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"

Тема Прерывания таймера в Winodws и Unix

Студент Цветков И. А.

Группа ИУ7-53 Б

Преподаватель Рязанова Н. Ю.

Москва — 2022 г.

1 Необходимые понятия

В процессе изложения нужны следующие понятия.

1. **Тик** - период времени между двумя последующими прерываниями таймера.

2. **Основной тик** - период времени, равный N тикам таймера, где число N зависит от конкретного варианта системы.

3. **Квант времени** - временной интервал, в течение которого процесс может использовать процессор до вытеснения его другим процессом.

2 Функции обработчика прерывания от системного таймера в защищенном режиме

Поскольку обработчик прерывания от системного таймера имеет наивысший приоритет, то никакая другая работа не может выполняться в системе во время его срабатывания.

2.1 ОС Unix

2.1.1 По тику

1. Инкремент часов и других таймеров системы.
2. Декремент кванта.
3. Инкремент счетчика тиков аппаратного таймера.
4. Инкремент поля *p_cpu* структуры *proc* до максимального значения в *127*, то есть инкремент счетчика использования процессора текущим процессом.
5. Декремент счетчика времени до отправления на выполнение отложенных вызовов. Когда счетчик становится равен нулю, то происходит выставление флага обработчика отложенного вызова.

2.1.2 По главному тику

1. Распознает отложенные вызовы функции, которые относятся к работе планировщика.
2. В нужные моменты пробуждает системные процессы (например, swapper и pagedaemon). *Пробуждать* - распознать отложенный вызов процеду-

ры *wakeup*, которая помещает дескрипторы процессов из списка *спящие* в очередь – *готовы к выполнению*.

3. Декремент счетчика времени, которое осталось до отправления одного из следующих сигналов:

   - *SIGPROF* - сигнал, который посылается процессу по истечении времени, заданного в таймере профилирования;
   - *SIGVTALRM* - сигнал, который посылается процессу по истечении времени, заданного в "виртуальном"таймере;
   - *SIGALRM* - сигнал, который посылается процессу по истечении времени, которое было предварительно задано функцией *alarm()*

2.1.3 По кванту

Сигнал *SIGXCPU* посылается текущему процессу, если он превысил выделенную для него квоту использования процессора (по умолчанию - обработчик сигнала прерывает выполнение процесса)

2.2 ОС Windows

2.2.1 По тику

1. Инкремент счетчика системного времени.
2. Декремент кванта.
3. Декремент счетчиков времени отложенных задач.
4. Когда активен механизм профилирования ядра, то инициализация отложенного вызова обработчика ловушки профилирования ядра происходит с помощью постановки объекта в очередь *DPC* (обработчик ловушки профилирования регистрирует адрес команды, которая выполнялась на момент прерывания).

2.2.2 По главному тику

Происходит 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 состоянии ожидания свыше *4 секунд*.

2.2.3 По кванту

Происходит инициализация диспетчеризации потоков, добавление соответствующего объекта в очередь *DPC*, где DPC (Deferred Procedure Call) - отложенный вызов процедуры.

3 Пересчет динамических приоритетов

Замечание: динамически могут пересчитываться только *приоритеты пользовательских процессов* (как в Unix, так и в Windows)

3.1 ОС Unix

Приоритет процесса задается любым целым числом в диапазоне от 0 до 127 (чем меньше число, тем выше приоритет), при этом:

- 0 - 49 – зарезервированы для ядра;
- 50 - 127 – прикладные (приоритеты прикладных задач могут изменяться во времени)

В традиционных системах механизм планирования базируется на приоритетах. Приоритетом планирования (который изменяется с течением времени) обладает каждый процесс. Задача планировщика выбрать процесс, который обладает наивысшим проритетом. Если же некоторые процессы имеют равный приоритет, то применяется вытесняющее квантование времени. Изменение приоритетов процессов происходит динамически на основе количества процессорного времени, которое они используют. Если какой-либо из процессов, который имеет высокий приоритет, будет готов к выполнению, то планировщик вытеснит ради него текущий процесс даже в том случае, если тот не израсходовал свой квант времени.

