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»

на тему: «Функции обработчика прерывания от системного таймера и пересчет динамических приоритетов в системах разделения времени»

Студент ИУ7-53Б (Группа) _____ (Подпись, дата) Авдейкина В. П. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Рязанова Н. Ю. (И. О. Фамилия)

2023 г.

СОДЕРЖАНИЕ

1 Функции обработчика прерывания от системного таймера в системах разделения времени

1.1 UNIX/Linux

Обработчик прерывания от системного таймера **по тику** выполняет задачи:

— инкремент счетчика тиков аппаратного таймера;

— декремент кванта текущего потока;

— обновление статистики использования процессора текущим процессом — инкремент поля **c_cpu** дескриптора текущего процесса до максимального значения 127;

— декремент счетчиков времени до отправки на выполнение отложенных вызовов, при достижении счетчиком нуля происходит установка флага для обработчика отложенных вызовов;

— инкремент часов и других таймеров системы.

Обработчик прерывания от системного таймера **по главному тику** выполняет задачи:

— инициализация работы планировщика;

— пробуждение системных процессов **swapper**, **pagedaemon**, то есть регистрация отложенного вызова процедуры **wakeup**, которая перемещает дескрипторы процессов из списка «спящих» в очередь готовых процессов;

— декремент счетчика времени до отправки одного из следующих сигналов:

1) **SIGALRM** — сигнал, посылаемый процессу по истечении времени, которое задано функцией **alarm()**;
2) **SIGPROF** — сигнал, посылаемый процессу по истечении времени, которое задано в таймере профилирования;

3) **SIGVTALRM** — сигнал, посылаемый процессу по истечении времени, которое задано в «виртуальном» таймере.

Обработчик прерывания от системного таймера **по кванту** выполняет задачу: посылка сигнала **SIGXCPU** текущему процессу, если он превысил выделенный ему квант.

1.2 Windows

Обработчик прерывания от системного таймера **по тику** выполняет задачи:

— инкремент счетчика тиков;

— декремент кванта текущего потока на величину, равную количеству тактов процессора, произошедших за тик. В случае, если количество затраченных потомком тактов процессора достигает квантовой цели, запускается обработка истечения кванта;

— декремент счетчиков тиков отложенных задач;

— в случае, если активен механизм профилирования ядра, инициализация отложенного вызова обработчика ловушки профилирования ядра путем постановки объекта в очередь **DPS**: обработчик ловушки профилирования регистрирует адрес команды, выполнявшейся на момент прерывания.

Обработчик прерывания от системного таймера **по главному тику** выполняет задачу: инициализация диспетчера настройки баланса путем освобождения объекта «событие».

Обработчик прерывания от системного таймера **по кванту** выполняет задачу: инициализация диспетчеризации потоков — постановка соответствующего объекта в очередь **DPS**.

2 Пересчет динамических приоритетов

В ОС семейств UNIX и Windows динамически пересчитываться могут только приоритеты пользоавтельских процессов.

2.1 UNIX/Linux

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

В современных системах UNIX ядро является вытесняющим. Это значит, что процесс в режиме ядра вытесняется более приоритетным процессом, находящимся так же в режиме ядра. Это сделано для того, чтобы система могла обслуживать процессы реального времени, например, видео и аудио.

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

Приоритет процесса — это целое числом, находящееся в диапазоне от 0 до 127. Чем меньше число, тем выше приоритет процесса. Приоритеты ядра находятся в диапазоне от 0 до 49. Приоритеты прикладных задач в диапазоне от 50 до 127. Приоритеты ядра являются фиксированными величинами, а приоритеты прикладных задач могут изменяться во времени.

Изменение приоритетов прикладных задач зависит от двух факторов:

— фактор «любезности»;

— последней измеренной величины использования процессора.

Фактор «любезности» — целое число в диапазоне от 0 до 39 (по умолчанию 20). Чем меньше значение фактора «любезности», тем выше приоритет

процесса. Фактор «любезности» изменяется только суперпользователем с помощью системного вызова **nice**. Фоновые процессы имеют более высокие значения фактора «любезности».

