Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №1 по курсу "Операционные системы"

Тема Прерывания таймера в Windows и Unix

Студент Гурова Н.А.

Группа ИУ7-54Б

Оценка (баллы) _____

Преподаватель Рязанова Н. Ю.

Москва — 2022 г.

СОДЕРЖАНИЕ

1 Основные определения

Тик — период времени между двумя последующими прерываниями таймера [1].

Основной тик — период времени, равный n тикам таймера (число n зависит от конкретного варианта системы) [1].

Квант времени — временной интервал, в течение которого потоку разрешено работать, пока не настанет очередь запускаться другому потоку с тем же уровнем приоритета [2].

2 Функции обработчика прерывания от системного таймера в защищенном режиме

Обработчик прерывания таймера запускается в ответ на возникновение аппаратного прерывания таймера, являющегося вторым по приоритету событием в системе[1] (после прерывания по сбою питания). Это говорит о том, что никакая другая работа не может выполняться в системе во время его срабатывания. Поэтому обработчик должен запускаться как можно быстрее, а его время работы желательно сводить к минимуму [1].

2.1 Unix

По **тику**:

— инкремент счетчика тиков аппаратного таймера;

— инкремент часов и других таймеров системы[2];

— декремент кванта текущего потока;

— декремент счетчика времени до отправления на выполнение отложенных вызовов[3], при достижении счетчиком нуля происходит выставление флага для обработчика отложенного вызова;

— обновление статистики использования процессора текущим процессом (инкремент поля `p_cpu` дескриптора текущего процесса до максимального значения, равного 127).

[1]Приоритет процесса — это параметр, который размещен в контексте процесса, и по значению этого параметра осуществляется выбор очередного процесса для продолжения работы или выбор процесса для его приостановки [3].

[2]Список некоторых таймеров в системе можно посмотреть с помощью команды `systemctl status *timer` [4].

[3]Отложенный вызов — запись функции, которую ядро системы должно будет вызвать через определенный промежуток времени [5].

По **главному тику**:

— регистрация отложенных вызовов функций, относящихся к работе планировщика[4], таких как пересчет приоритетов;

— пробуждение системных вызовов `swapper`[5] и `pagedaemon`[6] (то есть регистрация отложенного вызова процедуры `wakeup`, которая перемещает дескриптор процесса из списка «спящих» в очередь «готовых к выполнению»);

— декремент счетчика времени, оставшегося до посылки одного из следующих сигналов [9]:

  — `SIGALRM` — сигнал, посылаемый процессору по истечении времени, заданного функцией `alarm()` (будильник реального времени);

  — `SIGPROF` — сигнал, посылаемый процессору по истечении времени, заданного в таймере профилирования (будильник профиля процесса);

  — `SIGVTALRM` — сигнал, посылаемый процессору по истечении времени работы в режиме задачи (будильник виртуального времени);

По **кванту**:

— если текущий процесс превысил выделенный ему квант процессорного времени, отправка ему сигнала `SIGXCPU` [1].

[4]Планировщик — компонент ОС, определяющий, какой из процессов должен выполняться в данный момент времени и как долго он может занимать процессор [6].

[5]Исторически, системный вызов `swapper` отвечал за «процесс свопинга», то есть за перемещение всех страниц определенного процесса из/в память/резервное хранилище [7].

[6]Системный вызов `pagedaemon` — процесс-демон, управляющий страничной организацией памяти [8].

2.2 Windows

По **тику**:

— инкремент счетчика системного времени;

— декремент остатка кванта текущего потока;

— декремент счетчиков отложенных задач;

— если активен механизм профилирования ядра[7], то инициализация отложенного вызова обработчика ловушки[8] профилирования ядра добавлением объекта в очередь DPC (Deferred procedure call, отложенный вызов процедуры) [12]. Обработчик ловушки профилирования регистрирует адрес команды, выполнявшейся на момент прерывания.

По **главному тику**:

— инициализация диспетчера настройки баланса[9] путем освобождения объекта «событие», на котором он ожидает.

