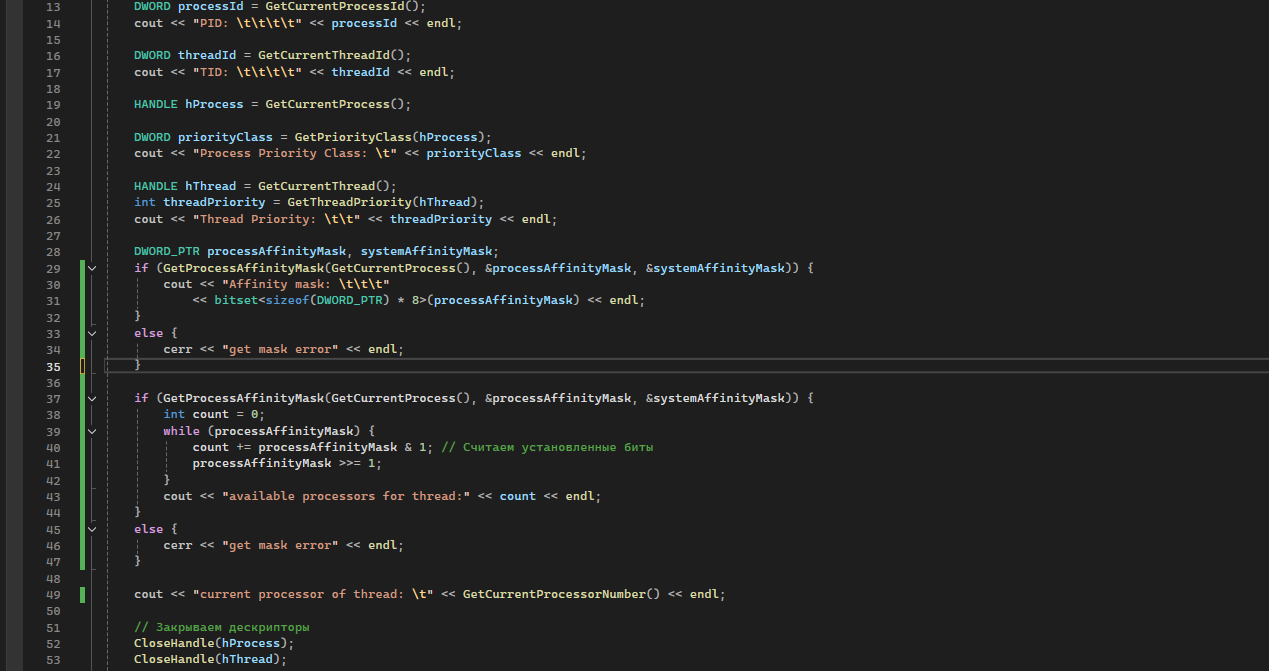
- идентификатор текущего (main) потока;

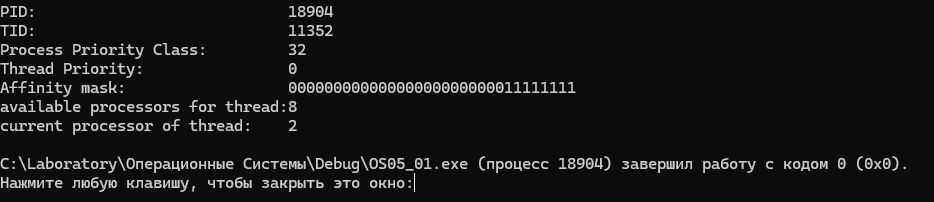
- приоритет (приоритетный класс) текущего процесса;

- приоритет текущего потока;

- маску (affinity mask) доступных процессу процессоров в двоичном виде;

- количество процессоров, доступных процессу;

- процессор, назначенный текущему потоку. 

