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

(часть 2)

по курсу «Операционные системы»

на тему:

«Обработчик прерывания от системного таймера и пересчет динамических приоритетов»

Студент Смирнов И.В.

Группа ИУ7-52Б

Преподаватели Рязанова Н.Ю.

2024 г.

СОДЕРЖАНИЕ

1 Функции обработчика прерывания от системного таймера

1.1 Windows

По тику

— инкремент счетчика системного времени;

— декремент кванта;

— декремент счетчиков времени отложенных задач;

— если активен механизм профилирования ядра, то инициализация отложенного вызова обработчика ловушки профилирования ядра добавлением объекта в очередь DPC (обработчик ловушки профилирования регистрирует адрес команды, выполнявшейся на момент прерывания);

По главному тику

— освобождение объекта «событие», которое ожидает диспетчер настройки баланса. Диспетчер настройки баланса по событию от таймера сканирует очередь готовых процессов и повышает приоритет процессов, которые находились в очереди в состоянии ожидания дольше 4 секунд.

По кванту

— инициализация диспетчеризации потоков добавлением соответствующего объекта в очередь DPC.

1.2 UNIX

По тику

— инкремент счетчика тиков аппаратного таймера;

— обновление системного времени (времени дня) и других связанных с ним таймеров;

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

— декремент счетчика времени, оставшегося до отправления на выполнение отложенных вызовов, при достижении нулевого значения счетчика — выставление флага, указывающего на необходимость запуска обработчика отложенного вызова;

— декремент кванта потока.

По главному тику

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

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

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

  – SIGALRM — сигнал, посылаемый процессу по истечении времени, заданного функцией alarm() (будильник реального времени);
  – SIGPROF — сигнал, посылаемый процессу по истечении времени, заданного в таймере профилирования (будильник профиля процесса);
  – SIGVTALRM — сигнал, посылаемый процессу 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.

2 Пересчет динамических приоритетов

В ОС семейств UNIX и Windows динамически пересчитываться могут только приоритеты пользовательских процессов.

2.1 Windows

В Windows реализуется приоритетная, вытесняющая система планирования, при которой всегда выполняется хотя бы один готовый поток с самым высоким приоритетом.

После того как поток был выбран для запуска, он запускается на время, называемое квантом. Но поток может и не израсходовать свой квант времени: если становится готов к запуску другой поток с более высоким приоритетом, текущий выполняемый поток может быть вытеснен.

Единого модуля под названием «планировщик» не существует. Процедуры, выполняющие обязанности по диспетчеризации, обобщенно называются диспетчером ядра. Диспетчеризации потоков могут потребовать следующие события:

— 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.

Уровни приоритета потоков назначаются из двух разных позиций: от Windows API и от ядра Windows.

Windows API систематизирует процессы по классу приоритета, который присваивается им при их создании:

— 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 потоков в рамках 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));

— простоя (idle (-15)).

Уровень, критичный по времени, и уровень простоя (+15 и -15) называются уровнями насыщения и представляют конкретные применяемые уровни вместо смещений. Относительный приоритет — это приращение к базовому приоритету процесса.

Соответствие между приоритетами Windows API и ядра Windows приведено в таблице 1.

Таблица 1 – 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.

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

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.

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:

— 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 2 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 управляются службой

планировщика класса мультимедиа MMCSS (это не является настоящим повышением, служба просто устанавливает по необходимости новые базовые приоритеты для потоков).

Таблица 2 – Рекомендуемые значения повышения приоритета.

Устройство	Приращение
Жесткий диск, привод компакт-дисков, параллельный порт, видеоустройство	1
Сеть, почтовый слот, именованный канал, последовательный порт	2
Клавиатура, мышь	6
Звуковое устройство	8

Рассмотрим последние 2 сценария подробнее.

MMCSS

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

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 1 показаны различные категории планирования.

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.

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 1 – Категории планирования

IRQL

Для обеспечения поддержки мультизадачности системы, когда исполняется код режима ядра, Windows использует приоритеты прерываний IRQL – Interrupt Request Level.

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 – Interrupt Service Routine.

После выполнения ISR диспетчер прерывания понижает IRQL процессора до исходного уровня 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.

В ядре IRQL-уровни представлены в виде номеров от 0 до 31, где более высоким номерам соответствуют прерывания с более высоким приоритетом.

На рисунке 2 показаны уровни IRQL.

