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 дисциплине "Операционные системы"

Тема Прерывание таймера в Windows и Unix

Студент Золотухин А. В.

Группа ИУ7-54Б

Оценка (баллы) _____

Преподаватели Рязанова Н.Ю.

Москва — 2022 г.

Тик – период времени между двумя последующими прерываниями таймера.

Основной тик – период времени равный n тикам таймера (число n зависит от конкретного варианта системы).

Квант времени (quantum, time slice) — временной интервал, в течение которого процесс может использовать процессор до вытеснения другим процессом.

1 Функции обработчика прерывания от системного таймера в защищенном режиме

Обработчик прерываний от системного таймера имеет наивысший приоритет. Никакая другая работа в системе не может выполняться во время обработчика прерывания от системного таймера. Обработчик прерывания от системного таймера должен завершаться как можно быстрее, чтобы не влиять на отзывчивость системы (отзывчивость – как быстро система отвечает на запросы пользователей).

1.1 Unix

По тику

- инкремент счетчика тиков аппаратного таймера
- инкремент счетчика использование процессора текущим процессом (то есть инкремент поля p_cpu структуры proc до максимального значения – 127)
- инкремент часов и других таймеров системы
- декремент кванта текущего потока
- декремент счетчика времени до отправления на выполнение отложенных вызовов, при достижении счетчиком нуля происходит выставление флага для обработчика отложенного вызова [1]

По главному тику

- пробуждает в нужные моменты системные процессы, такие как swapper и pagedaemon. ('пробуждает' тут понимается так: инициирование от-

ложенного вызова процедуры wakeup, которая перемещает дескрипторы процессов из списка "спящие"в очередь готовых к выполнению)

- инициирует отложенные вызовы функции, которые относятся к работе планировщика
- декрементирует счетчик времени, которое осталось до отправления одного из следующих сигналов:
  - SIGVTALRM – сигнал, посылаемый процессу по истечении времени, заданного в “виртуальном” таймере;
  - SIGPROF – сигнал, посылаемый процессу по истечении времени заданного в таймере профилирования;
  - SIGALRM – сигнал, посылаемый процессу по истечении времени, предварительно заданного функцией alarm().

По кванту

- посылка текущему процессу сигнала SIGXCPU, если он превысил выделенную для него квоту использования процессора. По получению сигнала обработчик сигнала прерывает выполнение процесса.

1.2 Windows

По тику

- инкремент счетчика системного времени
- декремент счетчиков времени отложенных задач
- декремент кванта текущего потока
- Если активен механизм профилирования ядра, то инициализация отложенного вызова обработчика ловушки профилирования ядра с помощью постановки объекта в очередь DPC (обработчик ловушки про-

филирования 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 состоянии ожидания дольше 4 секунд.)

По кванту

- Инициация диспетчеризации потоков (добавление соответствующего объекта в очередь DPC – Deferred procedure call — отложенный вызов процедуры)

2 Пересчет динамических приоритетов

В ОС семейства UNIX и в ОС семейства Windows только **приоритеты пользовательских процессов** могут динамически пересчитываться.

2.1 Unix

Планирование процессов в UNIX основано на приоритете процесса. Планировщик всегда выбирает процесс с наивысшим приоритетом. Приоритеты процессоров изменяются динамически системой в зависимости от использования вычислительных ресурсов, времени ожидания запуска и текущего состояния процесса. Если процесс готов к запуску и имеет наивысший приоритет, планировщик приостановит выполнение текущего процесса (с более низким приоритетом), даже если тот не «выработал» свой временной квант.

Традиционное ядро UNIX является строго невытесняющим, однако в современных системах UNIX ядро является вытесняющим – то есть процесс в режиме ядра может быть вытеснен более приоритетным процессом в режиме ядра. Ядро сделано вытесняющим для того, чтобы система могла обслуживать процессы реального времени, например видео и аудио.

