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 лабораторной работе №1 (часть 2)

Название «Функции обработчика прерывания от системного таймера в ОС Unix и Windows»

Дисциплина «Операционные системы»

Студент ИУ7-55Б _____ Клименко А.К.
(подпись, дата) (Фамилия И.О.)

Преподователь _____ Рязанова Н.Ю.
(подпись, дата) (Фамилия И.О.)

Москва, 2021

1 Обработчик прерывания от системного таймера в системах разделения времени

1.1 UNIX-системы

В UNIX-системах обработчик прерывания от системного таймера **по тику** выполняет следующие задачи:

— инкремент счётчика тиков;

— обновление статистики использования процессора текущим процессом;

— пересчёт часов, минут и секунд;

— декремент счётчика тиков до отправления на выполнение отложенных вызовов, и установка флага для обработчика отложенных вызовов при достижении этим счётчиком нуля.

По главному тику обработчик прерывания от системного таймера выполняет ниже перечисленные задачи:

— регистрацию отложенных вызовов функций, относящихся к работе планировщика, таких как пересчёт приоритетов процессов;

— пробуждение в нужные моменты системных процессов, таких как **swapper** и **pagedaemon** (пробуждение означает регистрацию отложенного вызова процедуры **wakeup**, которая помещает дескриптор процесса в очередь готовых к выполнению процессов);

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

- **SIGALRM** – сигнал, посылаемый процессу по истечении заданного времени функцией **alarm**;
- **SIGPROF** – сигнал, посылаемый процессу по истечении времени заданного в таймере профилирования;
- **SIGVTALRM** – сигнал, посылаемый процессу по истечении времени, заданного «виртуальным» таймером.

По кванту обработчик прерывания выполняет следующую задачу:

— посылает текущему процессу сигнал **SIGXCPU**, если он превысил лимит процессорного времени.

1.2 Windows-системы

Обработчик прерывания от системного таймера **по тику** выполняет следующие задачи:

— инкремент счётчика тиков;

— декремент остатка кванта текущего потока;

— декремент счетчика отложенных задач;

— постановка в очередь DPC объекта диспетчера настройки баланса (этот диспетчер активизируется каждую секунду для возможной инициации событий, связанных с планированием и управлением памятью).

По главному тику обработчик прерывания выполняет следующее действие:

— возвращает задействованный в системе объект «событие», который ожидает диспетчер настройки баланса.

По кванту обработчик прерывания от системного таймера выполняет следующую задачу:

— инициализирует диспетчеризацию потоков путем постановки соответствующего объекта в очередь DPC.

2 Пересчёт приоритетов

2.1 UNIX-системы

Очередь готовых к выполнению процессов формируется согласно приоритетам процессов и принципу вытесняющего циклического планирования: процессы с одинаковыми приоритетами выполняются в течении кванта времени циклически друг за другом. Если процесс, имеющий более высокий приоритет, поступает в очередь готовых к выполнению, планировщик вытесняет текущий процесс и предоставляет ресурс более приоритетному.

В современных системах Unix ядро является вытесняющим – процесс в режиме ядра может быть вытеснен более приоритетным процессом в режиме ядра.

2.1.1 Приоритеты процессов

Приоритет процесса в UNIX задаётся числом в диапазоне от 0 до 127, причём чем меньше значение, тем выше приоритет. Приоритеты 0-49 зарезервированы ядром операционной системы, прикладные процессы могут обладать приоритетом в диапазоне от 50 до 127.

Приоритеты ядра являются фиксированными величинами. Приоритеты прикладных задач могут изменяться во времени в зависимости от следующих двух факторов:

— фактор любезности – целое число в диапазоне от -20 до 19. Чем меньше значение фактора любезности, тем выше приоритет процесса. Фактор любезности процесса может быть изменён суперпользователем системным вызовом nice;

— степень загруженности процессора в момент последнего обслуживания им процесса.

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** – показатель уступчивости, устанавливаемый пользователем.

Планировщик использует поле p_pri для принятия решения о том, какой процесс отправить на выполнение. Значения p_pri и p_usrpri одинаковы, когда процесс

находится в режиме задачи. Когда процесс просыпается после блокировки в системном вызове, его приоритет временно повышается. Планировщик использует p_usrpri для хранения приоритета, который будет назначен процессу при переходе из режима ядра в режим задачи, а p_pri – для хранения временного приоритета для выполнения в режиме ядра.