По **кванту**:

— инициализация диспетчеризации[10] потоков добавлением соответствующего объекта в очередь DPC.

---

[7]Профилирование ядра — это выявление «узких мест» в производительности. С помощью профилирования можно определить, где именно произошла потеря производительности в той или иной программе. Специальные программы генерируют профиль — сводку событий, на основе которой можно выяснить, на выполнение каких функций ушло больше всего времени. Эти программы, однако, не помогают выявить причину снижения производительности [10].

[8]Термин ловушка (trap) относится к механизму, благодаря которому при прерывании или исключении процессор перехватывает контроль над выполняемым потоком и передает управление определенной части операционной системы. В Windows процессор передает управление обработчику ловушек (trap handler) — функции, специфичной для конкретного прерывания или исключения [11].

[9]Диспетчер настройки баланса — системный поток, который активизируется для возможной инициализации событий, связанных с планированием и управлением памятью [13].

[10]Диспетчеризация заключается в реализации найденного в результате планирования (динамического или статистического) решения, то есть в переключении процессора с одного потока на другой. Прежде чем прервать выполнение потока, ОС запоминает его контекст, с тем, чтобы впоследствии использовать эту информацию для последующего возобновления выполнения данного потока [14].

3 Пересчет динамических приоритетов

Планирование [15] — это организация очереди процессов, то есть постановка процессов в очередь. Планирование бывает с приоритетами и без приоритетов.

Приоритет — это некоторое число, характеризующее степень привилегированности процесса. Чем выше приоритет, тем меньше времени будет проводить поток в очередях. Различают два вида приоритетов [15]:

— статические приоритеты — такие приоритеты назначаются процессам до начала их выполнения и при выполнении не меняются;

— динамические приоритеты — приоритеты, изменяющиеся в процессе выполнения.

Динамически могут пересчитываться только приоритеты пользовательских процессов[1] (как в Unix, так и в Windows).

3.1 Unix

В современных системах Unix ядро является строко вытесняющим — процесс в режиме ядра может быть вытеснен более приоритетным процессом, также находящимся в режиме ядра. Это сделано для того, чтобы система могла обслуживать процессы реального времени, такие как аудио и видео.

Согласно приоритетам процессов и принципу вытесняющего циклического планирования формируется очередь готовых к выполнению процессов. Первыми будут выполняться процессы с большим приоритетом. Процессы с одинаковыми приоритетами выполняются в течение кванта времени, циклически друг за другом.

Приоритет задается любым целым числом, лежащим в диапазоне от 0 до 127. Чем меньше такое число, тем выше приоритет. Приоритеты от 0 до 49 зарезервированы для ядра, следовательно, прикладные процессы могут обладать приоритетом в диапазоне от 50 до 127 [17].

[1]Код пользовательского приложения запускается в пользовательском режиме, а код операционной системы запускается в режиме ядра. Предоставляя программному обеспечению операционной системы более высокий уровень привилегий, нежели прикладному программному обеспечению, процессор гарантирует, что приложения с неправильным поведением не смогут нарушить стабильность работы системы [16].

Таблица 3.1 – Поля структуры `proc`, относящиеся к приоритетам.

p_pri	Текущий приоритет планирования
p_usrpri	Приоритет режима задачи
p_cpu	Результат последнего измерения использования процессора
p_nice	Фактор «любезности», устанавливаемый пользователем

Для принятия решения о том, какой процесс отправить на выполнение, планировщик использует `p_pri`. Когда процесс находится в режиме задачи, значение его `p_pri` идентично `p_usrpri`. Когда процесс просыпается после блокирования в системном вызове, его приоритет будет временно повышен для того, чтобы дать ему предпочтение для выполнения в режиме ядра.

Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может заблокироваться. Приоритет сна является величиной, определяемой для ядра, и потому лежит в диапазоне 0-49. Когда замороженный процесс просыпается, ядро устанавливает значение его `p_pri` равным приоритету сна события или ресурса. Когда же процесс находится в состоянии возврата в режим задачи, его приоритет сбрасывается обратно в значение текущего приоритета в режиме задачи.