В *Unix* традиционное ядро является строго невытесняемым. Если процесс выполняется в режиме ядра (то есть в течение исполнения системного вызова или прерывания), то ядро не заставит такой процесс уступить процессор какому-либо высокоприоритетному процессу. Однако в современных системах Unix ядро является вытесняемым, поэтому процесс в режиме ядра может быть вытеснен процессом с более высоким приоритетом. Ядро было сделано вытесняемым, чтобы система могла обслуживать процессы реального времени (например, видео и аудио).

Приоритет прикладных задач зависит от:

- *фактора любезности (nice)* – целое число в диапазоне от 0 до 39 (по умолчанию – 20). Если значение увеличить, то приоритет уменьшится. При этом фоновые процессы автоматически имеют более высокие значения данного фактора (пользователь может поменять приоритет процесса, изменяя этот фактор, но только в сторону уменьшения приоритета, и только суперпользователь может повысить приоритет процесса);

- крайней измеренной величины использования процессора.

В структуре *proc* содержутся следующие поля, которые относятся 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

Рисунок 3.1 – Поля структуры proc

Значение в поле p_pri используется планировщиком для принятия решения о том, какой процесс нужно отправить на выполнение (p_pri и p_usrpri равны, если процесс находится в режиме задачи).

Значение p_pri может быть повышено планировщиком для того, чтобы выполнить процесс в режиме ядра. Тогда p_usrpri будет использоваться для хранения приоритета, который будет назначен процессу при возврате в режим задачи.

Поле p_cpu инициализируется нулем при создании процесса. Обработчик таймера на каждом тике увеличивает значение в этом поле текущего проесса на 1 до максимального значения в 127.

Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может блокироваться. Приоритет сна лежит в диапазоне от 0 до 49, так как определяется для ядра.

Когда процесс "просыпается", ядро устанавливает в поле p_pri приоритет сна (значение приоритера из диапазона системных вызовов, которое зависит от события или ресурса, по которому проиозошла блокировка).

На рисунке 3.2 представлены значения приоритета сна в системе 4.3BSD.

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

Рисунок 3.2 – Приоритет сна в ОС 4.3BSD

Также на рисунке 3.3 приведена таблица системных приоритетов сна из книги Андрея Робачевского:

Таблица 3.3. Системные приоритеты сна

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

Рисунок 3.3 – Приоритет сна из книги А. Робаческий "Операционная система UNIX"

Ядро системы каждую секунду инициализирует отложенный вызов процедуры *schedcpu()*, которая уменьшает значение p_cpu каждого процесса, исходя из фактора "полураспада который в системе *4.3BSD* считается так:

$$decay = (2 * load\_average)/(2 * load\_average + 1) \tag{3.1}$$

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

Процедура *schedcpu()* также пересчитывает приоритеты для режима задачи всех процессов по следующей формуле

$$p\_usrpri = PUSER + (p\_cpu/4) + (2 * p\_nice) \tag{3.2}$$

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

Таким обрзаом, если процесс в последний раз (до вытеснения другим

процессом) использовал большое количество процессорного времени, то его p_cpu будет увеличен. И это приведет к росту значения $p\_userpri$ и, следовательно, к понижению приоритета. Чем дольше процесс простаивает в очереди на выполнение, тем больше фактор полураспада уменьшает его p_cpu, что приодит к повышению его приоритета. Такая схема предотвращает бесконечное откладывание низкоприоритетных процессов в ОС (предпочтительна данная схема к процессам, которые осуществляют много операций ввода-вывода).

Таким образом, динамический пересчет приоритетов в режиме задачи позволяет избежать бесконего откладывания. Сам пересчет может быть произведен по следующим причинам:

- изменение фактора любезности процесса системным вызовом *nice*;
- ожидания процесса в очереди готовых к выполнению процессов;
- в зависимости от степени загруженнности процессора процессом p_cpu;
- приоритет может быть повышен до соответствующего приоритета сна вследствие ожидания ресурса или события.

3.2 ОС Windows

Система планирования в *Windows* является приоритетной и вытесняющей. При такой системе планирования всегда выполняется хотя бы один работоспособный (готовый) поток с самым высоким приоритетом (замечание - конкретные, имеющие высокий проритет и готовые к запуску потоки могут быть ограничены процессами, на которых им разрешено или предпочтительнее всего работать).