Ядро связывает приоритет сна (0-49) с событием или ожидаемым ресурсом, из-за которого процесс может быть заблокирован. Когда блокированный процесс просыпается, ядро устанавливает p_pri, равное приоритету сна события или ресурса, на котором он был заблокирован, следовательно, такой процесс будет назначен на выполнение раньше, чем другие процессы в режиме задачи.

В таблице 2.1 приведены значения приоритетов сна для систем 4.3BSD UNIX и SCO UNIX. Такой подход позволяет системным вызовам быстрее завершать свою работу. По завершении процессом системного вызова его приоритет сбрасывается в значение текущего приоритета в режиме задачи. Если при этом приоритет окажется ниже, чем приоритет другого запущенного процесса, ядро произведет переключение контекста.

Таблица 2.1 — Таблица приоритетов в системе 4.3BSD

Приоритет	**Значение**	**Описание**
PSWP	0	Своппинг
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 в терминал
PWAIT	30	Ожидание завершения процесса-потомка
PLOCK	35	Ожидание блокированного ресурса
PSLEP	40	Ожидание сигнала

Приоритет в режиме задачи зависит от уступчивости и последней измеренной величины использования процессора. Степень уступчивости – это число в диапазоне от -20 до 19 со значением 0 по умолчанию.

Системы разделения времени стараются выделить процессорное время таким образом, чтобы все процессы системы получили его в равных количествах, что требует слежения за использованием процессора. Поле p_cpu содержит величину последнего измерения использования процессора процессом. При создании процесса это поле инициализируется нулем. На каждом тике обработчик таймера увеличивает p_cpu на единицу для текущего процесса, вплоть до максимального значения – 127. Каждую секунду ядро вызывает процедуру schedcpu, которая уменьшает значение p_cpu каждого процесса исходя из фактора «полураспада».

В 4.3BSD для расчета применяется формула

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

где load_average – это среднее количество процессов, находящихся в состоянии готовности к выполнению за последнюю секунду.

Кроме того, процедура schedcpu также пересчитывает приоритеты режима задачи всех процессов по формуле

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

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, что приведет к увеличению значения p_usrpri и к понижению приоритета. Чем дольше процесс простаивает в очереди на выполнение, тем меньше его p_cpu. Это позволяет предотвратить зависания низкоприоритетных процессов. Если процесс бóльшую часть времени выполнения тратит на ожидание ввода-вывода, то он остается с высоким приоритетом.

Системы разделения времени пытаются выделить процессорное время таким образом, чтобы конкурирующие процессы получили его примерно в равных количествах. Фактор полураспада обеспечивает экспоненциально взвешанное среднее значение использования процессора в течение функционирования процесса. Формула,

применяемая в SVR3 имеет недостаток: вычисляя простое экспоненциальное среднее, она способствует росту приоритетов при увеличении загрузки системы.

2.2 Windows-системы

В системе Windows реализовано вытесняющее планирование на основе уровней приоритета, при которой выполняется готовый поток с наивысшим приоритетом.

Если поток с более высоким приоритетом готов к выполнению, текущий поток вытесняется планировщиком, даже если квант текущего потока не истёк.

В Windows за планирование отвечает совокупность процедур ядра, называемая диспетчером ядра. Диспетчеризация может быть вызвана, если:

— поток готов к выполнению;
— истёк квант текущего потока;
— поток завершается или переходит в состояние ожидания;
— изменился приоритет потока;
— изменилась привязка потока к процессору.

2.2.1 Приоритеты процессов

В системе предусмотрено 32 уровня приоритетов: уровни реального времени (16–31), динамические уровни (1–15) и системный уровень (0).

Windows API сортирует процессы по классам приоритета, которые были назначены при их создании:

— реального времени – real-time (4);
— высокий – high (3);
— выше обычного – above normal (6);
— обычный – normal (2);
— ниже обычного – below normal (5);
— простой – idle (1).

2.2.2 Приоритеты потоков

Процесс по умолчанию наследует свой базовый приоритет у того процесса, который его создал. Затем назначается относительный приоритет потоков в рамках процессов. Уровни приоритета потоков назначаются .

При создании потока, Windows API и ядро операционной системы назначают ему приоритет относительно процесса, в рамках которого он был создан. Данный приоритет называется относительным и является приращением к приоритету процесса, который в свою очередь является базовым для потока.

— критичный по времени – time critical (15);
— наивысший – highest (2);
— выше обычного – above normal (1);
— обычный – normal (0);
— ниже обычного – below normal (-1);
— низший – lowest (-2);
— простой – idle (-15).