Таким образом, планировщик использует `p_usrpri` для хранения приоритета, который будет назначен процессу при возврате в режим задачи, а `p_pri` — для хранения временного приоритета для выполнения в режиме ядра [17].

Таблица 3.2 – Приоритеты сна в системе 4.3BSD UNIX и SCO UNIX [18].

Событие	**4.3BSD**	**SCO**
Ожидание загрузки страницы/сегмента	0	95
Ожидание индексного дескриптора	10	88
Ожидание ввода/вывода	20	81
Ожидание буфера	30	80
Ожидание терминального ввода		75
Ожидание терминального вывода		74
Ожидание завершения выполнения		73
Ожидание события	40	66

Таблица 3.3 – Приоритеты сна в системе 4.3BSD UNIX [19].

Приоритет	**Значение**	**Описание**
PSWP	0	Свопинг
PSWP + 1	1	Страничный демон
PSWP + 1/2/4	1/2/4	Другие действия по обработке памяти
PINOD	10	Ожидание освобождения inode
PRIBIO	20	Ожидание дискового ввода-вывода
PRIBIO + 1	21	Ожидание освобождения буфера
PZERO	25	Базовый приоритет
TTIPRI	28	Ожидание ввода с терминала
TTOPRI	29	Ожидание вывода с терминала
PWAIT	30	Ожидание завершения процесса потомка
PLOCK	35	Консультативное ожидание блокир. ресурса
PSLEP	40	Ожидание сигнала

Приоритет в режиме задачи зависит от двух факторов [17]:

— фактор «любезности»[2] `p_nice` — число в диапазоне от 0 до 39 со значением 20 по умолчанию. Чем меньше значение фактора «любезности», тем выше приоритет процесса. Пользователи могут повлиять на приоритет процесса при помощи изменения этого фактора, используя системный вызов nice (но только суперпользователь имеет полномочия увеличивать приоритет процесса);

— фактор «утилизации» `p_cpu` — число в диапазоне от 0 до 127. При создании процесса `p_cpu` инициализируется нулем. На каждом тике обработчик таймера увеличивает значение этого поля на единицу, пока не дойдет до максимального значения равного 127. Этот фактор позволяет системе динамически изменять приоритет процесса.

Каждую секунду ядро системы вызывает процедуру `schedcpu()`, которая уменьшает значение `p_cpu` каждого процесса исходя из фактора «полураспада».

[2]Степень любезности называется так потому, что одни пользователи могут быть поставлены в более выгодные условия другими пользователями посредством увеличения кем-либо из последних значения уровня любезности для своих менее важных процессов [17]

В системе SVR3 используется фиксированное значение этого фактора, равное 1/2. В системе 4.3BSD фактор полураспада считается по формуле:

$$decay = \frac{2 \cdot load\_average}{2 \cdot load\_average + 1} \tag{3.1}$$

где $load\_average$ — это среднее количество процессов, находящихся в состоянии готовности к выполнению, за последнюю секунду.

Процедура `schedcpu()` также пересчитывает приоритеты для режима задачи всех процессов по формуле:

$$p\_usrpri = PUSER + \frac{p\_cpu}{2} + 2 \cdot p\_nice \tag{3.2}$$

где $PUSER$ — базовый приоритет в режиме задачи, равный 50.

Таким образом, если процесс в последний раз (то есть до вытеснения его другим процессом) использовал большое количество процессорного времени, его `p_cpu` будет увеличен. Это приведен к росту `p_usrpri` и, следовательно, к понижению приоритета. Чем дольше процесс простаивает в очереди на выполнение, тем больше фактор «полураспада» уменьшает его `p_cpu`, что приводит к повышению приоритета. Такая схема предотвращает зависание низкоприоритетных процессов по вине операционной системы [17].

Применение этой схемы предпочтительнее процессам, которые осуществляют много операций ввода-вывода (имеют высокий приоритет, так как много времени проводят в ожидании завершения своих операций), в противоположность процессам, производящим много вычислений (используют большое количество процессорного времени, производя много вычислений, поэтому их приоритет после вытеснения понижается) [17].