Очередь процессов, готовых к выполнению, формируется согласно приоритетам и принципу вытесняющего циклического планирования, то есть сначала выполняются процессы с большим приоритетом, а процессы с одинаковым приоритетом выполняются в течении кванта времени друг за другом циклически. В случае, если процесс с более высоким приоритетом поступает в очередь процессов, готовых к выполнению, планировщик вытесняет текущий процесс и предоставляет ресурс более приоритетному процессу.

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

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

Изменение приоритета прикладных задач зависит от следующих факторов:

- **фактор "любезности" (nice)** – это целое число в диапазоне от 0 до 39 со значением 20 по умолчанию. Увеличение значения приводит к уменьшению приоритета. Пользователи могут повлиять на приоритет процесса при помощи изменения значений этого фактора, но только суперпользователь может увеличить приоритет процесса. Фоновые процессы автоматически имеют более высокие значения этого фактора.

- **последней измеренной величины использования процессора**

Структура proc содержит 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', который устанавливается пользователем

p_pri используется планировщиком для принятия решения о том, какой процесс отправить на выполнение. **p_pri** и **p_usrpri** равны, когда процесс находится в режиме задачи.

Значение **p_pri** может быть изменено (повышено) планировщиком для того, чтобы выполнить процесс в режиме ядра. В таком случае **p_usrpri** будет использоваться для хранения приоритета, который будет назначен процессу при возврате в режим задачи.

p_cpu инициализируется нулем при создании процесса (и на каждом тике обработчик таймера увеличивает это поле текущего процесса на 1, до максимального значения равного 127).

Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может блокироваться (приоритет сна определяется для ярда, поэтому лежит в диапазоне 0 - 49). Когда процесс "про-

сыпается", ядро устанавливает в поле **p_pri** приоритет сна – значение приоритета из диапазона системных приоритетов, зависящее от события или ресурса по которому произошла блокировка. (событие и связанное с ним значение приоритета сна в системе 4.3BSD представлены в таблице 2.1).

Таблица 2.1 – Приоритеты сна в ОС **4.3BSD** (из книги [1] "UNIX изнутри")

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 блок. ресурса
PSLEP	40	Ожидание сигнала

Также приведена таблица из книги "Операционная система UNIX" Андрея Робачевского на рисунке 2.1. Заметим, что направление роста значений приоритета для этих систем (4.3BSD UNIX и SCO UNIX) различно.

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 2.1 – Системные приоритеты сна

Каждую секунду ядро системы инициализирует отложенный вызов процедуры schedcpu(), которая уменьшает значение **p_pri** каждого процесса исходя из фактора "полураспада" (в системе 4.3BSD считается по формуле 2.1)

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

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

Также процедура schedcpu() пересчитывает приоритеты для режима задачи всех процессов по формуле 2.2,

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

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

Таким образом, если процесс в последний раз использовал большое количество процессорного времени, то его **p_cpu** будет увеличен. Это приведет к росту значения **p_usrpri**, то есть 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. Такая схема предотвращает бесконечное откладывание низкоприоритетных процессов. Применение данной схемы предпочтительно процессам, осуществляющим много операций ввода-вывода, в противоположность процессам, производящим много вычислений.

То есть, если процесс большинство времени выполнения тратит на ожидание ввода-вывода, то он остается с высоким приоритетом и, таким образом, быстрее получает процессор при необходимости.

В тоже время вычислительные приложения обычно обладают более высокими значениями **p_cpu** и работают на значительно более низких приоритетах.

Динамический пересчет приоритетов процессов в режиме задачи позволяет избежать бесконечного откладывания.

Таким образом, приоритет процесса в режиме задачи может быть динамически пересчитан по следующим причинам:

- Вследствие изменения фактора любезности процесса системным вызовом nice
- В зависимости от степени загруженности процессора процессом p_cpu
- Вследствие ожидания процесса в очереди готовых к выполнению процессов
- Приоритет может быть повышен до соответствующего приоритета сна вследствие ожидания ресурса или события

2.2 Windows

В Windows процессу при создании назначается базовый приоритет. Относительно базового приоритета процесса потоку назначается относительный приоритет.