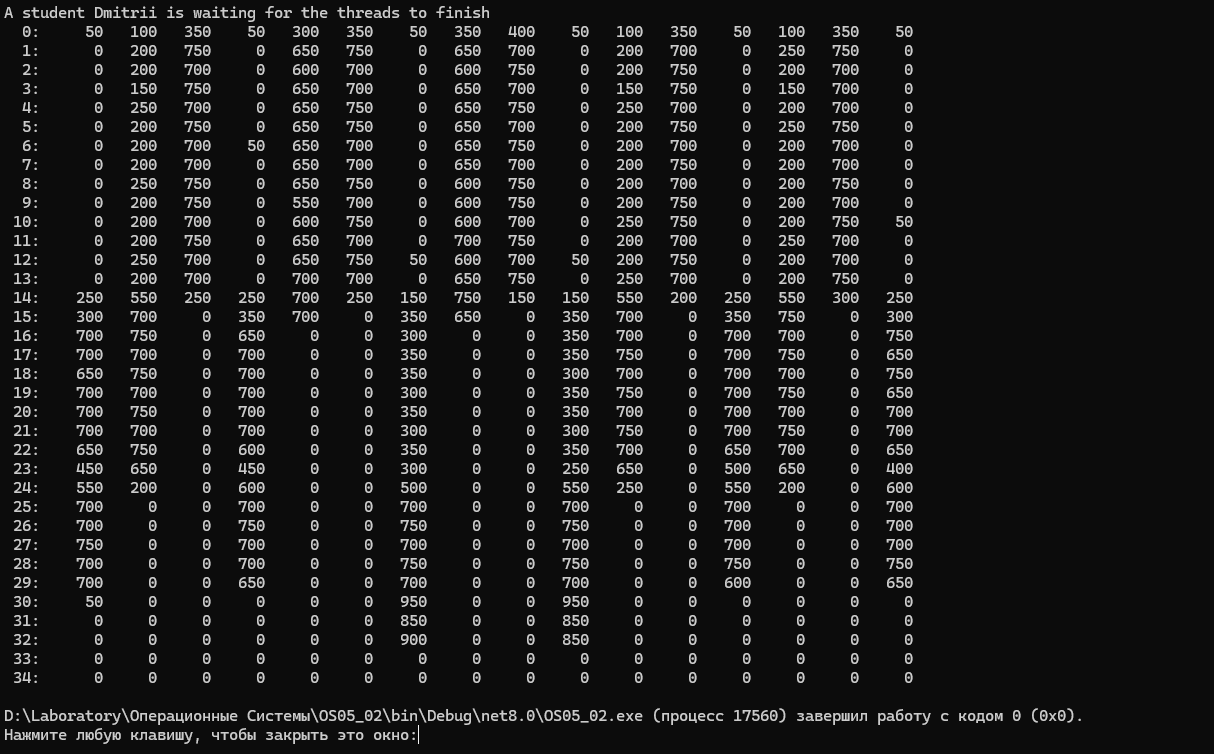


**Задание 02**

2. 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).

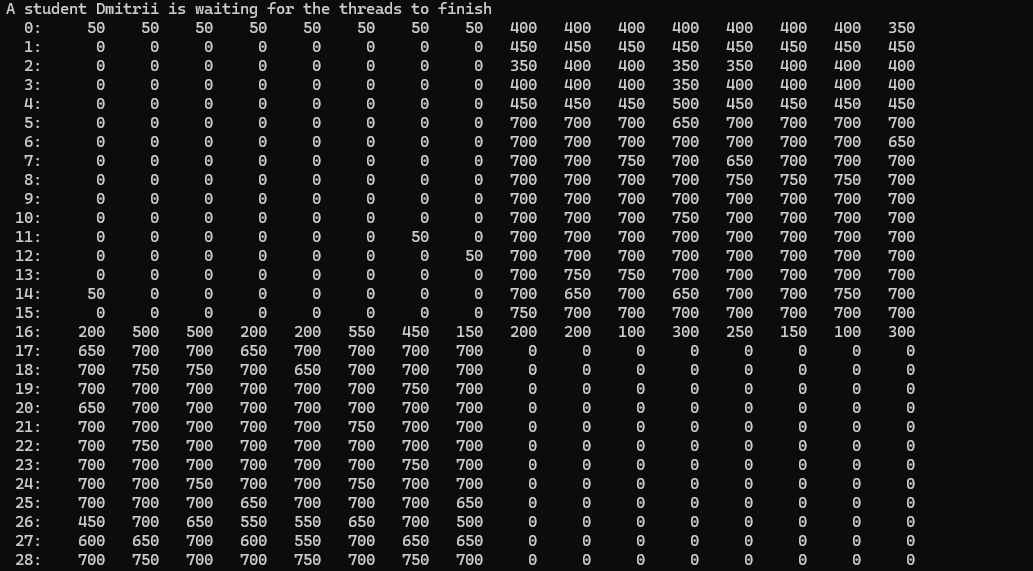
3. Выполните приложение, не забудьте про персонификацию вывода результатов.

