Государственное образовательное учреждение высшего профессионального образования
«Московский Государственный Технический Университет имени Н. Э. Баумана»

ОТЧЕТ

По лабораторной работе 1 (часть 2)

По курсу «Операционные системы»

Тема: «Функции обработчика прерываний от системного таймера»

Студент: Сиденко А. Г.

Группа: ИУ7-53Б

Преподаватель: Рязанова Н. Ю.

Москва, 2019

1. **Функции обработчика прерываний от системного таймера в Windows**
    1. **По тику**
        1. Обновление системного времени (инкремент);
        2. Декремент показания счетчика, отслеживающего продолжительность работы текущего потока;
        3. Декремент показаний счетчиков отложенных задач.
    2. **По главному тику**
        1. Инициализация диспетчера настройки баланса, который активизируется каждую секунду для возможной инициации событий, связанных с планированием и управлением памятью;
    3. **По кванту**
        1. Инициализация диспетчеризации потоков (добавление соответствующего объекта DPC в очередь).

2. **Функции обработчика прерываний от системного таймера в Unix/Linux**
    1. **По тику**
        1. Обновление статистики использования процессора текущим процессом;
        2. Обновление часов и других таймеров системы;
        3. Обработка отложенных вызов;
        4. Ведение счета тиков аппаратного таймера по необходимости.
    2. **По главному тику**
        1. Выполнение функций, относящихся к работе планировщика, таких как пересчет приоритетов и действия, выполняющиеся по истечении выделенного кванта времени;
        2. Пробуждение в нужные моменты системных процессов, такие как swapper и pagedaemon;
        3. Обработка сигнала тревоги (будильники).
    3. **По кванту**
        1. Посылка текущему процессу сигнала SIGXCPU, если тот превысил выделенную ему квоту(квант) использования процессора (процессорного времени).

3. **Пересчет динамических приоритетов (только у пользовательских процессов) в Windows**

В Windows реализуется приоритетная, вытесняющая система планирования, при которой всегда выполняется хотя бы один работоспособный (готовый) поток с самым высоким приоритетом.
Потоку разрешено работать в течение кванта времени. Но поток может и не израсходовать свой квант времени, поскольку в Windows реализуется вытесняющий планировщик: если становится готов к запуску другой поток с более высоким приоритетом, текущий выполняемый поток может быть вы- теснен еще до окончания его кванта времени.

Windows использует 32 уровня приоритета от 0 до 31.
— 16 уровней реального времени;
— 16 изменяющихся уровней, уровень 0 зарезервирован.

Уровни приоритета потоков назначаются исходя из двух разных позиций: одной от Windows API и другой от ядра Windows:
— Windows API систематизирует процессы по классу приоритета, который им присваивается при создании: Реального времени — Real-time (4), Высокий — High (3), Выше обычного — Above Normal (7), Обычный — Normal (2), Ниже обычного — Below Normal (5) и Простоя — Idle (1).
— Затем назначается относительный приоритет отдельных потоков внутри этих процессов. Здесь номера представляют изменение приоритета, применяющееся к базовому приоритету процесса: 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) и Простоя — Idle (–15).
Таким образом, у каждого потока два приоритета: текущий и базовый. Решения, связанные с планированием, принимаются на основе текущего приоритета. Отображение Windows-приоритета на внутренние номерные приоритеты Windows показано в таблице.

Класс приоритета/ Относительный приоритет	**Realtime**	**High**	**Above**	**Normal**	**Below Normal**	**Idle**
Time Critical (+ насыщение)	31	15	15	15	15	15
Highest (+2)	26	15	12	10	8	6
Above Normal (+1)	25	14	11	9	7	5
Normal (0)	24	13	10	8	6	4
Below Normal (–1)	23	12	9	7	5	3
Lowest (–2)	22	11	8	6	4	2
Idle (– насыщение)	16	1	1	1	1	1

В то время как у процесса имеется только одно базовое значение приоритета, у каждого потока имеется два значения приоритета: текущее и базовое. Решения по планированию принимаются исходя из текущего приоритета.

Однако система при определенных обстоятельствах на короткие периоды времени повышает приоритет потоков в динамическом диапазоне (от 0 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.

Исходный базовый приоритет потока наследуется из базового приоритета процесса. Процесс по умолчанию наследует свой базовый приоритет у того процесса, который его создал.
Как правило, пользовательские приложения и службы запускаются с обычным базовым приоритетом (normal), поэтому их исходный поток чаще всего выполняется с уровнем приоритета 8.
Повысить или понизить приоритет потока в динамическом диапазоне можно в любом приложении, но у вас должны быть привилегии, позволяющие повышать приоритет, использующийся при планировании для ввода значения в пределах динамического диапазона.

