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 по дисциплине «Операционные системы»

Тема Обработчик прерывания от системного таймера в системах разделения времени

Студент Нисуев Н.Ф.

Группа ИУ7-52Б

Преподаватель Рязанова Н.Ю.

2024 г.

СОДЕРЖАНИЕ

1 Функции обработчика прерывания от системного таймера

1.1 UNIX

По тику:

— инкремент счетчика реального времени;

— инкремент счетчика времени, прошедшего с момента запуска системы;

— инкремент счетчика процессорного времени, полученного текущим процессором;

— декремент кванта.

По главному тику:

— выполнение функций, относящихся к работе планировщика, такие как пересчет приоритетов и действия;

— пробуждение (регистрация отложенного вызова процедуры *wakeup*, которая перемещает дескриптор процесса из списка «спящих» в очередь «готовых к выполнению») системных процессов, таких как *swapper* и *pagedaemon*;

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

  — $SIGALRM$ — сигнал посылается процессу по истечении времени, которое было предварительно задано через системный вызов alarm;

  — $SIGPROF$ — сигнал, посылаемый процессу по истечению времени, заданного в таймере профилирования;

  — $SIGVTALRM$ — сигнал, посылаемый процессу по истечении времени, заданного с помощью системного вызова setitimer.

По кванту:

— посылка сигнала $SIGXCPU$ активному процессу, если тот превысил выделенный для него квант процессорного времени [1].

1.2 Windows

По тику:

— инкремент счетчика реального времени;

— декремент счетчика времени отложенных задач;

— декремент кванта.

По главному тику:

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

По кванту:

— инициализирует диспетчеризацию потоков с добавлением соответствующего объекта в очередь DPC [2].

2 Пересчет динамических приоритетов

2.1 UNIX

Планирование процессов в UNIX основано на приоритете процессов. Каждый процесс обладает приоритетом планирования, изменяющимся с течением времени. Планировщик всегда выбирает процесс с наивысшим приоритетом. Для диспетчеризации процессов с равным приоритетом применяется вытесняющее квантование времени. Изменение приоритетов процессов происходит динамически на основе количества используемого ими процессорного времени. Необходимо проводить пересчет приоритетов, чтобы исключить бесконечное откладывание.

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 такие процессы, как процессы на звуковой карте.

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

1. 0 - 49 – зарезервированы для ядра (приоритеты ядра фиксированы)
2. 50 - 127 – прикладные процессы (приоритеты прикладных процессов могут изменяться во времени)

Структура *proc* содержит следующие поля, относящиеся к приоритетам:

1. p_pri – текущий приоритет планирования
2. p_usrpri – приоритет режима задачи
3. p_cpu – результат последнего измерения использования процессора
4. p_nice – фактор «любезности», который устанавливается пользователем

Планировщик использует *p_pri* для принятия решения о том, какой процесс направить 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 2.1).

Таблица 2.1 – 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

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 [1].

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.

2.2 Windows

В ОС семейства Windows процессу при создании назначается базовый приоритет. Относительно базового приоритета процесса потоку назначается относительный приоритет. Планирование осуществляется на основе приоритетов потоков, готовых к выполнению. Поток с более низким приоритетом вытесняется потоком с более высоким приоритетом,в тот момент когда этот поток становится готовым к выполнению. По истечении кванта времени, выделенного текущему потока, ресурс передается самому приоритетному потоку в очереди готовых к выполнению. Каждый поток имеет динамический приоритет. Это приоритет, который планировщик использует для определения того, какой поток следует выполнить. Изначально динамический приоритет потока совпадает с базовым приоритетом процесса.

В ОС семейства Windows используется 32 уровня приоритета:

1. от 0 до 15 — изменяющиеся уровни (уровень 0 зарезервирован для потока обнуления страниц);

2. от 16 до 31 — уровни реального времени.

Система может повысить и понизить динамический приоритет, чтобы обеспечить скорость реагирования и отсутствие нехватки потоков на время процессора. Система не повышает приоритет потоков с базовым уровнем приоритета от 16 до 31. Только потоки с базовым приоритетом от 0 до 15 получают динамический приоритет.