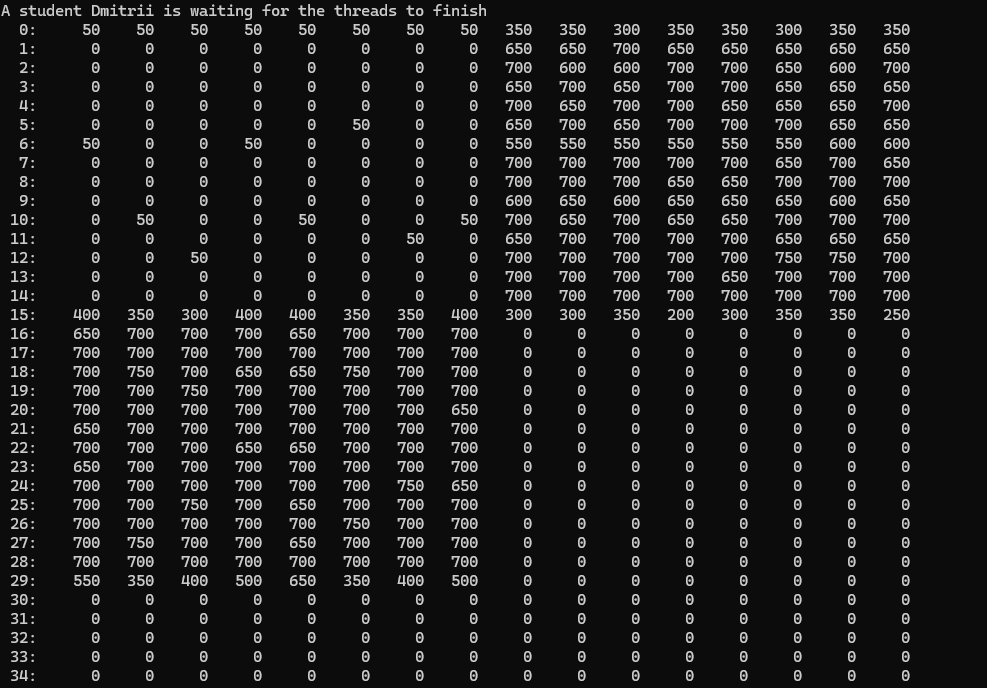
4. По зафиксированным скриншотам объясните полученные результаты.



**Задание 03**

5. 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.