Соответствие между приоритетами Windows API и ядра системы приведено в таблице 2.2.

Таблица 2.2 — 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

С точки зрения планировщика Windows важно только значение приоритета. Процесс обладает только базовым приоритетом, тогда как поток имеет базовый, который наследуется от приоритета процесса, и текущий приоритет. Процесс по умолчанию наследует свой базовый приоритет у того процесса, который его создал. Операционная система может на короткие интервалы времени повышать приоритеты потоков из динамического диапазона, но никогда не регулирует приоритеты пото-

ков в диапазоне реального времени. Приложения пользователя обычно запускаются с базовым приоритетом (normal). Некоторые системные процессы имеют приоритет выше 8, что гарантирует, что потоки в этих процессах будут запускаться с более высоким приоритетом.

Система повышает приоритет текущего потока в следующих случаях:

— по завершении операции ввода-вывода;
— по окончании ожидания на событии или семафоре исполнительной системы;
— по окончании ожидания потоками активного процесса;
— при пробуждении GUI-потоков из-за операции с окнами;
— если поток, готовый к выполнению, задерживается из-за нехватки процессорного времени.

Повышение приоритета применяется только к потокам 10 из динамического диапазона (1-15). Приоритет потока не может оказаться выше 15.

Рассмотрим каждый случай в отдельности.

2.2.3 Повышение приоритета по завершении операции ввода-вывода

По окончании определенных операций ввода-вывода Windows временно повышает приоритет потоков и потоки, ожидающие завершения этих операций, имеют больше шансов возобновить выполнение и обработать полученные от устройств ввода-вывода данные.

Драйвер устройства ввода-вывода через функцию IoCompleteRequest указывает на необходимость повышения приоритета после выполнения соответствующего запроса.

В таблице 2.3 приведены приращения приоритетов.

Таблица 2.3 — Рекомендуемые значения повышения приоритета

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

Приоритет потока повышается относительно базового приоритета. На рисунке 2.1 показано, что после повышения приоритета поток в течение одного кванта выполняется с повышенным приоритетом, а затем приоритет снижается на один уровень с каждым последующим квантом. Цикл продолжается до тех пор, пока приоритет не снизится до базового.