Вывод

Таким образом, приоритет процесса может быть динамически пересчитан по следующим причинам:

— приоритет может быт повышен до соответствующего приоритета сна вследствие ожидания ресурса или события;

— вследствие изменения фактора «любезности» процесса системный вызовом `nice`;

— в зависимости от степени загруженности процессора процессом;

— вследствие ожидания процесса в очереди готовых к выполнению процессов.

3.2 Windows

В Windows реализуется приоритетная, вытесняющая система планирования, при которой всегда выполняется хотя бы один работоспособный поток с самым высоким приоритетом, с той оговоркой, что конкретные, имеющие высокий приоритет и готовые к запуску потоки могут быть ограничены процессами, на которых им разрешено или предпочтительнее всего работать [20].

После того как поток был выбран для запуска, он запускается на время, называемое квантом. Поток может не израсходовать свой квант времени: если становится готов к запуску поток с более высоким приоритетом, текущий выполняемый поток может быть вытеснен. Фактически, поток может быть выбран на запуск следующим и вытеснен еще даже до начала своего кванта времени [20].

Код планировщик Windows реализован в ядре. Но единого модуля или процедуры под названием «планировщик»не существует, код разбросан по ядру, где происходят события, связанные с планированием. Процедуры выполняющие эти обязанности, обобщенно называются диспетчером ядра. Диспетчеризации потоков могут потребовать следующие события [20]:

— поток становится готовым к выполнению;

— поток выходит из состояния выполнения из-за окончания его кванта времени;

— поток завершается или переходит в состояние ожидания;

— изменяется приоритет потока;

— изменяется родственность процессора потока.

Windows использует 32 уровня приоритета, от 0 до 31. Эти значения разбиваются на части следующим образом:

— шестнадцать уровней реального времени (от 16 до 31);

— шестнадцать изменяющихся уровней (от 0 до 15), из которых уровень 0 зарезервирован для потока обнуления страниц.

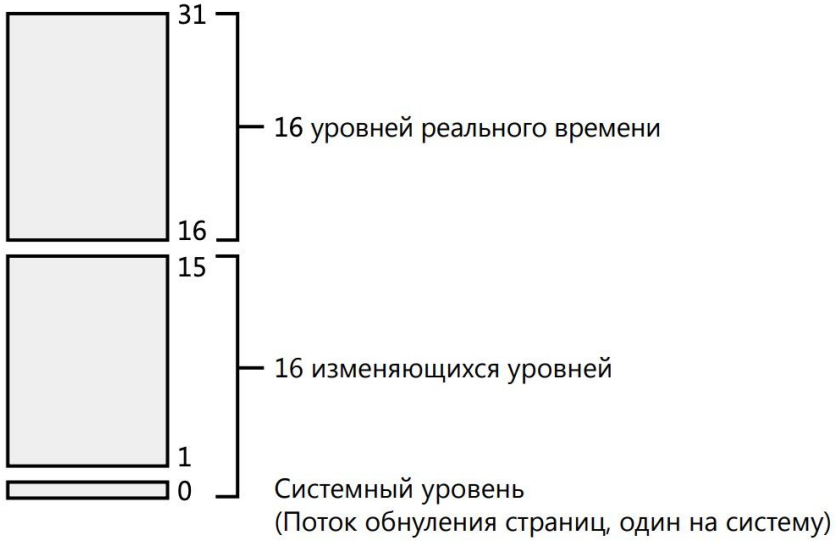

Рисунок 3.1 – Уровни приоритета потоков в Windows [20]

Уровни приоритетов потоков назначаются исходя из двух разных позиций: одной от Windows API и другой от ядра Windows. Сначала Windows API систематизирует процессы по классу приоритета, который им присваивается при создании:

Таблица 3.4 – Классы приоритетов в Windows API [20].

Приоритет (русс.)	**Приоритет (англ.)**	**Номер**
реального времени	real-time	4
высокий	high	3
выше обычного	above normal	7
обычный	normal	2
ниже обычного	below normal	5
простой	idle	1

Затем назначается относительный приоритет отдельных потоков внутри этих процессов. Здесь номера представляют изменение приоритета, применяющееся к базовому приоритету процесса. Другими словами, относительный приоритет — это приращение к базовому приоритету процесса.