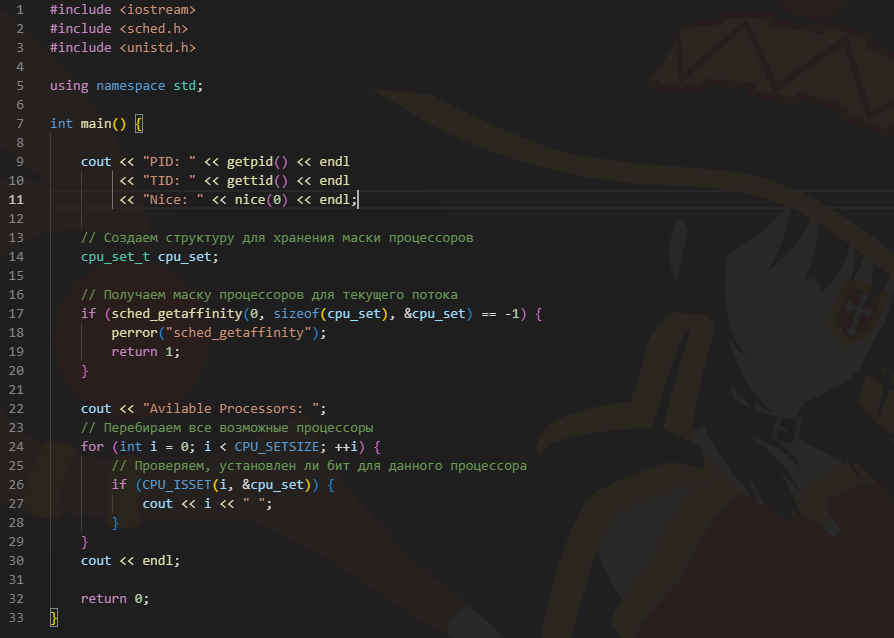
6. Выполните приложение OS05\_03. Удалось ли поработать низкоприоритетным потокам? (Чтобы уменьшить влияние случайности, можно повторить эксперимент несколько раз). 

7. Выполните приложение OS05\_03 с другими парами приоритетов, например, BelowNormal и Normal. Изменился ли характер работы потоков? 

8. По зафиксированным скриншотам объясните полученные результаты. При этом укажите числовые значения приоритетов потоков.

**Задание 04**

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

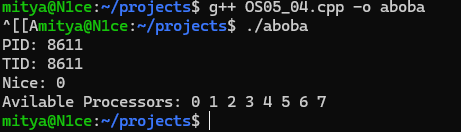


- идентификатор текущего процесса;

- идентификатор текущего (main) потока;

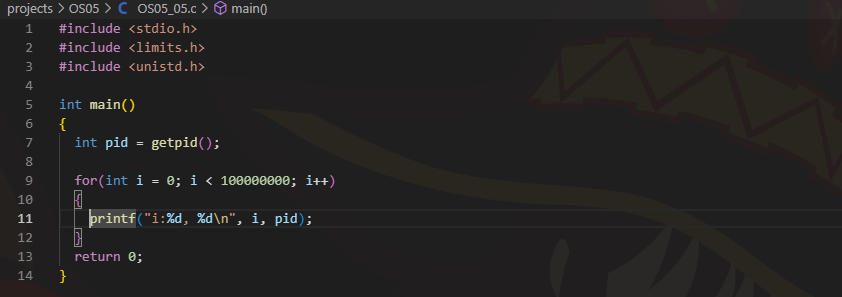
- приоритет (nice) текущего потока;

- номера доступных процессоров.



**Задание 05**

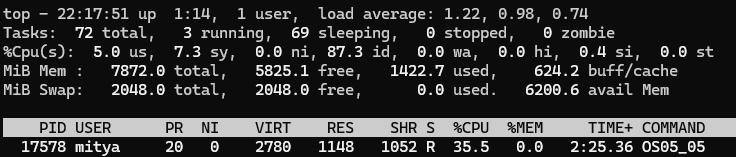
10. Разработайте консольное Linux-приложение OS05\_05 на языке С, выполняющее длинный цикл.