Планирование осуществляется только на основании приоритетов потоков, готовых к выполнению: если поток с более высоким приоритетом становится готовым к выполнению, поток с более низким приоритетом вытесняется планировщиком. По истечению кванта времени текущего потока, ресурс передается самому приоритетному потоку в очереди готовых к выполнению

Windows использует 32 уровня приоритета:

- от 0 до 15 – 16 изменяющихся уровней (из которых уровень 0 – зарезервирован для потока обнуления страниц)
- от 16 до 31 – 16 уровней реального времени

Уровни приоритета потоков назначаются исходя из двух разных позиций: одной от Windows API и другой от ядра Windows. 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)

После 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 низший — Lowest (–2)
- Простоя — Idle (–15)

Исходный базовый приоритет потока наследуется от базового приоритета процесса. Процесс по умолчанию наследует свой базовый приоритет у того процесса, который его создал. Соответствие между приоритетами Windows API и ядра системы приведено в таблице 2.2

Текущий приоритет потока в динамическом диапазоне – от 1 до 15 – может быть повышен планировщиком вследствие следующих причин:

- повышение вследствие событий планировщика или диспетчера (сокращение задержек);
- повышения приоритета, связанные с завершением ожидания;
- повышение приоритета владельца блокировки;
- Повышение вследствие завершения ввода-вывод;
- повышение при ожидании ресурсов исполняющей системы;

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

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

- повышение приоритета потоков первого плана после ожидания;
- повышение приоритета после пробуждения GUI-потока;
- повышения приоритета, связанные с перезагруженностью центрального процессора;
- повышение приоритетов для мультимедийных приложений и игр;

  Потоки, на которых выполняются различные мультимедийные приложения, должны выполняться с минимальными задержками. В Windows такая задача решается с помощью повышения приоритетов таких потоков драйвером **MMCSS (MultiMedia Class Scheduler Service)**. MMCSS работает с различными определенным задачи, например:

  – аудио

  – аудио профессионального качества

– игры

– захват

– воспроизведение

– низкая задержка

– задачи администратора многооконного режима.

– распределение

Важное свойство для планирования потоков – категория планирования – это первичный фактор, который определяет приоритет потоков, зарегистрированных с MMCSS (категории планирования указаны в таблице 2.4).

Функции MMCSS временно повышают приоритет потоков, зарегистрированных с MMCSS до уровня, который соответствует категории планирования. Потом их приоритет снижается до уровня, соответствующего категории планирования Exhausted, для того, чтобы другие потоки тоже могли получить ресурс.

Таблица 2.4 – 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

3 Вывод

Функции обработчика прерывания от системного таймера в защищенном режиме для ОС семейства UNIX и для ОС семейства Windows схожи, так как эти ОС являются системами разделения времени. Общие основные функции:

- декремент кванта текущего процесса в UNIX и декремент текущего потока в Windows.
- инициализация отложенных действий, которые относятся к работе планировщика (например пересчет приоритетов).
- декремент счетчиков времени (таймеров, часов, счетчиков времени отложенных действий, будильников реального времени)

Обе операционные системы (UNIX и Windows) – это системы разделения времени с вытеснением и динамическими приоритетами.

В ОС UNIX приоритет пользовательского процесса (процесса в режиме задач) может динамически пересчитываться, в зависимости от фактора "любезности", p_cpu (результат последнего измерения использования процессора) и базового приоритета (PUSER). Приоритеты ядра – фиксированные величины.

В ОС Windows при создании процесса ему назначается базовый приоритет, относительно базового приоритета процесса потоку назначается относительный приоритет, таким образом, у потока нет своего приоритета. Приоритет потока пользовательского процесса может быть динамически пересчитан.