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 ОС Windows и Unix

Студент Спирин М.П.

Группа ИУ7-54Б

Оценка (баллы) _____

Преподаватели Рязанова Н.Ю.

Москва — 2023 г.

1 Функции системного таймера в системах разделения времени

1.1 Функции системного таймера в семействе ОС Windows

По тику

- инкремент счетчика реального времени;
- декремент кванта текущего потока;
- декремент счетчиков времени до выполнения отложенных задач.

По главному тику

- Инициализация диспетчера настройки баланса путем сбрасывания объекта «событие», на котором он ожидает.

По кванту

- Инициация диспетчеризации потоков — постановка соответствующего объекта в очередь DPC.

1.2 Функции системного таймера в семействе ОС Unix

По тику

- инкремент счетчика времени с момента запуска системы (SVR4, переменная *lbolt*)
- инкремент счетчика реального времени;
- декремент счетчиков времени и при достижении счетчиками нулевого значение установка флага для обработчиков отложенных вызовов;
- инкремент счетчика процессорного времени, полученного процессором в режиме задачи и в режиме ядра.

По главному тику

- инициализация отложенного вызова функции планировщика;
- инициализация отложенного вызова процедуры wakeup, котораяменяет состояние процесса с «спящий» на «готовый к выполнению»; В системе SVR4 можно инициализировать отложенный вызов с помощью timeout(void (*fn)(), caddr_t arg, long delta); где fn() – функция, которую необходимо запустить, arg – аргументы, которые получит fn(), delta – временный интервал (в тиках процессора) через который fn() должна быть вызвана.
- пробуждение системных процессов, таких как pagedaemon.
- декремент счетчика времени, которое осталось до посылки одного из следующих сигналов:
  - SIGVTALRM – сигнал, посылаемый процессу по истечении промежутка реального времени;
  - SIGPROF – сигнал, посылаемый процессу по истечении времени, заданного в таймере профилирования;
  - SIGALRM – сигнал, посылаемый процессу по истечении времени, заданного в «виртуальном» таймере.

По кванту

- посылка сигнала SIGXCPU текущему процессу, если он превысил выделенную для него квоту процессорного времени.

2 Пересчет динамических приоритетов

Системы семейств Unix и Windows являются системами разделения времени с динамическими приоритетами и вытеснением. Динамические приоритеты могут иметь только пользовательские процессы, другие процессы имеют фиксированные приоритеты.

2.1 Системы семейства Windows

В Windows при создании процесса, ему назначается приоритет. Относительно приоритета процесса потоку назначается относительный приоритет.

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

Раз в секунду диспетчер настройки баланса сканирует очередь готовых потоков. Если обнаружены потоки, ожидающие выполнения более 4 секунд, диспетчер настройки баланса повышает их приоритет. Как только квант истекает, приоритет потока снижается до базового приоритета. Если поток не был завершен за квант времени или был вытеснен потоком с более высоким приоритетом, то после снижения приоритета поток возвращается в очередь готовых потоков.

Чтобы минимизировать расход процессорного времени, диспетчер настройки баланса сканирует лишь 16 готовых потоков. Кроме того, диспетчер повышает приоритет не более чем у 10 потоков за один проход: обнаружив 10 потоков, приоритет которых следует повысить, он прекращает сканирование. При следующем проходе сканирование возобновляется с того места, где оно было прервано в прошлый раз. Наличие 10 потоков, приоритет которых следует повысить, говорит о необычно высокой загруженности системы.

В Windows предусмотренно 32 уровня приоритета:

- приоритет 31 — наивысший;
- от 16 до 31 — процессы реального времени;
- от 0 до 15 — динамические уровни;

- 0 — зарезервирован для процесса обнуления страниц.

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)

Уровни приоритета потоков назначаются с двух позиций: Windows API и ядра операционной системы. Windows API сортирует процессы по классам приоритета, которые были назначены при из 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).

Затем назначается относительный приоритет потоков в рамках процесса:

- 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 2.1.

Таблица 2.1 – 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

Текущий приоритет потока в динамическом диапазоне может быть повышен планировщиком вследствие следующих причин:

- завершение операций ввода-вывода;
- повышение приоритета владельца блокировки;
- ввод из пользовательского интерфейса;
- длительное ожидание ресурса исполняющей системы;
- ожидание объекта ядра;
- готовый к выполнению поток не был запущен в течение длительного времени;

Таблица 2.2 – 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 службой планировщика MMCSS.

Текущий приоритет потока в динамическом диапазоне может быть понижен до базового приоритета путем вычитания всех повышений.

MMCSS

Потоки, на которых выполняются различные мультимедийные приложения, должны выполняться с минимальными задержками. В Windows эта задача решается путем повышения приоритетов таких потоков драйвером MMCSS – MultiMedia Class Scheduler Service. Приложения, которые реализуют воспроизведение мультимедиа, указывают драйверу MMCSS задачу из списка:

1. аудио;
2. возможность использовать функции записи;
3. воспроизведение звукового или видео контента;
4. задачи администратора многооконного режима.

Одно из наиболее важных свойств для планирования потоков — категория планирования — первичный фактор определяющий приоритет потоков, зарегистрированных в MMCSS. Различные категории планирования представленны в таблице ниже.