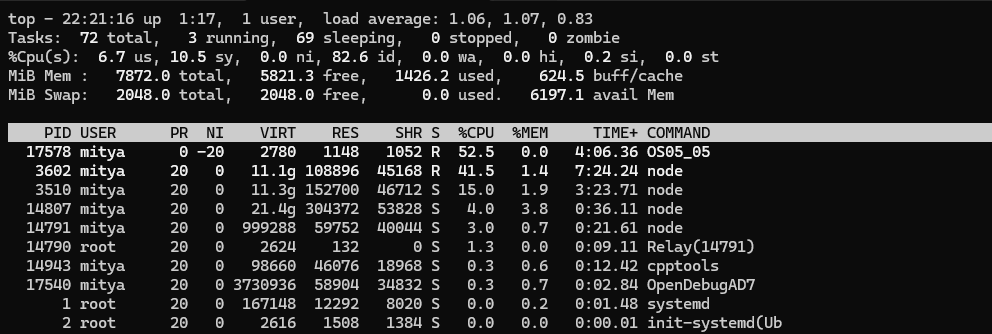


11. Запустите приложение OS05\_05.



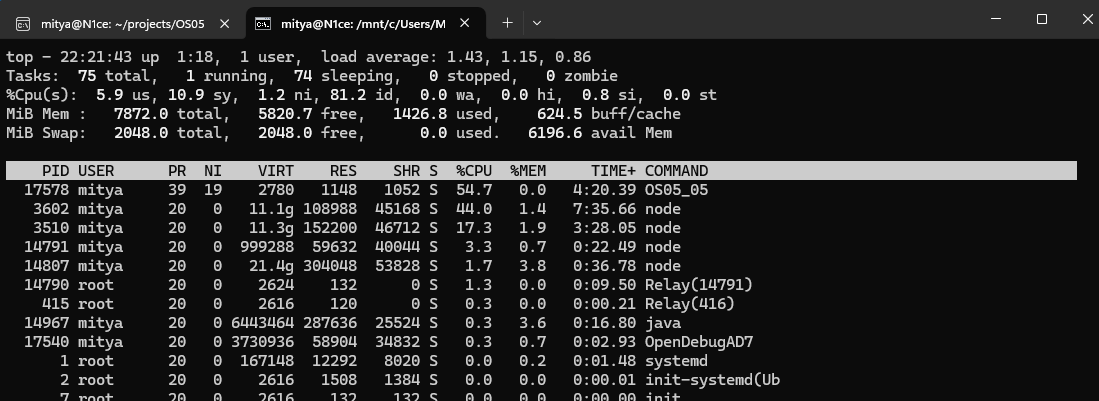
12. Зафиксируйте текущее значение nicе, полученное с помощью команды top.



13. Увеличьте приоритет для OS05\_05 до максимального значения (самого привилегированного). 

Зафиксируйте текущее значение nicе, полученное с помощью команды top.

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



**Задание 06. Ответьте на следующие вопросы**

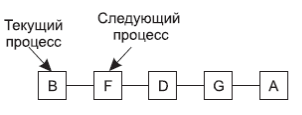
**15. Поясните понятие «мультизадачная OS с вытеснением».**

Мультизадачная операционная система с вытеснением (preemptive multitasking OS) — это система, которая поддерживает выполнение нескольких задач (процессов) одновременно и самостоятельно управляет переключением между ними, прерывая выполнение одной задачи для выполнения другой.

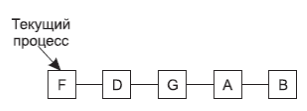
**16. Поясните понятие «циклическое планирование».**

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. Разумеется, 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. От планировщика требуется всего лишь вести список процессов, готовых к выполнению. Когда процесс исчерпает свой квант времени, он помещается в конец списка.

Список процессов, находящихся в состоянии готовности:



Список процессов после того, как B исчерпал свой квант времени:



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

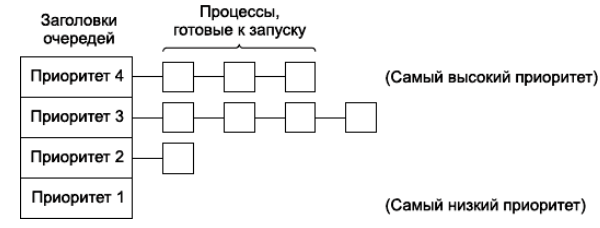
Каждому процессу присваивается значение приоритетности и запускается тот процесс, который находится в состоянии готовности и имеет наивысший приоритет.

