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»

Отчёт по 2 части лабораторной работы №1 по курсу "Операционные системы"

Тема Динамические приоритеты и функции обработчика прерывания от системного таймера

Студент Морозов Д.В.

Группа ИУ7-52Б

Оценка (баллы) _____

Преподаватель Рязанова Н.Ю.

Москва — 2022 г.

Оглавление

1 Функции обработчика прерывания от системного таймера

1.1 Функции обработчика прерывания от системного таймера в ОС семейства UNIX

По тику:

— увеличение значения поля *p_cpu* дескриптора текущего процесса до максимального значения, равного 127;

— инкремент счётчика реального времени;

— декремент кванта текущего потока;

— инкремент счётчика времени в будильниках 3 типов.

По главному тику:

— пробуждение системных процессов, таких как *pagedaemon* и *swapper*;

— обработка отложенных вызовов;

— декремент счётчика времени до отправки одного из сигналов среди SIGALRM, SIGPROF, SIGVTALRM.

По истечении кванта обработчик прерывания по кванту посылает текущему процессу сигнал SIGXCPU.

Чаще всего квант времени в традиционных системах UNIX равен 100 миллисекундам.

1.2 Функции обработчика прерывания от системного таймера в ОС семейства Windows

По тику:

— декремент кванта текущего потока;

— декремент счётчика отложенных задач;

— инкремент счётчика реального времени;

— инициализация отложенного вызова обработчика ловушки профилирования ядра с помощью постановки объекта в очередь DPC (Deferred Procedure Call), если активен механизм профилирования ядра.

По главному тику:

— освобождение объекта ядра «событие»;

— добавление в очередь DPC объекта диспетчера настройки баланса.

По истечении кванта обработчик прерывания инициализирует диспетчеризацию потоков.

Чаще всего на клиентской версии ОС Windows поток выполняется в течение двух интервалов таймера, на серверных — 12 интервалов. Однако значение кванта может варироваться от системы к системе.

2 Динамические приоритеты

В ОС семейств UNIX или Windows динамическими могут быть только пользовательские приоритеты, то есть приоритеты пользовательских процессов. Приоритеты процессов ядра являются фиксированными величинами.

Во всех системах учитываются время простоя процесса в очереди готовых процессов и причина блокировки процесса.

Планировщик (scheduler) — часть ядра, распределяющая процессорное время между процессами или потоками.

2.1 Динамические приоритеты в ОС семейства UNIX

В традиционных системах UNIX планировщик использует алгоритм Round Robin. Процессы, имеющие одинаковые приоритеты, будут выполняться циклически друг за другом, и каждому из них будет отведён для этого квант времени, обычно равный 100 миллисекундам. Если какой-то процесс с более высоким приоритетом становится выполняемым, то он вытеснит текущий процесс (если текущий процесс не находится в режиме ядра) даже в том случае, если текущий процесс не исчерпал отведённый ему квант времени.

Также в традиционных системах UNIX приоритет процесса определяется двумя факторами: фактором «любезности» и фактором утилизации. Фактор утилизации — время последнего обслуживания процесса процессором. Ядро периодически повышает приоритет процесса, пока тот не выполняется. а после того как процесс получит процессорное время, его приоритет будет понижен.

Приоритеты ядра являются фиксированными величинами и зависят только от причины засыпания процесса, поэтому приоритеты ядра также известны как приоритеты сна.

В таблице 2.1 приведены приоритеты сна в ОС 4.3BSD UNIX.

Таблица 2.1 – Приоритеты сна в ОС 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 освобождение 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 блокированного ресурса
PSLEP	40	Ожидание сигнала

В отличие от ОС семейства Windows, в ОС семейства UNIX планировщик является отдельным компонентом ОС. Изначально система UNIX создавалась как система разделения времени, что означает одновременное выполнение нескольких процессов.

Далее будет обсуждаться традиционный алгоритм планирования, применимый как в SVR3, так и в 4.3BSD.

Планировщик выбирает процессы, обаладающие наиболее высоким приоритетом. Для диспетчеризации процессов с равным приоритетом применяется вытесняющее квантование времени. Системой учитывается количество используемого процессами процессорного времени.

Традиционной ядро UNIX является невытесняющим.

Приоритет процесса задаётся любым целым числом, лежащим в диапозоне от 0 до 127. Чем меньше такое число, тем выше приоритет. Приоритеты от 0 до 40 зарезервированы ядром, остальные приоритеты используются прикладными процессами.

Структура *proc* содержит следующим 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.

Поля *p_pri* и *p_usrpri* применяются для различных целей. Поле *p_pri* используется планировщиком для принятия решения о том, какой процесс надо направить на выполнение. Если процесс находится в режиме задачи, то в его дескрипторе значение *p_pri* идентично *p_usrpri*. Когда процесс проснётся после блокировки в системном вызове, его приоритет будет временно повышен для того, чтобы дать ему предпочтение для выполнения в режиме ядра. Значит, планировщик использует поле *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.