Сценарии повышения приоритета (и их причины):
— Повышение вследствие событий планировщика или диспетчера (сокращение задержек).
Следующие сценарии имеют дело с объектом диспетчера, входящим в сигнальное состояние:
В очередь потока поставлен APC-вызов, событие установлено или отправило сигнал, системное время изменилось (таймеры должны быть перезапущены), мьютекс или семафор освобождены, изменилось состояние потока.
— Повышения связанные с завершением ожидания.
Пытаются уменьшить время задержки между потоком, пробуждающимся по сигналу объекта. Поскольку событие, которого ждал поток, может дать информацию на данный момент времени. В противном случае информация может стать неактуальной.
— Повышение вследствие завершения ввода-вывода (сокращение задержек).
Windows дает временное повышение приоритета при завершении определенных операций ввода-вывода, при этом потоки, ожидавшие ввода-вывода, имеют больше шансов сразу же запуститься и обработать то, чего они ожидали.
Рекомендуемые значения повышения приоритета представлены в таблице.

Устройство	**Повышение приоритета**
Жесткий диск, привод компакт-дисков, параллельный порт, видеоустройство	1
Сеть, почтовый слот, именованный канал, последователь-ный порт	2
Клавиатура, мышь	6
Звуковое устройство	8

— Повышение вследствие ввода из пользовательского интерфейса (сокращение задержек и времени отклика).

Потоки, владеющие окнами, получают при пробуждении дополнительное повышение приоритета на 2 уровня, из-за активности системы работы с окнами, например при поступлении сообщений. Приоритет для создания преимуществ, содействия интерактивным приложениям.

— Повышение вследствие слишком продолжительного ожидания ресурса исполняющей системы (предотвращение зависания).

Когда поток пытается получить ресурс исполняющей системы, который уже находится в исключительном владении другого потока, он должен войти в состояние ожидания до тех пор, пока другой поток не освободит ресурс. Для ограничения риска взаимных исключений исполняющая система выполняет это ожидание, не входя в бесконечное ожидание ресурса, а интервалами по 5 секунд.

Если по окончании этих 5 секунд ресурс все еще находится во владении, исполняющая система пытается предотвратить зависание центрального процессора путем завладения блокировкой диспетчера, повышения приоритета потока или потоков, владеющих ресурсом, до значения 14, перезапуска их квантов и выполнения еще одного ожидания.

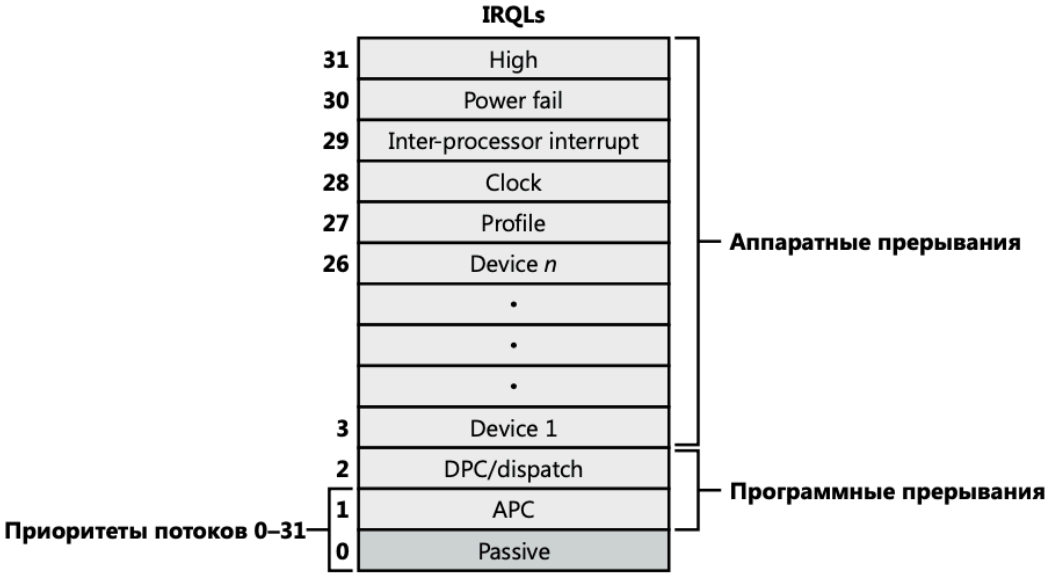
— Повышение в случае, когда готовый к запуску поток не был запущен в течение определенного времени (предотвращение зависания и смены приоритетов).

Один раз в секунду диспетчер настройки баланса сканирует очередь готовых потоков в поиске тех из них, которые находятся в состоянии ожидания (то есть не были запущены) около 4 секунд. Если такой поток будет найден, диспетчер настройки баланса повышает его приоритет до 15 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, где он опять становится подходящим для еще одного повышения приоритета, если будет оставаться в очереди следующие 4 секунды.

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 прерванный поток возобновляется с той точки, где он был прерван.

Потоки обычно запускаются на уровне IRQL0 (который называется пассивным уровнем, потому что никакие прерывания не обрабатываются и ника- кие прерывания не заблокированы) или на уровне IRQL1 (APC-уровень). Код пользовательского режима всегда запускается на пассивном уровне. Поэтому никакие потоки пользовательского уровня независимо от их приоритета не могут даже заблокировать аппаратные прерывания (хотя высокоприоритетные потоки реального времени могут заблокировать выполнение важных системных потоков).

См. рисунок.


