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»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №1(ЧАСТЬ 2)

Название: Изучение функций прерывания от системного таймера

Дисциплина: Операционные системы

Студент	ИУ7-56Б		А.Д. Ковель
	Группа	Подпись, дата	И. О. Фамилия
Преподаватель			Н. Ю. Рязанова
		Подпись, дата	И. О. Фамилия

Москва, 2022 г.

Содержание

1 Функции системного таймера в системах разделения времени

В разделе рассмотрены функции обработчика прерывания от системного таймера в операционных системах семейства Windows и Unix.

1.1 Windows системы

Обработчик прерывания по тику

Обработчик прерываний по тику выполняет следующие действия:

- инкремент счетчика реального времени;
- декремент кванта текущего потока;
- декремент счетчика отложенных задач;

Обработчик прерывания по главному тику

Обработчик прерываний по главному тику:

- ставит в очередь DPC объект диспетчера настройки баланса (диспетчер активизируется каждую секунду для возможной инициации событий, связанных с планированием и управлением памятью)
- возвращает системе ресурсы, задействованные объектом «событие», который ожидает диспетчер настройки баланса.

Обработчик прерывания по кванту

Обработчик прерывания по кванту инициализирует диспетчеризацию потоков, то есть ставит соответствующий объект в очередь DPC.

1.2 Unix системы

Обработчик прерывания по тику

Обработчик прерываний по тику выполняет следующие действия:

- инкремент таймеров системы — например, количество тиков, отсчитанных с момента загрузки системы ($SVR4$, переменная $lbolt$);
- декремент счетчика времени до отправления на выполнение отложенных вызовов — при достижении счетчиком нулевого значения выставление флага для обработчика отложенных вызовов;
- декремент кванта текущего потока.

Обработчик прерывания по главному тику

Обработчик прерывания по главному тику ставит в очередь на выполнение отложенные вызовы функций, относящиеся к работе планировщика, такие как пересчет приоритетов. Также он ставит в очередь на выполнение отложенный вызов процедуры wakeup, которая перемещает дескрипторы процессов из очереди «спящих» в очередь «готовых к выполнению».

Помимо этого, обработчик выполняет декремент счетчика времени, оставшегося до отправления одного из следующих сигналов:

- SIGALARM — сигнал тревоги реального времени, который отпарвляется по истечении заданного промежутка реального времени;
- SIGPROF — сигнал профилирования таймера;
- SIGVTALRM — сигнал тревоги виртуального времени, который измеряет время работы процесса в режиме задачи.

Обработчик прерывания по кванту

Обработчик прерывания по кванту посылает текущему процессу сигнала SIGXCPU — превышение лимита размера файла, если процесс израсходовал выделенный ему квант.

2 Пересчет динамических приоритетов

Системы семейства Unix и Windows являются системами общего назначения – это означает, что пересчитываться могут только пользовательские приоритеты. Остальные приоритеты являются статическими.

2.1 Windows системы

В системе Windows реализовано вытесняемое планирование на основе уровней приоритета, при котором выполняется готовый поток с наивысшим относительным приоритетом (базовый приоритет назначается соответствующему процессу).

Диспетчер настройки баланса сканирует очередь готовых процессов раз в секунду. Если он обнаруживает процессы, ожидающие выполнение более 4 секунд, то он повышает его приоритет и он перемещается в начало очереди, получая при этом процессорное время. Как только квант истекает, приоритет процесса снижается до базового приоритета. Если процесс не был завершен за квант времени или был вытеснен процессом с более высоким приоритетом, то после снижения приоритета процесс возвращается в очередь.

Для минимизации расхода процессорного времени, диспетчер настройки баланса сканирует только 16 позиций в очереди. Приоритет же повышает-

ся не более чем у 10 процессов за проход. При обнаружении 10 процессов, приоритет которых надо повысить, сканирование прекращается. При следующем проходе сканирование возобновляется с того места, где оно было прервано.

В системе предусмотрено 32 уровня приоритетов:

- приоритет 31 — наивысший приоритет;
- процессы реального времени имеют приоритет от 16 до 31;
- 0 - 15 — динамические уровни;
- 0 — зарезервирован для процесса обнуления страниц.

Звуковая карта имеет уровень приоритета выше, чем клавиатура и мышь, поскольку формирование звука — это процесс реального времени.

Уровни приоритета потоков назначаются с двух позиций: Windows API и ядра операционной системы. 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).

API-функция SetPriorityClass позволяет изменять класс приоритета процесса до одного из этих уровней. Затем назначается относительный приоритет потоков 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 Windows приведено в таблице:

	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 в динамическом диапазоне может быть повышен планировщиком по следующим причинам:

- 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;
- повышение приоритета проигрывания мультимедиа службой планировщика MMCSS (см. таблицу 2.1).