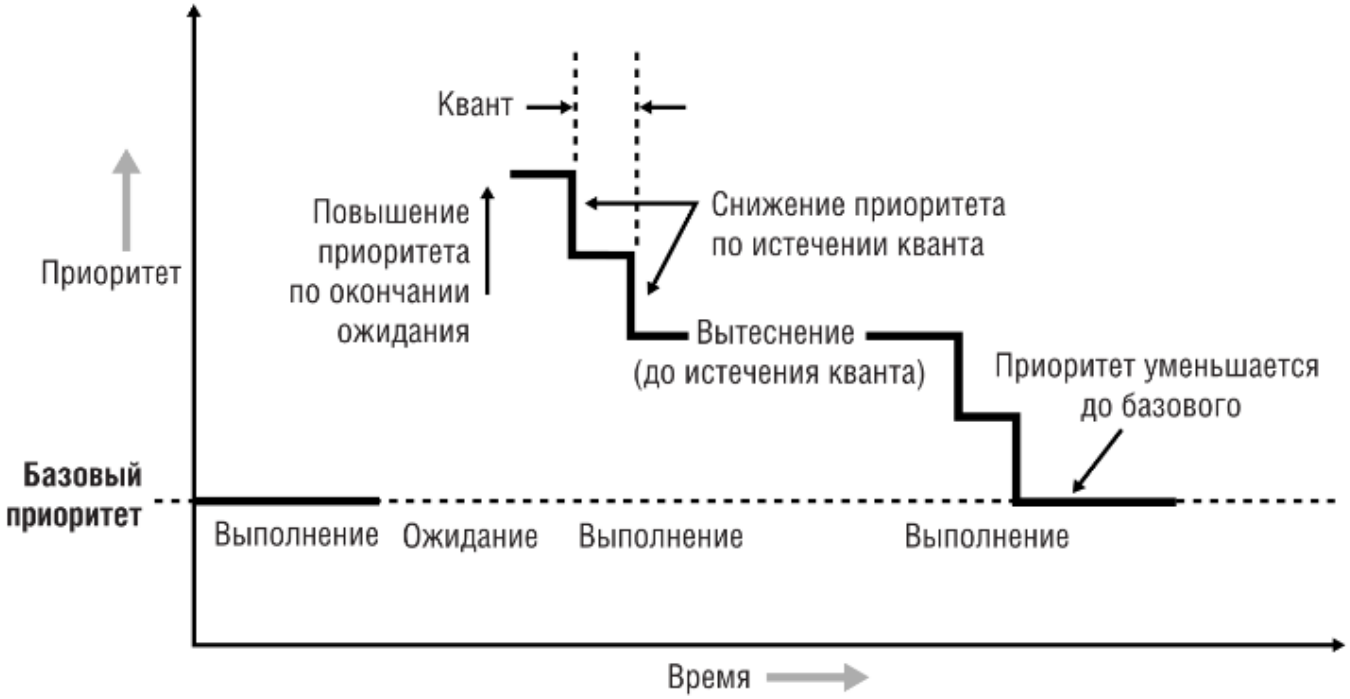

Рисунок 2.1 — Изменение приоритета

2.2.4 Повышение приоритета по окончании ожидания на событии или семафоре

Если ожидание потока на событии системы или семафоре успешно завершается из-за вы SetEvent, PulseEvent или ReleaseSemaphore, его приоритет повышается на 1. Такая регулировка позволяет равномернее распределить процессорное время – потокам, блокируемым на событиях, процессорное время требуется реже, чем остальным. В данном случае действуют те же правила динамического повышения приоритета.

К потокам, пробуждающимся в результате установки события вызовом функций NtSetEventBoostPriority и KeSetEventBoostPriority, повышение приоритета применяется особым образом.

2.2.5 Повышение приоритета по окончании ожидания потоками активного процесса

Если поток в активном процессе завершает ожидание на объекте ядра, функция ядра KiUnwaitThread повышает его текущий приоритет на величину значения PsPrioritySeparation. PsPrioritySeparation – это индекс в таблице квантов, с помощью которой выбираются величины квантов для потоков активных процессов. Какой процесс является в данный момент активным, определяет подсистема управления окнами.

Приоритет повышается для создания преимуществ интерактивным приложениям по окончании ожидания, в результате чего повышаются шансы на возобновление потока приложения. Важной особенностью данного вида динамического повышения приоритета является то, что он поддерживается всеми системами Windows и не может быть отключен даже функцией SetThreadPriorityBoost.

2.2.6 Повышение приоритета при пробуждении GUI-потоков

Приоритет потоков окон пользовательского интерфейса повышается на 2 после их пробуждения из-за активности подсистемы управления окнами. Приоритет повышается по той же причине, что и в предыдущем случае, – для увеличения отзывчивости интерактивных приложений.

2.2.7 Повышение приоритета при нехватке процессорного времени

Раз в секунду диспетчер настройки баланса – системный поток, предназначенный для выполнения функций управления памятью – сканирует очереди готовых потоков и ищет потоки, которые находятся в состоянии готовности в течение примерно 4 секунд. Диспетчер настройки баланса повышает приоритет таких потоков до 15. Причем в Windows 2000 и Windows XP квант потока удваивается относительно кванта процесса, а в Windows Server 2003 квант устанавливается равным 4 единицам. По истечении кванта приоритет потока снижается до исходного уровня. Если потоку все еще не хватило процессорного времени, то после снижения приоритета он возвращается в очередь готовых процессов. Через 4 секунды он может снова получить повышение приоритета.

Чтобы свести к минимуму расход процессорного времени, диспетчер настройки баланса сканирует только 16 готовых потоков за раз, а повышает приоритет не более чем у 10 потоков за раз. Диспетчер настройки баланса не решает всех проблем с приоритетами потоков, однако позволяет потокам, которым не хватает процессорного времени, получить его.

Выводы

Несмотря на то, что Windows и UNIX разные операционные системы, обработчики прерывания от системного таймера в этих системах выполняют схожие функции:

— инициализируют отложенные действия (такие как пересчет приоритетов);
— выполняют декремент счетчиков тиков;
— уменьшают квант процессорного времени, выделенного процессу.

Обе операционные системы являются системами разделения времени с вытеснением и динамическими приоритетами.

Приоритет пользовательского процесса в ОС UNIX может пересчитываться в зависимости от фактора любезности, p_cpu и базового приоритета. Приоритеты ядра являются фиксированными величинами.

При создании процесса в Windows, ему назначается приоритет, который является базовым относительно приоритетов потоков. Приоритет потока пользовательского процесса может быть пересчитан.

В любой системе у процесса базовый приоритет. Классическое ядро Unix не было многопоточным. Современные ядра и ядра Linux многопоточные.