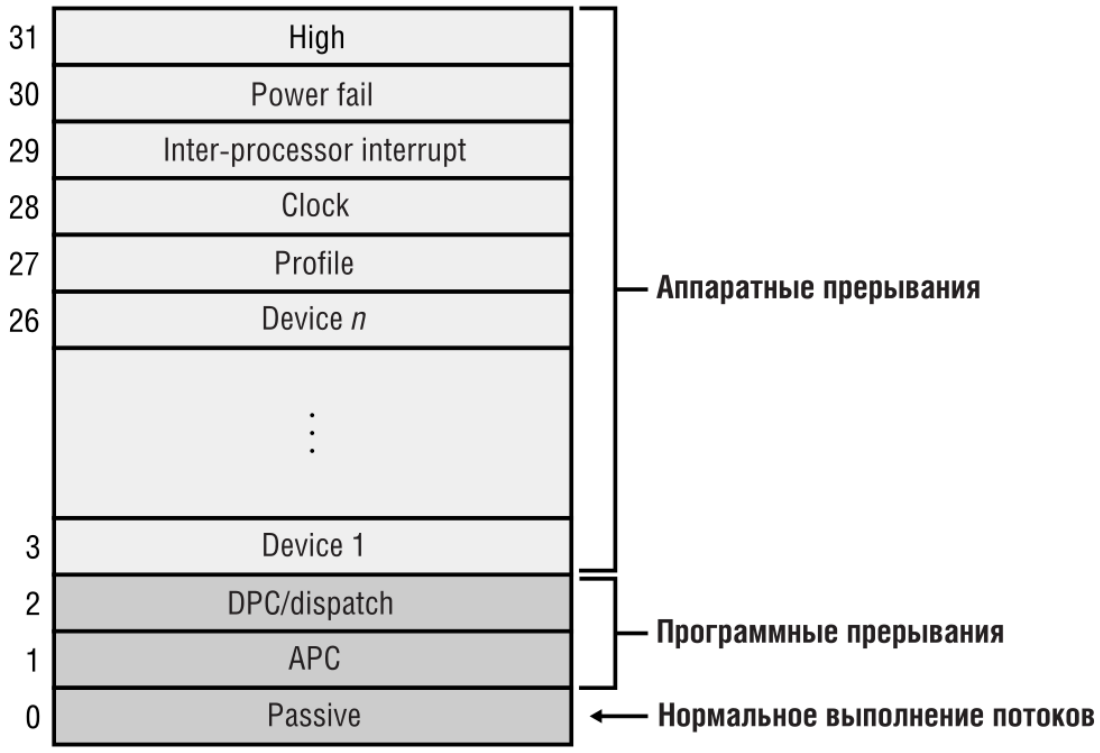

Рисунок 2 – Категории планирования

2.2 UNIX

В современных системах UNIX ядро является 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. В первую очередь выполняются 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 число, тем выше приоритет). Приоритеты от 0 до 49 зарезервированы для ядра, они являются фиксированными величинами. Прикладные процессы могут обладать приоритетом в диапазоне 50 – 127.

В традиционных системах UNIX приоритет процесса определяется двумя факторами:

— фактор «любезности» – целое число в диапазоне от 0 до 39 со значени-

ем 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 — степень загруженности CPU в момент последнего обслуживания им процесса. Этот фактор позволяет системе динамически изменять приоритет процесса.

Планировщик использует p_pri для принятия решения о том, какой процесс направить на выполнение. У процесса, находящегося в режиме задачи, значения p_pri и p_usrpri идентичны. Значение текущего приоритета p_pri может быть повышено планировщиком для выполнения процесса в режиме ядра. Когда процесс просыпается после блокировки в системном вызове, его приоритет временно повышается. При создании процесса поле p_cpu инициализируется нулем. На каждом тике обработчик таймера увеличивает поле p_cpu текущего процесса на единицу, до максимального значения, равного 127.

Таблица 3 – Таблица приоритетов в системе 4.3BSD UNIX

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

Каждую секунду обработчик прерывания таймера инициализирует отложенный вызов процедуры schedcpu(), которая уменьшает значение p_cpu

Таблица 4 – Системные приоритеты сна в системах 4.3BSD UNIX и SCO UNIX

Событие	Приоритет 4.3BSD UNIX	Приоритет SCO UNIX
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

каждого процесса исходя из фактора «полураспада» (decay factor). В 4.3BSD для расчета полураспада применяется следующая формула:

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

где load_average — это среднее количество процессов в состоянии готовности к выполнению, за последнюю секунду. Процедура schedcpu() также пересчитывает приоритеты 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.

Таким образом, если процесс до вытеснения другим процессом использовал большое количество процессорного времени, его p_cpu будет увеличен, что приведет к увеличению значения p_usrpri и, следовательно, к понижению приоритета. Чем дольше процесс простаивает в очереди на выполнение, тем меньше его p_cpu, и, соответственно, выше приоритет.

Такая схема позволяет исключить бесконечное откладывание низкоприоритетных процессов. При ее применении процессы, которые осуществляют много операций ввода-вывода, получают преимущество, так как они проводят много времени в ожидании завершения этих операций. Напротив, те про-

цессы, что производят много вычислений, используют большое количество процессорного времени, и их приоритет после вытеснения будет понижен.

Потоки UNIX

Некоторым приложениям требуется выполнять несколько крупных независимых задач, используя одно и то же адресное пространство и другие ресурсы. Традиционные системы UNIX либо вынуждают выполнять задачи последовательно, либо использовать неуклюжие и малоэффективные механизмы параллелизма. Традиционная модель не позволяет в полной мере задействовать возможности многопроцессорных систем, поскольку процесс одновременно может использовать только один процессор. Приложению приходится создавать несколько процессов и искать способы совместного использования памяти и ресурсов, а также синхронизации.

Современные системы UNIX предлагают решение приведенных проблем при помощи различных технологий нитей, реализованных в ОС с целью поддержки параллельного выполнения заданий. В каждом варианте UNIX для их обозначения применяются свои термины, например нити ядра, прикладные нити, прикладные нити, поддерживаемые ядром, C-threads, pthreads и «легковесные» процессы (lightweight processes).

ВЫВОДЫ

Операционные системы семейств Windows и UNIX являются системами разделения времени с динамическими приоритетами и вытеснением, поэтому обработчик прерывания от системного таймера выполняет в этих системах схожие функции:

— декремент кванта потока в UNIX и процесса Windows;

— инициализация отложенных задач, относящихся к работе планировщика (например, пересчет динамических приоритетов);

— инкремент счетчиков времени и декремент счетчиков таймеров, будильников реального времени, счетчиков времени отложенных задач.

Пересчет динамических приоритетов осуществляется только для пользовательских процессов, чтобы избежать бесконечного откладывания.

В UNIX приоритет пользовательского процесса (процесса в режиме задач) может динамически пересчитываться. Пользователь может только понижать приоритет, при этом значение поля nice увеличивается. Для повышения приоритета нужны права superuser. Приоритеты ядра – фиксированные величины.

В Windows при создании процесса ему назначается базовый приоритет, относительно базового приоритета процесса потоку назначается относительный приоритет.