Также ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может заблокироваться. Приоритет сна является величиной, определяемой для ядра, и поэтому располагается в диапазоне 0–49.

В таблице 2.2 приведены системные приоритеты сна.

Таблица 2.2 – Системные приоритеты сна

Событие	Приоритет 4.3BSD UNIX	Приоритет SCO UNIX
Ожидание загрузки в память сегмента /страницы	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

Когда процесс пробуждается, ядро устанавливает значение текущего приоритета процесса равным приоритету сна. Поскольку приоритет такого процесса находится в системном диапазоне и выше, чем любой приоритет

режима задачи, вероятность предоставления процессу вычислительных ресурсов весьма велика.

После завершения системного вызова перед возвращением в режим задачи ядро восстанавливает приоритет режима задачи, сохранённый перед выполнением системного вызова.

Каждую секунду ядро системы вызывает процедуру *schedcpu()*, которая уменьшает значение p_cpu каждого процесса исходя из фактора «полураспада» (decay factor). В системе SVR3 фактор «полураспада» равен $\frac{1}{2}$, однако в 4.3BSD для расчёта фактора используется следующая формула:

$$decay = \frac{2 \cdot load\_average}{2 \cdot load\_average + 1}, \tag{2.1}$$

где $load\_average$ — это среднее количество процессов, находящихся в состоянии готовности к выполнению, за последнюю секунду. Процедура *schedcpu()* пересчитывает приоритеты для всех процессов находящихся в режиме задачи по формуле:

$$p\_usrpri = PUSER + \frac{p\_cpu}{4} + 2 \cdot p\_nice, \tag{2.2}$$

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

В результате, если процесс до вытеснения другим процессом использовал большое количество процессорного времени, его p_cpu будет увеличен, что приведёт к росту значения p_usrpri, а это повлечёт за собой к понижению приоритета процесса. Такая схема предотвращает бесконечное откладывание.

2.2 Динамические приоритеты в ОС семейства Windows

В Windows реализуется приоритетная, вытесняющая система планирования, при которой выполняется хотя бы 1 поток с самым высоким приоритетом.

После выбора потока для запуска, он запускается на время, называемое квантом. Но поток может и не израсходовать свой квант времени, так

как в Windows реализуется вытесняющий планировщик: если становится готов к запуску другой поток с более высоким приоритетом, текущий выполяемый поток может быть вытеснен ещё до окончания его кванта времени. Возможна ситуация, при которой поток может быть выбран на запуск следующим и вытеснен ещё даже до начала своего кванта времени.

ОС назначает приоритет процессу и приоритет потоку относительно приоритета процесса.

В ОС семейства Windows нет отдельного модуля планировщика. Функции планировщика выполняют различные участки кода ядра системы, которые называются диспетчером ядра.

Также стоит отметить «родственность процессора» — явление, при котором готовые к запуску потоки с самым высоким приоритетом могут быть ограничены процессорами, на которых им разрешено или предпочтительнее всего работать.

По совокупности следующих обстоятельств Windows определяет, какой поток должен быть запущен следующим на логическом процессоре, или на каком логическом процессоре он не должен быть запущен:

1) Поток становится готовым к выполнению.

2) Поток выходит из состояния выполнения из-за окончания его кванта времени.

3) Изменяется приоритет потока.

4) Изменяется родственность процессора потока и он больше уже не может быть запущен на том процессоре, на котором выполнялся.

Решения по планированию принимаются только на основе потоков. Windows использует 32 уровня приоритетов, от 0 до 31.

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

— 16 уровней (от 0 до 15), 0-ой уровень зарезервирован для потока обнуления страниц.

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 (номера представляют внутренний индекс PROCESS_PRIORITY_CLASS, распознаваемый ядром).

Процесс наследует базовый приоритет от процесса, которые его создал. Потока наследует базовый приоритет от своего процесса. Это поведение может быть изменено.

Классы приоритета процесса:

— реального времени — Real-time (4);

— высокий — High (3);

— выше обычного — Above normal (7);

— обычный — Normal (2);

— ниже обычного — Below Normal (5);

— простой — Idle (1).

Далее назначается относительный приоритет отдельным потокам внутри этих процессов. Номера представляют изменение приоритета, применяющееся к базовому приоритету процесса.

Относительные приоритеты потоков:

— 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).

В Windows API каждый поток имеет базовый приоритет, который является функцией класса приоритета процесса и относительного приоритета потока. В ядре класс приоритета процесса преобразуется в базовый приоритет. Затем применяется относительный приоритет потока в качестве разницы для этого базового приоритета.

Далее будет представлено отображение Windows-приоритета на внутренние номерные приоритеты Windows.

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

Рисунок 2.1 – Отображение Windows-приоритета на внутренние номерные приоритеты Windows

Потоки с относительным приоритетом Time-Critical и Idle сохраняют номерные приоритеты вне зависимости от класса приоритета процесса (Только если этим классом не является Realtime). Дело в том, что Windows API требует насыщения приоритета от ядра. Ядро распознает запрос на насыщение и это приведёт к тому, что при положительном насыщении поток будет иметь наивысший приоритет внутри его класса приоритета, при отрицательном — наименьший приоритет.