Чтобы предотвратить бесконечное выполнение высокоприоритетных процессов, планировщик должен понижать уровень приоритета текущего выполняемого процесса с каждым сигналом таймера (то есть с каждым его прерыванием). Если это действие приведет к тому, что его приоритет упадет ниже приоритета следующего по этому показателю процесса, произойдет переключение процессов.

Можно выбрать и другую альтернативу: каждому процессу может быть выделен максимальный квант допустимого времени выполнения. Когда квант времени будет исчерпан, шанс запуска будет предоставлен другому процессу, имеющему наивысший приоритет.

Приоритеты могут присваиваться процессам в статическом или в динамическом режиме.

Система с четырьмя классами приоритетности:



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

Кооперативное планирование (cooperative scheduling) — это метод планирования задач в операционной системе, при котором переключение между задачами (процессами) происходит только тогда, когда сама задача добровольно уступает управление. В такой системе ОС не может принудительно прервать выполнение задачи; это делает её зависимой от "добросовестности" приложений.

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

Операционная система реального времени (Real-Time Operating System, RTOS) — это операционная система, которая предназначена для выполнения задач в строго ограниченные временные интервалы, обеспечивая точное и предсказуемое время отклика на внешние события. Она широко используется в системах, где время выполнения задач критично и даже небольшие задержки могут привести к нежелательным последствиям.

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

Приоритет процесса — это значение, которое определяет важность или срочность выполнения процесса в операционной системе, на основании которого происходит планирование.

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

Выражение «поток уступает процессор другому потоку» означает, что текущий поток, выполняющийся на процессоре, добровольно или принудительно прекращает своё выполнение, чтобы дать возможность другому потоку получить доступ к ресурсу центрального процессора.

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

1. Вызов функции Sleep. Функция Sleep позволяет потоку добровольно приостановиться на определенное количество миллисекунд. Когда поток вызывает Sleep, он временно освобождает процессор и помещается в состояние ожидания, чтобы другие потоки могли выполняться. Shced\_yield()
2. Использование функции SwitchToThread. Функция SwitchToThread позволяет текущему потоку уступить процессор другому потоку, который готов к выполнению и находится в этом же процессе или в другом. Если таких потоков нет, текущий поток продолжит выполнение.
3. Завершение квантового времени. В Windows потоки запускаются на процессоре на определенный квант времени. Когда квант времени истекает, планировщик автоматически переключается на другой поток. Поток может "уступить" процессор, просто дождавшись конца своего кванта времени.
4. Ожидание событий или объектов синхронизации. Поток может ожидать завершения определенного события или объекта синхронизации (например, мьютекса, семафора или объекта события). В этом случае поток переводится в состояние ожидания и освобождает процессор для выполнения других потоков.
5. Изменение приоритета потока. Приоритет потока влияет на его доступ к процессору. Потоки с более высоким приоритетом получают процессор чаще, чем потоки с более низким. Если поток временно снижает свой приоритет, это также позволяет другим потокам (с более высоким приоритетом) получить доступ к процессору.

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

Базовый приоритет потока – это начальный приоритет, с которым поток запускается. Он вычисляется на основе приоритета процесса и классов приоритетов (Real-time, high, above normal, normal, below normal, idle).

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

Функция ResumeThread работает с так называемым счетчиком приостановки (suspend count) для каждого потока. Этот счетчик показывает, сколько раз поток был приостановлен. Каждый вызов функции SuspendThread увеличивает значение счетчика приостановки на единицу, а каждый вызов ResumeThread уменьшает его на единицу.

|  |
| --- |
| DWORD ResumeThread(  [in] HANDLE hThread  ); |

[in] hThread – Дескриптор для перезагружаемого потока.

Если функция выполнена успешно, возвращаемое значение — это предыдущее число приостановки потока.

Если функция завершается сбоем, возвращается значение (DWORD) -1.