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:

1. реального времени (4);
2. высокий (3);
3. выше обычного (6);
4. обычный (2);
5. ниже обычного (5);
6. простой (1).

Исходный базовый приоритет потока наследуется от базового приоритета процесса. Процесс по умолчанию наследует свой базовый приоритет у того процесса, который его создал.

Таким образом, в Windows API каждый поток имеет базовый приоритет, являющийся функцией класса приоритета процесса и его относительного приоритета процесса. В ядре класс приоритета процесса преобразуется в базовый приоритет. В таблице 2.2 приведено соответствие между приоритетами Windows API и ядра системы приоритета.

В Windows также включен диспетчер настройки баланса, который сканирует очередь готовых процессов 1 раз в секунду. Если он обнаруживает потоки, ожидающие выполнения более 4 секунд, диспетчер настройки баланса повышает их приоритет до 15. Когда истекает квант, 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.

Текущий приоритет потока в динамическом диапазоне (от 1 до 15) может быть изменён планировщиком вследствие следующих причин:

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

1. повышение приоритета после завершения операций ввода-вывода;
2. повышение приоритета владельца блокировки;
3. повышение приоритета вследствие ввода из пользовательского интерфейса;
4. повышение приоритета вследствие длительного ожидания ресурса исполняющей системы;
5. повышение приоритета вследствие ожидания объекта ядра;
6. повышение приоритета в случае, когда готовый к выполнению поток не был запущен в течение длительного времени;
7. повышение приоритета проигрывания мультимедиа службой планировщика *MMCSS*.

Текущий приоритет потока в динамическом диапазоне может быть понижен до базового путем вычитания всех его повышений. В таблице 2.3 приведены рекомендуемые значения повышения приоритета для устройств ввода-вывода.

Потоки, на которых выполняются различные мультимедийные приложения, должны выполняться с минимальными задержками, причем выполнение потоков, работающих с аудио должно быть наиболее приоритетным, так как человеческий слух чувствителен к задержкам. В Windows эта задача решается путем повышения приоритетов таких потоков драйвером *MMCSS* — MultiMedia Class Scheduler Service.

Таблица 2.3 – Рекомендуемые значения повышения приоритета.

Устройство	*Повышение приоритета*
Звуковая карта	8
Клавиатура, мышь	6
Сеть, почтовый слот, именованный канал, последовательный порт	2
Жесткий диск, привод компакт-дисков, параллельный порт, видеоустройство	1

Одно из наиболее важных свойств для планирования потоков — категория планирования — первичный фактор определяющий приоритет потоков, зарегистрированных в *MMCSS*. Различные категории планирования представленны в таблице 2.4.

Функции драйвера 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*
Высокая	23-26	Потоки, связанные с обработкой аудио , работают с более высоким приоритетом, чем большинство потоков в системе, за исключением потоков, обеспечивающих выполнение критически важных системных задач.
Средняя	16-22	Потоки, поддерживающие выполнение активных приложений, например, таких как Windows Media Player, имеют умеренный приоритет.
Низкая	8-15	Потоки, не относящиеся к приложениям первого плана или задачам повышенного приоритета, выполняются с обычным уровнем приоритета.
Истощённые ресурсы	1-7	Потоки, использовавшие своё доступное время на процессоре, продолжают выполнение только тогда, когда потоки с более высоким приоритетом не готовы к работе.

ВЫВОДЫ

Обработчики прерывания от системного таймера в системах Unix и Windows выполняют похожие действия:

1. выполняют декремент кванта;
2. выполняют декремент счетчиков времени;
3. инициализирует отложенные действия, относящиеся к работе планировщика, такие как пересчёт приоритетов.

Обе системы являются системами разделения времени с динамическими приоритетами. Такой подход позволяет поддерживать работу процессов реального времени, такие как воспроизведение аудио и видео. Пересчёт динамических приоритетов пользовательских процессов выполняется для того, чтобы не допустить ситуации бесконечного откладывания.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Вахалия Ю.* UNIX изнутри. — 2003.
2. Внутреннее устройство Windows. 7-е изд. / М. Руссинович [и др.]. — Питер, 2022.