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

Таблица 2.3 – 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 продолжиться, только если не будут готовы к выполнению другие потоки с более высоким уровнем приоритета

IRQL

Хотя контроллеры прерываний устанавливают приоритетность прерываний, Windows устанавливает свою собственную схему приоритетности прерываний, известную как уровни запросов прерываний (IRQL). В ядре IRQL-уровни представлены в виде номеров от 0 до 31 (рисунок 2.1), где более высоким номерам соответствуют прерывания с более высоким приоритетом.

Прерывания обслуживаются в порядке их приоритета. При возникновении прерывания с высоким приоритетом процессор сохраняет информацию о состоянии прерванного потока и запускает связанные с прерывание диспетчер системных прерываний.

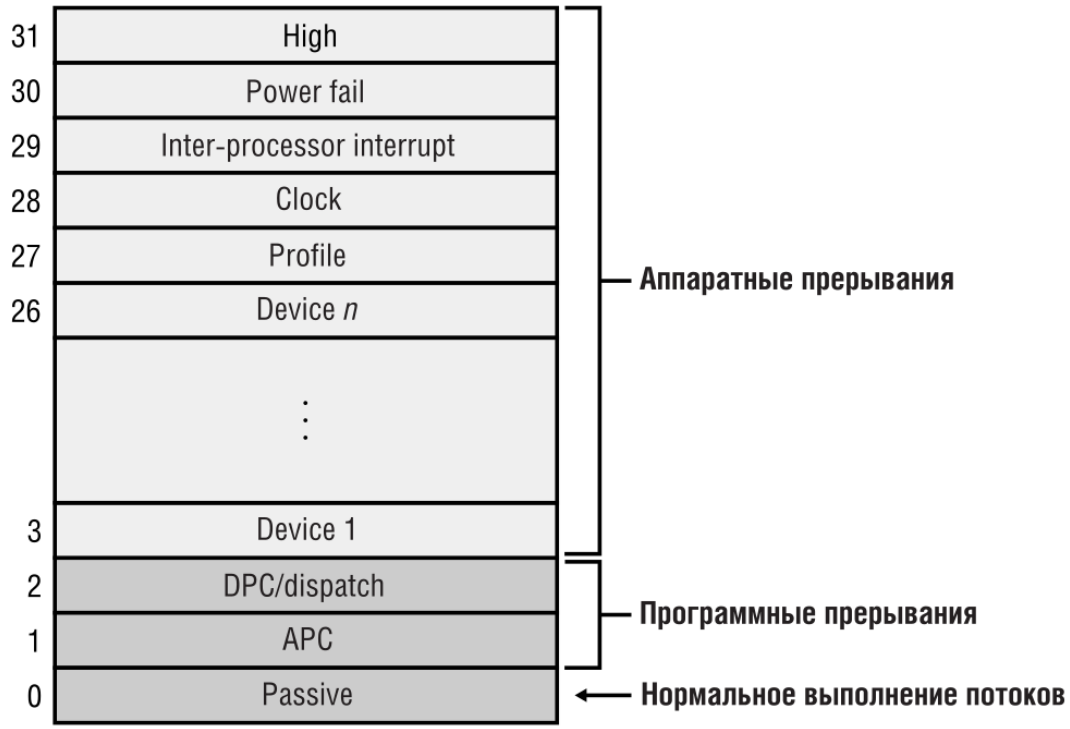

Рисунок 2.1 – Уровни запросов прерываний

2.2 Системы семейства Unix

Планирование процессов в UNIX основано на приоритете процесса. Планировщик всегда выбирает процесс с наивысшим приоритетом. Приоритеты планирования изменяются с течением времени (динамически) системой в зависимости от использования вычислительных ресурсов, времени ожидания запуска и текущего состояния процесса.

Традиционное ядро UNIX является строго невытесняющим, однако в современных системах UNIX ядро является вытесняющим – то есть процесс в режиме ядра может быть вытеснет 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';
- последней измеренной величины использования процессора.

Фактор любезности – это целое число в диапазоне от 0 до 39 (по умолчанию 20). Чем меньше значение фактора любезности процесса, тем выше приоритет процесса. Фактор любезности процесса может быть изменен с

помощью системного вызова **nice**, но только суперпользователем. Фоновым процессам задаются более высокие значения фактора любезности.

Дескриптор процесса 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 отправиь 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, когда тот вернется 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 'просыпается', ядро устанавливает **p_pri**, равное приоритету сна события или ресурса, по которому произошла блокировка (значение приоритета сна для некоторых событий в системе 4.3BSD представлены в таблице 2.4).

Также приведена таблица из книги «Операционная система UNIX» Андрея Робачевского на рисунке 2.2. Заметим, что направление роста значений приоритета для этих систем (4.3BSD UNIX и SCO UNIX) различно.

Таблица 2.4 – Приоритеты сна в ОС **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
PWAIT	30	Ожидание завершения процесса потомка
PLOCK	35	Консультативное ожидание блок. ресурса
PSLEP	40	Ожидание сигнала

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 2.2 – 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 => рост значения p_usrpri => понижение приоритета. Чем дольше процесс простаивает в очереди на выполнение, тем больше фактор полураспада уменьшает его p_cpu => повышение 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 динамический пересчет приоритетов процессов в режиме задачи позволяет избежать бесконечного откладывания.

3 Вывод

Операционные системы семейств Unix и Windows являются системами разделения времени с динамическими приоритетами и вытеснением, поэтому функции обработчика прерывания от системного таймера в этих системах выполняют схожие задачи:

- 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)

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