IRQLs
31
High
30
Power fail
29
Inter-processor interrupt
28
Clock
27
Profile
26
Device n
3
Device 1
2
DPC/dispatch
1
APC
0
Passive
Аппаратные прерывания
Программные прерывания
Приоритеты потоков 0–31

4. **Пересчет динамических приоритетов (только у пользовательских процессов) в Unix/Linux**

Традиционное ядро UNIX является строго невытесняющим. Если процесс выполняется в режиме ядра (например, в течение исполнения системного вызова или прерывания), то ядро не заставит такой процесс уступить процессор какому-либо высокоприоритетному процессу. Выполняющийся процесс может только добровольно освободить процессор в случае своего блокирования в ожидании ресурса, иначе он может быть вытеснен при переходе в режим задачи. Реализация ядра невытесняющим решает множество проблем синхронизации, связанных с доступом нескольких процессов к одним и тем же структурам данных ядра.

Механизм планирования процессов в UNIX базируется на приоритетах процесса. Каждый процесс обладает приоритетом планирования, изменяющимся с течением времени. Планировщик всегда выбирает процесс с наивысшим приоритетом. Изменение приоритетов процессов происходит динамически а основе количество используемого ими процессорного времени. Если какой-то высокоприоритетный процесс готов к выполнению, планировщик вытеснит текущий процесс, даже если тот не израсходовал свой квант времени.

Приоритет процесса задается любым целым числом, лежащим в диапазоне от 0 до 127. Чем меньше такое число, тем выше приоритет. В Unix приоритеты от 0 до 49 зарезервированы для ядра, следовательно, прикладные процессы могут обладать приоритетом в диапазоне 50–127. Потоки реального времени внутри системы Linux представлены приоритетами от 0 до 99.

Структура proc содержит следующие поля, 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

Планировщик использует p_pri для принятия решения о том, какой процесс отправить на выполнение.
Когда процесс находится в режиме задачи p_pri и p_usrpri идентичны.
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.

Процессу, ожидающему недоступного в данный момент ресурса, система определяет значение приоритет сна выбираемое ядром из диапазона 0-49 и связанное с событием, вызвавшее это состояние.
Когда замороженный процесс просыпается значение p_pri, устанавливается равным приоритету сна события или ресурса, на котором он был заблокирован.
Системные приоритеты сна представлены в таблице ниже.

Событие	Приоритет 4.3BSD UNIX	Приоритет SCO UNIX
Ожидание загрузки в память сегмента/страницы (свопинг/страничное замещение)	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 — низкоприоритетное состояние сна	40	66

Поскольку приоритеты ядра выше, такие процессы будут назначены на выполнение раньше, чем другие, функционирующие в режиме задачи. Такой подход позволяет системным вызовам быстро завершать работу, что является желательным, так как процессы во время выполнения вызова могут занимать некоторые ключевые ресурсы системы, не позволяя другим ими пользоваться.
Когда процесс завершил выполнение системного вызова, его приоритет сбрасывается обратно в значение текущего приоритета в режиме задачи, произойдет переключение контекста.

Приоритет в режиме задачи зависит от двух факторов: любезности (nice), число от 0 до 39, и результата последнего измерения использования процессора.
Увеличение значения nice приводит к уменьшению приоритета, при этом только суперпользователь может поднять приоритет.

Системы разделения времени пытаются выделить время таким образом, чтобы конкурирующие процессы получили его примерно в равных количествах, требуется следить за использованием процессора каждым процессом.
Поле p_cpu при создании процесса инициализируются нулем.
На каждом тике обработчик таймера увеличивает p_cpu на единицу для текущего процесса до максимального значения 127.

Каждую секунду ядро системы вызывает процедуру pyschedcpu() (запускаемую через отложенный вызов), которая уменьшает значение p_cpu каждого процесса исходя из фактора «полураспада» (decay factor).
По следующей формуле:

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

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:

$$p\_usrpri = PUSER + \frac{p\_cpu}{4} + (2 \cdot p\_nice)$$

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

Если процесс до вытеснения другим процессом использовал большое количество процессорного времени, его p_cpu будет увеличен. Это приведет к росту значения 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 его приоритета.

Фактор использования процессора обеспечивает справедливость и равенство при планировании процессов в режиме разделения времени. О
Приоритеты могут повышаться или понижаться в рамках этого диапазона в зависимости от того, сколько процессорного времени эти процессы получали в последний раз. Если приоритеты будут меняться слишком медленно, процессы, начавшие работу с низким приоритетами, сохранят их в течение долгого периода времени, что приведет к их фактическому зависанию.
Фактор полураспада обеспечивает экспоненциально взвешенное среднее значение использования процессора в течение всего периода функционирования процесса.

Выводы

Как видно, в ОС Windows и Unix на обработчик системного таймера возложены схожие функции. Это обусловлено тем, что обе ОС относятся к системам разделения времени с динамическим приоритетом.

Однако в планировании процессорного времени эти системы сильно различаются.

Классический Unix имеет невытесняющее ядро, а Windows является полностью вытесняющей.

Алгоритмы планирования основаны на очередях, но взаимодействия планировщика и потоков различны: планировщик в Windows вызывается самими процессами, тогда как в Unix/Linux за это отвечает ядро.