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

|  |
| --- |
| function WaitForSingleObject(  hHandle: THandle; *// идентификатор объекта*  dwMilliseconds: DWORD *// период ожидания*  ) |

Эта функция блокирует выполнение текущего потока и ожидает перехода объекта hHandle в сигнальное состояние в течение dwMilliseconds миллисекунд. Если в качестве параметра dwMilliseconds передать значение INFINITE, функция будет ждать в течение неограниченного времени. Если параметр dwMilliseconds равен 0, то функция проверяет состояние объекта и немедленно возвращает управление.

Функция возвращает одно из следующих значений:

* WAIT\_ABANDONED - поток, владевший объектом, завершился, не переведя объект в сигнальное состояние;
* WAIT\_OBJECT\_0 - объект перешел в сигнальное состояние;
* WAIT\_TIMEOUT - истек срок ожидания (обычно в этом случае генерируется ошибка либо функция вызывается в цикле до получения другого результата);
* WAIT\_FAILED - произошла ошибка, например неверное значение hHandle (более подробную информацию можно получить, вызвав функцию GetLastError).

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

Система может повысить и понизить динамический приоритет, чтобы гарантировать, что она реагирует и что потоки не голодают в течение времени процессора. Система не повышает приоритет потоков с базовым уровнем приоритета от 16 до 31. Динамические повышения приоритета получают только потоки с базовым приоритетом от 1 до 15.

Функцию повышения приоритета можно отключить, вызвав функцию [SetProcessPriorityBoost](https://learn.microsoft.com/ru-ru/windows/win32/api/processthreadsapi/nf-processthreadsapi-setprocesspriorityboost) или [SetThreadPriorityBoost](https://learn.microsoft.com/ru-ru/windows/win32/api/processthreadsapi/nf-processthreadsapi-setthreadpriorityboost) . Чтобы определить, отключена ли эта функция, вызовите функцию [GetProcessPriorityBoost](https://learn.microsoft.com/ru-ru/windows/win32/api/processthreadsapi/nf-processthreadsapi-getprocesspriorityboost) или [GetThreadPriorityBoost](https://learn.microsoft.com/ru-ru/windows/win32/api/processthreadsapi/nf-processthreadsapi-getthreadpriorityboost) .

|  |
| --- |
| BOOL SetThreadPriorityBoost(  [in] HANDLE hThread,  [in] BOOL bDisablePriorityBoost  ); |

[in] hThread – дескриптор потока.

|  |
| --- |
| BOOL SetProcessPriorityBoost(  [in] HANDLE hProcess,  [in] BOOL bDisablePriorityBoost  ); |

[in] hProcess – дескриптор процесса.

[in] bDisablePriorityBoost – если этот параметр имеет значение TRUE, динамическое повышение отключено. Если параметр имеет значение FALSE, включено динамическое повышение.

|  |
| --- |
| BOOL GetThreadPriorityBoost(  [in] HANDLE hThread,  [out] PBOOL pDisablePriorityBoost  ); |

[in] hThread – дескриптор потока.

|  |
| --- |
| BOOL GetProcessPriorityBoost(  [in] HANDLE hProcess,  [out] PBOOL pDisablePriorityBoost  ); |

[in] hProcess – дескриптор процесса.

[out] pDisablePriorityBoost – указатель на переменную, которая получает состояние элемента управления повышения приоритета. Значение TRUE указывает, что динамическое повышение отключено. Значение FALSE указывает на нормальное поведение.

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

Каждый процесс имеет уникальный идентификатор — целое число в диапазоне от 0 до 32767, которое называется *идентификатором процесса* (*process identifier*, *PID*). Идентификаторы процессов PID 0 и 1 имеют для системы специальной значение; все остальные идентификаторы присваиваются «обычным» процессам. Процесс имеет только один PID, и он сохраняется неизменным в течение его жизненного цикла.

TID (Thread ID) – идентификатор потока, уникально определяет поток внутри процесса. В однопоточной программе TID совпадает с PID, так как процесс включает только один поток. В многопоточной программе каждый поток имеет свой TID, но все потоки одного процесса будут иметь одинаковый PID.