Дескриптор процесса **struct proc** содержит следующие поля, которые относятся к приоритету процесса:

— **p_pri** — текущий приоритет планирования;

— **p_usrpri** — приоритет процесса в режиме задачи;

— **p_cpu** — результат последнего измерения степени загруженности процессора (процессом);

— **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** повышается планировщиком для выполнения процесса в режиме ядра, а **p_usrpri** используется для хранения приоритета, который будет назначен процессу, когда тот вернется в режим задачи.

При создании процесса **p_cpu** инициализируется нулем. На каждом тике обработчик таймера увеличивает это поле на 1, до максимального значения, которое равно 127. Каждую секунду обработчик прерывания инициализирует отложенный вызов процедуры **schedcpu()**, которая уменьшает значение **p_pri** каждого процесса исходя из фактора «полураспада». В системе 4.3BSD фактор «полураспада» рассчитывается по формуле 2.1:

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

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

Приоритеты для режима задачи всех процессов пересчитываются в процедуре schedcpu() 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.

Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может блокироваться. Приоритет сна — значение приоритета в диапазоне от 0 до 49, зависящее от события или ресурса, по которому произошла блокировка. Когда процесс «просыпается» после того, как был блокирован в системном вызове, ядро устанавливает приоритет сна в поле **p_pri**. В таблице 2.1 приведены значения приоритетов сна для некоторых событий в системе 4.3BSD.

Таблица 2.1 – Таблица приоритетов 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	ый 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

Если процесс в последний раз использовал большое количество процессорного времени, то его **p_cpu** будет увеличен. Это приведет к росту значения **p_usrpri**, из чего последует понижение 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 предпочтительнее процессам, осуществляющим много операций ввода-вывода, в противоположность процессам, производящим много вычислений.

То есть динамический пересчёт приоритетов низкоприоритетных процессов позволяет избежать бесконечного откладывания.

2.2 Windows

В Windows процессу при создании назначается базовый приоритет. Процесс по умолчанию наследует свой базовый приоритет у того процесса, который его создал. Относительно базового приоритета процесса потоку назначается относительный приоритет.

Планирование осуществляется только на основе 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 на выполнение.

В Windows используется 32 уровня приоритета, которые описываются целыми числами от 0 до 31:

— 0 — зарезервирован для процесса обнуления страниц;

— от 0 до 15 — динамически изменяющиеся уровни;

— от 16 до 31 — уровни реального времени;

— 31 — наивысший приоритет.

Уровни приоритета потоков назначаются в зависимости от двух разных позиций: Windows API и ядра Windows. Сначала Windows API сортирует процессы по классам приоритета, которые были назначены им при создании:

— реального времени (Real-time, 4);

— высокий (High, 3);

— выше обычного (Above Normal, 6);

— обычный (Normal, 2);

— ниже обычного (Below Normal, 5);

— уровень простоя (Idle, 1).

Затем назначается относительный приоритет отдельных потоков в рамках 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);

— уровень простоя (Idle, –15)

В таблице 2.2 показано соответствие между приоритетами Windows API и ядра системы.

Таблица 2.2 – Соответствие между приоритетами Windows API и ядра Windows

Класс приоритета	Real-time	High	Above	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

Планировщик повышает текущий приоритет потока в динамическом диапазоне (от 1 до 15) вследствие следующих причин:

— повышение приоритета владельцем блокировки;

— повышение приоритета после завершения операций ввода/вывода;

— повышение приоритета вследствие ввода из пользовательского интерфейса;

— повышение приоритета вследствие длительного ожидания ресурса исполняющей системы;

— повышение вследствие ожидания объекта ядра;

— повышение приоритета в случае, когда поток, готовый к выполнению, не был запущен в течение длительного времени;

— повышение приоритета проигрывания мультимедио службой планировщика **MMCSS**.

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

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

2.2.1 Диспетчер настройки баланса

В Windows включен общий механизм ослабления загруженности центрального процессора, который называется **диспетчером настройки баланса** и является частью системного потока.