Таблица 3.5 – Относительный приоритет потока внутри процесса [20].

Приоритет (русс.)	**Приоритет (англ.)**	**Номер**
критичный по времени	time-critical	15
наивысший	highest	2
выше обычного	above normal	1
обычный	normal	0
ниже обычного	below normal	-1
самый низший	lowest	-2
простой	idle	-15

Соответствие между приоритетами Windows API и ядра Windows приведено в таблице 3.6 [20].

Таблица 3.6 – Соответствие между приоритетами Windows API и ядра Windows

	real-time	**high**	**above normal**	**normal**	**below normal**	**idle**
time critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Приложения пользователя обычно запускаются с базовым приоритетом (`normal`), поэтому их исходный поток чаще всего выполняется с уровнем приоритета 8.

У процесса имеется только одно базовое значение приоритета, а у каждого потока имеется два значения приоритета: текущее (динамическое) и базовое.

Решения по планированию принимаются исходя из текущего приоритета. Система при определенных обстоятельствах на короткие периоды времени повышает приоритет потоков в динамическом диапазоне (от 1 до 15). Windows никогда не регулирует приоритет потоков в диапазоне реального времени (от 16 до 31), поэтому они всегда имеют один и тот же базовый и текущий приоритет [20].

Исходный базовый приоритет потока наследуется от базового приоритета процесса. Процесс по умолчанию наследует свой базовый приоритет у того процесса, который его создал [20].

Планировщик Windows периодически настраивает текущий приоритет потоков, используя внутренний механизм повышения приоритета. Во многих случаях это делается для уменьшения различных задержек и повышения восприимчивости системы, а также чтобы у потоков была возможность выполнения и освобождения ресурсов. Другими словами, это повышение применяется для предотвращения сценариев смены приоритетов и зависаний [21].

Повышение приоритета вступает в действие немедленно и может вызвать изменения в планировании процессора. Однако если поток использует весь свой следующий квант, то он теряет один уровень приоритета. Если же он использует второй полный квант, то он перемещается вниз еще на один уровень, и так до тех пор, пока не дойдет до своего базового уровня.

Сценарии повышения приоритета [21]:

— повышение вследствие событий планировщика или диспетчера (сокращение задержек);

— повышение вследствие завершения ввода-вывода (сокращение задержек — поток может вновь запуститься и начать новую операцию ввода-вывода). В таблице 3.7 приведены рекомендуемые значения повышения приоритета для устройств ввода-вывода;

— повышение вследствие ввода из пользовательского интерфейса (сокращение задержек и времени отклика);

— повышение приоритета владельца блокировки;

— повышение вследствие слишком продолжительного ожидания ресурса исполняющей системы (предотвращение зависания);

— повышение в случае, когда готовый к запуску поток не был запущен в течение определенного времени (чтобы исключить бесконечное откладывание процессов);

— повышение вследствие ожидания объекта ядра;

— повышение приоритета потоков первого плана после ожидания (улучшение отзывчивости интерактивных приложений);

— повышение приоритета после пробуждения GUI-потока (потоки-владельцы окон получают при пробуждении дополнительное повышение приоритета на 2);

— повышения приоритета, связанные с перезагруженностью центрального процессора (CPU Starvation);

— другие псевдоповышающие механизмы, проявляющие себя при проигрывании мультимедиа. В отличие от других повышений приоритета, эти механизмы применяются непосредственно в режиме ядра. Повышение приоритета проигрывания мультимедиа управляется службой планировщика класса мультимедиа MMCSS (это не является настоящим повышением).

Таблица 3.7 – Рекомендуемые значения повышения приоритета [21].

Устройство	**Приращение**
Диск, CD-ROM, параллельный порт, видео	1
Сеть, почтовый ящик, именованный канал, последовательный порт	2
Клавиатура, мышь	6
Звуковая плата	8

Рассмотрим последние два сценария подробней.