В *Windows* планировка потоков происходит на основании приоритетов готовых к выполнению потоков – поток с более низким приоритетом вытесняется планировщиком, когда поток с более высоки приоритетом становится готовым к выполнению.

В *Windows* нет единого модуля или процедуры с названием "планировщик потому что этот код рассредоточен по ядру (код, отвечающий за плани-

рование, реализован в ядре). А совокупность процедур, которые выполняют эти обязанности, называются диспетчером ядра.

Диспетчеризация потоков может быть вызвана:

- поток готов к выполнению (только что создан или вышел из состояния "ожидание")
- поток выходит из состояния "выполняется"(его квант истек, поток завершается или переходит в состояние "ожидание");
- изменилась привязка к процессорам (поток больше не может работать на процессоре, на котором он выполнялся);
- поменялся приоритет потока.

В Windows 32 уровня приоритетов: 0-15 – 16 изменяющихся уровней (0 – зарезервирован для потока обнуления страниц); 16-31 – 16 уровней реального времени.

Исходя из двух разных позиций – *Windows API* и *ядра Windows*, назначаются уровни приоритета.

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

- реального времени (Real-time) - (4);
- высокий (High) - (3);
- выше обычного (Above Normal) - (6);
- обычный (Normal) - 2;
- ниже обычного (Below Normal) - (5);
- простоя (Idle) - (1);

Затем назначается относительный приоритет отдельных потоков внутри этих процессов:

- критичный по времени (Time-critical) - 15;

- наивысший (Highest) - (2);
- выше обычного (Above Normal) - (1);
- обычный (Normal) - (0);
- ниже обычного (Below Normal) - (-1);
- самый низкий (Lowest) - (-2);
- простоя (Idle) - (-15);

Процесс по умолчанию наследует свой базовый приоритет у того процесса, который его создал.

На рисунке 3.4 можно увидеть соответствие между приоритетами Windows API и ядра системы.

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 3.4 – Соответствие между приоритетами Windows API и ядра системы

Планировщик может повысить текущий приоритет потока (в динамическом диапазоне от 1 до 15) из-за следующих причин.

- Повышение приоритета вследствие событий планировщика или диспетчера (сокращение задержек).
- Повышение приоритета, связанное с завершением ожидания.
- Повышение приоритета владельца блокировки.
- Повышение приоритета вследствие заврешения ввода-вывода.
- Повышение приоритета при ождиании ресурсов исполняющей системы.

- Повышение приоритета потоков первого плана после ожидания.
- Повышение приоритета после пробуждения GUI потока.
- Повышение приоритета, связанные с перезагруженностью ЦП.
- Повышение приоритетов для мультимедийных приложений и игр.

На рисунке 3.5 предсталвены рекомендуемые значения повышения приоритета.

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 3.5 – Рекомендуемые значения повышения приоритета

Также стоит упомянуть, что важным свойством планирования потоков является категория планирования. Это – первичный фактор, который определяет приоритет потоков, зарегестрированных с MMCSS. Функции MMCSS временно повышают приоритет потоков, зарегестрированных с MMCSS до уровня, который соответствует категории планирования. Затем их приоритет снижения до уровня Exhausted, чтобы другие потоки тоже могли получить ресурс.

На рисунке 3.6 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 3.6 – Категории планирования

Вывод

ОС Unix и ОС Windows – системы разделения времени с вытеснением и динамическими приоритетами.

У обеих операционных систем функции обработчика прерывания от системного таймера схожи, так как они являются системами разделения времени.

Основные общие функции обработчика прерывания от системного таймера:

- декремент кванта;
- декремент счетчиков времени (таймеров, часов, счетчиков времени отложенных действий, будильников реального времени);
- инициализация отложенных действий, которые относятся к работе планировщика.

В Unix приоритет пользовательского процесса (процесса в режиме задач) может динамически пересчитываться в зависимости от фактора любезности, результата последнего измерения использования процессора и базового приоритета.

В Windows когда создается процесс, ему назначается базовый приоритет, и относительно базового приоритета потоку назначается отнносительный приоритет, то есть у потока нет своего приоритета (приоритет потока пользовательского процесса может быть динамически пересчитан).