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

Планировщик потоков в Linux — это компонент ядра, который отвечает за распределение времени процессора между процессами и потоками. Планировщик решает, какой процесс или поток должен выполняться в данный момент, исходя из различных критериев: приоритета, значения nice, политики планирования и других параметров. В Linux потоки рассматриваются как процессы с общими ресурсами, поэтому планировщик работает с потоками так же, как и с процессами, обеспечивая их справедливое и эффективное выполнение.

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

Приоритет nice и приоритет планировщика процессов ядра ОС — разные числа. Число nice — приоритет, который пользователь хотел бы назначить процессу. Приоритет планировщика — действительный приоритет, назначенный процессу планировщиком.

Планировщик процессов ядра ОС Linux поддерживает приоритеты от 0 (реальное время) до 139 включительно. Приоритеты -20…+19 утилиты или команды nice соответствуют приоритетам 100…139 планировщика процессов.

Приоритет nice может принимать значения от -20 до 19. Чем ниже значение nice, тем выше приоритет процесса.

* Например:
  + nice = -20 означает самый высокий приоритет (сам процесс будет "жадно" использовать процессор). Статический приоритет равен 100.
  + nice = 19 означает самый низкий приоритет (процесс получает минимальное время процессора). Статический приоритет равен 139.

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

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. OTHER (или SCHED\_OTHER)

Используется: для обычных (не привилегированных) процессов.

Статический приоритет равен 0. Для политики OTHER важнее не абсолютный приоритет, а "nice" — параметр, который определяет относительный приоритет процесса. Чем выше значение nice, тем меньше приоритет у процесса (он будет меньше конкурировать за процессорное время).

Процессы с политикой OTHER будут выполняться, если нет процессов с более высокоприоритетными политиками, такими как FIFO или RR.

2. BATCH (или SCHED\_BATCH)

BATCH от OTHER отличается тем, что BATCH снижает частоту переключений задач, что делает её лучше для длительных фоновых вычислений. OTHER же оптимизирована для обычных задач, требующих более быстрого отклика, и распределяет ресурсы с учётом интерактивности процессов.

Задачи с BATCH запускаются, когда нет активных интерактивных задач.

3. FIFO (или SCHED\_FIFO)

Используется для процессов реального времени, которым нужно более строгое планирование.

У каждого процесса FIFO фиксированный статический приоритет, который определяет порядок его выполнения. Процессы с более высоким приоритетом подавляют процессы с более низким приоритетом.

Если процесс с политикой FIFO имеет самый высокий приоритет, он будет выполняться до тех пор, пока либо не блокируется (например, ожидая ввод-вывод), либо не уступит процессор по вызову sched\_yield().

Когда процесс блокируется, он помещается в конец своей очереди, и очередь продвигается.

Новый процесс с политикой FIFO добавляется в начало своей очереди, то есть начинает исполняться сразу, если у него более высокий приоритет.

4. Round Robin (или SCHED\_RR)

Используется также для процессов реального времени, но с квантованием времени.

Приоритет также определяется статическим приоритетом, как и в FIFO, и процессы с более высоким приоритетом подавляют более низкие.

Отличие от FIFO в том, что здесь процесс получает "квант времени" — определенное количество времени, в течение которого он может выполняться.

Когда квант времени заканчивается, процесс помещается в конец своей очереди, и CPU переходит к следующему процессу с тем же приоритетом.

Такая политика позволяет всем процессам с одинаковым приоритетом работать по очереди, предотвращая бесконечное выполнение одного процесса.

Для установки и получения политики планирования процесса используются функции sched\_setscheduler и sched\_getscheduler, соответственно.

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

Процесс может освободить процессор без блокировки, вызвав **sched\_yield**. Процесс будет перемещен в конец очереди процессов с одинаковым статическим приоритетом, и управление будет передано другому процессу.