Перезагруженность центрального процессора

В Windows включен общий механизм ослабления загруженности центрального процессора, который называется диспетчером настройки баланса и является частью системного потока [22].

Один раз в секунду этот поток сканирует очередь готовых потоков в поиске тех из них, которые находятся в состоянии ожидания около 4 секунд. Если такой поток будет найден, диспетчер настройки баланса повышает его приоритет до 15 единиц и устанавливает квантовую цель эквивалентной тактовой частоте процессора при подсчете 3 квантовых единиц. Как только квант истекает, приоритет потока тут же снижается до обычного базового приоритета. Если поток не был завершен и есть готовый к запуску поток с более высоким уровнем приоритета, поток с пониженным приоритетом возвращается в очередь готовых потоков.

Для минимизации времени своей работы, диспетчер настройки баланса сканирует только 16 готовых потоков. Если на данном уровне приоритета имеется больше потоков, он запоминает то место, на котором остановился, и начинает с него при следующем проходе очереди. Кроме того, он за один проход повысит приоритет только 10 потоков. Если найдет 10 потоков, заслуживающих именно этого повышения, он прекратит сканирование на этом месте и начнет его с этого же места при следующем проходе.

MMCSS

MMCSS работает с вполне определенными задачами, включая следующие: аудио, захват, распределение, игры, проигрывание, аудио профессионального качества, задачи администратора многооконного режима [23].

Каждая из этих задач включает информацию о свойствах, отличающих их друг от друга. Одно из наиболее важных свойств для планирования потоков называется категорией планирования — Scheduling Category, которое является первичным фактором, определяющим приоритет потоков, зарегистрированных с MMCSS. На рисунке 3.2 показаны различные категории планирования.

Категория	Приоритет	Описание
High (Высокая)	23–26	Потоки профессионального аудио (Pro Audio), запущенные с приоритетом выше, чем у других потоков на системе, за исключением критических системных потоков
Medium (Средняя)	16–22	Потоки, являющиеся частью приложений первого плана, например Windows Media Player
Low (Низкая)	8–15	Все остальные потоки, не являющиеся частью предыдущих категорий
Exhausted (Исчерпавших потоков)	1–7	Потоки, исчерпавшие свою долю времени центрального процессора, выполнение которых продолжится, только если не будут готовы к выполнению другие потоки с более высоким уровнем приоритета

Рисунок 3.2 – Категории планирования изменение приоритета

Механизм, положенный в основу MMCSS, повышает приоритет потоков внутри зарегистрированного процесса до уровня, соответствующего их категории планирования и относительного приоритета внутри этой категории на гарантированный срок. Затем он снижает категорию этих потоков до Exhausted, чтобы другие, не относящиеся к мультимедийным приложениям потоки, также могли получить ресурс.

IRQL

Для обеспечения поддержки мультизадачности системы, когда исполняется код режима ядра, Windows использует приоритеты прерываний IRQL [24].

Прерывания обслуживаются в порядке их приоритета. При возникновении прерывания с высоким приоритетом процессор сохраняет информацию о состоянии прерванного потока и активизирует сопоставленный с данным прерыванием диспетчер ловушки. Последний повышает IRQL и вызывает процедуру обслуживания прерывания (ISR). После выполнения ISR диспетчер прерывания понижает IRQL процессора до исходного уровня и загружает сохраненные ранее данные о состоянии машины. Прерванный поток возобновляется с той точки, где он был прерван. Когда ядро понижает IRQL, могут начать обрабатываться ранее замаскированные прерывания с более низким приоритетом. Тогда вышеописанный процесс повторяется ядром для обработки и этих прерываний.

Вывод

Функции обработчика прерывания от системного таймера в защищенном режиме для ОС семейства Unix и для ОС семейства Windows схожи, так как эти ОС являются системами разделения времени. Общие основные функции:

— декремент кванта текущего процесса в Unix и декремент текущего потока в Windows.

— инициализация отложенных действий, которые относятся к работе планировщика (например пересчет приоритетов).