Один раз в секунду этот поток сканирует очередь готовых потоков в поиске тех из них, которые находятся в состоянии ожидания около 4 секунд. Если такой поток будет найден, диспетчер настройки баланса повышает его приоритет до 15 единиц и устанавливает квантовую цель эквивалентной тактовой частоте процессора при подсчете 3 квантовых единиц. Как только квант истекает, приоритет потока тут же снижается до обычного базового приоритета. Если поток не был завершен и есть готовый к запуску поток с более высоким уровнем приоритета, поток с пониженным приоритетом возвращается в очередь готовых потоков.

Для минимизации времени своей работы диспетчер настройки баланса сканирует только 16 готовых потоков. Если на данном уровне приоритета

имеется больше потоков, он запоминает то место, на котором остановился, и начинает с него при следующем проходе очереди. За один проход диспетчер повышает приоритет только у 10 потоков. Если он обнаруживает 10 потоков, заслуживающих именно этого повышения, он прекращает сканирование на этом месте и начинает его с этого же места при следующем проходе.

2.2.2 MMCSS

Для того, чтобы мультимедийные потоки могли выполняться с минимальными задержками, драйвер *MMCSS* (*MultiMedia Class Scheduler Service*) временно повышают приоритет потоков до уровня, соответствующего их категориям планирования (таблица 2.4). Для того, чтобы другие потоки могли получить ресурс, приоритет снижается до уровня, соответствующего категории *Exhausted*.

Категории планирования — первичный фактор, определяющий приоритет потоков, зарегистрированных в MMCSS:

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

2.2.3 IRQL

Для обеспечения поддержки мультизадачности системы, когда исполняется код режима ядра, Windows использует приоритеты прерываний IRQL – Interrupt Request Level.

Прерывания обслуживаются в порядке их приоритета. При возникновении прерывания с высоким приоритетом процессор сохраняет информацию о состоянии прерванного потока и активизирует сопоставленный с данным прерыванием диспетчер ловушки. Последний повышает IRQL и вызывает процедуру обслуживания прерывания ISR – Interrupt Service Routine.

После выполнения ISR диспетчер прерывания понижает IRQL процессора до исходного уровня загружает сохраненные ранее данные о состоянии машины. Прерванный поток возобновляется с той точки, где он был прерван. Когда ядро понижает IRQL, могут начать обрабатываться ранее замаскированные прерывания с более низким приоритетом. Тогда вышеописанный процесс повторяется ядром для обработки и этих прерываний.

В ядре IRQL-уровни представлены в виде номеров от 0 до 31, где более высоким номерам соответствуют прерывания с более высоким приоритетом.

Уровни IRQL:

— 31 — наивысший (например, ошибка шины) ;

— 30 — питание;

— 29 — межпроцессорное прерывание;

— 28 — таймер;

— 27 — устройство n;

— ...;

— 3 — устройство 1;

— 2 — DPC (deferred procedure call)/dispatch);

— 1 — APC (asynchronous procedure call);

— 0 — низший.

ВЫВОД

Функции обработчика прерывания от системного таймера для операционных систем семейства UNIX и для семейства Windows схожи, так как они являются системами разделения времени. Схожие задачи обработчика прерывания от системного таймера:

— декремент кванта (текущего процесса в UNIX или потока в Windows);

— инициализация (но не выполнение) отложенных действий, которые относятся к работе планировщика (например, пересчет приоритетов);

— декремент счетчиков тиков (таймеров, часов, счетчиков времени отложенных действий, будильников реального времени).

Пересчет динамических приоритетов осуществляется только для пользовательских процессов, чтобы избежать бесконечного откладывания. ОС обоих семейств UNIX и Windows — это системы разделения времени с динамическими приоритетами и вытеснением.

В UNIX приоритет пользовательского процесса (процесса в режиме задачи) может динамически пересчитываться, в зависимости от трех факторов. Приоритеты ядра — фиксированные величины.

В 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.