Таблица 2.1 – Рекомендуемые значения повышения приоритета

Устройство	Повышение
Диск, CD-RO M , параллельный порт, видео	1
Сеть, почтовый слот, именованный канал, последовательный порт	2
Клавиатура, мышь	6
Звуковая карта	8

MMCSS

Потоки, на которых выполняются различные мультимедийные приложения, должны выполняться с минимальными задержками. В Windows эта задача решается путем повышения приоритетов таких потоков драйвером Multi Media Class Scheduler Service — MMCSS.

MMCSS работает со следующими задачами:

- звук;
- возможность использования функции записи;
- воспроизведение звукового или видео контента;
- аудио профессионального качества;
- задачи администратора многооконного режима.

Одно из наиболее важных свойств для планирования потоков — категория планирования (Scheduling Category) — первичный фактор определяющий приоритет потоков, зарегистрированных в MMCSS:

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

Таблица 2.2 – Категории планирования

Категория	Приоритет	Описание
High (Высокая)	23-26	Потоки профессионального аудио 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 (Исчерпавшиеся потоки)	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

2.2 Unix системы

В данной работе рассмотрен классический UNIX, разработанный в начале 70-х годов. В современных системах могут использоваться различные алгоритмы планирования.

В системах UNIX ядро использует принцип вытесняемого циклического планирования. Это значит, что процессы, имеющие одинаковые приоритеты, будут выполнятся циклически друг за другом, и каждому из них будет отведен квант времени, обычно равный 100 миллисекундам. Более высокий по приоритету процесс, выполняющийся в режиме ядра, вытеснит более низкий по приоритету процесс даже если последний не использовал весь отведенный ему квант времени. Приоритет процесса определяется фактором «любезности» и фактором утилизации. Суперпользователь может

повлиять на приоритет процесса, используя системный вызов nice.

Дескриптор процесса pro с содержит следующие поля, которые относятся к приоритету:

- `p_pri` — текущий приоритет планирования;
- `p_usrpri` — приоритет процесса в режиме задачи;
- `p_cpu` — результат последнего измерения степени загруженности процессора;
- `p_nice` — фактор любезности.

Когда процесс находится в режиме задачи, значения `p_pri` и `p_usrpri` равны. Значение `p_pri` может быть повышено планировщиком для выполнения процесса в режиме ядра, а `p_usrpri` будет хранить приоритет, который будет назначен при возвращении в режим задачи.

Приоритеты ядра фиксированы и зависят от причины засыпания процесса. В таблице 2.3 приведены события и связанные с ними значения приоритета ядра в системе 4.3BSD.

Таблица 2.3 – Таблица приоритетов 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
`PWAIT`	30	Ожидание завершения процесса потомка
`PLOCK`	35	Консультативное ожидание блокированного ресурса
`PSLEP`	40	Ожидание сигнала

При инициализации процесса поле `p_cpu` равно нулю. На каждом тике обработчик прерывания инициализирует отложенный вызов процедуры

`schedcpu()`, которая уменьшает значение `p_cpu` каждого процесса исходя из фактора «полураспада», который рассчитывается по формуле:

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

где $load\_average$ — среднее количество процессов, находящихся в состоянии готовности к выполнению за последнюю секунду. Процедура `schedcpu()` пересчитывает приоритеты для режима задачи всех процессов следующим образом:

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

где `PUSER` – базовый приоритет в режиме задачи, который равен 50.

В результате, если процесс до вытеснения другим процессом использовал большое количество процессорного времени, то его `p_cpu` будет увеличен. Это приведет к росту значения `p_usrpri` и, следовательно, к понижению приоритета. Чем дольше процесс простаивает в очереди на выполнение, тем больше фактор полураспада уменьшает его `p_cpu`. Такая схема предотвращает бесконечное откладывание низкоприоритетных процессов.

Вывод

Обработчик системного таймера в операционных системах Windows и Unix выполняют схожие основные функции — инициализируют отложенные действия, выполняют декремент различных счетчиков времени.

Пересчет динамических приоритетов пользовательских процессов выполняется для того чтобы исключить их бесконечное откладывание.

Приоритет пользовательского процесса в классическом Unix может динамически пересчитываться, в зависимости от фактора любезности, `p_cpu` и базового приоритета, приоритеты ядра являются фиксированными.

При создании процесса в Windows, ему назначается базовый приоритет. Потоку назначается относительный приоритет согласно соответствующему процессу.