— декремент счетчиков времени (таймеров, часов, счетчиков времени отложенных действий, будильников реального времени)

Unix и Window — это системы разделения времени с вытеснением и динамическими приоритетами.

В ОС Unix приоритет пользовательского процесса может динамически пересчитываться, в зависимости от фактора «любезности», а также от `p_cpu` (результат последнего измерения использования процессора) и базового приоритета (`PUSER`). Приоритеты ядра — фиксированные величины.

В ОС Windows при создании процесса ему назначается базовый приоритет, относительно базового приоритета процесса потоку назначается относительный приоритет, таким образом, у потока нет своего приоритета. Приоритет потока пользовательского процесса может быть динамически пересчитан.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Вахалия Ю.* Unix изнутри. — СПб.: Питер, 2003. — С. 188.
2. *Руссинович М.*, *Соломон Д.* Внутреннее устройство Microsoft Windows. — СПб.: Питер, 2013. — С. 477.
3. Организация планирования в Unix [Электронный ресурс]. — Режим доступа: `https://studfile.net/preview/9792843/page:8/` (дата обращения 11.12.2022).
4. Использование таймеров [Электронный ресурс]. — Режим доступа: `https://habr.com/ru/company/ruvds/blog/512868/` (дата обращения 11.12.2022).
5. *Вахалия Ю.* Unix изнутри. — СПб.: Питер, 2003. — С. 189.
6. *Вахалия Ю.* Unix изнутри. — СПб.: Питер, 2003. — С. 186.
7. Управление памятью на основе свопинга [Электронный ресурс]. — Режим доступа: `https://studfile.net/preview/1640882/page:18/` (дата обращения 10.12.2022).
8. Управление процессами [Электронный ресурс]. — Режим доступа: `https://studfile.net/preview/1175400/page:28/` (дата обращения 11.12.2022).
9. *Вахалия Ю.* Unix изнутри. — СПб.: Питер, 2003. — С. 191.
10. Профилирование и трассировка ядра [Электронный ресурс]. — Режим доступа: `https://habr.com/ru/company/selectel/blog/280322/` (дата обращения 11.12.2022).
11. *Руссинович М.*, *Соломон Д.* Внутреннее устройство Microsoft Windows. — СПб.: Питер, 2013. — С. 104.
12. *Руссинович М.*, *Соломон Д.* Внутреннее устройство Microsoft Windows. — СПб.: Питер, 2013. — С. 131.
13. *Руссинович М.*, *Соломон Д.* Внутреннее устройство Microsoft Windows. — СПб.: Питер, 2013. — С. 94.

14. Планирование и диспетчеризация потоков [Электронный ресурс]. — Режим доступа: https://studfile.net/preview/4001841/page:5/ (дата обращения 11.12.2022).

15. *Рязанова Н. Ю.* Лекции по курсу «Операционные системы». — МГТУ им. Н. Э. Баумана, 2022.

16. *Руссинович М.*, *Соломон Д.* Внутреннее устройство Microsoft Windows. — СПб.: Питер, 2013. — С. 35.

17. *Вахалия Ю.* Unix изнутри. — СПб.: Питер, 2003. — С. 194-196.

18. *Робачевский А. М.* Операционная система Unix. — СПб.: БХВ-Петербург, 2002. — С. 224.

19. *Вахалия Ю.* Unix изнутри. — СПб.: Питер, 2003. — С. 82.

20. *Руссинович М.*, *Соломон Д.* Внутреннее устройство Microsoft Windows. — СПб.: Питер, 2013. — С. 476-479.

21. *Руссинович М.*, *Соломон Д.* Внутреннее устройство Microsoft Windows. — СПб.: Питер, 2013. — С. 500-504.

22. *Руссинович М.*, *Соломон Д.* Внутреннее устройство Microsoft Windows. — СПб.: Питер, 2013. — С. 510.

23. *Руссинович М.*, *Соломон Д.* Внутреннее устройство Microsoft Windows. — СПб.: Питер, 2013. — С. 517.

24. *Руссинович М.*, *Соломон Д.* Внутреннее устройство Microsoft Windows. — СПб.: Питер, 2013. — С. 111.