Планировщик Windows периодически настраивает текущий приоритет потоков, используя внутренний механизм повышения приоритета. Повышение применяется для предотвращения сценариев смены приоритетов и зависаний.

Ряд сценариев повышения приоритета:

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

— повышение вследствие завершения ввода-вывода (сокращение задержек);

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

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

— повышение в случае, когда готовый к запуску поток не был запущен в течение определённого времени (предотвращение зависания и смены приоритетов);

— повышение приоритета владельца блокировки;

— повышение, связанное с завершением ожидания;

— повышение приоритета потоков первого плана после ожидания;

— повышение приоритета, связанные с перезагруженностью центрального процессора.

Windows даёт временное повышение приоритета при завершении определённых операций ввода-вывода потокам, ожидающих ввод-вывод.

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

Рисунок 2.2 – Рекомендуемые от Microsoft значения повышения приоритета

Ядро реализует код подачи сигнала в API-функции *IoCompleteRequest*, используя либо APC, либо событие. При передаче драйвером параметра *IO_DISK_INCREMENT* в функцию *IoCompleteRequest* для асинхронного чтения диска ядро вызывает *KeInsertQueueApc* с параметром повышения *IO_DISK_INCREMENT*. Когда ожидание потока разрушается из-за APC, он получает повышение, равное 1.

Также стоит отметить отдельно повышение приоритета проигрывания мультимедиа. MMCSS (MultiMedia Class Scheduler Service) — служба

планировщика класса мультимедиа. Упомянутое повышение **не является настоящим повышением**, служба устанавливает новые базовые приоритеты для потоков.

Одно из наиболее важных свойств для планирования потоков, зарегистрированных с MMCSS, называется категорией планирования. Далее представлены 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 2.3 – 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 время. По истечении времени он снижает категорию этих потоков до Exhausted.

Windows устававливает свою собственную схему приоритетности прерываний — уровни запросов прерываний (IRQL, Interrupt Request Level). Уровни представлены в виде номеров от 0 до 31 на системах x86. Более высоким номерам соответствуют прерывания с более высоким приоритетом.

Прерывания обслуживаются в порядке их приоритета, и прерывания с более высоким уровнем приоритета получают преимущество в обслуживании. Процессор сохраняет состояние прерванного потока и запускает связанный с прерыванием диспетчер системных прерываний при возникновении прерывания с более высоким уровнем приоритета.

В Windows включён общий механизм ослабления загруженности центрального процессора, который называется диспетчером настройки баланса.

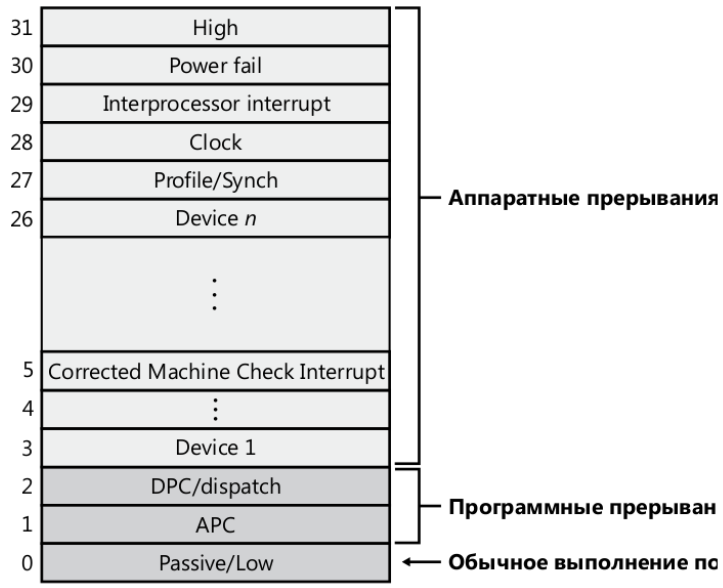

Рисунок 2.4 – Уровни запросов прерываний для архитектуры x86

Раз в секунду системный поток, который существует для выполнения функций управления памятью, сканирует очередь готовых процессов в поиске тех, 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 эквивалетной тактовой частоте процессора при подсчёте 3 квантовых единиц. По истечении кванта приоритет потока будет снижен до обычного базового. Однако диспетчер на самом деле не сканирует все потоки, а только 16 готовых потоков и за 1 проход может повысить приоритет только 10 потоков.

Заключение

Операционные системы семейств Windows и UNIX являются системами разделения времени с динамическими приоритетами с вытеснением, поэтому функции обработчика прерывания от системного таймера указанных операционных систем схожи:

— декремент кванта текущего потока;

— инкремент счётчика реального времени;

— инициализация отложенных действий, относящихся к работе планировщика.

В ОС семейств Windows и UNIX единицей диспетчеризации является поток. Пересчёт приоритетов пользовательских процессов необходим для того, чтобы предотвратить бесконечное откладывание процессов.